A CLOUD-BASED ARCHITECTURE FOR DISTRIBUTED PROCESSING IN NETWORKED GAMES

A thesis submitted in fulfilment of the requirements

For the degree of

Doctor of Philosophy

In the

Faculty of Computing and Engineering

School of Computing and Information Engineering

Of

Ulster University

By

Craig James Hull

May 2017

I confirm the word count of this thesis is less than 100,000 words

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

This journey would not have been possible without the support of my parents and my brother. Thank-you for your constant encouragement and reassurance that I could complete this, it made this entire process a lot easier. This is for you and for absent family.

Thanks to my supervisors, Dr Darryl Charles, Prof Philip Morrow and Prof Gerard Parr for their insight and assistance throughout this process.

To Conor, Adam and Aditya, thanks for the laughs in the office which made the long days go in that bit quicker. One way or another we got there in the end.

To my friends, thank-you for all your support.

ABSTRACT

This thesis presents a framework for the improvement and maintenance of the QoS (Quality of Service) of networked video games. Hardware and software are monitored by certain variables and decisions are made based on the range of these. A high QoS is sought after by all players of video games, however delivering this has proven to be difficult at times. Cloud gaming technology has greatly improved distributed processing, though there are still factors inhibiting it. High load on servers and high round trip time to the user's devices and consoles are preventing the users from achieving a high QoS. With games becoming more accessible, the range of devices that they can be executed on increases, though the quality varies from device to device. The servers which the game providers utilise can come under stress. For example, the hugely popular augmented reality game Pokémon Go came under fire from users as the servers could not handle the stress of the huge number of connected users resulting in server outages [1]. Another challenge is the issue of latency as many users may suffer from a low bandwidth internet connection which results in a poor user experience. It would be ideal for all game players to achieve a high QoS regardless of the device they are using, their connection to the server and the condition of the server. The research hypothesis underpinning the work described here is that cloud gaming techniques can be utilised to improve a user's QoS

A novel and adaptable architecture that combines cloud and fog assistance with self-adaptation techniques, in which the client adapts to a situation, is proposed as a solution to this problem. By employing available resources from the game server (cloud) and other under-utilised network nodes local to the device (fog), a game player's QoS may be improved. Self-adaptation procedures are the last resort solution of the architecture should there be no available resources both locally and globally. Testing of this architecture is carried out under various conditions from varying latencies and packet loss to data packets of differing size being distributed and self-adaptation occurring due to different in-game elements. Results from experiments based on varying pressures in the game world and network conditions show that, by constantly monitoring the QoS of the game and the network, effective decisions can be made to improve a declining QoS. A smaller data packet transmitted frequently provides a greater improvement in comparison to a larger data packet transmitted less frequently.

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

QoS	-	Quality of Service
QoE	-	Quality of Experience
FPS	-	Frames Per Second
DB	-	Database
MIB	-	Management Information Base
MCVG	-	Mobile Cloud Video Gaming
MBG	-	Mobile Browser Gaming
MCG	-	Mobile Cloud Gaming
SNMP	-	Simple Network Management Protocol
RPC	-	Remote Procedure Call

NOTE ON ACCESS TO CONTENTS

I hereby declare that with effect from the date on which the thesis is deposited in the Library of Ulster University, I permit:

1. The librarian of the University to allow the thesis to be copied in whole or in part without reference to me on the understanding that such authority applies to the provision of single copies made for study purposes or for the inclusion within the stock of another library.
2. The thesis to be made available through the Ulster Institutional Repository and/or Ethos under the terms of the Ulster eTheses Deposit Agreement which I have signed.

CHAPTER 1 INTRODUCTION

1.1 OVERVIEW

This thesis proposes an architecture for improving and maintaining the QoS provided in video games. A wide variety of factors can affect QoS, for example there is the specification of the client device and the current scene within the video game within the client. Example metrics from the client and game include FPS (Frames per Second), GPU (Graphical Processing Unit) and CPU (Central Processing Unit) usages. From all the factors it can be argued that the most important of these is the FPS value of the game [2]. By improving these metrics, the overall QoS provided improves. It can be argued that with an improvement in QoS, the Quality of Experience (QoE) also improves since if a good QoS is provided then a good QoE is only dependent on subjective measures from the user such as their current mood [3]. With devices becoming increasingly network aware, the utilisation of distributed resources can help prevent and reverse a declining QoS. As some nodes on a network, such as other computers, can be under-utilised, processing can be distributed to these. These nodes must also be monitored regarding resources and the network between them.

