The Impact of Decentralisation on Networked Computer Games

Ashley Vaughan Smith
School of Computing
University of Derby

A project submitted for the degree of
Computer Games Programming B.Sc. (Hons)

2009 May

Acknowledgements

I would like to thank my friends and family for their patience and support when I was locked away in my room for forever, and my supervisor Adam Thornett for his help in this project. I would also like to thank everyone at Monumental Games for teaching me so much.

Abstract

There are many advantages to distributing a networked computer game, including removing bottlenecks from the network, the ease of handling broken links in the network and the extensibility of not having a dedicated central location for the game servers. However, making that game decentralised is something that game developers avoid due to concerns about security, cheating and complexity. In this report I aim to give a detailed description of these concerns and some possible solutions to them. I will also analyse the impact that these decisions would have on game developers.

Contents

List of Figures **v**

List of Tables **vii**

1 Introduction **1**

1.1 Overview 2

2 Literature Review **3**

2.1 State Consistency 4

2.2 Security 7

2.3 Performance 8

2.4 Discussion 10

3 Leverage Benefits **11**

3.1 Unbounded Size 12

3.2 Static Data 13

3.3 Latency 14

3.4 Bandwidth 14

3.5 Disconnection 16

3.6 Cost 16

3.7 Discussion 17

4 Impact **19**

4.1 Connections 20

4.2 Static Data 20

4.3 Revenue 21

4.4 Possessions 22

4.5 New Users 23

4.6 Collusion and Cheating 24

4.7 AI 25
4.8 Performance 26
4.9 Discussion 26

5 Implementation **27**
5.1 Fulfilment 28
5.2 Discussion 33

6 Conclusion **35**
6.1 Summary 35
6.2 Future Work 37

Glossary **39**

References **41**

List of Figures

2.1 Network states 5

3.1 Latency 14
3.2 Bandwidth 15
3.3 Connections 15

LIST OF FIGURES

List of Tables

3.1 List of online game genres 12

1

Introduction

In this section we introduce this report and the topics of decentralisation, distribution and networking in relation to networked computer games. Providing clarification on game network topologies, decentralisation and distribution.

There are a certain number of different network topologies that are used in computer games. Online multi-player computer games typically fall into one of four categories:

1. A single client connected to a single server.
2. A single client connected to a network of servers.
3. A single client connected to other clients and to a server.
4. A client connected to multiple clients.

Items one, two and three can be thought of as centralised - there are one or more single points in the network where clients must communicate. This report will focus on item four, peer-to-peer and how a computer game can leverage the benefits of not requiring a central location for communication.

From experience working on the MMOG "Football Superstars" at Monumental Games Ltd. (2008a) when using the centralised approach to networking all of the actions that a client in the game makes are tracked and authorised through one or more central locations. This means that the game requires high performance, redundant and expensive servers in places throughout the world for everyone in the game to be able to play equally.

An interesting development in recent networks around the house and office means that some devices are now connected in a mesh-like, heterogeneous topology as opposed to a tree-like topology used in normal internet connections. A good example of this is wireless networking,

as in a recent paper by Brik et al. (2008) showing the success of this mesh-like topology in a real-life example. This brings about the possibility for games to take advantage of this heterogeneous network and relieve some of the difficulties of getting dedicated high-speed access points for clients and providers of the game. It also means that each peer will act autonomously in the network bringing impacts for the game and the games design. These impacts will be discussed in the following chapters of the report.

1.1 Overview

This report discusses and gathers research on decentralised networks and their applicability to networked computer games. In chapter two we look at the literature on the subject and its relevance to games. In chapter three we will look at how the decentralised model can be leveraged by games for better performance or new features not present in client-server games. In chapter four we will look at the impact that using this model would have on a game and its design and gather these requirements into a specification. Finally in chapter five we will look at each of the requirements and how they could be implemented. In the conclusion we will weigh up the benefits and the impact of decentralisation on computer games. In short the aims of this report are:

- Find how different game designs or genres could leverage the advantages of being decentralised.
- Investigate the issues with using a decentralised architecture in a game as a specification.
- Review implementation details that a game would go through in development for each specification point.
- Present reasons for and against using a decentralised network in a structured and understandable way with specifics that each game design would need and implementation possibilities.

2

Literature Review

This chapter reviews the current research and literature on the subject of distribution, peer-to-peer and decentralised networking for games.

Networked computer games have state: a sequence of mutable data representing an entity in a game (Knutsson et al. (2004)). Games are played over a network, the time that it takes to propagate the changes in state from one client to the next (latency) is important in a real-time computer game because if there is too much latency between the users of the game then the user will notice and game play will be hindered (Tom et al. (2004)).

Synchronising state in a networked computer game is one of the main difficulties for game developers. A recent article by McCoy et al. (2003) gives an overview of state consistency and the challenges faced when implementing state consistency. It shows that the four main challenges are latency, bandwidth, reliability and complexity.

Latency is an issue because of the real-time nature of computer games. If the time taken for the action to get to the desired location is too high then the peer will notice the jump between the two differing states either visually or internally. Algorithms can be used to mask this transition and these will be discussed.

Bandwidth is limited because of the slow network speeds over the internet compared to higher LAN speeds. This brings into consideration the fact that peers need to reduce the amount of network traffic they produce.

The reliability of servers in a client-server (CS) network is essential because of the tree-like hierarchy of the network. If a server fails then the clients connected to that server will disconnect. In a peer-to-peer (P2P) network, reliability is less of an issue because peers will have more than one connection to other peers. If a peer has more than one physical or logical

connection to another peer, such as in a mesh-like network, then reliability is less of an issue as the peer has redundant connections to use in case of failures in the network.

The complexity of using a P2P network is greater than a CS network because it needs a more complex way of synchronising state than a simple CS network. In a CS network all of the servers can be authenticated and trusted to give correct data. In a P2P network the peers in the network have the ability to cheat and give false information because there is no central authority. There are also algorithms to combat this ability to cheat and they will be discussed further on in the report.

2.1 State Consistency

The design and architecture of a networked computer game is highly dependant on the game state. The state of a game is, from the point of view of the server and every client, the data containing every piece of information that the peer needs to participate in the game. This includes things like the position of players, health and items. Keeping the game state in a single location can be done, however it creates a bottleneck at that location, meaning it becomes centralised. Another way of keeping state is to keep state locally for all of the participants of the game and receive changes to that state from the peers themselves. i.e. when a player moves, this action will need to be updated for all of the participants of the game that know about the player - this is distributed. If the state were only maintained by the peers of the game - not stored in a central location like CS networks - then that would be decentralised.

Fig. 2.1 shows some examples of the game state and it's position in the network. Assume in the diagram that client c2 can see and needs to know about c1 and c2. c3 is not close enough to be interested in or know about the state changes of c1.

Fig. 2.1(a) shows a client-server topology. The state of the game is kept in a single location on the server s1. Each of the clients c1-c3 are representative of a client that is controlling an entity on the server e1-e3. Each client needs to update it's own entity when the user performs actions on it and needs to know about changes to the other entities that it is interested in. This is the simplest way of storing state. However the bottleneck is clear and this topology does not scale well.

Fig. 2.1(b) shows a distributed client-server topology. In this case the bottleneck has been removed at the servers. In the diagram the proxy-entities p1-p3 need to replicate their state changes to other servers and the concrete entities need to broadcast changes to proxy entities on all other servers and this then propagates to the clients that are interested in that entity.

