2008-11-14

The Hybrid Game Architecture: Distributing Bandwidth for MMOGs While Maintaining Central Control

Jared L. Jardine
Brigham Young University - Provo

THE HYBRID GAME ARCHITECTURE: DISTRIBUTING BANDWIDTH FOR MMOGS WHILE MAINTAINING CENTRAL CONTROL

by

Jared Lee Jardine

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

December 2008

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Jared Lee Jardine

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Daniel Zappala, Chair
Date	Dan Olsen
Date	Mark Clement

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Jared Lee Jardine in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date	Daniel Zappala Chair, Graduate Committee
Accepted for the Department	Kent Seamons Graduate Coordinator
Accepted for the College	Thomas W. Sederberg Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

THE HYBRID GAME ARCHITECTURE: DISTRIBUTING BANDWIDTH FOR MMOGS WHILE MAINTAINING CENTRAL CONTROL

Jared Lee Jardine

Department of Computer Science

Master of Science

Current Massively Multi-player Online Games (MMOGs) have enormous server-side bandwidth requirements. The costs of providing this bandwidth is in turn passed on to the consumer in the form of high monthly subscription fees. Prior work has primarily focused on distributing this bandwidth using peer-to-peer architectures, but these architectures have difficulty preventing cheating, overwhelming low resource peers, and maintaining consistent game state. We have developed a hybrid game architecture that combines client-server and peer-to-peer technologies to prevent cheating, maintain centralized and consistent game state, significantly reduce central server bandwidth, and prevent lower capacity players from being overwhelmed. By dramatically reducing the bandwidth needed to host a game without introducing additional liabilities, our hybrid architecture reduces the costs associated with that bandwidth and allows MMOG developers to reduce the cost of monthly subscription

fees. In addition, because the central server will need less bandwidth per player, a single server is able to support considerably more concurrent players. Our experiments show that bandwidth can be reduced by up to 95% and a single server can support a game twice as large.

Contents

1 Introduction **1**

1.1 Client-Server Architectures 1

1.2 Peer-to-Peer Architectures 2

1.3 The Hybrid Game Architecture 3

2 Related Work **5**

2.1 DHT-Based Game Architectures 5

2.2 Hierarchical Architectures 7

2.3 Critique of Peer-to-Peer Architectures 10

3 The Hybrid Game Architecture **13**

3.1 Measuring and Monitoring Players 16

3.2 Advantages / Analysis 17

3.3 Security Concerns 18

4 Methodology **21**

4.1 The Game 21

4.2 Experimental Setup 22

4.3 Metrics 23

5 Results **25**

5.1 Varying the Game World Size 26

5.2 Varying Capable Player Percentage 29

5.3 Varying State-Changing Move Percentage 36

6 Conclusion **41**

6.1 Bandwidth Consumption 41

6.2 Latencies 42

6.3 Player Moves per Second 43

6.4 Future Work 43

Chapter 1

Introduction

Game developers spend a large amount of money on bandwidth to support a single MMOG (Massively Multi-player Online Game). This investment in game hosting bandwidth is a significant consideration alongside software development costs. These extra costs prevent many game developers from entering the field. The bandwidth costs for hosting an online game are tied to the common client server architecture. As a result, many researchers are developing peer-to-peer architectures that distribute these costs among the game players. Both of these architectures have advantages: the client-server architecture is simple and the game publisher can maintain control of the game, while the peer-to-peer architecture reduces bandwidth costs. Our hybrid-game architecture incorporates the advantages of both architectures.

1.1 Client-Server Architectures

The chief benefit of the client-server architecture is that control of the game can be maintained by the game publisher. The central server provides four critical controls: First, the server is able to control the sequence of events in game, for example deciding who got a treasure first. Second the server maintains control of game state, including character inventory and statistics. Third, the server can enforce game rules and prevent cheating by limiting what it tells the clients. For example, users should not know about where other users are unless the server tells them. Fourth, because the

server is controlled by the game company, it can be updated directly without the bother of patching all game users.

Another benefit of the client-server architecture is that it is generally simpler to implement than other architectures. Peer to peer architectures require a peer discovery mechanism, global event ordering, distributed storage, and distributed computation, but the central server of the client-server architecture easily handles all of these critical game elements.

The two main drawbacks of the client-server architecture are, first, the high cost of centralized bandwidth, and second, the client-server architecture is less robust due to its central point of failure. Server clusters are often used to mitigate failure risks. With these clusters all of the responsibilities of the server are distributed among the servers in the cluster. This is done with redundancies built in so that the cluster is fault tolerant.

1.2 Peer-to-Peer Architectures

The peer-to-peer architecture has two primary benefits. First, the peer-to-peer architecture eliminates several significant server related costs. Communication, storage, and computation costs are distributed to the peers. Second, the peer to peer architecture provides a fault tolerant platform. Several peer-to-peer game systems use distributed systems that are so robust they can handle as high as 50% node failure without measurably degrading system performance [7].

Peer-to-peer architectures can be complicated, but their most fundamental problem is maintaining coherent game state. This problem can be reduced to difficulties with global event ordering. How do the peers decide who picked up the treasure first or who was shot first by whom? This is a critical issue, because users are directly competing with one another, and the transposition of two events can be the difference between victory and defeat. With all peers authoritatively equal, it is difficult

to properly and efficiently order events. Most peer-to-peer architectures do not even attempt to tackle this problem, while the few that do increase latency and have not been widely adopted. Another difficulty is that peer-to-peer architectures often rely on time synchronization. This synchronization must meet real-time requirements. A popular time synchronization protocol, NTP, only gives assurances on the order of seconds, while game state needs to be synchronized on the order of 10s of milliseconds.

1.3 The Hybrid Game Architecture

Our hybrid architecture saves bandwidth by having the central game server process only state-changing moves. A *state-changing move* is a move where some action is performed that changes the general state of the game, for example, moving to a new game world region, attacking another player, or picking up treasure or equipment. The majority of moves in a game are *positional moves*, which occur when a player is moving around in the game without crossing any important boundary or performing any action other than moving within a game world region. Our hybrid architecture uses peer-to-peer methods to pass these moves among players, without using the server's resources. To make this distinction we assume that positional moves that do not cross a regional boundary or trigger a significant event do not affect the global game state.

To effectively use players to distribute positional moves, the central server must measure them. Once the server knows which peers have capacity it can delegate certain responsibilities to them. The central server then assigns capable peers to be regional servers, which will serve regional state updates to all players in their given region. This can account for tremendous bandwidth savings. Regional servers also need to pass moves on to the central server when a player crosses a regional boundary. These moves are sent to the server so that the new regional server can be notified. Otherwise the regional server handles all simple positional moves, while

state-changing moves are handled at the central server. This allows the central server to maintain control of game state, while offloading bandwidth costs associated with positional moves.

Our hybrid-game architecture possesses the key attributes of both the client-server and peer-to-peer game architectures. First, by utilizing peer resources it can dramatically reduce the bandwidth required to host a MMOG. Second, the distinction between simple positional moves and state-changing moves allows the central server to maintain control, which helps to maintain consistent game state and prevent cheating. Third, because our hybrid game architecture's regional servers act in many ways like a typical game server, the implementation of our architecture should be simple and low cost.

To evaluate our architecture, we create a simple multi-player online game in which two teams of players search for hidden treasure. The game is played in either client-server or hybrid architecture mode, which allows us to demonstrate the hybrid architectures advantages. We use emulation to restrict player bandwidth, so that we can test how our architecture performs when most players are unable to assist in distributing the game's bandwidth. Using this game, we run our experiments with 50 automated players on PlanetLab. Our experiments show that the hybrid architecture scales better than the client-server architecture, requires as much as 95% less bandwidth at the central server, lowers average latency, and almost doubles the throughput of the game. All of these benefits are dependent on having enough players that are capable of sharing the central server's load.

Chapter 2

Related Work

While the client-server architecture currently dominates commercially-deployed MMOGs, most academic work has focused on peer-to-peer architectures. The two main classes of peer-to-peer architectures are those that are based on a Distributed Hash Table (DHT), and those that rely on a hierarchical organization of peers.

2.1 DHT-Based Game Architectures

The core of several peer-to-peer architectures is what is known as a Distributed Hash Table or DHT [19] [20]. DHTs are used in order to effectively distribute storage across a large number of distributed machines. DHTs provide a basic storage and lookup service of the form *put(key, value)* and *value = get(key)*. The service also includes some security guarantees, extensive fault tolerance, caching, and other performance enhancements. Probably the chief of these is that DHTs typically guarantee $O(logN)$ bounds on item lookup. Many services have been built on top of DHTs, including file systems [7] [18] and multicast [4].

In a DHT-based game, all of the players of a game collectively form a DHT. To deliver game state to players, the system uses a publish-subscribe model. When a player enters a region of the game, they subscribe to the region's events. Then as events are generated in the region, they are broadcast to all players who have

a subscription to the region's events. When used in this way, a DHT has three primary benefits. First, the DHT gives a structure on which to base a purely peer-to-peer system. DHTs have proven methods for joining and leaving without damaging the effectiveness of the underlying architecture. Second, the DHT provides a fault tolerant system. Most DHTs can survive many peers failing simultaneously without interrupting playability on the other peers. Third, the DHT can guarantee state lookup in $O(logN)$ time. This is a great advantage when looking at games that are of the scale of a MMOG.

SimMud [14] is a game simulation that is based on a DHT [18]. In addition they use Scribe [4] to multicast the state changes. As players subscribe to a game region, SimMud builds them into a multicast tree to receive the region's published events. One of the problems they encounter is that certain messages take incredibly long to arrive, which the developers attribute to the Scribe multicast mechanisms [14]. Another liability of SimMud is its reliance on multicast trees. It is a competitive advantage for a player higher in the tree to receive moves earlier than those lower in the tree. This presents a real opportunity for cheating by unscrupulous peers. In addition SimMud is susceptible to catastrophic failures. This is not acceptable to game developers.

Mercury [2] uses a DHT with a coordinate system, but makes a fundamental change that allows it to support range based queries in a single request. Standard DHTs would be required to store an entity list for each discrete region. Players in the game would then need to subscribe to each region list that fell within their sphere of influence independently. As they received these lists they would then need to subscribe individually to each item in each of the lists. Mercury's range queries solves these cumbersome lookup chains. However, to support these range based queries, Mercury is only able to claim $O(log^2 n)$ bounds.

Colyseus [3] is an architecture built on top of Mercury's [2] DHT, that distributes the various game items uniquely. Instead of needing to access the original objects all of the time, Colyseus stores local copies. A primary concern with Colyseus [3] is view inconsistencies, which occur when one player sees a different game world than another. Another problem with it is that the experiments do not take each node's available bandwidth into consideration. Because they are treating each peer equally, a peer can become overwhelmed when it does not have sufficient bandwidth.

The Zoned Federation [12] is another game architecture that relies on the use of a DHT. Rather than basing all traffic on the DHT, the Zoned Federation utilizes the DHT as a backup system. First, each game region or zone is stored in the DHT. Then, the first player that enters a zone becomes the zone owner until they leave. The zone owner acts as the central server for the zone. All subsequent players then send their moves to the zone owner while the zone owner sends all of the players region state updates. During a session on a zone, the zone owner periodically stores the zone's state back to the DHT. In this way, if the zone owner fails or leaves, the other nodes have a base point to work with. The chief drawback of this approach is that each in game zone will be limited to the capacity of the zone owner. If a dial up client visits a zone first they will still have the responsibilities of zone ownership, although they may not have the capacities to support any other clients.