In this thesis, an adaptive architecture will be proposed in response to low QoS in video games. This architecture will draw upon cloud and fog assistance and self-adaptation techniques to maintain a high QoS. The main features of a video game will be identified and methods of their distribution detailed. Each of these features will have a set of resource requirements which will be fulfilled somewhere along the network or be reduced until the client device itself can handle it. The constant change in resource requirements and availability requires a Decisions Manager who will optimise the distribution process.

The chapter continues as follows: In Section 1.2, distributed processing is examined and becomes more specific with video streaming and cloud distribution. The section also explains the benefits of fog computing for improving the QoS of a video game. Section 1.3 sees the introduction of the research objectives of this thesis that go towards creating an architecture that will improve a user's QoS. Section 1.4 is a summary of the thesis contributions and section 1.5 is an outline of the thesis structure.

1.2 THE DISTRIBUTION OF PROCESSING

The phrase "many hands make light work" crosses over into the realms of computing and becomes the term distributed processing or distributed computing. Distributed processing is defined as hardware or software components, located in a computer network, which communicates and coordinate their actions by passing messages for higher overall efficiency [4], [5]. There are many benefits to distributed processing, one of which is performance enhancement. Multiple computers or

devices working on a problem can solve the problem much faster than a computer or device working alone. Games are applications that can benefit greatly from distributed processing.

At present, there are two methods of utilising distributed processing to benefit video games:

1. Streaming media
    a. Video Streaming
    b. Graphics Streaming
2. Adding additional hardware to the network

Of the two methods, the process of streaming media is the most popular with Video Streaming being used the most in the current industry. A live example of this is PlayStation Now [6]. Video streaming is the process of rendering on the server and sending the resulting images to the client device while Graphics streaming is the process of sending graphics commands to the client device which will then render the game images. Much research in the literature focuses on streaming media approaches which has resulted in many different frameworks:

1. Co-operative Video Sharing

    Co-operative video sharing is the process of sharing video contents with other users. Relating to games, the video that is streamed down to a group of users can be decoded co-operatively. This process of streaming video and the decoding of players videos who are in the same gaming scene, via a secondary network, that most modern devices can be connected to, can reduce server transmission rate [7].

2. Asymmetric Graphics Rendering

    Asymmetric Graphics Rendering is the process of encoding the left or right view of a 3D scene differently from its opposite, which can reduce the bandwidth required to transmit the 3D stream. It has been shown, [8]–[10], that encoding one view at a high enough quality and encoding the other at a quality above a set threshold does not noticeably lessen the quality.

3. Mobile Cloud Gaming

    Mobile cloud gaming is the process of playing a game that resides on the cloud on a mobile device. The two current approaches to this are Mobile Cloud Video Gaming (MCVG) and Mobile Browser Gaming (MBG). MCVG is the streaming of video down to the mobile device whereas MBG is graphics rendering on the client side. In [11], a system is proposed which is adaptable and changes between MCVG and MBG based on network conditions.

4. Games@Large

Games@Large was an EU project focused on enabling consumer electronics platforms and devices to be video game ready [12], however, it was shut down in 2010. This example also combines Video Streaming and Graphics Streaming into an adaptable architecture as seen in [13]–[16].

1.3 RESEARCH OBJECTIVES

Section 1.2 presented the current methods of distributed processing in video games which are to stream video and to distribute to the cloud. The number of issues with these methods such as the potential for server overload, can be reduced by implementing fog assistance as well as self-adaptation. By utilising fog computing, round trip times can be reduced as well as reducing server strain, and self-adaptation will allow the device to adapt to its current situation when there are no resources available on the network. Therefore, this thesis proposes that by distributing processing to under-utilised resources and focusing on specific QoS metrics, a user's QoS can be improved. The focus on these is channelled into the combination of cloud assistance with fog assistance and self-adaptation. The overall aim of the thesis is to provide an architecture which monitors a game's state and makes decisions taking account of the state of the client device, network nodes and the state of the network in its decision-making process. The results of these decisions are intended to improve the QoS provided and therefore improve the user's overall QoE. The following research objectives for this work have been identified:

1. To review existing methods of distribution in video games, in particular, how they distribute data and which data they choose to distribute.