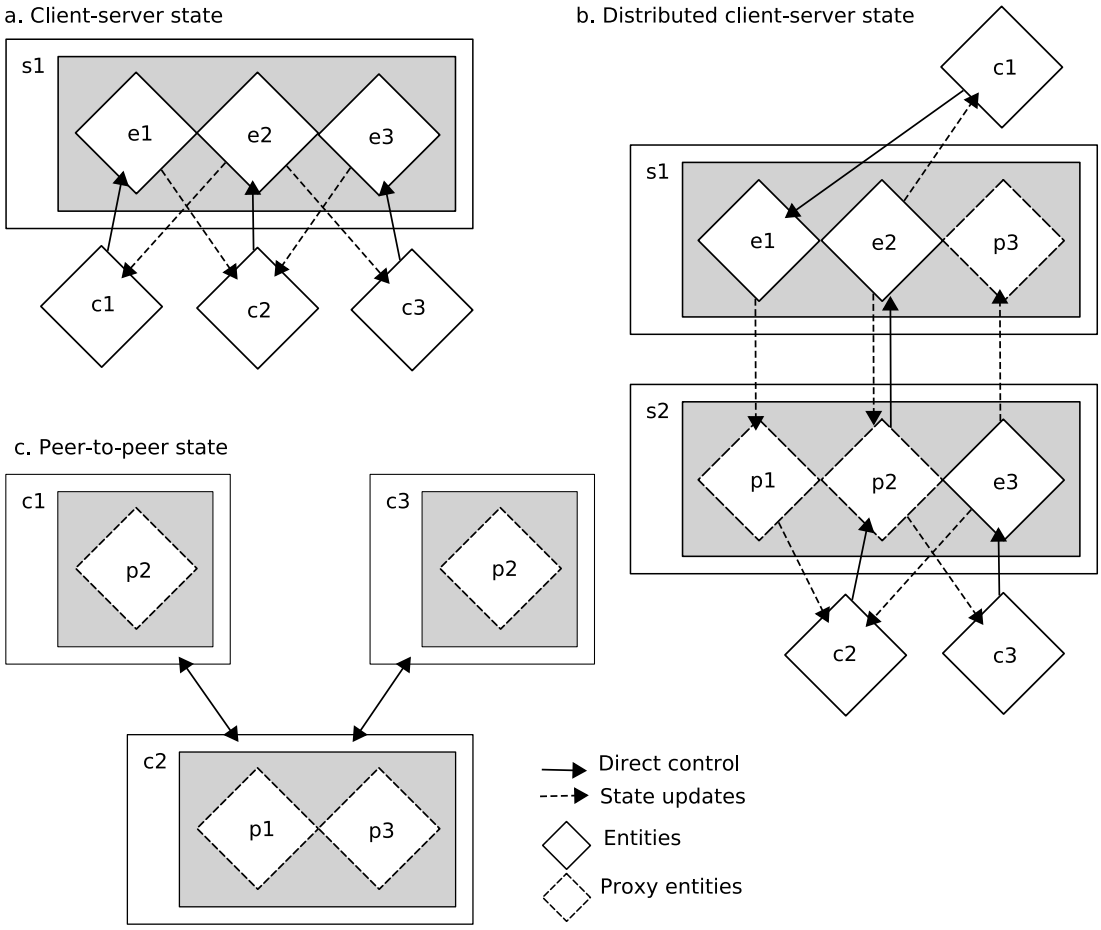

Figure 2.1: Network states - Different locations of state in a networked computer game.

Fig. 2.1(c) shows a peer-to-peer topology. Each client keeps the state of the other clients that it is interested in. When any client changes state it replicates that state change to the clients that are interested in it. This may look simpler that the previous two, but because the previous two have trusted servers that can authenticate the state changes, they use a much simpler algorithm to deal with changes in state. With peer-to-peer the state changes cannot be authenticated in a central location so another, relatively more complicated method is needed to authenticate the state changes.

Having worked on a MMOG technology API (Monumental Technology Suite Monumental Games Ltd. (2008b)), I have experience of commercial-grade technology that employs a distributed server architecture. In this architecture every client of the game will connect to one of the many servers of the game. Each of these servers will have access to the entirety of the persistent game data; things like the list of all users in the world and their last known location. However they will only have access to the game state that the server is managing. The game state is in memory and consists of temporary variables like the conversation state the player is in. Each of the distributed servers manages a slice of the game world and when a player moves over the boundary of those slices the server transitions the players temporary information from it to the server the player is entering. This method is commonly used in online games (Beskow et al. (2008)). This removes the problems with bandwidth and having a single server. However it means that a server may have to query other servers for the actual state of a player. It also means that a players data will have to be moved when the player moves to another server's slice of the game world. If the network of servers is set up with high speed links then this will give little if any noticeable difference to the client. However the link between the server and clients needs to be low latency for the clients to not experience lag. This is difficult and expensive.

A paper by Pellegrino & Dovrolis (2003) reflects the fact that a CS architecture has higher player-to-player latency than a true P2P architecture, however it also shows that a much more complex synchronisation protocol is needed. It also introduces a composite architecture that combines the benefits of CS and P2P where a hierarchy is introduced. This is similar to the distributed server architecture above except that peers will be connected in a hierarchy. This means that clients gain the benefit of lower latency but as a disadvantage have higher CPU processing requirements.

This does not mean that distributed server architectures are without their complications. As explained in Cronin et al. (2004), state consistency in concentrated networks is usually done in a sequential manner. However because each of the servers in a distributed network are working

asynchronously to (and usually far apart from) each other, it means that a paradox could occur. E.g. if a player receives some gold on one server and at the same time has some gold removed from another server, one of the servers will find that the data is inconsistent. This brings about the need for being able to roll-back (undo) an action.

Splitting up the game and the game state because each entity has a limited perception field is something that is commonly practised in games as discussed in Knutsson et al. (2004). The game state then becomes loosely coupled and each logical area that entities are gathered in can be viewed as a single state by entities in other logical areas. Locality of reference (Denning (2005)) shows that data accessed in a short period of time is clustered together. Taking advantage of this we can group states together so that it is more likely data will be accessible.

The reason that most applications use a client-server approach is that due to the server being a piece of equipment that can be physically identified and secured, it is trusted to take care of the security of authentication, communication and identification. In a decentralised network this is not the case, as each participant in the network cannot be physically identified without prior communication.

2.2 Security

Security in a decentralised network revolves around the fact that none of the peers in the network can be identified unless they have communicated previously in some way (Identification). The communication between peers of information that should be kept private (Communication) and how to identify previously met peers (Authentication).

2.2.1 Identification

Identification of a peer that has not been communicated with before is a new issue. In client-server architectures people rely on the fact that each client is given a username and password that is securely stored internally. This is then used to identify them in future sessions. If this were used in a decentralised way then each client would need to have a copy of that username and password to identify the connecting peer, which is possible with cryptography, but when the peer is first introduced there is no proven means by which they can trust each other.

2.2.2 Authentication

Authentication is looked at in Suryanarayana et al. (2006), introducing a system to communicate in a decentralised network using PKI (See below). The system also manages decentralised trust.

The paper shows four different algorithms to manage trust between peers in the network, using the distributed-trust, NICE, REGRET and complaint-based models noted. This provides games with a way of communicating securely and being able to authenticate peers in the network.

2.2.3 Communication

One way to communicate securely between peers in a decentralised network is to use PKI (public key infrastructure) Gu et al. (2007). This is a way of avoiding man-in-the-middle attacks. This model uses a key to identify a peer in the network. Once the keys are exchanged securely the two communicating peers can exchange information without the data being able to be read by anyone that does not have those keys. One shortcoming of PKI is that it is possible for a peer in the network to impersonate another peer.

The Tor Project (2009) is an example of a distributed and decentralised network communication protocol. TOR is used for anonymous communication over the internet. This could also be applied to a decentralised game to make playing the game completely private, because a decentralised network cannot rely on IP addresses, like TOR, it could use TOR implicitly and provide anonymity to the users of the game.

2.3 Performance

Performance in a decentralised network is critical because unlike a client-server network, there will be many multiple connections between peers. Meaning increased resources are required to process the data on these connections.

The paper by Boulanger et al. (2006) explains that every game state change cannot be sent to every player and gives an overview of techniques for managing state change interest - a way of minimising the state information that clients receive. It shows multiple ways of splitting up the space that the game world occupies so that it is fast to compute the state changes that a peer should receive and efficient at filtering out messages that it should not. This is reflected by Li et al. (2004) who shows that reducing the amount of state data on the client to as little as possible is essential to prevent cheating in on-line games and also helps with bandwidth limiting.