2.2 Hierarchical Architectures

Another set of peer-to-peer architectures relies on the peers to be organized into some form of hierarchy. These hierarchies handle the problems of original peer discovery, adjacent region discovery and other top level game concepts [9]. Most of this work assumes an overall hierarchy and focuses on the communication and the ordering of events among peers within a single game world region. One of the reasons that they rely on a overarching hierarchy is that within the game region each player updates

every other player. This means that they generally scale at $O(logN^2)$. By restricting these all-to-all groups to the number of players within a single game region they can avoid being unbearable. In addition, by passing messages directly between players these architectures can reduce the message latency considerably.

MiMaze [11], a pioneer in distributed game architectures, relies on a bucket synchronization method, in which virtual time is divided into buckets and moves are identified by the bucket in which they take place. When every player's move has been received, the players can move forward. Bucket synchronization methods force the players to move forward at the speed of the slowest connection. This is also vulnerable to attacks made by those who are just trying to slow the game down for others by delaying their communications.

Lockstep [1] builds on the bucket synchronization method in MiMaze to prevent timing cheats. With bucket synchronization alone, any player can see other players moves prior to committing their move. With Lockstep each player sends a hash of his move for the given bucket to every other player. When players have received a hash of every other player's move, they then send the plaintext move to all other players. In this way the players must commit their moves for a given time bucket before then can find out other players moves. By requiring each player to receive an update from every other player in the group, latency is bounded by the slowest peer. However, in Lockstep, not only do you need to wait for the slowest peer, you need to wait for him twice, since you will first send your hashed move, and then follow it up with your plaintext move.

NEO [10] extends Lockstep by adding a bound on the maximum time allowed for each round. For each move after the first round, each player sends their move along with a list of those players from which it received the last move within the given round time. Then, the majority vote is used to determine if each player's move has been received within the round time and will be accepted. Those moves which

are accepted are processed, while those that failed to get to a majority of the other players are rolled back. In essence, it provides a guarantee on the speed with which the game will progress by choosing to ignore move messages that took too long. While the game will progress at the given speed, a poor performing player may proceed at a much slower pace.

Trailing State Synchronization (TSS) [6] is an unique event ordering architecture that uses a time based state "roll back" to cope with inconsistencies. In TSS the current state is based on guesses of where other peers will be based on current position and velocity. As messages arrive from each peer, their position and velocity are updated. When TSS detects an inconsistency, it rolls back to a prior game state and continues computation from there. Unfortunately, TSS relies on millisecond clock synchronization between all players, so if an inconsistency occurs within a few milliseconds of delay, the state must be rolled back. In addition, rollback states require ongoing computation as a complete, alternate version of the game. This greatly increases the computational complexity required by the game.

Load Balanced Trees [21] assigns each region a "responsible node". All events in a region are sent to the responsible node for that region, and the responsible node forwards the region state to all who have subscribed to the region's updates. This responsible node essentially becomes the central server for the region. The main contribution of this work is in dealing with scaling within a given region. When enough peers are in a given region that the responsible node becomes overloaded, a Load Balanced Tree of peers is constructed. The responsible node then disseminates region updates through this tree.

Another hierarchical architecture uses Software Multicast Reflectors [17] as quasi-servers. This structure is set up in a hierarchy and relies on in-game move being assigned varying priorities. As a player subscribes to a game region, they also stipulate at what priority they have assigned for events from that region. Each region

then has a Software Multicast Reflector, which is best described as nested multicast trees. The root of the multicast reflector is the root of the highest priority multicast tree, while each subsequently lower priority tree is connected as a leaf in the previous tree. For example, an in-game move is published, and enters at the root of the highest priority multicast tree. It then is sent to every subscriber at that priority level, one of which will be the next lower priority multicast tree. This process is repeated until all of the priority levels are handled. Generally a player will subscribe to the events that are closer at a high priority, while those at a distance could be subscribed to at a low priority. This is intended to allow those who view actions with a higher priority to receive those events first. Unfortunately it relies on additional dedicated resources to form the heart of its hierarchical structure. This prevents the realization of the bandwidth savings generally associated with peer-to-peer architectures.

2.3 Critique of Peer-to-Peer Architectures

Whenever choosing to spend the time necessary to formulate and test a new solution it is necessary to describe why the current solutions are insufficient. With all of the game architectures that are built on peer-to-peer architectures we feel a specifically pressing need to describe why we believe that purely peer-to-peer architectures are not the right solution.

First, DHTs do not place a bound on storage update time. Players and creatures are moving around, picking up, using, and dropping equipment as well as damaging themselves and others on a constant basis. These changes need to be recorded swiftly and the true state needs to be available to all of the players; so that they can maintain consistent state. A DHT does not bound the time it takes for a write to be performed so that every interested party is able to access it. This question has not been addressed primarily because the original purpose of DHT was to create a file system or similar file sharing architecture that did not depend on quick writes.

Second, peer-to-peer architectures struggle to enforce access control and security - anyone can join at any time. Several peer-to-peer architectures do not ensure that the nodes that are placed in a position of responsibility will be fair. In the Zoned Federation of Servers [12] approach the zone and its players are totally at the mercy of the zone owner. In the context of a MMOG the zone owner can unfairly pillage the equipment and resources of the zone, and its members. Likewise, in a publish-subscribe design, nothing prevents a player from subscribing to information that they should not be able to obtain. This can allow certain players to see through walls, or track other players outside of the rules of the game. Finally, there is no scrutiny of those publishing new events or data to the system.

Third, peer-to-peer architectures assume that each node in the group of peers has identical capacities. While this assists in simplifying the problem, it does not present an accurate view of MMOGs customers. Relying on these assumptions will either drive the architecture to the least common denominator, or else lock out nodes that do not have sufficient resources to handle the load. If 30% of the games clients access the Internet via a dial-up connection, the game developer does not want to close the door to the revenue that these nodes provide, nor do they want to degrade their game service to the other 70% of their clients to maintain the peer-to-peer architecture's attributes. For instance, a dial-up client should not serve as a responsible node in the Load Balanced Trees [21]. Neither should an entire region of players in Lockstep [1] be limited to a dial-up client's latency.

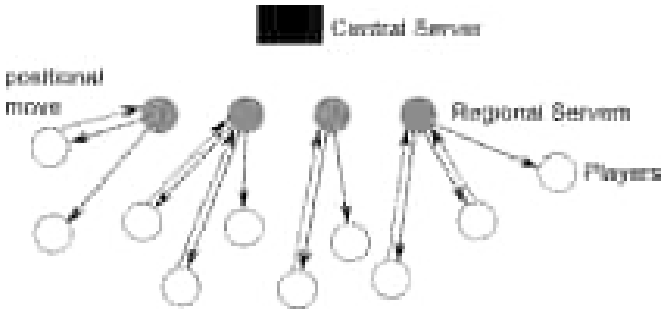
Chapter 3

The Hybrid Game Architecture

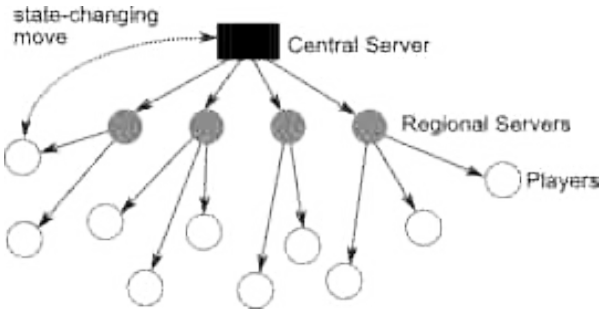
Our hybrid game architecture provides security by using a central server to control access to the game state. This provides the server a great deal of power, because the ultimate goal of most games is to acquire or control game state in the form of treasure, power, or the lives of other players. Any time players must interact with game state, the server acts as the arbiter, ordering state-changing events and issuing state updates. The central server also registers players, monitors their actions, and can evict them from the game if it detects that they have misbehaved.

The hybrid game architecture achieves scalability by using peers to distribute game updates. The server divides the game into regions and assigns a player to distribute updates for that region. Whenever a player makes a positional move, it sends the move to its regional server, which distributes the move to the other players in the region, illustrated in Figure 3.1(a). Whenever a player makes a state-changing move, it sends the move directly to the central server. If the state change is accepted, the central server sends the player the result and also issues an update to the appropriate regional server, which in turn distributes this update to the players in its region, illustrated in Figure 3.1(b).

For example, a player walks from one side of the foyer of an inn to the other. This move is a simple positional move and is sent from the player directly to the region server, which updates the region state and sends the update to all of the region's players who can see the player walking. The same player then picks up a



(a) Positional Moves



(b) State-Changing Moves

Figure 3.1: Distributing Moves

coin left on the inn's floor. This move is state-changing is sent from the player directly to the central game server. The central server then updates the player's inventory and send the player their new state. The central server will also pass the region server a region update that shows that the coin is no longer on the floor. This message is passed from the region server to the region's players who can notice it.

Even with region servers, players always maintain a direct link to the central server to be used for access control and for region navigation. When a player first accesses the system they validate through the central server. This link is then preserved and used when a player is moving from one region to another. For example, when a player approaches a new region his current region server includes that information as a state-changing move that is passed on to the server. The server then communicates the address of the new region server directly to the player so that it can switch. In addition, players can inform the central server of problems with region servers. The central server will also handle cases when a region server is overwhelmed or fails.

(a) Undivided (b) Divided into Regions

Figure 3.2: Game World Map

Before region servers can be assigned, the game world needs to be divided into regions. Different games world maps need to be divided differently. The game world prior to region division is shown in Figure 3.2(a), while a divided game world is illustrated in Figure 3.2(b). Game regions will generally have one or more paths to other regions, these paths will probably be an alley, doorway, or other similar transition area specific to the map. The game world is divided so that regions will be large enough to generally have more than one player node, but small enough that they can be handled by a region server without overwhelming it.

With region servers donating system resources and bandwidth to help the architecture's performance, special care should be given to ensure that they are not abused. If a region server's player experience is degraded because they are overwhelmed, they will not want to participate in the architecture. To give capable nodes incentive to share their resources, game developers may want to offer lower subscription fees to those who share their resources as region servers.

3.1 Measuring and Monitoring Players

To determine whether players can act as region servers, the central server needs to classify the players according to bandwidth and latency. For the bandwidth classification the server asks the player to classify their bandwidth. In addition, for those who are capable, players will be asked if they would like to opt in and share their bandwidth as a region server. If they do not wish to share their resources as a region server or they have a connection that will not support acting as a region server they will remove themselves from the region server pool. Those that report that they are capable then move on to be classified according to latency. To measure latency we have the central server ping the connecting player following sound Internet measurement strategies [16]. We then take the maximum latency returned by the ping and use it as that player's latency. If a node has a latency over 200 milliseconds they are classified as incapable and are never used as a region server. If their latency is less than 200 milliseconds and there is currently a region that is being hosted by the central server, the capable player is promoted to the region server over that game region. If they have wrongfully claimed to be capable, the server will shortly recognize that they are bogged down with even the smallest region and remove them from the pool.