2. To determine which parameters can be used in a decision-making architecture, in which the outcome of all decisions is to improve the QoS provided.

3. To develop an adaptable architecture which will utilise the cloud and fog resources available to improve QoS. This architecture can then be tested against differing data types and connection variations. Self-adaptation will be included as a last resort - in the unlikely event that no resources are available from either the cloud or fog, the client device can adapt itself.

1.4 THESIS CONTRIBUTIONS

In this thesis, an adaptable architecture is presented that improves upon a declining QoS and then maintains the improvement. This architecture makes decisions based on information from all over a network and employs under-utilised resources to provide a good QoS. The information is stored in Management Information Bases (MIBs) which are present on all devices. A MIB is a virtual information

store which holds objects “whose values collectively reflect the current state of the network” [17]. The MIBs here hold data such as the game’s current FPS rate, the bandwidth of the connection and the current GPU usage. If the resource values within a MIB, such as CPU usage, are low then the device that the MIB belongs to will be flagged as being able to provide some assistance with CPU intensive tasks. Under-utilised resources, such as low CPU and GPU usage rates of devices, can be employed to provide assistance. The closer an under-utilised device is to a client requiring assistance the more likely it will be drafted in to assist. By employing a device much closer to the client to provide assistance, processing pressure is taken from the server which reduces the likelihood of a server crash. The QoS provided is improved as the server can operate efficiently, any tasks handed off for outside processing can be executed, and results returned faster. If there are no available resources to assist with processing, a fail-safe procedure is in place in the form of self-adaptation. In the case of self-adaptation, unnecessary objects not core to the gameplay can be completely removed so as to free some of the client device’s resources.

The main goal of this research is to develop an adaptable architecture which will utilise cloud and fog resources to improve the QoS provided. The hypothesis being worked from is that cloud gaming techniques can be utilised to improve a user’s QoS.

The main contributions of this research are:

- The evaluation of current Cloud and Fog Computing methods in relation to video games and the improvement in the QoE provided.
- The development of an architecture which utilises Cloud and Fog computing concepts and combines these with a self-adaptation component to create an adaptable architecture for the improvement of QoS in networked games.
- The analysis of the architecture and the results from experimental testing.

The proposed contributions have been peer reviewed in the following conference proceeding:

- Hull, C. et al. FRAGED: A Framework for the Adaptive Game Execution and Delivery to Improve the Quality of Experience in Network Aware Games. PGNet 2014, Liverpool.

1.5 THESIS STRUCTURE

The remainder of the thesis contains five further chapters: Background and Literature Review, Architecture, Experimental Method and System Setup, Results, and Conclusion.

Chapter 2 is the literature review of related work and covers a wide variety of areas. The related research is divided into five areas: Streaming Games, Distributed Environments, Distributed Management, Fog Computing and Saving Energy. The area of streaming games focuses mainly on the cloud and the streaming of data to the client. Some of the methods proposed have an adaptable architecture, which is similar to one of the core objectives (Section 1.3).

Chapter 3 explains the proposed architecture that utilises distributed resources to improve a user's QoS in network aware games. The architecture is discussed in high-level to low-level detail from the range of possible scenarios for a client to the decision-making involved and the role of Remote Procedure Calls (RPCs) and the network within the architecture.

Chapter 4 focuses on the architectures ability to make decisions. The decision making is a theoretical analysis of the architecture with regard to its ability to decide how best to assist a client and decisions such as which network node will assist are explored.

Chapter 5 presents the results of the distribution of data and the adaptation of the client to its current situation. The distribution of data in this architecture is explored in a number of ways as a variety of methods are used with different network conditions. The self-adaptation ability is a component which executes only when there are no resources available on the network; this is the fail-safe of the architecture.

Chapter 6 concludes the thesis by evaluating the objectives established in section 1.3. Suggestions for future work are proposed to improve upon the completed work and extend the scope of research before presenting a statement concluding this work.

CHAPTER 2 BACKGROUND AND LITERATURE REVIEW

2.1 OVERVIEW

This chapter presents background research and a literature review carried out in a variety of areas. The related research can be divided into five areas: Streaming Games, Distributed Environments, Distributed Management, Fog Computing and Saving Energy. Each area is related to the overall aim of this work which is to develop an architecture that utilises distributed resources to improve the QoE of networked games.

