Galaxpeer: P2P networking for time-sensitive Massively Multiplayer Online Games

Thijs Boumans
4214854
t.boumans-1@student.tudelft.nl

Leon Helsloot
4235991
l.j.helsloot@student.tudelft.nl

April 2016

Abstract

This paper presents an architecture for a 3D space shooter using peer-to-peer networking. By using the locations of players in the virtual world in the topology of the overlay network, nodes communicate primarily with nodes that are close in the virtual world. This ensures low latencies suitable for time-sensitive games, while still providing high scalability and fault tolerance. We demonstrate that this system scales up to at least 1100 nodes, with strong indications it can handle far more, and provides resilience to failing nodes.

1 Introduction

Online gaming is a rapidly growing multi-billion dollar industry, with 29% of the most frequent video game players in the U.S. paying to play games online [6]. An important part of the cost of running an online game service is formed by running game servers - a 2011 infographic estimated the server costs of the massively multi-player online role-playing game World of Warcraft to total $136,986 per day [4]. These costs are especially problematic as the load varies greatly over time and is linked to the popularity of a game, which is hard to predict, possibly resulting in under- or over-provisioning. Using a peer-to-peer (P2P) network architecture rather than the classical client-server architecture could alleviate these problems, resulting in reduced operating costs of online games.

In previous work, Knutsson et al. identified three potential problems of P2P gaming: performance, availability and security [5]. Moreover, they present a proof-of-concept P2P Massively Multi-player Online Game (MMOG) using a distributed hash table (DHT). Similarly, [3] and [2] used distributed hash tables to create generic P2P MMOG architectures. While these DHT-based approaches were found to perform sufficiently well for game types that are not time-critical, such as real-time strategy or role playing games, the latencies introduced by message routing in the P2P network is not acceptable in games where the player directly controls the avatar, such as first person shooter games [5].

We therefore present *Galaxpeer*, a 3D space shooter game using a P2P overlay network which mirrors the topology of the virtual game space, and analyse its properties in terms of performance and availability. The rest of this paper is organized as follows: section 2 describes Galaxpeer and its requirements, followed in section 3 by a description of its design. Experimental results to analyse the properties of Galaxpeer are given in section 4, after which the findings are discussed in section 5. Finally, section 6 gives an overview of future work.

2 Background

Galaxpeer is a massively multiplayer online space shooter, in which each player controls a spacecraft with which they must destroy other players' spacecraft while navigating through an asteroid field. Players can move around freely in any direction through the virtual space, practically unlimited by enforced boundaries. There are three types of objects in Galaxpeer: spacecraft, each controlled by a player, rockets, and asteroids. All objects are mutable, with a health parameter that is decreased when the object collides with other objects, and are destroyed when the health parameter reaches zero. When a player spacecraft is destroyed, it will respawn near another player after a few seconds. Unity is used for the graphics. All game data is transient, and lives only as long as it is relevant.

To provide players of the game with a good experience, several qualities must be met. First of all, the application should be able to run in real time, processing events as they happen, without a noticeable delay to the user. The game engine should process at least five frames per second, as the graphics engine can further smooth out movements.

Secondly, the game should be scalable to run with at least 100 concurrent players. The amount of objects (and thus players) present in any small region in space is limited due to constraints on the capabilities of both the player and the machine. However, the game should automatically scale in such a way that 100 players can join without the players nor their machines reaching a capacity limit.

Moreover, the system should be fault tolerant in such a way that it should remain operational when players leave the game. As noted in [5], machines participating in a peer to peer architecture are much more likely to disconnect in an uncontrollable fashion than centralised servers in a data center. While we expect players to occasionally disconnect in groups, disconnections are likely to happen several seconds apart. Since it is still possible that two players leave within the same second, the game should be able to handle two players leaving within the same second, and be unlikely to fail if

more players leave at the same time.

Finally, we require the game state to be locally consistent, meaning that from the viewpoint of every single player, the events that occur must logically follow from what the player has already seen. The state of the game as a whole may be somewhat inconsistent, as long as no individual player is able to notice.

One major assumption we make is that all players of Galaxpeer are honest, and that cheating will not occur. As noted in [5], cheat prevention in peer-to-peer games is a problem of its own, and is thus outside the scope of this project.

3 System Design

To ensure high performance when Galaxpeer scales to large numbers of players, the game operates differently on a local scale than on a global scale. While each player must have highly detailed knowledge about all objects they can see, objects that are beyond their line of sight require much less detail. Since the speed at which objects can move is limited, objects that are outside a player's line of sight will not immediately affect the player. On the other hand, interaction between players at a global scale is necessary to provide robust fault tolerance.