The server picks players to act as region servers as capable players join the game. Although smarter methods can be determined in the future, our current architecture simply picks the next lowest region number and hands it out to the next capable player. Although the server currently prevents players from acting as region servers if they connect to the Internet from behind a NAT or similar mechanism that prohibits them from being connected to by other players, there are some emerging methods that may make this a non-issue [15].

While our hybrid game architecture leverages player node capacities to lower bandwidth costs, this is not accomplished without creating some dependency on these nodes. It is important that our architecture is able to smoothly transition when a

region server fails or becomes overwhelmed. To achieve this, each region server sends status messages to the central server every 150 ms if they have not already sent a state update within the last 150 ms. If the central server goes for three of these 150 ms intervals without a message from the region server, it knows that the region server has failed or become overwhelmed and can quickly step in to take over the region server duties. The central server then informs the region's players to send their positional moves to it until it appoints another player-based region server. This enables players to recover from an unresponsive region server.

3.2 Advantages / Analysis

Our hybrid game architecture produces most of its bandwidth savings by our approach to message passing. There are two key ways in which we benefit over the client-server architecture. First, by distinguishing between simple positional and state-changing moves, and then handling positional moves at the region servers, the server does not need to devote any bandwidth to these moves. Second, by introducing the region server into the architecture, the server's bandwidth is limited more by the size of the game map than by the number of players in the game.

The hybrid game architecture saves bandwidth for the central server by distinguishing between simple positional and state-changing moves. This savings is tightly coupled with the specific MMOG. When players are moving around without performing any state-changing actions, these simple positional moves can be served by the region servers without changing the global game state. The bandwidth savings of this approach is directly tied to the average proportion of simple positional moves within a specific game. For example, if 33% of a game's moves are simple positional moves, the central server requires 33% less bandwidth.

The hybrid game architecture also saves bandwidth for state updates by using capable players as region servers. Studies have shown that a player's move messages

are approximately 20 bytes, and that state update messages are approximately 320 bytes [13]. Consider the case when a player sends a state changing move to the central server. First, In the client server architecture, the player sends a 20 byte move message to the server and receives a 320 byte state update message back from the server. The central server in this case processed 340 bytes. Next, in our hybrid game architecture a player sends a 20 byte state changing move to the central server, which then sends a corresponding 20 byte update message to the correct region server, which finally sends a 320 byte state update message to the player. The central server in this case processed only 40 bytes, for a savings of 300 bytes or 88% of its required bandwidth. Unfortunately, it isn't always this simple. There are small messages that add overhead, and there may not be enough capable players to fill all of the region server roles. In regions where there is not a player acting as the region server, our hybrid-game architecture reverts to the client-server architecture.

3.3 Security Concerns

In most peer-to-peer architectures malicious clients can be a real problem. By placing the nodes into a system with central server controls, most of this problem is removed. The central server maintains access control, and thus can ban or otherwise restrict access to a misbehaving client. In addition, the clients have no control of game state. This prevents a malicious client from purposely altering the game state to benefit themselves or to attack the system. While a client can mount a denial of service on either the central server or a region server, this risk exists in all architectures that rely on a central component. This has not prevented commercially deployed MMOGs from utilizing the client-server architecture, nor would it prevent the use of our hybrid game architecture.

The first attack that we want to address is a player falsely manipulating the game state. At first glance our hybrid game architecture may appear vulnerable to

this attack because we place players in a position of responsibility over a region of the game. However, region servers do not serve state-changing moves. All moves that change a player's stored attributes or inventory are sent directly to the central server. This includes any move that changes a monster or player's inventory, experience, characteristics, or profile in any way. All that a region server does in response to the central server's state change message is adjust the region state update to reflect any visible changes. Otherwise, the region server only deals with simple positional moves. This severely restricts the attacks that a region server can perform.

Second, consider a region server using their position of responsibility to gain an advantage over other players. As a safeguard against this, our hybrid-game architecture does not allow a region server to play in the region that it is serving. When the central game server sees that the player is moving to the region that they are serving, they will be replaced as the server of that region. This removes the region server's temptation to exploit the region that they are serving to benefit their player. While a game region server and a player within that region could possibly collude to the benefit of the player, region servers cannot know before they join the game which region they will be selected to serve. Also, by limiting the region server's influence on state changing moves, the only benefits a player could gain from this collusion are: possibly moving faster through the region than is allowed by game rules; seeing portions of the region that they should not be allowed to see; and having their position somehow masked to other players of the game. To prevent these collusion attacks, the server can periodically audit region state to check that players are moving appropriately.

Finally, a region server may choose to mount a denial of service attack on a given player within its region. If the region server does not properly update a player, the player will be able to detect this action. Players expect a region state update every 200ms. If they are not getting their messages or feel that they are being

abused, they can lodge a complaint via their direct connection to the central server. Whenever players in a region lodge a complaint against a region server, that server will be removed. The central server will log all of the complaints that it receives from players, and if players complain much more than usual, there future complaints will be ignored. In addition, if game developers offer reduced subscription fees to region servers they will have a financial motive to play fairly and treat players properly.

Chapter 4

Methodology

We created a simple MMOG to test our hybrid-game architecture. It can run in either client-server or hybrid-game modes. We choose to compare our architecture with a client-server architecture because the client-server architecture dominates deployed MMOGs. By comparing against client-server we are making a case for the deployment of our architecture as its replacement. We measure our performance based on server bandwidth consumption, network latency, and average player moves per second.

4.1 The Game

In our MMOG players quest for treasure and bring it back to their start location. The world map is divided into between 5 and 40 regions. The central server then assigns players to one of two teams. Their given team determines the region that the players are assigned to, with even numbered players being assigned to the first region and odd players being assigned to the last region. The start locations begin at the top of the far end of the starting region and move down the far edge. After the player enters the game world, they query the central server as to the region that their treasure is in. Players then send move messages to, and receive region state updates from their region server. When a player reaches a region boundary they send a region move message to both the central server and their region server. Once the player reaches

the region where their treasure is placed they query the central server for the exact location of the treasure. After the player has picked up the treasure, a state-changing move, they move at 1/2 their normal speed. Each time a player returns to their start location with a treasure, a point is added to their team's score. A replacement treasure is then generated and randomly placed, and the cycle begins again. For the purposes of this game, players are programmed with automated behaviors.

4.2 Experimental Setup

Our tests run on PlanetLab, a global network of computers used for Internet research. PlanetLab allows us to perform actual Internet experiments so that we do not need to rely on a simulator. PlanetLab nodes generally have broadband connections; to properly represent players of different capacities, we use an emulator that lets us limit a player's bandwidth to a fixed value. This allows us to test with different numbers of players representing dial-up and broadband clients. Each test is run on both the client-server architecture and our hybrid-game architecture five times to increase the statistical significance of our results.

Our experiments are divided into two different stages, the ramp-up followed by the steady state. During ramp-up players join the game with an exponential arrival rate averaging one player per second. After all of the players have joined the game the experiment enters steady state, which lasts for ten minutes. Our bandwidth measurements are taken during steady state because the server is under the most strain during this period. This allows us to illustrate the capabilities of our hybrid game architecture with everything up and running. While an average client duration has been measured to be around 180 minutes [13], we chose a 10 minute period. This gives us a sufficiently large interval to measure, while allowing us to fit within the bandwidth limits enforced by PlanetLab.

In our game, move messages and state update messages are very small, because our game is simple. We choose to pad our messages so that they represent a more realistic MMOG message load. In accordance with the sizes measured in [13] we padded our move messages to 20 bytes and our state update messages to 320 bytes. Every move message and region state update is time-stamped and logged. Players also measure and log the time between the sent move and the received region state update as a the player's experienced latency.

4.3 Metrics

Whenever a player sends a move message, that player first records the message and a time stamp. They then set a timer and wait to receive a corresponding region state update from their server. When it is received, the player records the time it took as the move's latency. Each player sends moves no faster than once every 200 milliseconds. As the moves are received either by the central server or a region server, they are also logged and time stamped. At the end of an experiment the data is then taken and logged for both the messages in and the messages out. Using these message logs the bandwidth used both at the central server and region servers is calculated. To ensure that the ramp-up portion of our experiment doesn't skew our results we only use data from the last 600 seconds of the central server, region servers and players.

Although our experiments are primarily focused on bandwidth usage at the central server we also calculated latency and average player moves per second. The hybrid game architecture is successful if it lowers the bandwidth used at the central server. The other two metrics ensure that our central server bandwidth measurement is useful. If the latency in our hybrid game architecture is too much, then regardless of how much bandwidth savings there is the game is unplayable [8]. Also if the central server bandwidth is reduced, but the player moves per second is also reduced it shows that the bandwidth was reduced because the game did not progress at the same pace.

On the flip side if bandwidth is reduced, latency is kept within a playable range, and player moves per second increases it shows that our hybrid game architecture is being successful.

Chapter 5

Results

We ran three experiments: varying the size of the game world, varying the percentage of players with broadband connections, and varying the percentage of moves that were state-changing. In our first experiment, we vary the size of the game world, using 50 players, 24% broadband players. For our second experiment we varied the number of players with a broadband connection, also using 50 players, but always on a game world with 8 regions. In our final experiment we vary the percentage of moves that are state-changing, with an 8 region game world and 24% broadband players. In this experiment we are limited to 35 players because PlanetLab was undergoing major updates and over 100 of the nodes we rely on for our tests were unresponsive. While we prefer to have 50 players for all of our tests we believe that the results from this final experiment with 35 players is still sufficient to show the benefits of the hybrid game architecture. We run each experiment ten times at each setting, with five runs using the client-server architecture and five runs using the hybrid game architecture. In each experiment the players are restricted to send moves no faster than one every 200 milliseconds or five per second.

All of our bandwidth results show the central server in the client server architecture, the central server in the hybrid game architecture and the region servers in the hybrid game architecture. In addition, we graph all of our results with the 25th and 75th percentiles.