2.2 INTRODUCTION

In this chapter, existing research relating to distributing processing is discussed. In this research, there are many variations of distributed data and many variations in the type of environment the data is distributed within. As discussed in Chapter 1, by monitoring and improving QoS measures within a network, both hardware and software, the QoE that is observed by the user can be improved.

The area of streaming games focuses mainly on the cloud where client devices connected to the cloud have the game streamed to them. Data streamed to the client can take one of two forms, video or graphics commands. Some of the methods proposed under streaming games are found to be highly adaptable to the current situation of the network. One of the research objectives of Chapter 1 highlights the need for such an adaptable architecture.

A distributed environment is one in which networked computers cooperate to achieve a common goal. In such an environment, there can be multiple servers over which load balancing can take place or one server which can delegate processing to remote clients. This research considers multiple server situations as well as distributed virtual environments. The employment of additional hardware to assist with processing falls under this area.

Distributed Management is required to oversee the distribution of data over the current environment. There are many different methods of distributed management, the research for this thesis will focus on agents. These agents can be better understood as a set of rules. If a condition is met, then the agent will act. For example, if the frames per second at which a game is running drops below a set threshold then a process will be executed.

Fog computing is very similar to cloud computing with the difference being that the processes executed in the Fog are being executed much closer to the client device. An advantage is that the result of the executed process is received much quicker by the client in comparison to the result being transmitted from the cloud.

Through distribution, it is possible to save energy. By employing underutilised nodes within a network clients or servers under pressure can be relieved of tasks. This can reduce the load on the system and therefore reduce its own energy consumption.

The final section of this chapter summarises the literature review and identifies opportunities for the development of an adaptable architecture.

2.3 STREAMING GAMES

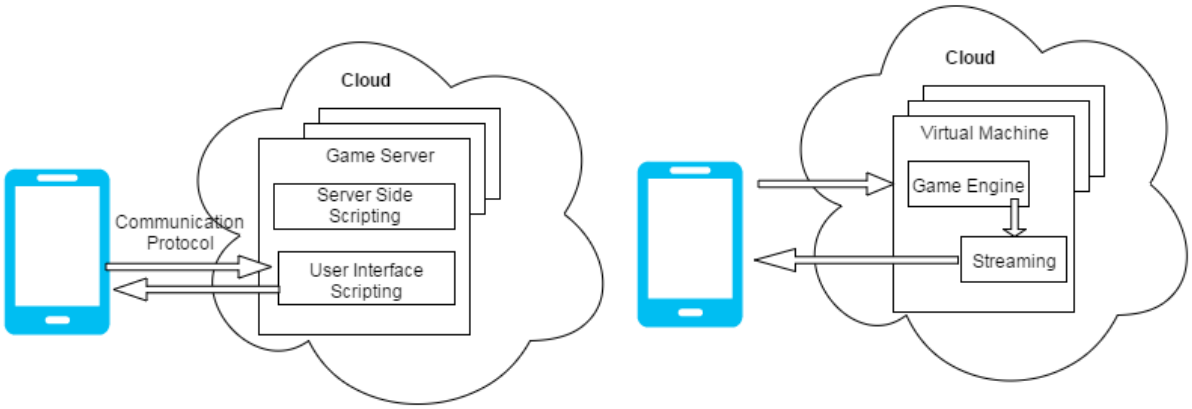
The concept of media streaming is not new with attempts made to display media on computers dating back to the mid-20th century. Little progress was made with this due to the hardware's limited capabilities [18]. Nowadays significant developments have been made on home networking including streaming technology associated with game giants Nintendo, Sony and Microsoft. For example, the Wii-U can wirelessly stream data to the gamepad controller with only a 1/60th of a second delay [19], and PlayStation Vita and a variety of smartphones and Tablets can provide a similar function for PlayStation and Xbox consoles respectively. Steam OS and associated computer hardware can stream games over the home network direct to a television [20]. Other significant developments include the Nvidia Shield [21] which can come in the form of an Android TV box or tablet. Owners have the option to stream games from their home PC or Nvidia's own streaming service GeForce Now, which is specifically for the Shield.