3.1 Local scale

Since only those objects that are local to a player in the virtual world are relevant for that player, two areas of interest are defined for each player, outside of which objects are ignored. Both areas of interest used in Galaxpeer are simple *zoneless* regions [1], forming a sphere with a fixed radius around the player. The first area of interest, AoI_o defines the region in which game objects are relevant. The second area of interest, AoI_n, with a radius twice as large as the first, defines the region in which other nodes are relevant. The use of two areas of interest gives each node knowledge of all players with whom its AoI_o overlaps, and thus of all players to whom an object known to the node may be relevant.

Every node communicates directly with all other nodes within their AoI_n to ensure network latencies are as low as possible. Contrary to aforementioned DHT approaches, knowledge and responsibility of objects in Galaxpeer is not distributed throughout the entire network, but limited to nodes with proximity to the objects within the virtual world. While this approach may impact resilience to cheating, it improves performance as it allows for direct communication with the whole subset of relevant nodes.

To limit the load on the nodes, each node is responsible only for the objects that are closer to their local player than to any other player. The node responsible for an object sends information about that object to all players with the object in their AoI_o. All players with the object in their AoI_o keep track of the physics of the object, but only the responsible node sends updates about the object. This approach allows the use of dead reckoning for objects, extrapolating object movement under the assumption that their current behaviour will not change. Dead reckoning greatly reduces the number of messages required to maintain synchronicity, as well as the impact of failures of the responsible node. If an object is outside the region of interest of any player, it is destroyed. While this may create some artifacts in the form of objects disappearing from the game, it avoids putting additional load on nodes to keep track of irrelevant objects.

The replication of physics on all nodes to whom an object is relevant also means that any node with the object in their AoI_o can take ownership of an object, should the responsible node disconnect or fail, or stop sending updates for any other reason. Since any object is the responsibility of the closest player, ownership is also transferred when the closest player changes. During normal operation, object handovers are initiated by the current owner. In the case of failures, however, other nodes have to initiate takeovers. Should an ownership conflict occur, for instance due to a timing failure, ownership is granted to the node with the lowest unique identifier.

Because the scalability of Galaxpeer is limited in the density of players and objects within the same area of interest, over-full areas of interest must be prevented. However, players must be able to reach other players quickly to make the game fun. Therefore, the spawn positions of players are dependent on the locations of other players. To achieve this, a random location in space is picked, after which the player closest to that location is located. The player then spawns near the edge of the AoI_o of that player. This technique automatically scales the used size of the virtual world with the amount of players.

3.2 Global scale

In order to achieve local consistency, each player has to know which other players are close to them, and thus needs some knowledge of other nodes within the network. Galaxpeer mirrors the location of players in the virtual world in the topology of its overlay network, since nearby players are more relevant than far-away players. To achieve this, the virtual space is divided into octants, and a connection is maintained to the closest player in each of the eight octants. When a node A initiates a connection to another node B, A requests the address and location of each of the 8 players closest to B. If a node C with which B responds is closer to A than the original closest node, A sets up a connection to C, and requests the nodes closest to C. This process is repeated until a stable state is reached, thus allowing a player to discover the players closest to any location, regardless of the initial connection.

To increase fault tolerance, each node maintains a cache of information of all second-order nodes, i.e. the nodes closest to the closest nodes in each octant. This cache is updated every 15 seconds to avoid keeping information on offline nodes. If any of the closest nodes fails, a connection to a new closest node is set up based on the cached information. Moreover, when a request for the closest nodes is received, each node will forward not only their closest nodes, but also a node from their cache which is as far away from the requesting node as possible. This additional node increases fault tolerance of nodes on the edge of the virtual space, with connections in only a few octants, as well as the speed by which players can teleport through the system to find the nodes closest to their pseudo-random spawn location.

To ensure that each node is aware of all other nodes within

their AoI_n, including nodes that are not included in the list of first- and second-order connections, connection information of other nodes is flooded to all known nodes within the AoI_n of that node. While this results in an increased amount of messages in densely populated regions, it greatly improves the reliability of relevant peer detection.

4 Experimental Results

4.1 Local scale

To assess the performance of Galaxpeer during normal operating conditions, we measured the performance of one node to which around 600 simulated players connected. The behaviour of these players consisted of random movement and shooting rockets as often as possible. These nodes were run on 4 different machines distributed between two locations in different cities. The used machines were 1 iMac with a 3.2 GHz Intel Core i5 quad-core, 1 MacBook late 2009 edition, and two laptops with a 2.30 GHz Intel Core i7-3610QM quad-core running Linux. The node on which performance was measured ran alone on one machine, all other nodes were distributed across the other three machines. During this test, some nodes were killed due to resource exhaustion. Of the 600 nodes that were launched, 518 were able to stay alive over a prolonged amount of time.