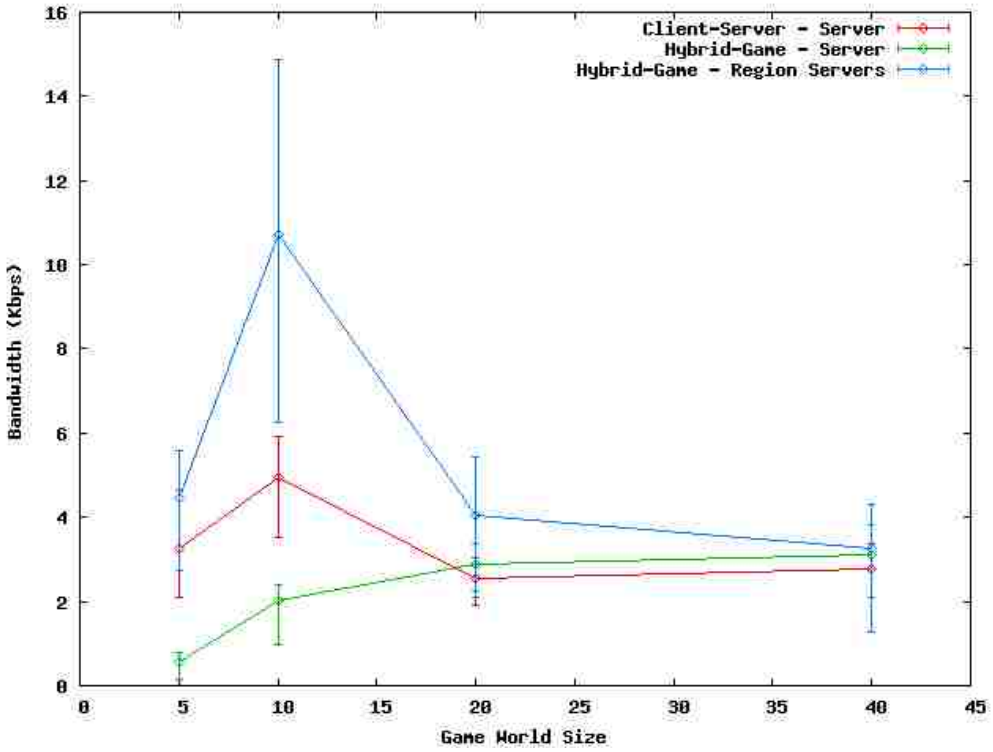


Figure 5.1: Map Region Density Experiment: Incoming Bandwidth

5.1 Varying the Game World Size

Our first experiment varies the game world size. We vary the size of the game world to see how our hybrid game architecture performs as the average regional player density shifts. This is because it is far simpler for us to shrink the game world than it is to increase the number of players we use in our tests.

For this experiment we use a capable player percentage of 24%, which gives us an average of 12 capable players for each run. We start with a game world that is 40 regions large. We then progressively reduce the game world size until we reach a game world of 5 regions. This includes the following game world sizes: 40, 20, 10, and 5. Because our experiments have 50 players in the game, these game world sizes yield average regional player densities of 1.25, 2.50, 5.00, and 10.00.

Our results show that as the game region player density increases the central server in the hybrid game architecture uses less bandwidth. The pattern found in

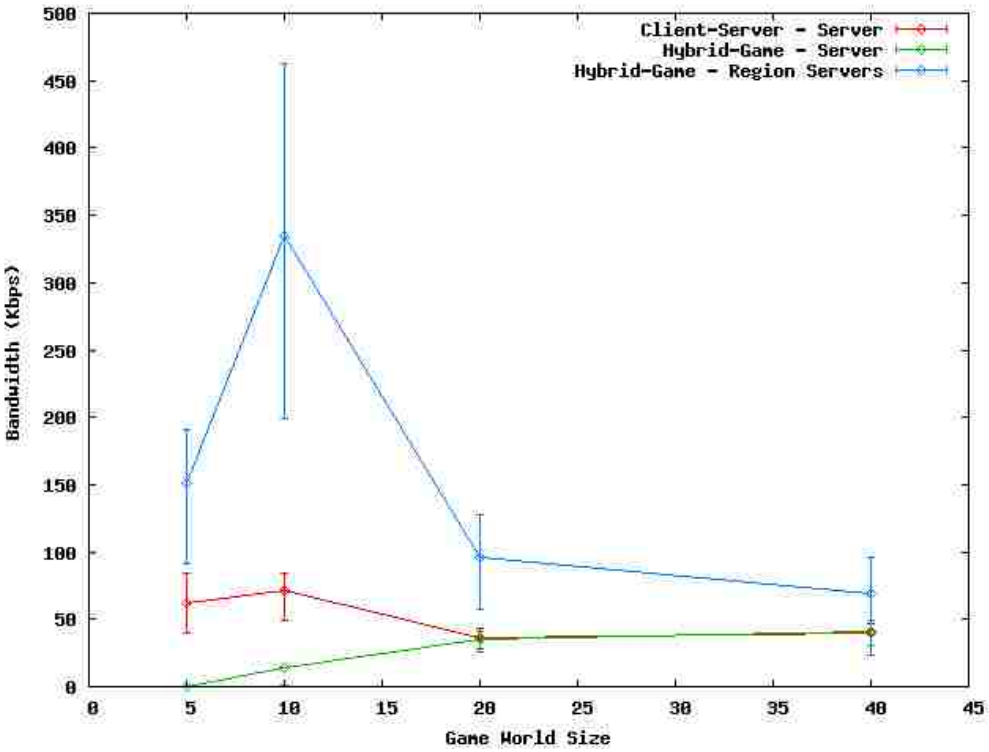


Figure 5.2: Map Region Density Experiment: Outgoing Bandwidth

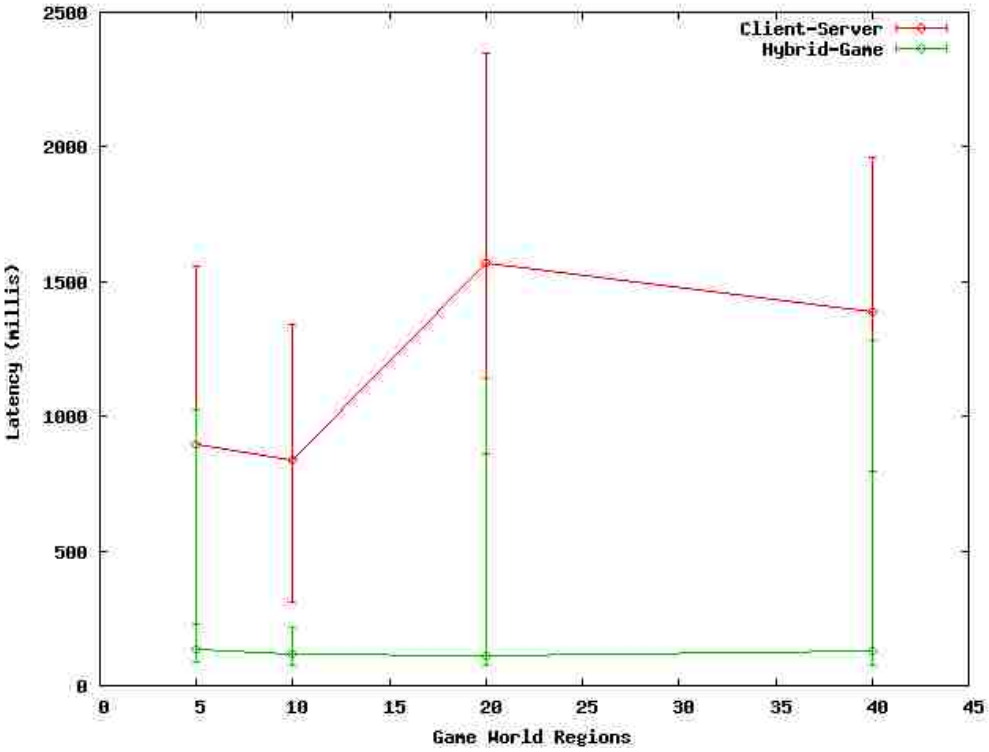


Figure 5.3: Map Region Density Experiment: Latency

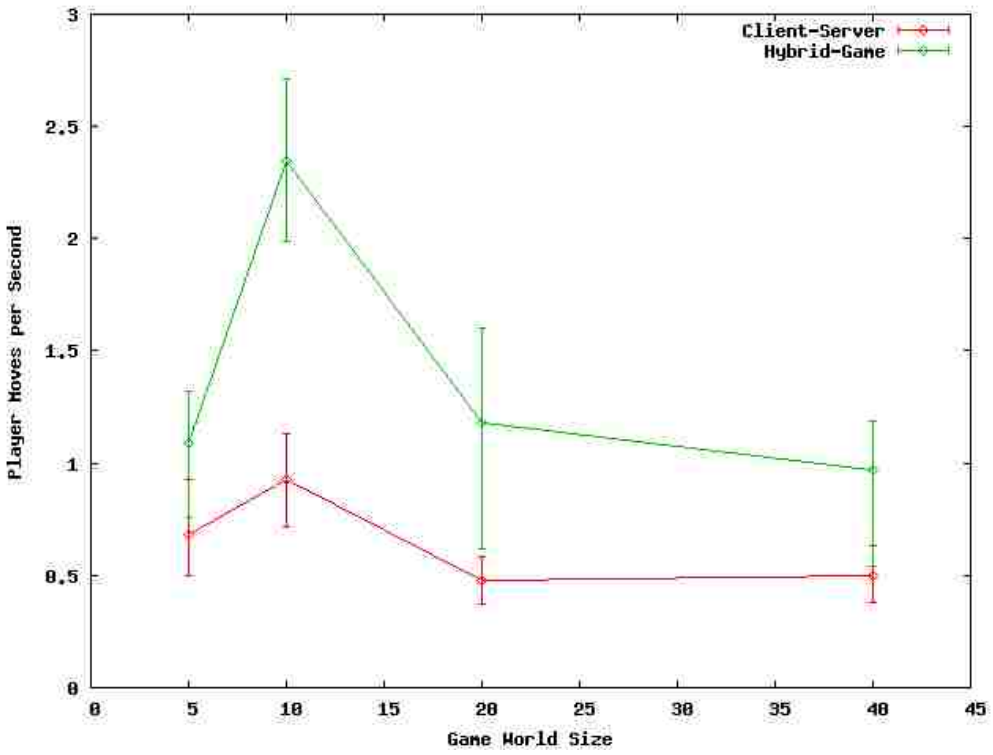


Figure 5.4: Map Region Density Experiment: Moves per Second

Figure5.1 and Figure5.2 illustrates that both incoming and outgoing bandwidth are behaving similarly. They also show that the central server is being overwhelmed in both the client-server architecture and our hybrid game architecture when the density is low. We have, on average, 12 capable players in each test. This means that on tests with low region density, with 40 and 20 regions, the central server is only offloading a portion of its regions. For mkoexample, in our tests with 40 regions, 28 regions are served by the central server because the 12 capable players are only assigned one region each. However, in our hybrid architecture tests with high player density, our 10 and 5 regions, the central server only needed to serve state-changing moves, and all of the positional moves were handled by region servers. These tests show the real bandwidth savings of the hybrid-game architecture. Not only is the central server's bandwidth reduced, but the total game throughput soars. Finally, in the tests with the game world size reduced to 5 regions we anticipated the most savings. Our central

server was free to only serve state changing moves, but all of our simple positional moves were being served by only 5 capable players, and it appears that they were overwhelmed. Although the bandwidth savings of our hybrid-game architecture as a percentage is improved, the total game throughput is less than in the 10 region tests.

The latency results for this experiment are shown in Figure 5.3. It shows that the latency in the hybrid game architecture is significantly reduced. It also shows that the variance in latency is best in our test with 10 game regions, where the density is high and all regions are served by region servers. This data also reinforces that the central server in the client-server architecture was overwhelmed. In the client-server architecture the median latencies are all over 700 milliseconds. In the hybrid-game architecture, while the 75th percentile latencies are still higher than we would have liked, the medians are around 200 milliseconds. One might expect the added level of hierarchy would give us a slight latency disadvantage, but that is really only for state-changing moves, and because the server in the hybrid game architecture is free from serving positional moves it can serve the state-changing moves quickly.