The lock-step algorithm Wolfson (1987) is a protocol that enables a distributed network to collect actions of its participants and make sure no participant can modify or spoof their action after receiving other participants actions and before sending their own. This is especially useful to networked computer games where being able to see a players action before making your own

would enable cheating. The fact that the protocol needs to receive all actions from all players in order for the next turn to begin means that it is better suited to a turn-based game such as a RTS rather than a real-time, latency dependant game such as a FPS. A paper by Baughman et al. (2007) builds on the lock-step protocol by optimising it and adds asynchronicity to the protocol. Which is perfect for a decentralised network where the peers act autonomously and have slices of the game world that can act autonomously.

Dead-reckoning is a common algorithm used to predict the movement of an actor over time Zhang et al. (2006). Where the velocity of an entity is communicated instead of the absolute position. Meaning applications can predict the movement of the actor if a piece of data is delayed or missing. Variants of this algorithm are also merged and optimised, such as globally synchronised dead-reckoning Aggarwal et al. (2004) and local-lag Vogel & Mauve (2001). Such algorithms are necessary and used commonly in networked computer games to disguise the real-time, non-deterministic nature of networked games.

Another algorithm is discussed in Singhal & Zyda (1999); the frequent state regeneration. This reduces the time between waits in the Lock-step protocol (latency) but incurs a penalty in bandwidth. So this is more applicable to real-time FPS games rather than RTS games.

The algorithm bucket synchronisation is discussed in Gautier & Diot (1998) is similar to the lock-step algorithm in that it requires the participants to wait until every other participant has sent their action, but introduces a delay to synchronise with a time.

Singhal (1997), Sharkey et al. (1998) and Jefferson (1985) also introduce algorithms to combat the inconsistencies that are incurred with using a networking model that introduces lag into games. These all rely on estimating what the actors in the world are going to do next and predicting their movements. I.e. The time-warp algorithm abstracts away the time that the application is at into time that the network can work on. It also utilises rolling-back states when inconsistencies occur.

The paper Moon et al. (2007) researching into P2P and distributed networking for games shows that a tree-based method of path finding to the best node will make the frame rate (Rate at which the game can play at over a network) actually increase with the number of players on that network. The paper also shows nicely many algorithms to maintain consistency in the distributed network.

2.4 Discussion

From looking at these protocols and systems, we can deduce that the following issues are present in all distributed applications:

- State consistency and synchronisation.
- Identification/introduction of new peers to the network.
- Authentication of peers as they connect to others already present in the network.
- Communication in the network of private information.
- Performance of applications in the decentralised network.
- Security aspects of communicating with peers in the network.

There are multiple ways to solve these issues and the optimal one for the situation depends on how the game is designed. For example a RTS game may be more suited to using the lock-step protocol because unlike real-time, latency dependant games a RTS game can use prediction to emulate the position of the units. These points will be discussed in more depth in chapter 4 where we will look at designs of games and what they would require if they were decentralised.

In the following chapter we will look at how a game can take advantage of being distributed/decentralised and how the game design or genre of the game defines how and if it can be done.

2.4.1 Static Data

One item that does not have much literature is decentralised static data. Static data in a game is the meshes, the textures and the executable data that does not change (The immutable data). This data needs to be accessible by the user, but each user needs to have the same static data to keep gameplay consistent. Although if two users had different static data (Two different games running on the same network) then they could just be logically separated but still participate in the network. This issue will be looked at in later chapters.

3

Leverage Benefits

This section covers the advantages or incentives for using a decentralised networking model for different genres of game.

The main advantages from reviewing the literature on the subject of decentralised networking shows that it can be applied to games to gain the following advantages over a centralised networking model:

- Self-propagating, limitless network means that there is no cost to adding new physical resources to the network.
- Static data transfer can take advantage of distributed peers bandwidth.
- Homogeneous networks create opportunities for games to gain redundancy in connections, latency improvements via path-finding and the fact that the game network can never be shut-down.
- Multiple connections per user means that path-finding can be used to expand the bandwidth of a peer.
- Redundancy in homogeneous networks means that disconnections in the network will not bring the entire network down.
- No dedicated servers or high-speed access points means that the cost of a decentralised game is minimal.

In this section we will visit each of these points and look at the relevance to a different genre of game. Game genres are a subjective topic, so the list of genres that we will look at are taken

from Entertainment Software Association (2007). Taking out the genres that are not usually played online leaves table 3.1. It shows the genres of game that can be played online:

List of online game genres
Strategy
Role playing
Shooter
Flight
Sports games
Racing

Table 3.1: List of online game genres

This list defines the game genres that are different in the way that they handle networked design and operations that we will look at and contrast the typical design to that of a decentralised game of the same genre.

3.1 Unbounded Size

One advantage of using a decentralised networking model for a networked computer game as opposed to a CS model is that the network will be self-propagating. This is not the case with centralised networks because to improve or expand the network, new servers need to be physically added to provide access for users that are not near a server already or servers that are already starved for resources. In a decentralised network, each user provides some of the services of the network. This means that if a new user connects to the network they are in effect increasing the capacity that the network can hold. This is all without having to change the network physically in any way as opposed to a centralised network where new connections would have to be added physically. This unbounded size means that some games could take advantage of it and that all games could take advantage of not having to physically expand the network.

Strategy games such as the Command And Conquer (Electronic Arts Inc. (2009)) series usually have multiple users connected to the same game at the same time but are restricted because of the vast number of individual units that it needs to deal with. Strategy games are usually split up into levels and a maximum number of players are allowed into the levels instance at one time. They also usually have a single server to authenticate the actions in the game. At the same time there is an option of creating a local game without having to connect to the central server. If a strategy game were to consider decentralisation the main advantage

it could leverage would be the unlimited number of players that could be connected to the same game at the same time. However the problem with the vast number of units still remains. One way to combat this would be to look at concatenation of units into groups. I.e. When a group of individual units gets too large the units are grouped together and thought of as a single unit until resources are freed.

Games with a single entity controlled by the user like role-playing, adventure and flight games would be able to take advantage of the unlimited number of players without having to worry about resources being taken up by infinite number of units - as there will be less resources required by the entities in the game than a strategy game.

An interesting development would be in games where there has been traditionally a limited number of players like fighting or racing games and removing the limit of players. There would be some complications with this in regard to performance on the clients computer however. An example of this with a CS architecture is Beyond Protocol (Dark Sky Entertainment (2009)).

This also leads onto the fact that because there is no central authority and infinite players the network can never be stopped. Meaning that unless everyone independently disconnects from the network it will keep going - giving games an easy way to deal with redundancy as explained in section 3.5.

3.2 Static Data

Static data in a game is the immutable data that does not change over time. It is the meshes, textures and executable data of the game.

Looking at the transfer of static data, there are big advantages with distributed protocols compared to the client-server approach. Because each user contributes to the network they can use the network to send the data to each other in a BitTorrent like fashion (BitTorrent (2009)). Compare this to the client-server way, where dedicated high-speed and expensive equipment is needed to serve clients the static data of the game. This development of peer-to-peer transfers is expanding into commercial games; Playstation 3 is using a peer-to-peer process for transferring data and there are many noted advantages for using this distribution method in all types of online game (McCoy et al. (2003)).

However the problem lies in the fact that in a decentralised network there can be no authority that decides which static data is the correct one. This problem is different to actually transferring the data once it has been decided what the correct data is and will be discussed in the following chapter.

3.3 Latency

The inconsistencies in a networked game are entirely dependant on the latency of packets. If there were no latency then all games and applications would be able to run perfectly given appropriate bandwidth. So reducing this latency is a practice that is crucial to making a game work. The time between one player making an action and that action being reflected on another players computer is the inconsistency time T_i (Pellegrino & Dovrolis (2003)).

In a client-server setting the inconsistency time is dependant on the latency L between the server S and the two (or more) participants P. So for players P_a and P_b, $S(T_i) = P_a(L) + P_b(L)$ (Pellegrino & Dovrolis (2003)).