The measurements from this experiment are displayed in figure 1. The figure shows both the amount of time spent simulating physics and the amount of time needed to process incoming messages. Both measurements show the time spent in ticks per second, with 10,000 ticks equal to 1 millisecond. While the average amount of time spent on networking and physics are not far apart, the time needed for networking is far more erratic. It is also worth noticing that the physics engine of Galaxpeer theoretically does not scale as well as the networking part. The relatively low overall load of approximately 20,000 ticks, or 2 ms, per second, suggests that a single node could handle a far larger network than we could simulate using 4 machines.

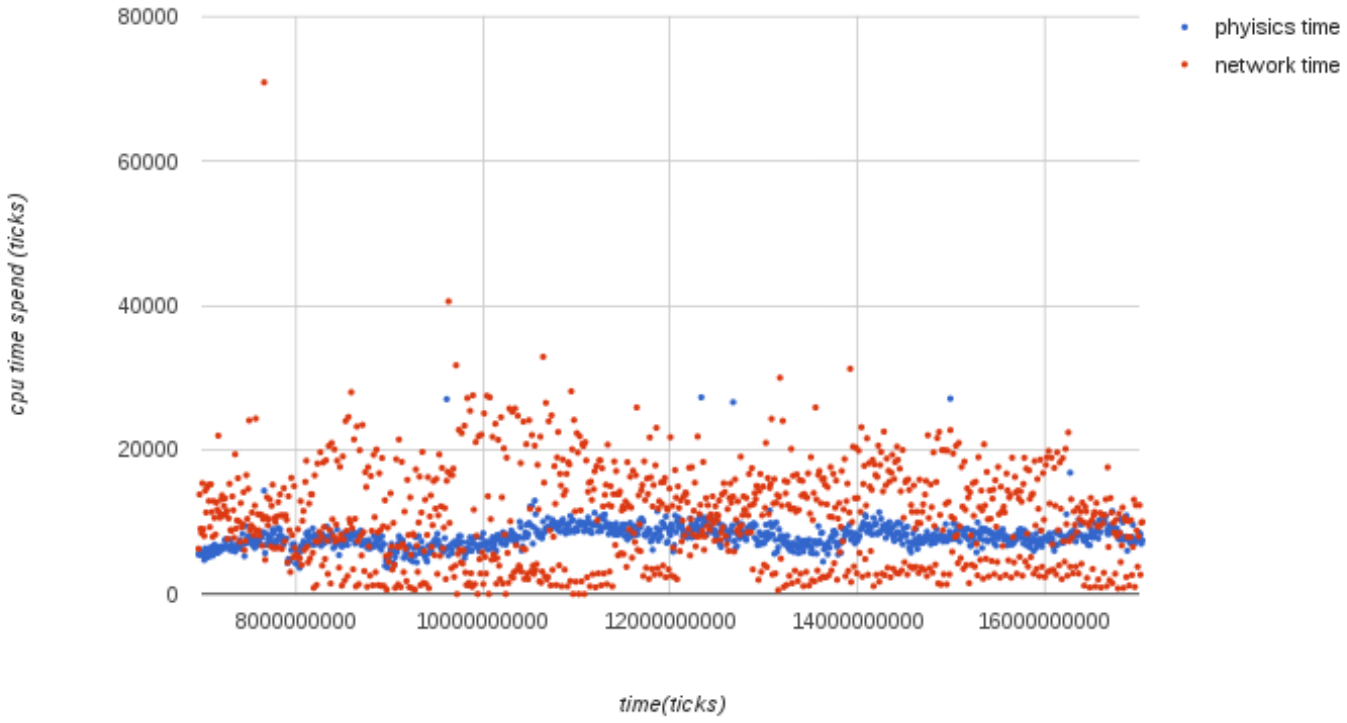

Figure 1: Time spent on physics and network for local scale

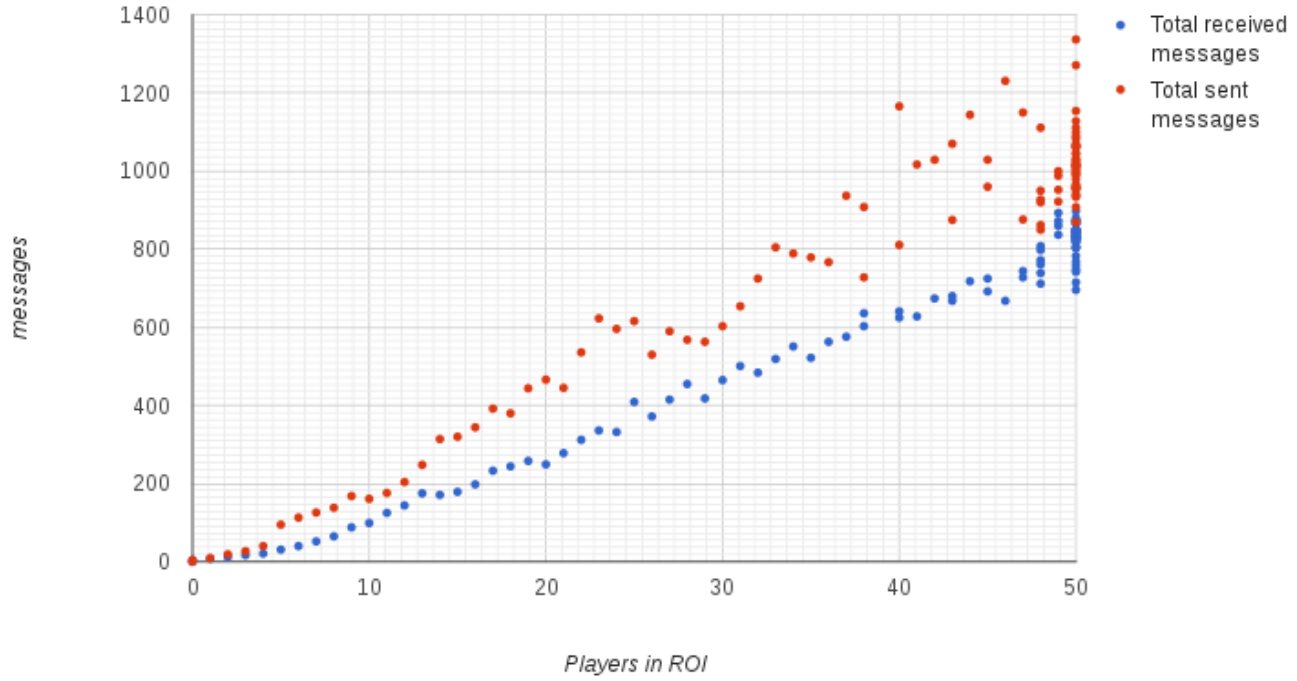

Figure 2: Number of messages sent and received per second, for an increasing number of clients in the area of interest.