Figure 5.4 illustrates that the game in the hybrid-game architecture is moving much more quickly than the client-server game. The ideal for these experiments would be an average of 5.0 moves per second for every player. Both the hybrid-game architecture and the client-server architecture fall short of that ideal. In this experiment the client-server architecture is averaging between .5 and 1.0 moves per second, while the hybrid-game architecture is averaging between 1.0 and 2.25 moves per second. This shows that the hybrid-game architecture is able to get substantially more game throughput, and is thus doing a better job of serving the game.

5.2 Varying Capable Player Percentage

Our hybrid-game architecture depends on capable players being promoted to act as region servers. To tell to what degree this reliance is a liability, our second experiment

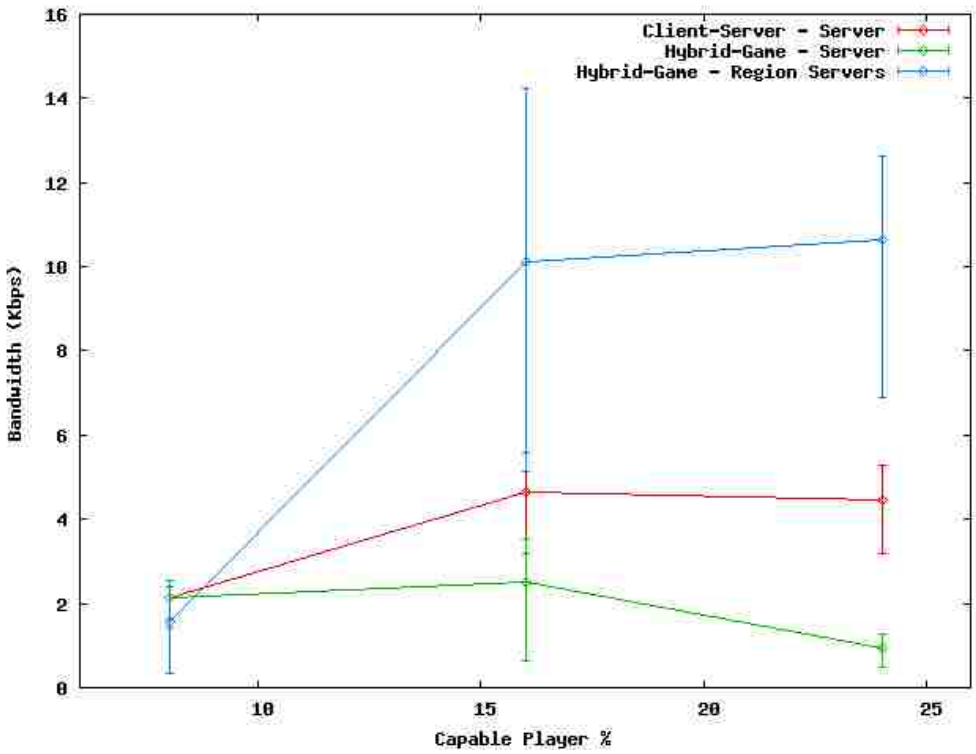


Figure 5.5: Player Capacity Experiment: Incoming Bandwidth

varies the portion of players that are capable of being region servers. To illustrate the hybrid-game architecture's limitation we use a game world size of eight regions and the following capable player percentages: 8%, 16%, and 24%. This gives us test results with an average of 4, 8, and 12 players capable of being regional servers.

Broadband access should be more prevalent in the future; studies have shown that as residential users increase their bandwidth capacities they are more likely to participate in resource sharing peer-to-peer architectures [5]. As broadband and fiber-optic Internet access becomes increasingly prevalent, the portion of players that would be qualified as capable players should increase.

These results dramatically illustrate how the hybrid-game architecture saves bandwidth as the number of capable players increases. The pattern is once again the same for both incoming bandwidth Figure 5.5, and outgoing bandwidth Figure 5.6. In the first tests we use a capable player percentage of 8. This translates to an

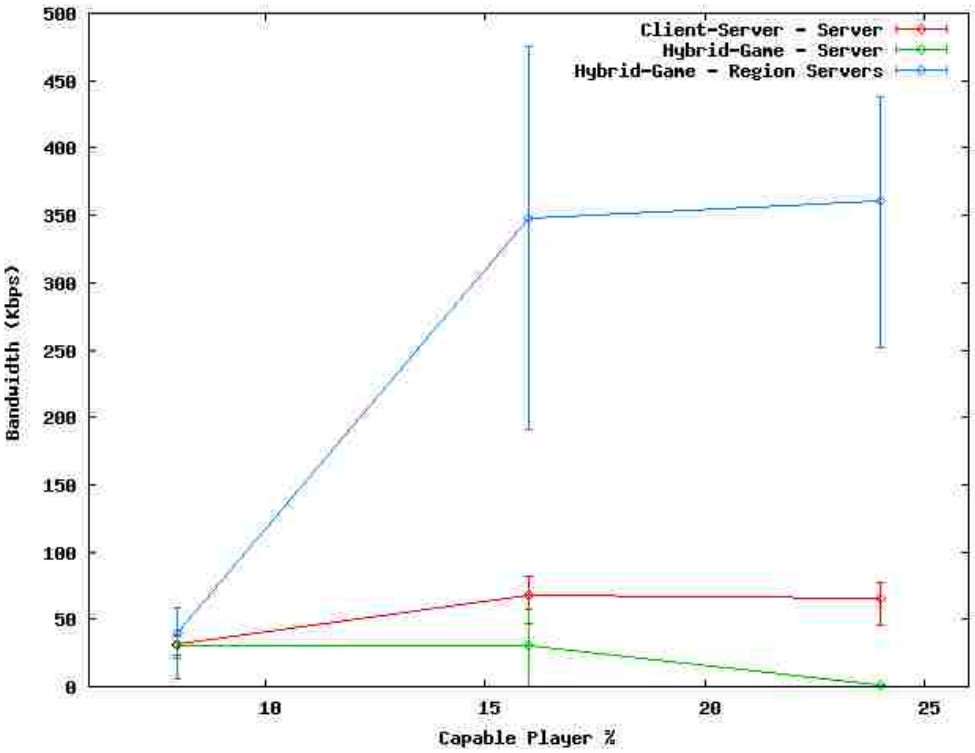


Figure 5.6: Player Capacity Experiment: Outgoing Bandwidth

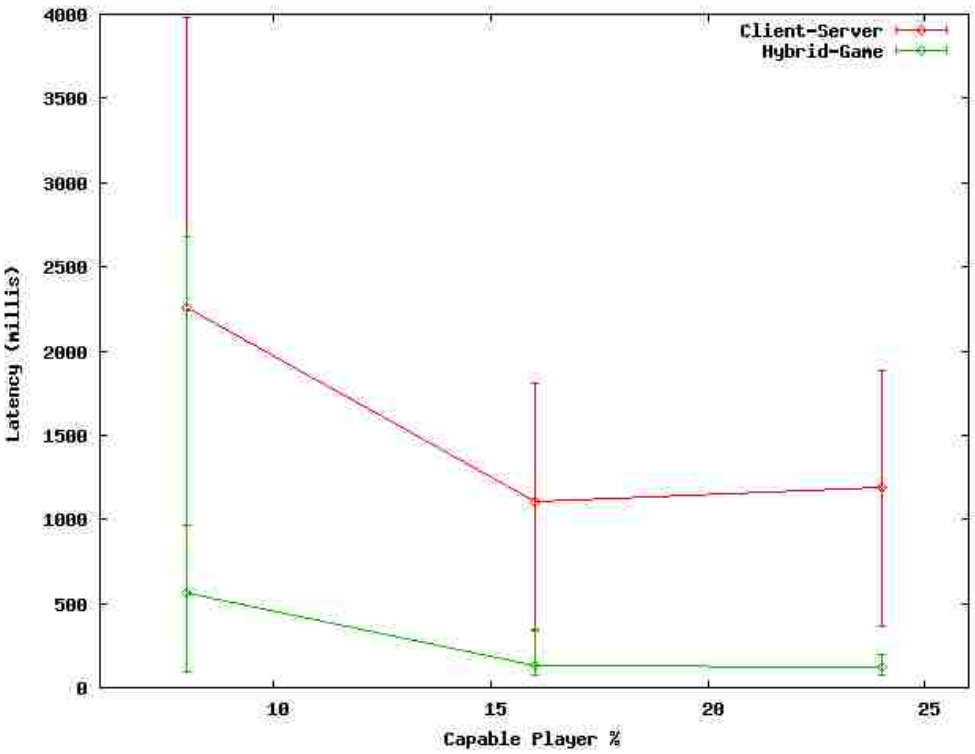


Figure 5.7: Player Capacity Experiment: Latency

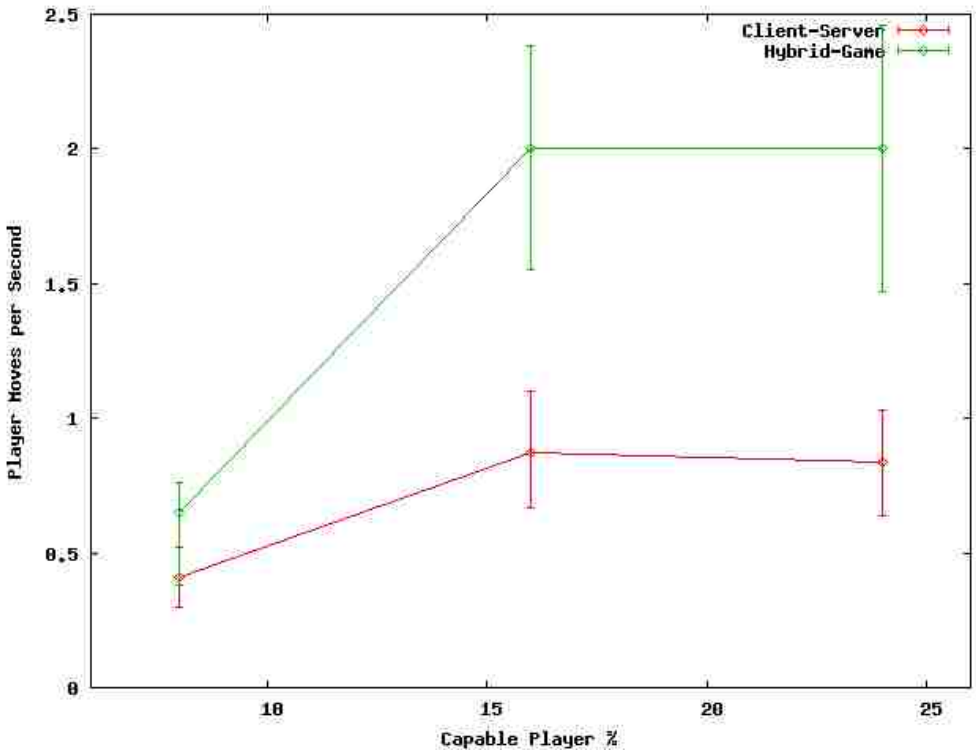


Figure 5.8: Player Capacity Experiment: Moves per Second

average of 4 capable players for each test. The central server in these tests still serves 4 of the regions as well as all of the state-changing moves. The bandwidth used at the central server is the same for the client-server and the hybrid-game server, although the hybrid game architecture processes more moves and has better latency. The next tests use a capable player percentage of 16, or an average of 8 broadband players. In these tests the central server is generally only required to serve the state-changing moves and the bandwidth savings is considerable. Then in our final tests we use a capable player percentage of 24, or 12 broadband players. With these test we always have more than enough capable players to act as region servers and so the central server never needs to serve simple positional moves. This illustrates the ideal situation for the hybrid-game architecture, and the bandwidth savings as well as the increase in overall game throughput are substantial.