In a peer-to-peer setting the inconsistency time is dependant on the latency between the two peers. $P_a(T_i) = P_a(L)$. So because it is implausible for a user to know that a client, server or peer has a low latency between them; having less factors in the equation, as the decentralised method does, a peer to peer transfer is more likely to have lower latency than a client to server to client transfer.

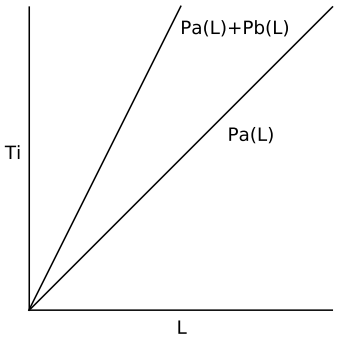

Figure 3.1: Latency - Comparing latency in peer-to-peer networks to client-server networks.

This shows that the latency of a client server game is always more than that of a decentralised game.

3.4 Bandwidth

From a low-level view it appears that peer-to-peer uses less bandwidth in a broadcast of an action than client-server network does.

In the example shown in Fig. 3.2 below there is a action of size x that needs to be broadcast to the interested participants P_b, P_c. In a client-server network this takes one upstream trip and two downstream trips of $3x$ in total. In a peer-to-peer network it takes two upstream

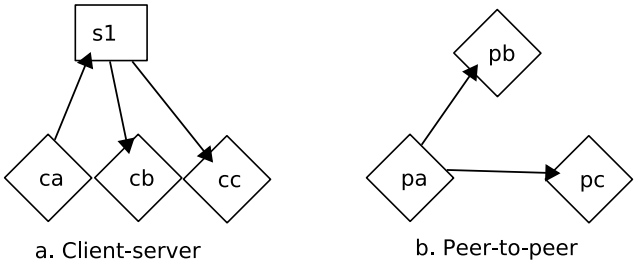

Figure 3.2: Bandwidth - Comparing bandwidth in peer-to-peer networks to client-server networks.

trips of $2x$ in total. This seems like the obvious choice because it involves less bandwidth and less packets, however because the peer-to-peer system will require a much more complicated protocol than client-server it means that overall the total bandwidth may be more than in a client-server depending on the protocol used. For example the action that the peer makes may need to be signed (as in a PKI (Suryanarayana et al. (2006))) thus increasing the bandwidth requirements for peer-to-peer transfer.

Looking at connections, a peer-to-peer network will have more connections in the network than a client-server. Because each peer needs to be connected to every other peer this means an exponential increase in connections with the number of peers on the network. This can be worked out as $\frac{(n-1)n}{2}$ connections where n is the number of peers in the network (A triangle number). Whereas a client-server setup will have n; assuming that there is only one server - one for each client connected.

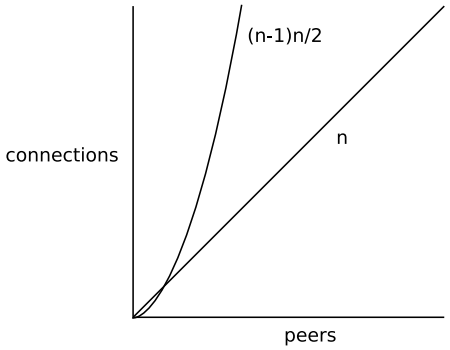

Figure 3.3: Connections - Comparing network connections in peer-to-peer networks to client-server networks.

We can see that the number of connections in the network increases exponentially as the number of peers in the network increases. This means that a peer will be overloaded with

the number of connections quickly. Therefore there needs to be some way to avoid this large number of connections.

3.5 Disconnection

In a client-server network, if the server gets disconnected then all of the clients that are connected to that server at the time of the disconnection will be disconnected or reconnected elsewhere. If there are multiple servers available at the time then the clients may be disconnected for only a brief moment while they are reconnected to another server. If a client gets disconnected from the server on the clients side of the network (I.e. there is a problem with the clients internet connection) then they will have no way to reconnect.

In a peer-to-peer network a client can have multiple physical and logical connections to another peer and even have connections to peers through other peers acting as forwarding proxy's (Brik et al. (2008)). This is also possible in a client-server setup but is less likely because the game design is usually simplified by not letting users have more than one connection. Because of the tree-like nature of client-server games, there will always be one or more points of failure. In a peer-to-peer network this can be eliminated, and in a heterogeneous network can create redundant connections that will both reduce the risk of failure and increase the possibility of getting a lower latency and higher bandwidth connection to other peers. (See sections 3.3 and 3.4).

3.6 Cost

Because a decentralised network has no central, costly servers (although it is possible for users to run servers that host the game for a more reliable connection) it means that this way of handling a game provides hosting for developers that is cheaper than a CS design. Of course, compared to having one or more central locations as in a client-server architecture, a decentralised architecture can be completely free to play. However this also brings about the question of how to generate revenue from the game. There is no easy way of having a decentralised game and having the ability to make users pay for the service. This is because usually some data will need to be kept in a trusted location where a subscribed user is different to a non-subscribed user and private information such as credit card details are stored. This is not possible in peer-to-peer situations because the data sent from the peers cannot be trusted to be correct.

One way to generate revenue could be real-world money: there are a few different ways to encrypt something without changing or knowing about it's contents Damgard & Jurik (2000), D. et al. (1990). These methods of cryptography could be applied to decentralised games to create a real-world value currency in the game. This could be used in a decentralised game whereby a company creates and sells these encrypted tokens as money in the world. When the user wants to use the money token they will go to another peer and the token is then verified, re-encrypted and transferred to the selling peer. When the peer wants to cash-in the token, they send it back to another company that decrypts it.

There are other methods to use real-world currency and they will be discussed in chapter 5. Another way of generating revenue would be in-game advertising. Akin to the advertising that is ongoing in Second Life where companies are buying land to create large in-game advertising posters. This could be accompanied by advertisements in the game executable/user interface itself.

Another way would be to use the game sections described in later chapters. As a user moves into a new game section, they would download the static data for that section. This is discussed in the following chapter, but essentially this downloading of static data would have to be concentrated. This means that the developers could charge for users wanting to expand into new areas of the game. However this would mean that users could send each other the data that they have paid for to other users without paying the developers. In the paper DeCandia et al. (2007) it shows how a distributed filesystem is used to avoid bottlenecks in the Amazon network. This could be applied to the decentralised game too to avoid bottlenecks.

3.7 Discussion

In this section we have investigated the ways we can leverage the benefits of a decentralised network, the advantages of decentralisation and have looked at some of the issues that games would face. The issues that have been discovered are:

- How to identify the static data that is correct and up-to-date.
- How to avoid the exponential growth of connections in the network with relation to the number of peers in the network.
- How to generate revenue from a decentralised game.

These issues will be discussed in the following chapter and from the issues gathered a specification will be proposed for games wishing to use decentralisation with possible implementations proposed in chapter 5.

4

Impact

This section covers the impact of using a decentralised model for networking in a game. It will take the issues raised when researching decentralised games in the previous chapters and create a specification for a decentralised game.

One of the reasons that decentralised networking is avoided in games is that data is kept purely in the hands of the users of the game. This compromises security of the users because other users have to trust them to provide the correct data. Another reason is that there is no easy way to authenticate the data that a user provides to another user, making it relatively easy to cheat in a game without methods to prevent cheating. One reason that it is hard to authenticate users is that a user cannot be identified only by the data that they provide. As in a CS architecture a user is registered and identified by a username and password. This would be difficult to implement in a distributed manner as each peer would have to have access to all other peers usernames and passwords. This means that impersonation of other users is possible without usernames and passwords to identify them. All of these problems can be overcome but the main factor that game developers avoid decentralised networks is that they are inherently complex. This is due to the need to overcome the factors described above. So this section will cover all of these problems and how they relate to the design of game: RTS, FPS and RPG (See chapter 3 for description).