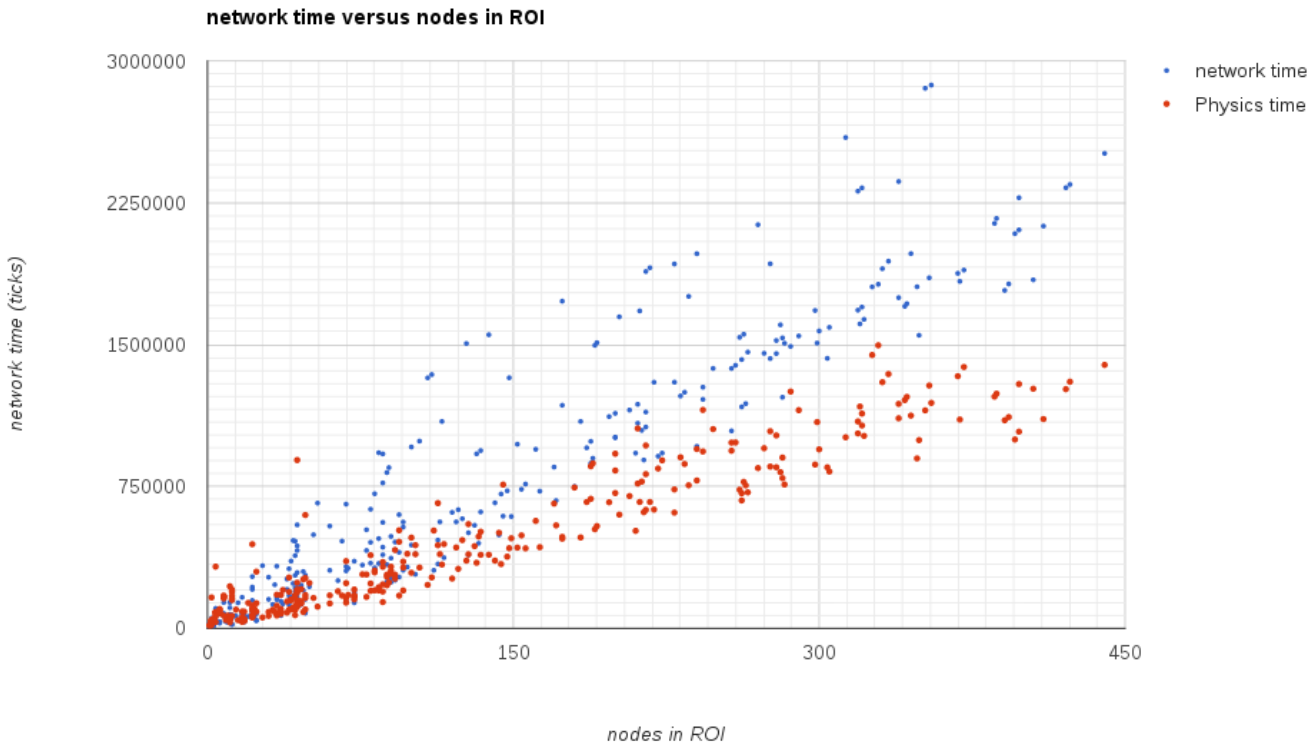

Figure 3: Time spent on physics and message processing, in ticks per second, for an increasing number of clients in the area of interest (10,000 ticks = 1 ms).

To investigate the scalability of the number of clients within the same area of interest, we spawned an increasing amount of simulated players within the same area of interest, and measured the amount of sent and received messages. To ensure that these players did not move apart, the players only rotated on a fixed position. Figure 2 shows the results of this experiment, with the total amount of messages sent and received per second. The diagram shows a linear relation between the number of players in the area of interest and the amount of messages, suggesting a reasonable scalability of the number of clients within an area of interest. Interestingly, the number of received messages is lower than the number of sent messages, suggesting some packet loss.

Comparing the time spent on physics and network processing for an increasing number of nodes in the area of interest in figure 3 paints a similar picture of local scalability. The time spent on networking increases slightly faster than the time spent in the physics engine, indicating networking is more likely to eventually become a bottleneck.

4.2 Global scale

To test the scalability of our architecture at a global scale, we spawned a total of 1100 players at 40 times the size of the area of interest from each other, such that none of the players had overlapping regions of interest. To also test the performance of the joining process we had all players join the same player and record its performance. The result is shown in figure 4. This test was done across 5 machines, 1 iMac with a 3.2 GHz Intel Core i5 quad-core, 1 MacBook late 2009 edition, 1 Acer Aspire S3 laptop running Windows and two HP laptops with a 2.30 GHz Intel Core i7-3610QM quad-core running Linux. The measured node was on one of the HP laptops and separate from the rest of the network, and was used as an initial connection point for all other nodes. The bottleneck of this test was our inability to generate more nodes rather then the performance of this single node. We think that if we had had more machines we could have ran far more nodes with no scalability issues.

The large initial spike in network time that is visible in figure 4 is caused by the large number of connecting nodes, each of which requests a list of nodes closest to the measuring nodes. The selection of an additional node furthest away from the requesting node, being a very inefficient operation, put a large burden on the measuring node. Once the network stabilised, the time spent on processing messages decreased again. This test shows that, although the network scales very well to large amounts of players, using a single access point for all players may deteriorate the performance of that access point. Instead, newly connecting players should ideally be distributed over different nodes.