Figure 5.7 shows that the median latency for the hybrid game architecture is much faster than the client-server architecture even when broadband players are rare. This data shows that in the client-server architecture the central server was always overwhelmed, with the average latency over a second in all of our tests. It also shows that when there are not enough capable players to assist in sharing the load, the central server in the hybrid game architecture can also slow down. However, when the capable player percentage gets high enough to serve the game regions the hybrid game architecture is much more responsive than the client-server architecture. What this is really showing is that when there are sufficient capable players, the hybrid game architecture allows the central server to run a much larger game.

Figure 5.8 shows that the hybrid game architecture is getting roughly two times the move throughput of the client-server architecture. Though the ideal for these experiments would be an average of 5.0 moves per second for every player, it is not achieved in either the architecture. The client-server architecture averages between .4 and .75 while the hybrid-game architecture ranges between .65 and 2.0. By being able to get substantially more game throughput the hybrid game architecture is doing a better job of serving the game.

Figure 5.9 and Figure 5.10 illustrate the CDF of latency for the best and worst case. Both of these graphs show that while the majority of the latencies are minimal they have long tails. In the best case, the hybrid-game architecture is getting well over 50% of its latencies below 200 milliseconds. However, when there are insufficient broadband players the worst case shows that the server on both the hybrid-game and client-server architectures becomes overwhelmed. Even the hybrid game architecture appears to have over 40% of its players experiencing latencies over a full second, and [8] shows that latencies any higher than 400 milliseconds make a real-time game unplayable.

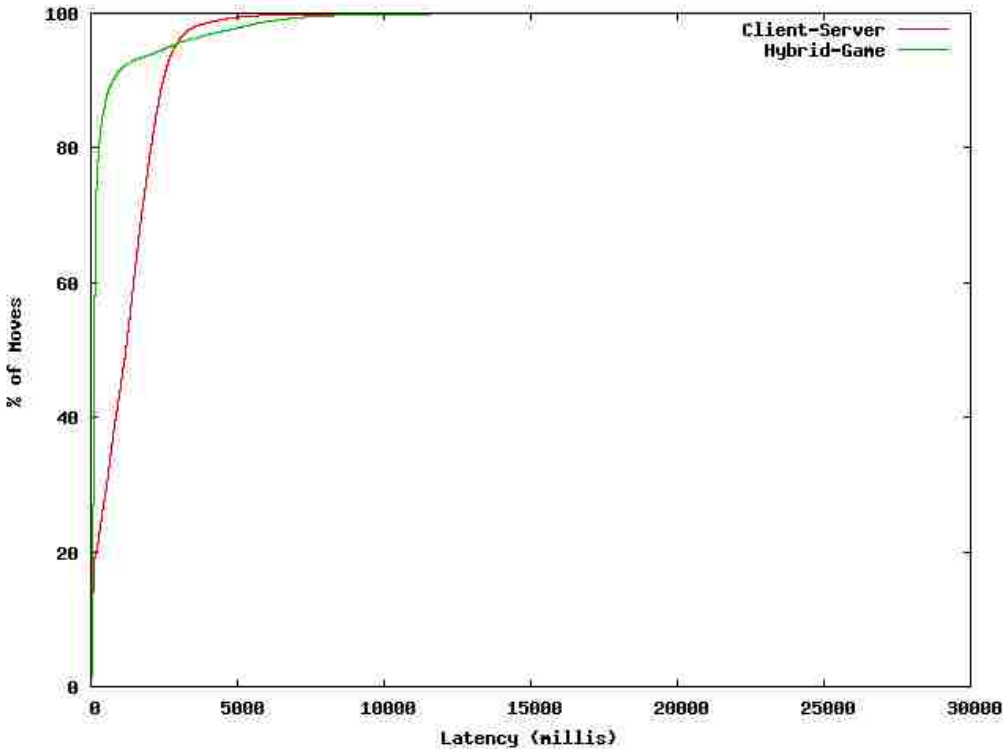


Figure 5.9: Best Latency Results: 12 Broadband Players

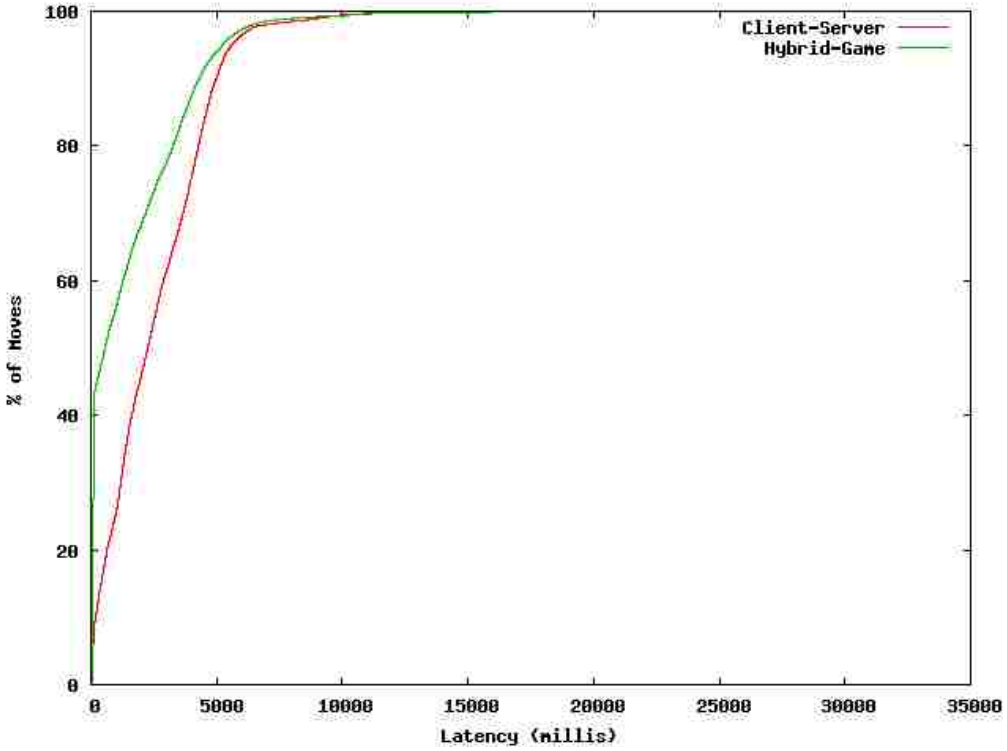


Figure 5.10: Worst Latency Results: 4 Broadband Players

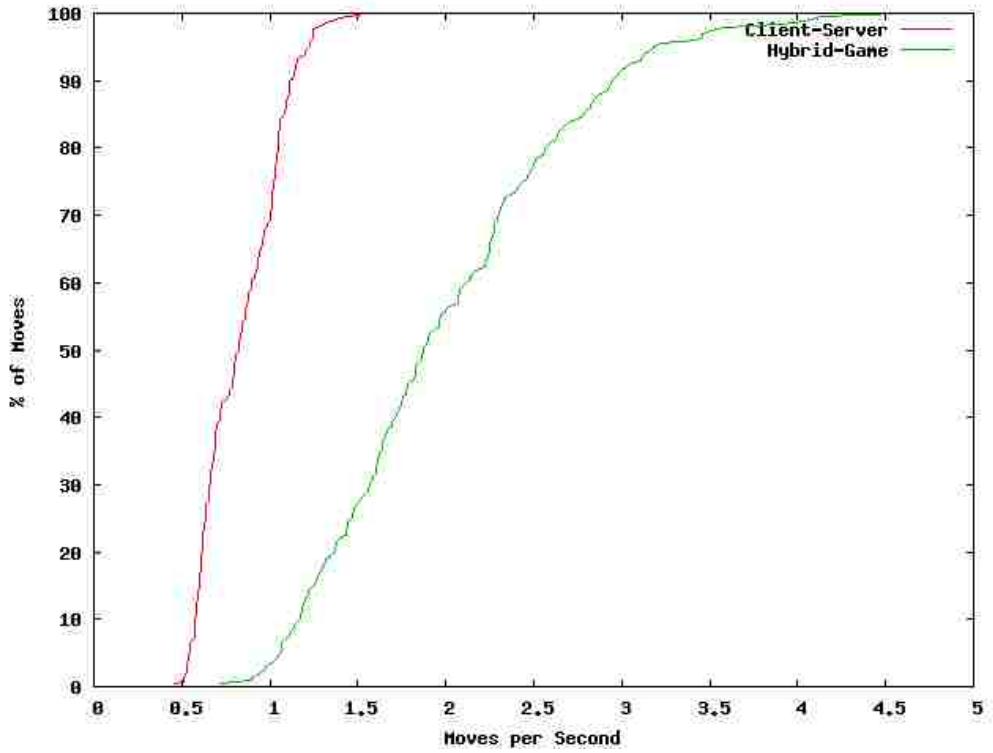


Figure 5.11: Best Moves per Second Results: 12 Broadband Players

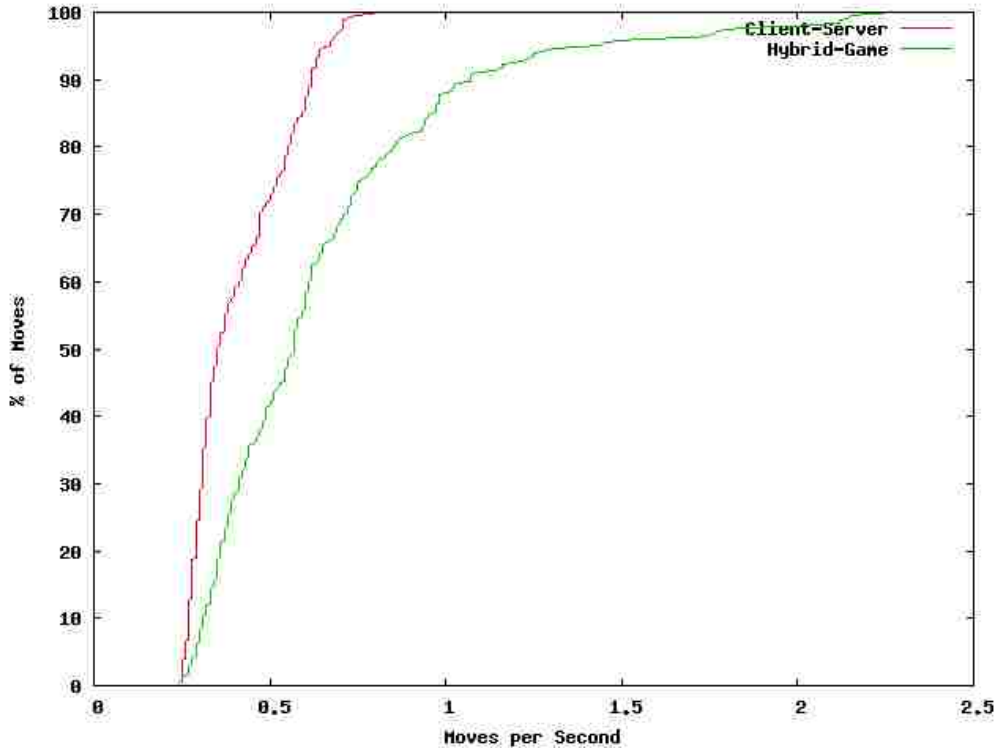


Figure 5.12: Worst Moves per Second Results: 4 Broadband Players

Figure 5.11 and Figure 5.12 show how even in the worst case the hybrid game architecture's game is progressing considerably faster than the client-server's. While in the best run, Figure 5.11, the hybrid-game architecture has a majority of players getting between 1.5 and 2.5 moves per second, a few players that are getting close to the ideal of 5.0 moves per second. The client-server architecture is a different story, in its best run it is getting a few players above 1.0 moves per second, while the majority of its players are getting between 0.5 and 1.0. In its worst run, Figure 5.12, while the two architectures are much closer the players in the client-server architecture entirely range between 0.25 and 0.75 moves per second.