Being a networked computer game means that an application is susceptible to the normal issues of being on a network such as cheating. Therefore a decentralised application must have the requirements of a client in a normal CS networked computer game of the same general design. How these requirements are implemented are going to need to be different but the requirement is still there. Therefore adding the requirement:

- A decentralised application must have the requirements of clients in a normal CS networked computer game of the same general design.

4.1 Connections

From the previous chapter we have found that the number of connections in a network increases exponentially with the number of peers in the network. This is reflected by Chen et al. (2008). This means that eventually a peer in the network will be unable to function due to the number of connections it needs to keep track of and manage. There are methods available to split up the game world either into spatial locations so that peers are only connected to peers that are in the same spatial region. The game could also be split up logically so that each peer is only connected to peers that it needs information about to proceed in the game. There is also the option to use some users as proxy users. I.e. A message of one peer will utilise the already established connections to route the message to another peer that it does not have a connection to.

These options will be discussed in the following chapter and the following requirement will be added to the specification:

- A decentralised game needs to be able to split up the peers in the game so that the number of connections in the network does not expand exponentially.

4.2 Static Data

The problem of the static data; the model, texture and executable data that does not change throughout the game; is a new one when looking at decentralised games. Because there is not a single place for users of the game to get the static game data from, it would have to be acquired by users from various different places and versions of the data would have to be differentiable and verifiable. A developer would create the game and release it to the public. Users would then download that game data, play and eventually update their data with new data. Implicitly the distribution point becomes the central location for users of the game to update, but other distribution points could be added therefore making distribution distributed.

Because users of the decentralised game are more likely to have differing versions of the game than CS games, then the game design should reflect the fact that this can happen. It should allow users play the game with the earlier version, but only with users that are still running that earlier version. This is already done in many games and could be applied to many genres.

This is beneficial to the game to avoid having to disconnect players with differing versions, but not required.

Therefore the following items can be added to the specification of a decentralised game:

- A way to differentiate versions of immutable, static data of a game from each other.
- A way to verify the integrity of the data.
- A way to distribute this data to users of the game.

4.3 Revenue

As discussed previously there are no easy ways to generate revenue for a decentralised game without introducing a single point of failure into the game. Some ideas that could overcome this are:

- Real-world money.
- In-game advertising.
- Selling sections of static game data.

With real-world money the game would have its own currency and this currency would be exchangeable with real currency. However if this were done, it would mean that the place that money were converted would be centralised. It also introduces the problem of mutable data and decentralisation. This is explained in the following section, but basically data in a decentralised game is stored on all peers and not in a central trusted location like in a client-server architecture. One way of making real-world currency work would be to encode a unique identifier at one end of the transaction and make sure that it cannot be decoded whilst it is in the hands of the users but that it can be verified to be authentic. This could be done with a version of Paillier encryption whereby a token is encoded with a unique id and passed around inside the game. A token that can be verified from any user before being transferred and eventually when a user wants to change their token from virtual currency to real-world currency the token is decrypted and the unique id is read to verify the token and to transfer the money to the user (Damgard & Jurik (2000), D. et al. (1990)).

In-game advertising could either be built into the immutable static data of the game or it could be dynamic and retrieved whilst in game-play.

If a company were to produce game data that users could acquire for a cost then the main issue is that users could copy that data to others without paying for it, much like a user can copy a music file to another without paying for it. The only way to solve this problem involves centralisation. This is because there needs to be a central authority that sells these sections of data and authorises their use.

Therefore if a game design requires the ability to generate revenue then one of these options should be chosen. Adding the requirements:

- If a decentralised game requires the option to generate revenue then this can be accomplished in many ways and one or more of these approaches should be chosen before the game is implemented.

4.4 Possessions

In a traditional game the idea of items or possessions is a simple one. An entity has different properties attached to its entry in a database and when a user of that entity wants to pickup an item, that item is moved from one place on that database to another. In a decentralised game the idea of a possession, item or property attached to a entity is different. In the previous example the database is kept in a trusted zone that normal users cannot access. In a decentralised game this data is kept on the peers and is not only accessible by those peers but the users of the game are able present false information when requested and cheat. Therefore in order to create any sort of stable item in a decentralised game there are a few things that need to be looked at.

In a client-server architecture a client will send to the server the action that it would like to happen. The server will then check this action is legal and then broadcast what has happened to the interested clients. This means that the clients implicitly trust the server. If we apply this to a decentralised game and assume that peers trust each other then this allows the peers to cheat by presenting false information when it is implicitly trusted to provide information that is true. For example a user could send an action to sell an item that it should not have but has a copy of. However in a decentralised game the peers have all the data available at the time of the broadcast of the action to be able to check if that action is legal. So a system of implicit trust between peers means that actions could be broadcast and then be checked for legality by other peers. This requires that there be at least one other peer on the network to check the legality of the items that the user has. It also assumes that no collusion between peers would happen (See section 4.6).

When a peer goes offline this also impacts the game. Because in a client-server game, the database is kept up all of the time it can be counted upon to know the state of the game when a client left and restore that state when the client next enters the game. A decentralised game does not have that benefit because each peer is able to disconnect at any time and there is no always-on place to store data. If a peer goes offline it will need to ensure that when it comes back online there is another peer that can verify that it's data is the same as when it left. Meaning that each peer will need to keep a persistent record of the data about each other peer that it knows about so that it can check the legality of the peers data later on when requested to.

From this we can conclude that a game that is decentralised will require the following:

- The ability to save and load data about an entity in the game; persistence of mutable data.
- The ability to verify the legality of an action, and data in that action, that has been broadcast by an entity in the game.
- At least one user in the game at the time of a user entering.

4.5 New Users

The problem of new users to the game has not been dealt with yet. The issue here is that in a normal game, a new user is signed up through a central system and this central system will assign the items that the user has when it starts the game. We will look at the issue in relation to the design of the game.

In an online role playing game the adding of a new user is a process for the trusted, central servers. The clients only receive the information for the users that are online at a point in time and trust that data to be correct. When a new user is added it does not affect the client. This is different if the game were decentralised, as a new user would mean that a peer would have to interact with the new user and if they do this then the peer may have to trust data coming from the new user. For example it would have to trust that the new user has the right amount of gold and has not altered this value. This is true for all persistent, mutable data for the user but is different to verifying the data of the peer. This all means that there is required a way to check that the persistent data on a user is correct and legal as outlined an the previous section, but there is also a requirement for a new user to be able to receive items. And for the

requirement of being able to check the legality of the items there must be a way for a peer to have reference peers that can verify the legality of the items. This will enable a peer to enter the game with a new user and have referees that can vouch for the legality of the new items added to the user when questioned by other peers.

Therefore the following requirements can be added to the specification:

- A new user must be able to receive items as persistent, mutable data.
- The new user must have other peers that can be used as referees to vouch for the legality of the new items when questioned by other peers.

The situation is different in a real-time strategy or first-person shooter design where games in progress may be volatile. In these games when a user begins play they will usually start from scratch. There is only a small amount of data that is persistent in these games like high-scores. So the issue of a new user is not applicable in these situations and can be ignored. However if a real-time strategy that has persistence were introduced then this would have to adhere to the specification outlined for role-playing games to work. Therefore the above specification will be used for games that use persistence instead of just role-playing designs.

4.6 Collusion and Cheating

Collusion is defined as a secret agreement between two or more people to deceive or defraud another. There are two types of collusion: collusion that is performed in the game and collusion in the physical world. For example a two players sitting next to each other could collude about the position of another player; this is physical collusion. Collusion using the game protocol would be when players spawn-camp in a FPS game. We cannot prevent the problem of physical collusion but can with collusion using the protocol of the game. In Lepinksi et al. (2005) it is proven that a finite game can avoid all possible instances of collusion using the games communication protocol. So this requirement should be added to the specification.

In a decentralised game collusion is harder to detect than a client-server situation. This is because as outlined in chapter 2, a peer on the network cannot be identified easily and a new user can be created without cost. So a peer could create two users for the game, transfer the money from one user to the other and delete the empty one. Leaving the peer with a user that is legal with double the money of one. Whereas in a client-server situation the trusted central location would be aware of the same client creating multiple users.