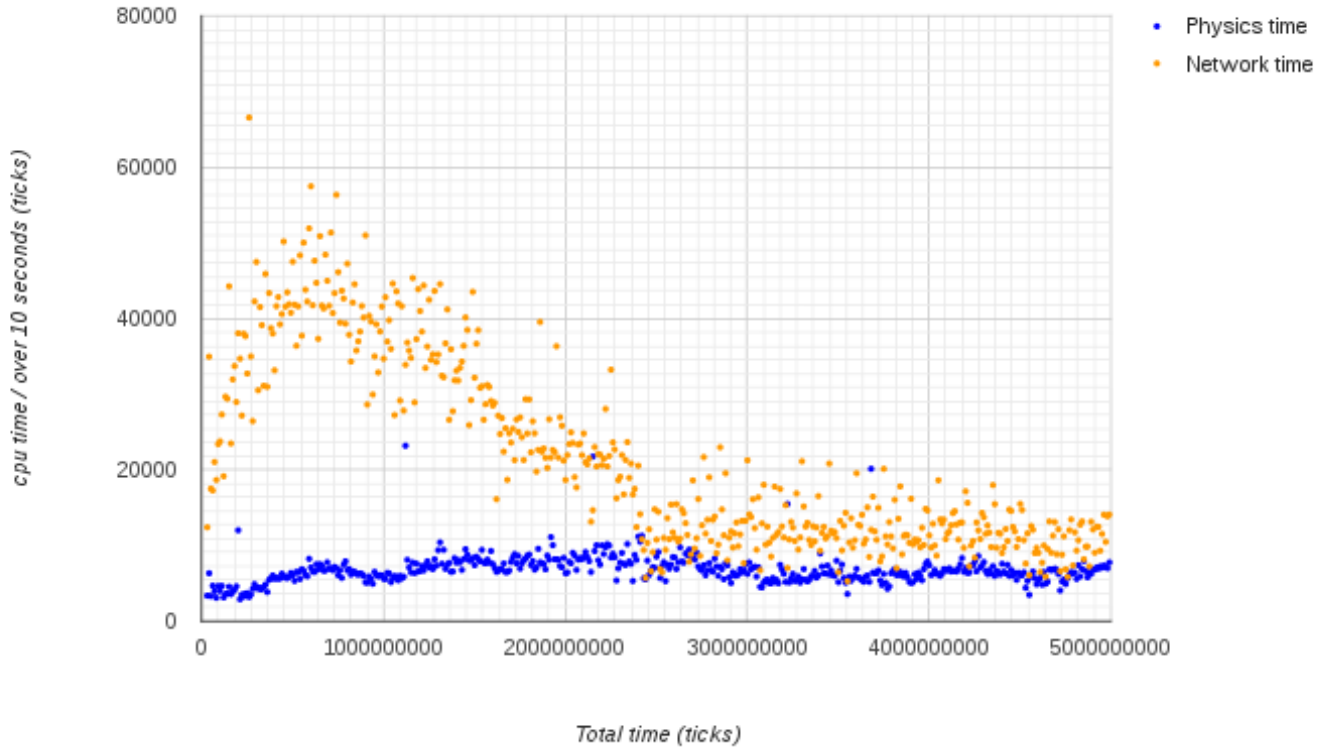

Figure 4: Time spent on physics and network for global scale

4.3 Fault tolerance

To assess the resilience to failing nodes in the network, we set up a small network with a total of 32 nodes. All nodes repeatedly teleported from a random point far away in space to a shared rendezvouz point and back, i.e. from a location with no other clients in the area of interest to a location where all other clients should be in the area of interest. This cycle, with 60 seconds between teleportations, ensured that clients constantly had to re-discover other clients in the network. During this cycle, each node had a random failure pattern following an exponential distribution with a mean time to failure of 10 minutes. After failing, a node had a 50% chance of resuming operation, and a 50% chance of being replaced by an entirely new node. In either case, the time to come back online followed an exponential distribution with a mean time of 3 minutes and 20 seconds.

The results of this experiment are displayed in figure 5. The diagram shows the number of nodes present in the largest group of nodes that are interconnected. The other partitions are all nodes that have at least 1 other node in their area of interest, and are not part of the main cluster. The lone nodes are the nodes that believe to be alone in their region of interest. While the high failure rate caused many small errors in the form of incomplete views of the network, nodes appear to recover quickly from these errors. Moreover, nodes that are not within the main cluster at one point do join the main cluster at a later point, indicating the capability of the network to recover from temporary partitions.

Manual inspection of the data suggests that most of the potential partitions are false positives, as either half of the network is only 1 node off, suggesting that one node is simply not close enough to the centre of the group to be fully counted. This indicates an error in the measuring method. Most lone nodes are caused by nodes that came back online between the teleportation step and the recording step, and thus have not teleported yet.