5.3 Varying State-Changing Move Percentage

Our final experiment changes the percentage of moves that are state-changing. In an actual game, the state changing move percentage would be fixed and a function of the game. In our experiment we are able to simulate a change in the state-changing move percentage by sending what would otherwise be a positional move directly to the central server. The central server then passes the move on to the region server and the region server sends the player the region state update directly. By rerouting these positional moves through the central server, the server follows the same pattern as it does when the player sends in an actual state changing move. This allows us to maintain the same game mechanics while effectively changing the percentage of state-changing moves. In this way these rerouted messages can illustrate how the hybrid game architecture performs with different state-changing move load.

For this experiment we use a game world of 8 regions and a capable player percentage of 24%, but we were restricted in the number of players we could use to 35. This means that there is an average of seven capable players for each run. We test 25% , 50% , and 100% of the messages being rerouted through the central server.

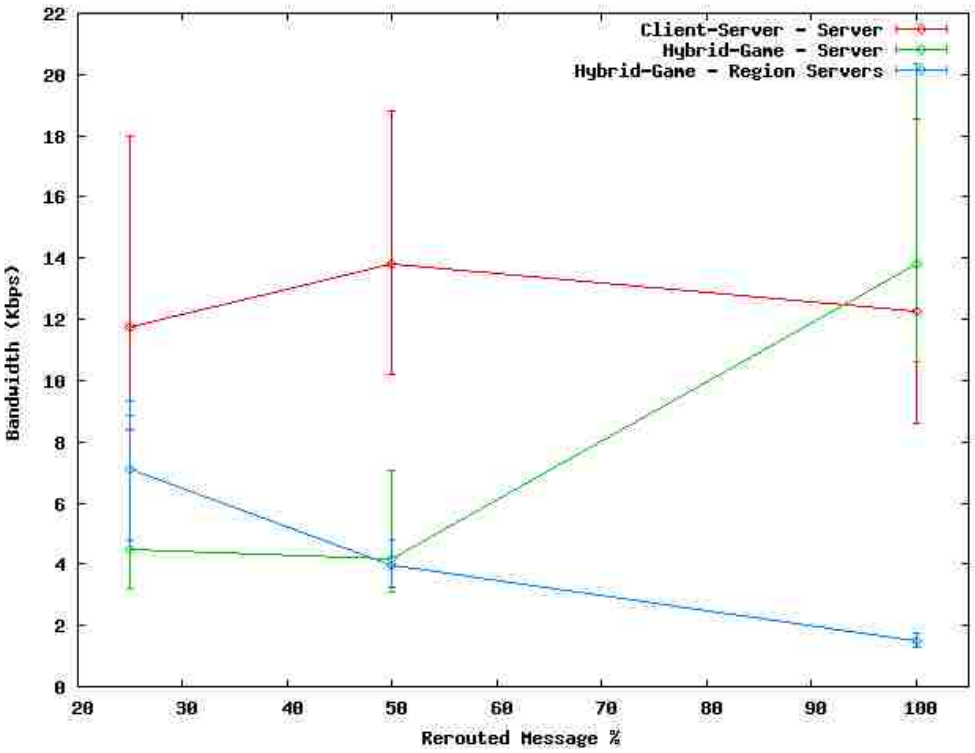


Figure 5.13: Extra State-Changing Moves Experiment Results: Incoming Bandwidth

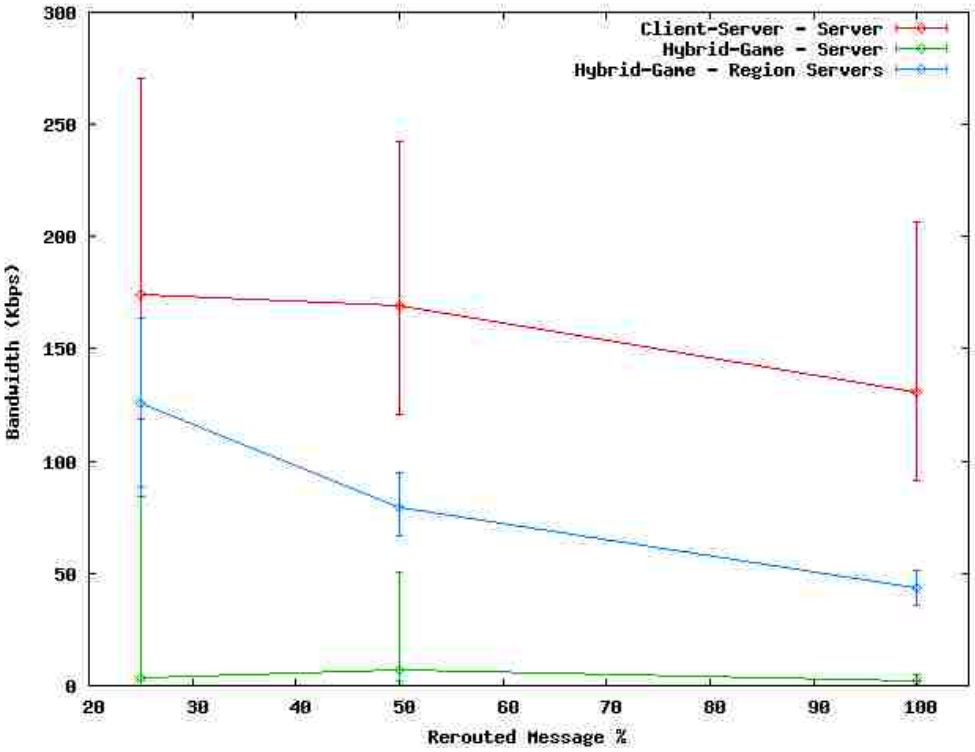


Figure 5.14: Extra State-Changing Moves Experiment Results: Outgoing Bandwidth

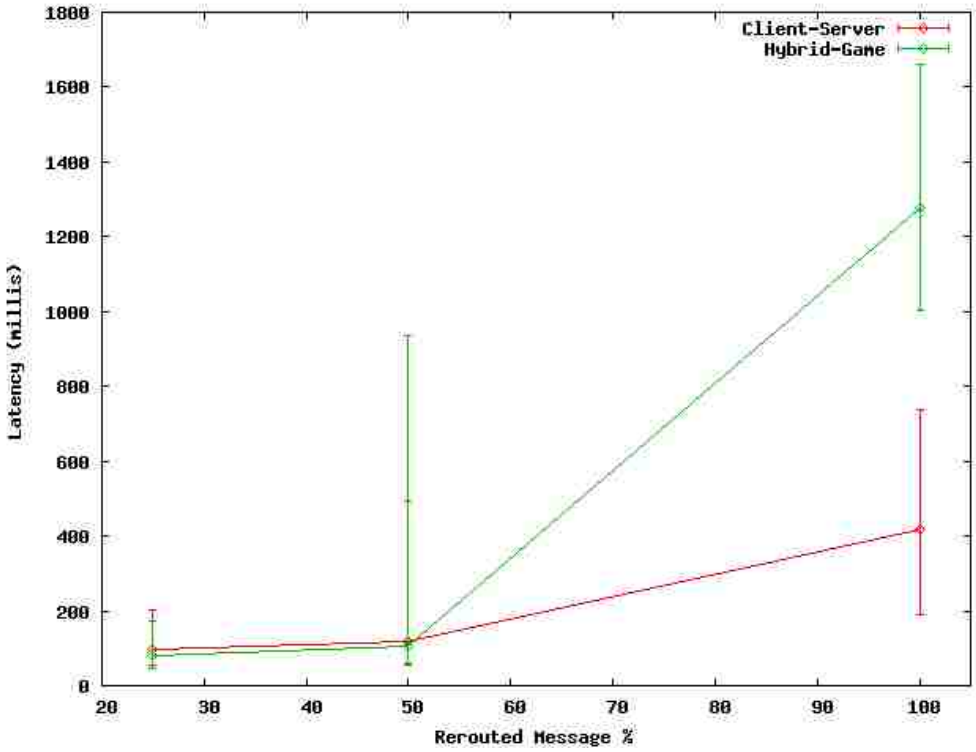


Figure 5.15: Extra State-Changing Moves Experiment Results: Latency

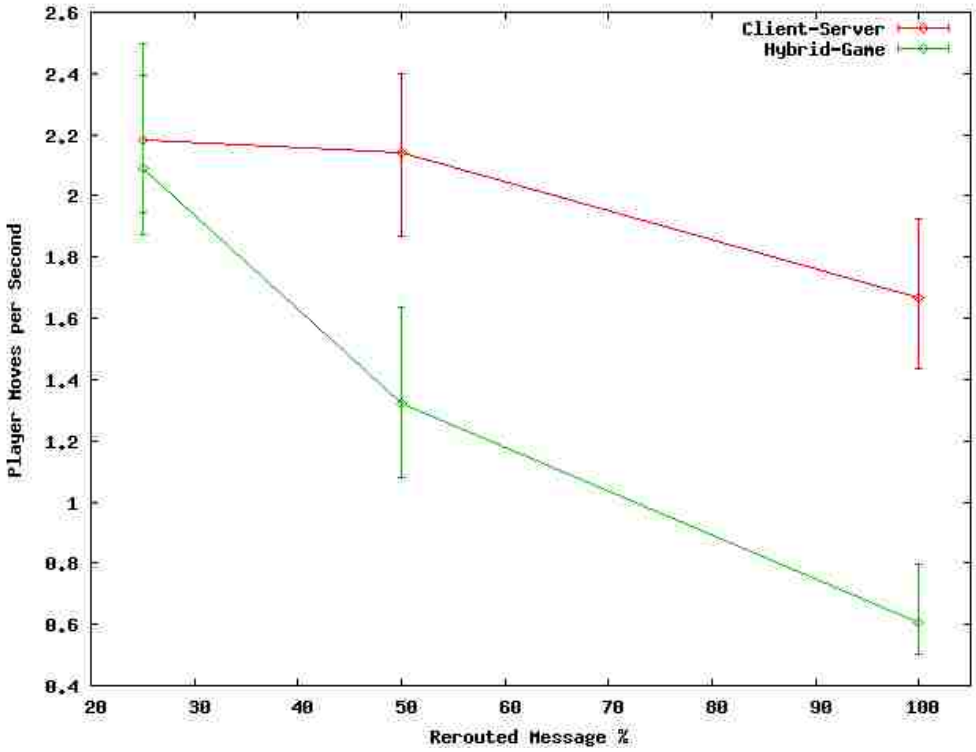


Figure 5.16: Extra State-Changing Moves Experiment Results: Moves per Second

This is the first bandwidth graph that shows a distinctly different pattern for incoming bandwidth Figure 5.13, than for outgoing bandwidth Figure 5.14. As the number of messages that are changed to state-changing moves and sent through the central server increases, the bandwidth into the central server in the hybrid game architecture also increases, whereas the bandwidth out of the region servers decreases. This is showing that the extra messages in and out of the central server in the hybrid game architecture are slowing the system down. When the bandwidth handled by the central server and region servers are added together they are not as much as the bandwidth handled by the client-server architecture. This is most likely due to the monolithic thread that runs the central server. In our original calculations we did not anticipate even beginning to overwhelm any of the PlanetLab nodes. Any future experiments should address this liability. If the various connections were threaded, and thus responded independently, this overhead could be avoided.