This problem is also present in non role-playing games. For example in a RTS game a peer could be playing two different users in the game and the other peers would not know this. This would allow the user to have double the resources than other players that are playing fairly.

Using combinations of protocols described in section 2.3 means that the protocol that we produce can be free from possibilities for peers cheating using the protocol. For example using the lock-step algorithm (Wolfson (1987), Baughman et al. (2007)) means that peers would not have the ability to see other peers actions before deciding their own - look-ahead cheating. However this does not eliminate all forms of cheating. Cheating is much like collusion in that the policed part of the game can be cheat-free, but the rules outside the game cannot. This leads onto the fact that the games design effects that way that a peer can cheat in the physical world.

The specification points raised in this section are:

- Remove all possible collusion opportunities in the game protocol.
- Use a protocol that does not allow cheating.

4.7 AI

Artificial intelligence in a decentralised game is a relatively new topic. We can take much of the research from non-decentralised game AI and apply it to a distributed game. In games AI is required to be directable and to produce the illusion of intelligence. This means that an AI entity in a decentralised game must be directable. This is usually done through scripting in client/server games: in a traditional CS environment the server would make all of the AI decisions and then relay those actions to the interested clients. In a decentralised game there is no central place to make all of the AI decisions therefore traditional scripting used in centralised games cannot be used in decentralised games without modification. AI decisions usually involve some sort of randomness - be it in script or through differences in hardware - and because each of the peers would retrieve a different AI outcome from this randomness means that the AI decisions could not be made asynchronously and independently.

This presents the fact that AI would not be run in a location that has access to all of the data as is usually the case in client/server games. Therefore the AI script would only have access to the data of the peer that it is running on. This is not feasible because AI in games is complex and requires more than the view of one peer to execute. Therefore we require a

protocol that can collaboratively execute an AI script fairly given each of the peers information on the game.

Therefore we can reason that a decentralised game that uses AI will need:

- A directable AI system that can be replicated to other peers in a consistent manner.
- A protocol that can collaboratively execute an AI script fairly given each of the peers information on the game.

4.8 Performance

As we discovered in the previous section 3.4, there are many more connections, more CPU usage and more bandwidth is required in a decentralised game than a CS game. Therefore the issue of performance is more prominent in a decentralised game than a CS game.

We can use some of the protocols described in section 2.3 to alleviate some of the performance issues in the game much like cheating and collusion, performance is an issue that client/server games face and we can take the design from those games and use it in a decentralised game. The implementation may differ but the design of protocols can be used and modified to be used in decentralised games.

Unlike CS games, a decentralised game will require to be able to broadcast messages and it will require the ability to manage a mesh-like network. These are both points that have not been used in client/servers games before and points that will need to be added to the specification.

Adding to the specification:

- Leverage the already existing methods that client/server games use to get the best performance.
- The ability to broadcast messages effectively.
- The ability to manage a mesh-like network effectively.

4.9 Discussion

In this chapter we have looked at the issues regarding decentralisation and networked computer games. We have created a specification that a game that is decentralised would have for different designs of games. In the next chapter we will take the specification and look at possible implementation details for each of the requirements.

5

Implementation

In this chapter we will take the specification created in the previous chapter and look at possible implementations for each of the points raised. We will begin by presenting the points raised in the previous chapter in a more concise format:

1. A way to differentiate versions of immutable, static data of a game from each other.
2. A way to verify the integrity of the static data.
3. A way to distribute this data to users of the game.
4. If a decentralised game requires the option to generate revenue then this can be accomplished in many ways and one or more of these approaches should be chosen before the game is implemented.
5. The ability to save and load data about an entity in the game; persistence of mutable data.
6. The ability to verify the legality of an action that has been broadcast by an entity in the game.
7. At least two users in the game at the time of a user entering.
8. A new user must be able to receive items as persistent, mutable data.
9. The new user must have other peers that can be used as referees to vouch for the legality of the new items when questioned by other peers.
10. Remove all possible collusion opportunities in the game protocol.

11. Use a protocol that does not allow cheating.

12. A directable AI system that can be replicated to other peers in a consistent manner.

13. A protocol that can collaboratively execute an AI script fairly given each of the peers information on the game.

14. Leverage the already existing methods that client/server games use to get the best performance.

15. The ability to broadcast messages effectively.

16. The ability to manage a mesh-like network effectively.

17. A decentralised game needs to be able to split up the peers in the game so that the number of connections in the network does not expand exponentially.

5.1 Fulfilment

We will now look at each of the requirements in turn and then look at implementation technologies that may be able to be used to fulfil the requirement.

Point 1: *A decentralised game requires the ability to differentiate the multiple versions of static data that a game may produce.* Implementing this requirement would be a matter of adding a version identifier to the data itself. This would fulfil the requirement and could be implemented in the application without using any new technology. The application could then check the version of the data that it has against the version that a remote peer has to check if the versions are compatible.

There is also the argument that the check against the versions could be omitted and the application could assume that every remote peer it connects to will have the same version that the local application has. The application would have to check for inconsistencies in any actions that the peers send and disconnect from the peer if they violate the actions. This would mean that inconsistencies would occur with the peers that do not have the right versions. This would remove or disconnect the peers from each other. Therefore this requirement is needed to prevent unnecessary disconnections of other peers.

Point 2: *A decentralised game requires the ability to verify the integrity of static game data.* This requirement could be implemented with a checksum. The checksum would be applied to the whole version of the data. This checksum would then be checked against other peers checksums to verify that the data is integral between all communicating peers once it has been downloaded. The game should use the versioning requirement above to check static data of the same or compatible versions.

Again, this could be omitted but this would also create inconsistencies and possibly lead to security problems if a user were to have a buffer overflow that might cause some malicious code to be executed for example.

Point 3: *A decentralised game requires a way to distribute the static data to the users of the game.* This could be implemented by using a distributed file transfer protocol. BitTorrent (2009) and Gnutella are examples of distributed file transfer. This requirement is linked with the requirement of versioning and integrity. The version of the static data that needs to be acquired must be known before it can be downloaded, also the integrity of the data must be checked after downloading. This also means that there must be some way to find the peers that already have the version of data that is required. This can be accomplished by identifying the data differently if it is a different version of data. This will enable user to find the version that is required in the same way that they find the correct data to download.

Point 4: *If a decentralised game requires the option to generate revenue then one or more of the discussed approaches should be chosen.* The methods being: real-world money, advertising or selling parts of the game world, or mixtures of the three.

There are a variety of anonymous money transfer implementations such as Bitcoin by Nakamoto (2009) and eCache. These are ways to transfer money anonymously and securely. Bitcoin is a decentralised money transfer implementation that would be suitable for a decentralised game. It does not use trust to secure the transfer but uses proof-of-work Jakobsson & Juels (1999), so this implementation would fit in with any decentralised game that wishes to transfer money.

Advertising could be accomplished purely through the game design or could be built into the application.

Selling sections of the game world could be accomplished by centralisation of the game data, however this would introduce bottlenecks into the network and the issue of users being able to

copy the data without having to pay for it is also present. This requirement may need future work to find a way to enable a decentralised interface with the outside economy and to prevent copyright infringements.

Point 5: *A decentralised game requires the ability to save and load persistent data.* One way of implementing this requirement would be to add some sort of serialisation protocol to the application. There are many serialisation techniques, Boost's serialisation by Ramey (2009) could be used. Serialisation needs to be utilised by other requirements in this specification so it should be outside of the application and available to those requirements, such as the requirement to be able to verify the integrity of the data.

Point 6: *A decentralised game requires the ability to verify the legality of an action it receives from another peer.* This requirement could be implemented in many ways. TCP ensures data integrity however this is not the same as verifying a legal action. A legal action is data that will not conflict with the state of the game or the game protocol. Checking if the action is compatible with the game protocol is usually a feature of the protocol. For example a state machine in a TCP server would know that a read request cannot come before an accept request. Therefore this requirement could be implemented in many ways in the protocol. The ability to verify data is used in multiple requirements so this should be available outside of the application.