The rise of Nvidia's GeForce Now isn't the only commercial change with regards to streaming games. For a long time the giants of this arena were OnLive [22] and Gaikai [23], however, as of 2012 Sony acquired Gaikai, and in 2015 Sony acquired OnLive's patents meaning that OnLive services would be discontinued. During 2015 Sony released a cloud gaming service dubbed 'PlayStation Now'. This service allows the streaming of PlayStation 3 games on PlayStation 4 and other compatible devices including PCs via a thin client on the device. Currently, there are two approaches to streaming games, namely video streaming and graphics streaming. These are implemented in a variety of research platforms including Mobile Cloud Gaming (MCG), the Games@Large framework, Remote Visualisation and Asymmetric Graphics Rendering.

2.3.1 MOBILE CLOUD GAMING

MCG is the process of playing a game that resides on the cloud on a mobile device. This is now seen as a form of Gaming as a Service. Being able to tap into the seemingly unlimited power of the cloud, mobile cloud gaming can overcome the limitations of the mobile device.

A detailed summary of MCG is provided in [11]. Two current approaches to MCG are discussed below: Mobile Cloud Video Gaming (MCVG) and Mobile Browser Gaming (MBG) as seen in Figure 2.1a and 2.1b.



(a) Mobile Browser Gaming (MBG) (b) Mobile Cloud Video Gaming (MCVG)

Figure 2.1. Two approaches to Mobile Cloud Gaming adapted from [11]

MCVG allows the mobile device to act as a receiver for the in-game video as the video is streamed, whereas for MBG the web browser of the mobile device is the game container and the rendering of the game graphics is carried out on the mobile side.

The key advantages, such as utilising a Thin-Client, and disadvantages, such as the need for a strong network, of a MCG system are discussed. In [11] a framework is proposed for the next generation of MCG and the main findings of this work are the proposed framework and the issues with regards to creating it. The main advantage of the designed framework is its adaptability, also known as dynamic cloud integration. The proposed system can switch between the video streaming of MCVG and the client-side graphical rendering of MBG based on the network conditions by on-loading and off-loading game components between the device and the cloud.

The key finding in this paper is that the division of game components and splitting them between the cloud and the device can improve a player's gaming experience as more processing power is available via the cloud.

The research approach used in [11] defines two case studies, an augmented-reality cloud game and a context-aware cloud game to illustrate the next generation of MCG. The augmented reality cloud game is Google's own Project Glass [24] in which the player's view is captured, and the cloud responds by providing gaming content such as coins displayed in the player's view. The context-aware cloud game takes the location of the player, via GPS and delivers gaming content.

This research is connected to the wider research field as other authors are looking at adaptable frameworks that can switch between video and graphics streaming and others are looking at offloading processes to the cloud to improve processing capability. Some research looks specifically at video streaming and proposes a system in which gateways are placed between access points and the media servers [25]. Based on the device the client is using, the video stream can adapt its quality. The client may also be able to connect to another gateway allowing them continuous access to their video stream. This is similar to the work in [11] as a system is created that can adapt based on the client's device. The research in these papers is very relevant to our project as we will be creating an adaptable game delivery system in which game components will be offloaded to the cloud and other devices available.

2.3.2 GAMES@LARGE

Games@Large was a four-year EU project focused on enabling consumer electronic platforms and devices to be video game ready. According to [26] the goal of Games@Large was to "provide instantaneous, ubiquitous access to high-end videogames". This was to be achieved without specialised hardware at any end of the network and without the requirement of significant network resources. The Games@Large framework has two approaches to streaming games; these are graphics and video streaming [13]–[16].

Graphics streaming is used mainly for end devices, such as a PC, with accelerated graphics support typically having screens of higher resolution [14], [16]. All calls on the OpenGL or DirectX library are intercepted, encoded and streamed. Using this streaming method, encoding is much less demanding and independent from the image resolution of the device. With this, high image quality is achieved as the game scenes are directly rendered for the desired screen. However, bit-rates are less predictable, and high peaks of data rate are expected, especially for scene changes where numerous textures have to be loaded by the graphics card [15].

Video streaming in the Games@Large approach is only used when graphics streaming is not applicable, an example of this type of situation is when the client lacks the required graphics hardware. With video streaming, the rendering takes place on the server and the resulting frame buffers are captured and encoded as a H.264 video stream [13]. This is explained in more detail in [14] which goes further in saying that the rendering commands are intercepted and modified before their execution to exactly meet the client's properties without any image degradation or processing delay. This type of streaming is computationally demanding due to the H.264 encoding on the server side and the decoding on the client side [16].