Figure 5.15 shows the median latency for this experiment. The server in the client-server architecture is much less overwhelmed than in the previous experiments due to the reduction in total number of players. The latency in both architectures is acceptable with 25% additional state-changing moves. However, as the number of state-changing moves increases beyond that, the hybrid game architecture slows more than the client-server architecture does, to where at 100% the hybrid game architecture's latency is four times that of the client-server architecture. This demonstrates how the hybrid game architecture relies on the differentiation of state-changing and positional moves. If all moves are truly state-changing moves, then the hybrid game architecture adds another layer of hierarchy. While this adds to the message latency it also decreases the size of the messages that the central server needs to handle.

Figure 5.16 shows that the average moves per second that each player makes decreases as more moves are state-changing. These numbers are the direct inverse of the latency numbers. If a player is having to wait over a second to get the response

for each move they will have less than one move per second. Once again this should improve with some threading adjustments to the central server.

Chapter 6

Conclusion

For games that can distinguish between positional and state-changing moves, not only does our hybrid-game architecture distribute upwards of 90% of the central server bandwidth, it allows the central server to serve twice as many player moves per second. This means that our hybrid-game architecture can save game hosting companies 90% of their bandwidth costs and 50% of their server costs while maintaining central game control. This will provide for an incredible advantage in an increasingly competitive global market.

6.1 Bandwidth Consumption

Our original hypothesis regarding the game world size experiments was that "we predict that the hybrid-game architecture's performance should increase with the player density of a given region." While this is partially true we found that when the density got too high, the region server was overwhelmed. More interesting though, the results show that having sufficient capable players to serve all of the regions is much more important than the density of the players in the regions.

Our original hypothesis regarding capable player percentage was that "We predict that the hybrid game architecture will produce great bandwidth savings so long as their are sufficient capable players to serve as regional servers; once this number is reached we expect to have similar performance with or without additional capable

players." Our results back this up by showing that the hybrid game architecture has substantial bandwidth savings so long as we had enough capable players. Once we have them serving all of the regions, there is only a small improvement found by increasing the number of capable players, for example going from 16% to 24%.

Our original hypothesis regarding state-changing moves was that "we predict that as the proportion of state-changing moves increases our bandwidth savings will decrease, but that is will remain in the 80% range." Once again this hypothesis ended up part true and part false. With regard to bandwidth savings, although the central server maintained the 80+% bandwidth reduction the savings is not as strong as it should have been because the game progressed at a much slower pace. We believe that this is an artifact of our central server being a monolithic thread rather than accepting each incoming connection on its own thread. Had we done this we believe that this this hypothesis would be proven correct.

6.2 Latencies

Our original hypothesis regarding latencies has three parts. The first is that the hybrid-game architecture will have an increased latency due to its hierarchy. This turns out to have been generally false. The first reason is that most of our communication does not require two levels of hierarchy, it is just back and forth with the region servers. The second reason is that with the central server so overwhelmed, the distribution of bandwidth really freed up the central server in the hybrid-game architecture to respond more quickly. The second part of our latency hypothesis is that latency should still generally be less than 200 milliseconds. This sprung from our thoughts that our game would never really come near the capacities of the system we were running on. While the results for the hybrid game architecture were generally less than 200 milliseconds, our hypothesis really never took into account its incredibly long tail. The final part of our hypothesis stated that we believed that

when more messages were state-changing moves our latency would nearly double the client-server architecture's latency. Our results show that the latency does indeed increase. Rather than being limited to two times the latency it climbed to four times the latency. This further illustrates that it is important to distinguish correctly between state-changing and positional moves. With the proper number of broadband players and a reasonable portion of positional moves the hybrid game architecture will reduce latency.

6.3 Player Moves per Second

Our original hypothesis regarding Player Moves per Second is that both architectures will easily maintain 5.0 moves per second. This is obviously disproved by our data. We didn't have a single player that averaged the full 5 moves per second. What the data tells us though is just as important. With the central server capacities as they were, and with the right settings, the hybrid-game architecture allowed for more than double the player moves per second than the client-server architecture. This means that the hybrid-game architecture not only distributed the central server bandwidth and lowered those costs, but that if the same server could then serve twice as many player moves the game hosting company would only need half as many servers.

6.4 Future Work

While we maintained the message and region update sizes discussed in [13] we would like to do a future battery of tests that maintain those message's relationship but make them smaller so that we can more accurately test our hybrid-game architecture on PlanetLab. If we can reduce the load on the game's central server for both the client-server and hybrid game architectures we can get a more realistic view of its potential impact.

We would also like to re-examine the state-changing moves tests with a fully threaded central server. We believe that the lack of threading in our experiments is the key factor that limited the total game throughput and latency in this experiment. We postulate that the bandwidth savings would remain high, latency for both architectures would improve, and total game throughput would be comparable to that in our other experiments.

Also, as broadband becomes more prevalent we would like to test certain runs where all players are capable. While we have demonstrated that the benefit of moving from 16% to 24% yielded only minor gains, we believe that this test would further prove our assertion.

We would also like to do tests that further vary the layout of the game world. In all of our tests the start regions for the two teams logged by far the most traffic. If these regions were divided and served by separate region servers we believe that it would demonstrate that the game world shape and organization can dramatically effect our hybrid-game architecture. We believe that this would prove that the hybrid-game architecture can be optimized by a game developer to take full advantage of its resource savings.

Another test we want to perform is one that illustrates the difficulty of cheating. To do this we may need to complicate the interactions that go on between players to present avenues where cheats may be used. We also want to refine the mechanism that is used to swap out overwhelmed region servers, and test the hybrid game architecture in even larger games to see how it scales up.

Bibliography

[1] N. E. Baughman and B. N. Levine. Cheat-Proof Playout for Centralized and Distributed Online Games. *INFOCOM 2001. Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications*, 1(1):104–113, 2001.

[2] A. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting Scalable Multi-Attribute Range Queries. *ACM SIGCOMM Computer Communication Review*, 34(4):353–366, October 2004.

[3] A. Bharambe, J. Pang, and S. Seshan. Colyseus: A Distributed Architecture for Online Multiplayer Games. Technical report, Carnegie Mellon University, September 2006.

[4] M. Castro, M. Jones, A. Kermarrec, and A. Rowstron. An Evaluation of Scalable Application-Level Multicast Built Using Peer-to-Peer Overlay Networks. In *Proceedings of IEEE INFOCOM 2003*, page , Washington, DC, USA, 2003. IEEE Computer Society.

[5] K. Cho, K. Fukuda, H. Esaki, and A. Kato. The Impact and Implications of the Growth in Residential User-to-User Traffic. In *ACM SIGCOMM*, pages 207–218, New York, NY. USA, Aug. 2006. ACM Press.

[6] E. Cronin, A. Durc, B. Filstrup, and S. Jamin. An Efficient Synchronization Mechanism for Mirrored Game Architectures. *Multimedia Tools and Applications*, 23(1):7–30, 2004.

[7] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *ACM Symposium on Operating System Principles*, pages 202–215, New York, NY, USA, 2001. ACM Press.

[8] T. Fritsch, H. Ritter, and J. Schiller. The Effect of Latency and Network Limitations on MMORPGs: A Field Study of Everquest 2. In *NetGames 05*, pages 1–9, New York, NY, USA, 2005. ACM Press.

[9] C. GauthierDickey, V. Lo, and D. Zappala. Using N-Trees for Scalable Event Ordering in Peer-to-Peer Games. In *International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 87–92, New York, NY, USA, 2005. ACM Press.

[10] C. GauthierDickey, D. Zappala, V. Lo, and J. Marr. Low-Latency and Cheat-Proof Event Ordering for Peer-to-Peer Games. In *Proceedings of the 14th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 134–139, New York, NY, USA, 2004. ACM Press.

[11] L. Gautier and C. Diot. Design and Evaluation of MiMaze, a Multi-Player Game on the Internet. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, pages 233–236, Washington, DC, USA, 1998. IEEE Computer Society.

[12] T. Iimura, H. Hazeyama, and Y. Kadobayashi. Zoned Federation of Game Servers: a Peer-to-peer Approach to Scalable Multi-player Online Games. In *Proceedings of 3rd ACM SIGCOMM workshop on Network and System Support for Games*, pages 116–120, New York, NY, USA, 2004. ACM Press.

[13] J. Kim, J. Choi, D. Chang, T. Kwon, Y. Choi, and E. Yuk. Traffic Characteristics of a Massively Multi-Player Online Role Playing Game. In *Proceedings of 4th ACM SIGCOMMworkshop on Network and System Support for Games*, pages 1–8, New York, NY. USA, 2005. ACM Press.

[14] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-Peer Support for Massively Multiplayer Games. *INFOCOM 2004. Proceedings of the Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, 1:107, 2004.

[15] J. Maassen and H. E. Bal. SmartSockets: solving the connectivity problems in grid computing. In *Proceedings of the 16th internation symposium on High Performance distributed computing*, pages 1–10, New York, NY, USA, 2007. ACM Press.

[16] V. Paxson. Strategies for Sound Internet Measurement. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, pages 263–271, New York, NY, USA, 2004. ACM Press.

[17] S. Rooney, D. Bauer, and R. Deydier. A Federated Peer-to-Peer Network Game Architecture. *Communications Magazine, IEEE*, 42(5):114–122, May 2004.

[18] A. Rowstron and P. Drushel. Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Middleware 2001: IFIP/ACM International Conference on Distributed Systems Platforms*, page 329, Heidelberg, Germany, November 2001. Springer Berlin Heidelberg.

[19] A. Rowstron and P. Drushel. Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility. In *ACM Symposium on Operating Systems Principles*, pages 188–201, New York, NY, USA, 2001. ACM Press.

[20] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Applications, Technologies, Architectures and Protocols for Computer Communication*, pages 149–160, New York, NY, USA, 2001. ACM Press.

[21] S. Yamamoto, Y. Murata, K. Yasumoto, and M. Ito. A Distributed Event Delivery Method with Load Balancing for MMORPG. In *Proceedings of the 4th ACM SIGCOMM Workshop on Network and System Support for Games*, pages 1–8, New York, NY, USA, 2005. ACM Press.