Point 7: *A decentralised game requires at least one user in the game at any one time.* This requirement does not have an implementation, the only thing that can be done to accommodate this requirement is to have two dedicated computers to run the game online all of the time. This would defeat the benefit of being decentralised though so this requirement should be optional. If there are enough users on the network then there does not need to be an always up server for the game because there will most likely be a user online at any point in time.

Point 8: *A new user of a decentralised game must be able to receive items from other peers initially.* This requirement is different to just receiving an item through an action from any peer because the user must be marked as a new user and the new user must be able to contact the other peers to initiate the exchange of new items. This is part of the application/design of the game and is specific to the type of game that is being developed.

Point 9: *A user of a decentralised game must have referee peers that can check and vouch for the integrity and legality of the peer in question.* The implementation of this requirement could be developed in the games protocol. Each peer would hold persistently the data of other peers that it is in communication with and would record the transactions that the peer engages in. These could then later be checked against to verify data.

Point 10: *A decentralised game must remove all possible collusion opportunities that peers could take advantage of using the decentralised protocol.* This requirement could be implemented in many ways and this depends on many things like the protocol used and the game design. However it has been shown that collusion can be eliminated (Not collusion in the real-world though) and this needs to be addressed in order for the decentralised game to function correctly. This requirement could utilise some of the protocols discussed in the previous section 2.3, such as the lock-step protocol (Wolfson (1987)) to stop users colluding on the moves that other users have sent before they are sent themselves.

Point 11: *A decentralised game must eliminate all opportunities for peers to cheat in the game.* This requirement is similar to the one above except that it refers to cheating rather than collusion. Cheating can be removed and it needs to be in order for the game to function. This requirement could also utilise some of the protocols discussed in the previous section 2.3, like the above requirement. For example the use proof-of-work (Jakobsson & Juels (1999)) to stop users creating items illegally in the game world.

Point 12: *A decentralised game that needs AI will need a directable AI system that can be replicated in a decentralised manner.* This requirement could be met by implementing a scripting language into the game, much like a CS game. This script would be the same script amongst the peers of the game and it would be packaged with the static data of the game. The script would be an interface to actions in the game and the script would run as a normal server. This would then enable the game to broadcast the actions of the AI script to other peers on the network. This is not a way to execute the AI, but a way to distribute the actions of the AI script. The next requirement will look at how to interpret the actions that are sent through the network and execute them in a collaboratively decentralised way.

Point 13: *A decentralised game must have the ability to collaboratively execute an AI script that will interface with the AI system above.* This requirement is beyond the scope of this report so we will leave this for future work.

Point 14: *A decentralised game must use all possible previous client/server technology to gain the best performance.* This requirement can be implemented in many ways. The protocols described in section 2.3 are a few examples of ways to increase performance in a distributed network. For example using the asynchronous lock-step to decrease the coupling between turns in the game (Baughman et al. (2007)).

Point 15: *A decentralised game must have the ability to broadcast messages effectively.* Chen et al. (2008) gives a good example of how to multicast messages efficiently in a peer-to-peer situation. Splitting up of the game world and the connections between peers will help to broadcast messages to only those that need to receive the messages and reduce the overall number of messages on the network as explained in Boulanger et al. (2006).

Point 16: *A decentralised game must have the ability to manage a mesh-like network effectively.*

There are a variety of systems that manage a decentralised peer-to-peer network. Mithos Waldvogel & Rinaldi (2003), Pastry object routing Rowstron & Druschel (2001), Chord a distributed lookup protocol Stoica et al. (2001), Tapestry object routing Zhao et al. (2001).

These are all examples of a content addressable network (CAN) and varieties of distributed hash tables (DHT) (Ratnasamy et al. (2001)). These implementations could be used to manage the mesh-like network of peers. Mithos for example uses a coordinate system to find the closest peers to connect to. This also helps with the requirement to split up the network below.

Point 17: *A decentralised game needs to be able to split up the peers in the game so that the number of connections in the network does not expand exponentially.* This could be implemented spatially, logically or by using users as proxys.

To implement this spatially, the game world would need to be split up into sections and each section would hold a set number of peers. Each of the sections would then need at least one user to be connected to another section so that all of the sections are connected. This is because for a user to pass between sections a user needs to verify the legality of the data that the user presents to the users in the next section.

To split up the peers logically means that each peer would only be connected to the peers that it needs to be connected to in order for the peer to participate in the game correctly. This means that a method to find the peers that are needed to participate in the game is needed. This could be extracted from the current game state and the peers that the user is currently connected to. However a new user would need to connect to a peer and be referred onto other peers that it would need to connect to.

The game could also be sectioned by allowing peers to become proxys for other peers. Meaning that using the existing connections, a peer could send a message to another peer that it does not currently have a connection to. However this brings with it some problems: a peer could read the data that it is forwarding onto another peer. Chen et al. (2008) shows that routing works and can be used to reduce latency as described above.

5.2 Discussion

In this section we have looked at the specification created in the previous chapter. We have inspected each of the points raised and provided examples of implementation technologies for each. In the next chapter we will conclude the paper with a discussion of the findings.

6

Conclusion

This section concludes the paper and offers a summary of the points discussed in previous chapters.

The initial chapter of this report introduced the topic of decentralised games. In this chapter we looked at traditional networked computer games categories and how decentralised games are different.

In the second chapter of this report we looked into the research and literature that has already been created in the field of decentralised networking, games and design. We found many papers on decentralised networking for games but few that were not centralised in some way.

Next we looked into how a game could leverage and take advantage of being decentralised. We found many benefits for a game being decentralised compared to centralised. However we discovered there were some issues with decentralised games that required additional design for the idea to work.

We looked at the issues that were found and any new issues in the next chapter. We found many issues and discussed how these may impact the game. These issues were then collected into a specification that a decentralised game must follow in-order for it to succeed.

The specification created in the previous chapter was then looked at to find any possible implementations that could fulfil each of the specification points.

6.1 Summary

In summary the report created the following specification that developers should follow when considering a decentralised design for their networked computer game:

1. A way to differentiate versions of immutable, static data of a game from each other.
2. A way to verify the integrity of the static data.
3. A way to distribute this data to users of the game.
4. If a decentralised game requires the option to generate revenue then this can be accomplished in many ways and one or more of these approaches should be chosen before the game is implemented.
5. The ability to save and load data about an entity in the game; persistence of mutable data.
6. The ability to verify the legality of an action that has been broadcast by an entity in the game.
7. At least two users in the game at the time of a user entering.
8. A new user must be able to receive items as persistent, mutable data.
9. The new user must have other peers that can be used as referees to vouch for the legality of the new items when questioned by other peers.
10. Remove all possible collusion opportunities in the game protocol.
11. Use a protocol that does not allow cheating.
12. A directable AI system that can be replicated to other peers in a consistent manner.
13. A protocol that can collaboratively execute an AI script fairly given each of the peers information on the game.
14. Leverage the already existing methods that client/server games use to get the best performance.
15. The ability to broadcast messages effectively.
16. The ability to manage a mesh-like network effectively.
17. A decentralised game needs to be able to split up the peers in the game so that the number of connections in the network does not expand exponentially.

Various implementations were looked at to provide each of the requirements of the specification in the previous chapter.

6.1.1 Applicability

Throughout the report we have looked at the applicability of decentralisation to games. A game that wishes to be decentralised in design will need to have major features re-designed to accommodate for the game being decentralised. This may put off developers from creating a decentralised game, but the advantages of making the game decentralised are quite extensive. For example there is no cost for increasing the capacity of the network and some games could take leverage features that were not possible before. Although there is a big difficulty in generating revenue from this type of game it can be accomplished.

6.2 Future Work

This report has created a specification that a game developer should follow in order to create a decentralised game. A follow-on project for this would be to create a decentralised game and examine how implementation progresses and how well or not the game succeeds. There is also opportunity for a library to be specifically created to implement each of the specification points raised for games to use to easily create decentralisation.