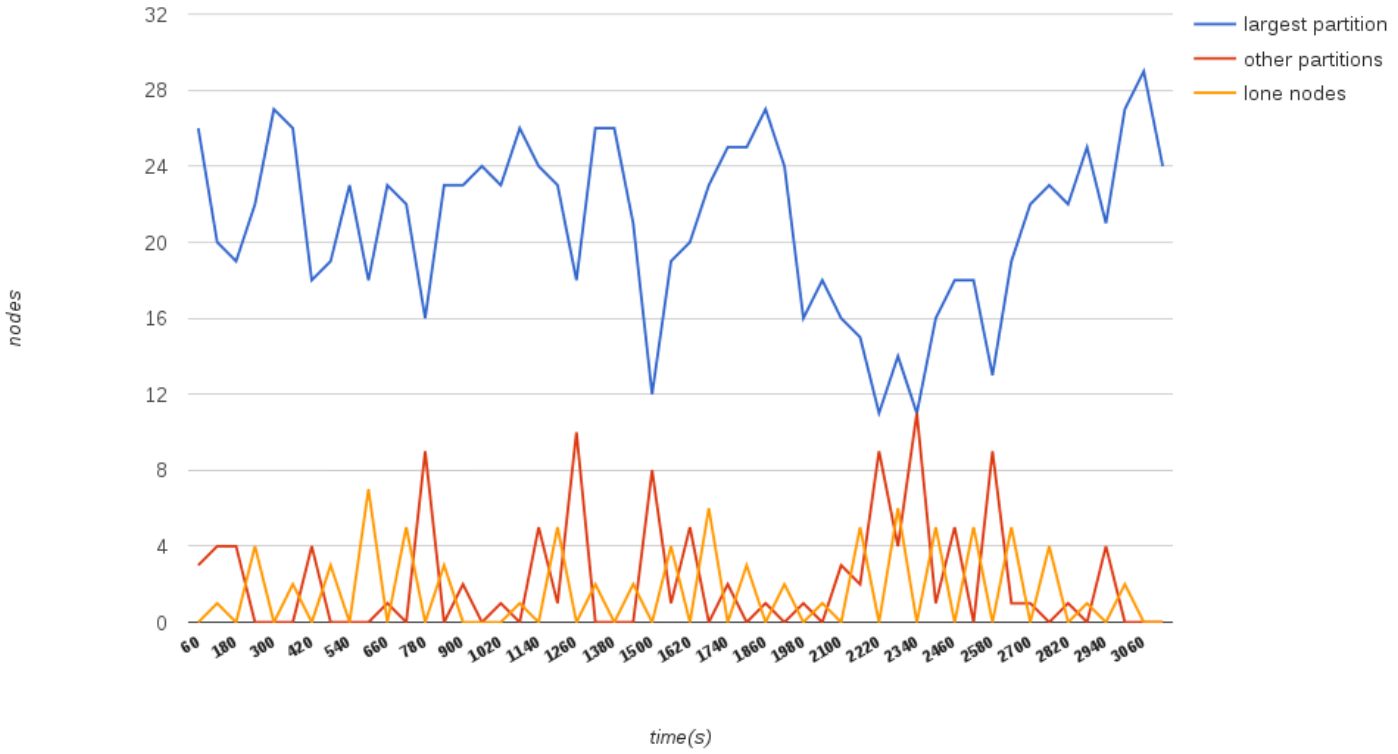

Figure 5: Potential partioning in the network

5 Discussion

As reported in the previous section, Galaxpeer achieves its scalability requirements very well on a global scale, easily scaling to over 1000 clients. However, some limitations apply on a local scale, due to latency, consistency and fault tolerance requirements. When implementing a time-sensitive peer-to-peer game, these limitations should be taken into account.

5.1 Trade-off network load and fault tolerance

To create a fault tolerant network, each node keeps track of nodes slightly further away than the closest nodes in each octant, even if those nodes are not within the area of interest. While this provides a better protection against partitioning of the network and decreases the time required for locating closest nodes when joining and spawning, it puts a strain on the network due to the increase in communication between nodes. We do, however, believe that each player should be aware of at least three other players, such that partitioning cannot occur

even if two players disconnect simultaneously. Depending on the performance and fault tolerance requirements of specific applications, the frequency at which information about nodes is exchanged can be adapted.

Similarly, updates about objects are sent more frequently than necessary (once per second for each object in Galaxpeer) to allow for quick detection of orphaned objects, as well as provide resilience to dropped network packets. With large amounts of objects within an area of interest, however, these frequent updates result in a high network and message processing load. Since dead reckoning is used to extrapolate object movement, the frequency at which object updates are sent could be decreased, at the cost of less resilience to failing nodes and packet loss. Moreover, less frequent updates may result in jumpy movement of objects if clocks of different players are skewed.

5.2 Physics mirroring

The use of dead reckoning for simple physics calculations, while sharing uncommon and complex calculations across the system, forms a trade-off between load on the physics engine and load on the network. While the load on the physics engine is drastically reduced by distributing collision detection across nodes, and the load on the network is reduced by locally extrapolating object movement, this comes at the cost of a slight difference in simulation between players. This is usually resolved within a short time due to frequent re-synchronisations. A trade-off must be made between acceptable differences in simulation and load on the physics engine and the network.

5.3 Cheating

During development we decided that all players are honest. This was so that we could limit the scope of our project. In the future works we will discuss some possible measured to prevent cheating.