From this research, it can be seen that a low-end device with low graphics capability will require video streaming while a more capable device such as a desktop will utilise graphics streaming. They describe a framework that, based on the end user's device, will adapt and use either graphics or video streaming.

The research approach used in these papers varies. In [15] and [16] the Games@Large framework is tested with experiments such as adding artificial traffic to the network with tools such as jPerf [27]. In [15] the graphics commands were compressed before being transmitted on the network for the graphics streaming approach using real-time compression, the result was that the delay was significantly reduced. The research in [16] provided no experimental results, this paper discusses accelerated video streaming and presents the advancements in video encoding which is enhanced for games.

The Games@Large framework is a significant step in the direction of adaptive game streaming and is linked closely to Mobile Cloud Gaming (Section 2.3.1). The research in [28] discusses the Games@Large framework, with a particular focus on the video streaming approach. The framework is applied to games which employ a skybox or skydome, these are backgrounds which make the game world seem larger than what it is. The skybox/skydome will take up the majority of any scene and with these encoded in a faster manner, the stream can be presented to the user quicker. This is similar to the research in [13]–[16] as an aspect of the Games@Large framework is being utilised. This literature is significant as, with this project, we are looking to implement a framework that can adapt based on the conditions of the network and keep the players QoS high.

The research here contributes to the knowledge of the Games@Large framework, providing detail on graphics and video streaming and results from experiments that prove its validity against traditional approaches.

2.3.3 REMOTE VISUALISATION

Remote Visualisation is the process of connecting to a remote server and utilising its CPU and GPU power. It is claimed that "image compression alone cannot guarantee interactive framerates" [29]. To reduce the strain on bandwidth, several techniques are used including reducing quality during animations. The main finding of this research is that lossless image compression algorithms coupled with parallel processing can significantly increase framerates and that at the time of writing, hardware compression had very little impact.

A key finding of this research is video streaming can be made more efficient with the bandwidth reducing techniques found in [29]. In the wider research field, this work is similar to [28] as it also

focuses on video streaming to make it a faster process. Another paper of note is [30] in which remote visualisation of large 3D models occurs with a MPEG-4 streaming architecture. The MPEG-4 encoding is seen to be the bottleneck and more specifically the motion estimation. Results in this paper show that with motion estimation being carried out elsewhere, the encoding process is carried out at a faster rate.

The work in remote visualisation is relevant to this project as processing is being passed off to another PC and improving upon the QoS that the client is receiving.

2.3.4 ASYMMETRIC GRAPHICS RENDERING

Asymmetric graphics rendering is the process of encoding the left or right view of a scene differently in comparison to the other; this applies to 3D gaming. An example of this is that the left view of a scene has 150 metres of view while the right scene only has 100 metres of view. By reducing the bandwidth required to transmit the stream, 3D gaming can be enhanced.

3D display gaming on mobile devices is on the rise [31], [32] and with this, an asymmetric graphics rendering approach has been proposed [33]. The research shows the left and right views of a scene being encoded at different bitrates to overcome the challenge of ensuring a good QoE when streaming 3D video over a network.

The key finding of this research area is that by transmitting one view as medium quality and the other as low, a better peak signal to noise ratio is gained over a case where both views are transmitted as medium quality. “In this way, the user experience can be greatly improved whether under the same network condition (increase video quality) or the same video quality (decrease network delay).” [33]

The research gap addressed by [33] is that by altering a view on a 3D video stream, bandwidth usage can be reduced while maintaining the QoE for the user. In the wider research field, this research links in with mobile cloud gaming and video streaming as a 3D video stream that is modified to require less bandwidth is proposed for mobile devices. This is relevant to the project as mobile devices may be incorporated into the system and reducing bandwidth usage is a priority regardless of the client device.

2.4 DISTRIBUTED ENVIRONMENTS

Multiple devices working together on a problem can solve the problem much faster in comparison to a single device. Many games operate over a distributed environment, most notably the area of Massively Multiplayer Online Games (MMOs). The most famous of MMOs is World of Warcraft [34].

This giant of the video games industry has a many servers to many clients set-up. The core idea is that with many servers, the massive number of clients connected can be looked after.

With the increasing number of devices on a network, it can be argued that there is much processing power going to waste. Video game data can be distributed to these under-utilised nodes and therefore not require the need for additional hardware.

The research discussed below addresses client-server distribution, distributed virtual environments, adding additional hardware in the form of micro clouds and AI Partitioning.