There is also the implementation of a decentralised AI system that was not looked at because it was outside the scope of the project. This could be looked at in future work.

Glossary

LAN local area network; a set of systems linked over a small area.

MMOG massively multi-player on-line game; a game with many clients connected at the same time.

P2P peer-to-peer; an alternate to the client-server model of networking where each participant is both a client and a server at the same time.

Lag latency; the lowest time between transmission of a message and receiving the message due to mainly physical hardware limitations.

CS client-server; a client-server architecture.

API application programming interface;

PKI public key infrastructure.

FPS first person shooter.

RTS real-time strategy.

IP internet-protocol.

CPU central processing unit.

DHT distributed hash table; a table of key-value pairs that is distributed amongst peers.

TCP transmission control protocol.

References

S. Aggarwal, et al. (2004). 'Accuracy in dead-reckoning based distributed multi-player games'. In *NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, pp. 161–165, New York, NY, USA. ACM. 9

N. E. Baughman, et al. (2007). 'Cheat-proof playout for centralized and peer-to-peer gaming'. *IEEE/ACM Trans. Netw.* **15**(1):1–13. 9, 25, 32

P. B. Beskow, et al. (2008). 'Latency reduction by dynamic core selection and partial migration of game state'. In *NetGames '08: Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*, pp. 79–84, New York, NY, USA. ACM. 6

I. BitTorrent (2009). 'BitTorrent'. http://www.bittorrent.com/download.html. 13, 29

J.-S. Boulanger, et al. (2006). 'Comparing interest management algorithms for massively multiplayer games'. In *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, p. 6, New York, NY, USA. ACM. 8, 32

V. Brik, et al. (2008). 'A measurement study of a commercial-grade urban wifi mesh'. In *IMC '08: Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, pp. 111–124, New York, NY, USA. ACM. 2, 16

M. Chen, et al. (2008). 'Utility maximization in peer-to-peer systems'. In *SIGMETRICS '08: Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pp. 169–180, New York, NY, USA. ACM. 20, 32, 33

E. Cronin, et al. (2004). 'An Efficient Synchronization Mechanism for Mirrored Game Architectures'. *Multimedia Tools Appl.* **23**(1):7–30. 6

C. D., et al. (1990). 'Untraceable electronic cash'. In *In Proceedings on Advances in Cryptology*, pp. 319–327, New York, New York, NY. S. Goldwasser, Ed. Springer-Verlag. 17, 21

I. B. Damgard & M. J. Jurik (2000). 'A Generalisation, a Simplification and some Applications of Paillier's Probabilistic Public-Key System'. *BRICS (Basic Research in Computer Science)* **RS-00-45**. 17, 21

Dark Sky Entertainment (2009). 'Beyond Protocol'. http://www.beyondprotocol.com/. 13

G. DeCandia, et al. (2007). 'Dynamo: amazon's highly available key-value store'. *SIGOPS Oper. Syst. Rev.* **41**(6):205–220. 17

P. J. Denning (2005). 'The locality principle'. *Commun. ACM* **48**(7):19–24. 7

Electronic Arts Inc. (2009). 'Command and Conquer Series'. http://www.commandandconquer.com/. 12

Entertainment Software Association (2007). *Best-selling video game super genres by units sold, 2007*. ESA. 12

L. Gautier & C. Diot (1998). 'Design and Evaluation of MiMaze, a Multi-Player Game on the Internet'. In *International Conference on Multimedia Computing and Systems*, pp. 233–236. 9

J. Gu, et al. (2007). 'Decentralized Trust in Distributed Networks: a Delegate-based Security Hardening Approach'. *IJCSNS International Journal of Computer Science and Network Security* **7**(10):265–272. 8

M. Jakobsson & A. Juels (1999). 'Proofs of Work and Bread Pudding Protocols'. In *Communications and Multimedia Security*, pp. 258–272. Kluwer Academic Publishers. 29, 31

D. R. Jefferson (1985). 'Virtual time'. *ACM Trans. Program. Lang. Syst.* **7**(3):404–425. 9

B. Knutsson, et al. (2004). 'Peer-to-peer support for massively multiplayer games'. vol. 1, pp. 106–97. IEEE INFOCOM. 3, 7

M. Lepinksi, et al. (2005). 'Collusion-free protocols'. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pp. 543–552, New York, NY, USA. ACM. 24

K. Li, et al. (2004). 'Analysis of state exposure control to prevent cheating in online games'. In *NOSSDAV '04: Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*, pp. 140–145, New York, NY, USA. ACM. 8

A. McCoy, et al. (2003). 'Game-state fidelity across distributed interactive games'. *Crossroads* **9**(4):4–9. 3, 13

Monumental Games Ltd. (2008a). *Football Superstars* ©. Cyber Sports Ltd., Dunstan House, 14a St Cross Street, London, EC1N 8XA. 1

REFERENCES

Monumental Games Ltd. (2008b). *Monumental Technology Suite*. Monumental Games Ltd., Price House, 37 Stoney Street, The Lace Market, Nottingham, NG1 1LS. 6

K. S. Moon, et al. (2007). 'Efficiently Maintaining Consistency Using Tree-Based P2P Network System in Distributed Network Games'. 9

S. Nakamoto (2009). 'Bitcoin: A Peer-to-Peer Electronic Cash System'. http://www.bitcoin.org/. 29

J. D. Pellegrino & C. Dovrolis (2003). 'Bandwidth requirement and state consistency in three multiplayer game architectures'. In *NetGames '03: Proceedings of the 2nd workshop on Network and system support for games*, pp. 52–59, New York, NY, USA. ACM. 6, 14

R. Ramey (2009). 'Boost Serialization'. http://boost.org/doc/libs/release/libs/serialization. 30

S. Ratnasamy, et al. (2001). 'A scalable content-addressable network'. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 161–172, New York, NY, USA. ACM. 32

A. Rowstron & P. Druschel (2001). 'Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems'. *Lecture Notes in Computer Science* **2218**:329–?? 32

P. M. Sharkey, et al. (1998). 'A Local Perception Filter for Distributed Virtual Environments'. In *VRAIS '98: Proceedings of the Virtual Reality Annual International Symposium*, p. 242, Washington, DC, USA. IEEE Computer Society. 9

S. Singhal & M. Zyda (1999). *Networked virtual environments: design and implementation*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA. 9

S. K. Singhal (1997). *Effective remote modeling in large-scale distributed simulation and visualization environments*. Ph.D. thesis, Stanford, CA, USA. 9

I. Stoica, et al. (2001). 'Chord: A scalable peer-to-peer lookup service for internet applications'. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 149–160, New York, NY, USA. ACM. 32

G. Suryanarayana, et al. (2006). 'Architectural support for trust models in decentralized applications'. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pp. 52–61, New York, NY, USA. ACM. 7, 15

I. The Tor Project (2009). 'Tor Project'. http://www.torproject.org/. 8

B. Tom, et al. (2004). 'The Effects of Loss and Latency on User Performance in Unreal Tournament 2003'. In *ACM Network and System Support for Games Workshop*. ACM. 3

J. Vogel & M. Mauve (2001). 'Consistency control for distributed interactive media'. In *MULTIMEDIA '01: Proceedings of the ninth ACM international conference on Multimedia*, pp. 221–230, New York, NY, USA. ACM. 9

M. Waldvogel & R. Rinaldi (2003). 'Efficient topology-aware overlay network'. *SIGCOMM Comput. Commun. Rev.* **33**(1):101–106. 32

O. Wolfson (1987). 'The overhead of locking (and commit) protocols in distributed databases'. *ACM Trans. Database Syst.* **12**(3):453–471. 8, 25, 31

Y. Zhang, et al. (2006). 'Globally synchronized dead-reckoning with local lag for continuous distributed multiplayer games'. In *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, p. 7, New York, NY, USA. ACM. 9

B. Y. Zhao, et al. (2001). 'Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and'. Tech. rep., Berkeley, CA, USA. 32

Appendices