6 Conclusion and Future work

Our design of establishing connections based on player locations in the virtual world is very well scalable to more than 1000 players, and seems capable of scaling near limitless as long as players are not close to each other. By connecting only to close nodes, low latencies for relevant information can be achieved. Moreover, partial mirroring of physics and sharing information about other nodes make Galaxpeer resilient to failing nodes.

While Galaxpeer provides a basis for networking in time-sensitive peer-to-peer games, it is by itself not suitable to be released as a game. Two major problems in peer-to-peer systems, and particularly games, are the possibility of cheating and bootstrapping of the system. Both problems are future work.

6.1 Cheating

One problem with a peer to peer system as described in this paper is that players are easily capable of cheating. While we did not implement any method to prevent cheating, we do propose one possible fix for this issue.

First of all, we have to ensure that one player knows another player is cheating. Because we already have a dead reckoning system available we can do this by checking if the location of each object is within the error margin of the system since the last message arrived. This will allow other players to detect at the very least blatant cheating when they are nearby, but will not work against more subtle cheating or cheating when you are out of the region of interest of any other player. This is not very problematic as cheating in these ways is not as noticeable for other players.

Then once we have detected cheating we need a reputation system, as our system requires every player in range to be in agreement over whether or not a given player should be banned. We propose a system where every player polls every other player in range, after which each player can state whether or not they have witnessed any cheating, and if they have witnessed the player long enough to make a vote. All players who have witnessed the player for a long enough period (1 minute) will take a vote if they witnessed any cheating and what follows is an algorithm for solving the byzantine agreement problem which, if it succeeds, will have all other players ban the cheating player and drop their connections with him, effectively dropping him from the network.

The weakness of this system is that a malicious agent could create a large number of fake players with which he can ban other players at will. However, we do not believe this is solvable while keeping the entire system distributed.

6.2 Bootstrapping

The current system requires each player to know the IP address and port of another active player to join. This can be problematic, as each node of the network can only handle a few players joining at the same time (as described in section 4.2), and is not user-friendly. Any kind of centrally hosted static address would not be truly peer-to-peer, and as such partly defeat the purpose of a peer-to-peer game. One possible solution is to nevertheless create *super peers*, which are nodes that are either purposely created or have been online for a long period of time, and try to join the game by connecting to these super peers. Another option is to create a server that keeps track of players who recently joined and simply respond to every request with the ip address of another recently made request. If a node has already gone offline another request is simply made. This would probably work but not be truly peer to peer and could be a potential bottle neck once several hundred players start joining the game at once, which could then be alleviated by building a distributed server system complete with load balancing.

References

[1] 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. Washington, DC, USA: IEEE Computer Society, 2008, pp. 27–34. ISBN: 978-0-7695-3425-1. DOI: 10.1109/DS-RT.2008.20. URL: http://dx.doi.org/10.1109/DS-RT.2008.20.

[2] Luther Chan et al. "Hydra: A Massively-multiplayer Peer-to-peer Architecture for the Game Developer". In: *Proceedings of the 6th ACM SIGCOMM Workshop on Network and System Support for Games*. NetGames '07. Melbourne, Australia: ACM, 2007, pp. 37–42. ISBN: 978-0-9804460-0-5. DOI: 10.1145/1326257.1326264. URL: http://doi.acm.org/10.1145/1326257.1326264.

[3] Thorsten Hampel, Thomas Bopp, and Robert Hinn. "A Peer-to-peer Architecture for Massive Multiplayer Online Games". In: *Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games*. NetGames '06. Singapore: ACM, 2006. ISBN: 1-59593-589-4. DOI: 10.1145/1230040.1230058. URL: http://doi.acm.org/10.1145/1230040.1230058.

[4] Brenna Hillier. *Infographic: World of Warcraft by the numbers*. June 1, 2011. URL: http://www.vg247.com/2011/06/01/infographic-world-of-warcraft-by-the-numbers/.

[5] B. Knutsson et al. "Peer-to-peer support for massively multiplayer games". In: *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*. Vol. 1. Mar. 2004, p. 107. DOI: 10.1109/INFCOM.2004.1354485.

[6] *The 2015 Essential Facts About the Computer and Video Game Industry*. Report. Entertainment Software Association, 2015.

Appendix A: Time sheets

name	Leon Helsloot
total-time	85 hours
think-time	4 hours
dev-time	55 hours
xp-time	10 hours
analysis-time	3 hours
write-time	13 hours
wasted-time	0

name	Thijs Boumans
total-time	88 hours
think-time	12 hours
dev-time	38 hours
xp-time	10 hours
analysis-time	6 hours
write-time	18 hours
wasted-time	4