2.4.1 CLIENT SERVER DISTRIBUTION

From the research presented in [35], [36] it is evident that eventually, a client server system will slow down when supplying a game service. When facilitating a massively multiplayer online game (MMOG), "the single server becomes a bottleneck due to insufficient network bandwidth" [35]. The major problem with client-server systems is scalability.

A possible solution to the single server problem is to incorporate a distributed client-server architecture [36]. A group of clients is connected to a server with each server being connected to run-time infrastructure services. Figure 2.2 illustrates this approach.

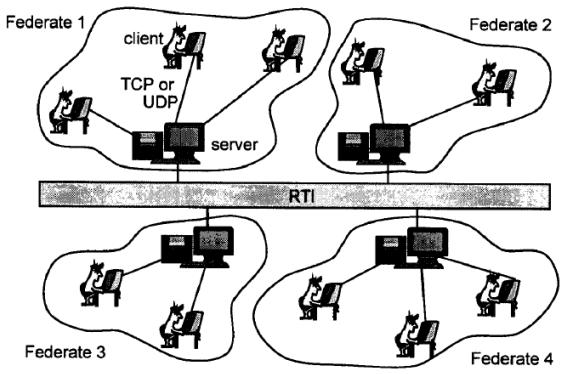


Figure 2.2: Distributed Client Server Architecture [36]

To distribute the work amongst the server's, two approaches have been suggested: virtual world subdivision and participant subdivision (Table 2.1).

Table 2.1: Two approaches to dividing work amongst servers.

Type of Division	**Explanation**
Virtual World Subdivision	With each group assigned to a server, the client connects to the server which their avatar belongs to.
Participant Subdivision	Clients connect to the server based on their geographical location.

With the increased number of servers, this solution allows for more players to enter a game. Problems with this potential solution are highlighted in [35] with the main problem being server overload. Game players tend to group in towns and cities; some zone servers could suffer from heavy load. Load balancing will be needed in a situation such as this, and a server would be dedicated to devising a load balancing strategy. Now the problem lies with the load balancing server as it would become a bottleneck.

The answer to creating a more responsive environment is not as simple as adding more servers. One approach presents us with a hybrid peer to peer MMOG cloud architecture [35]. Figure 2.3 presents this hybrid peer to peer MMOG cloud and its setup with regards to a section devoted to game servers, another for game regions data storage and a third for the character database.

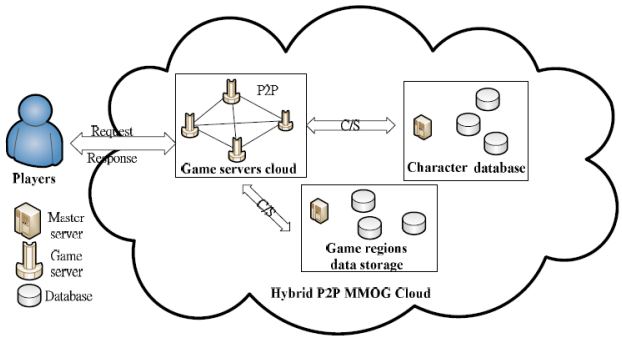


Figure 2.3: Hybrid P2P MMOG Cloud Architecture [35]

With the architecture on the cloud, there is more opportunity for resource allocation. The game servers have a P2P connection with each other allowing them to communicate load information and player information. As seen in Figure 2.3 there is a client server connection between the game servers and character database and the game regions storage. Having this data stored elsewhere allows for the game servers to focus more on the gameplay. As with all game servers, there is the potential to become overloaded, and the proposed architecture has a server load management procedure to prevent this. The top priority is to serve the players with a low response time which can be achieved

by choosing the server closest to them, if this becomes overloaded a neighbouring server is chosen that serves the same game region and has a low load. If this server is also overloaded a neighbouring server which serves another region and has a low load is chosen, if this too is overloaded then a distant server which can take on more players is chosen.

When compared to a multi-server architecture this approach performed better as it allowed more players to be catered for. The average response time for these players was lower, and when under load there is a smaller deadline miss ratio.

2.4.2 DISTRIBUTED VIRTUAL ENVIRONMENTS

A distributed virtual environment is defined in [37] as a software system that can "connect geographically dispersed users into a shared virtual space and support the interaction between the users and the shared world."

The research in [38] proposes a model of event communication within distributed virtual environments (DVE). The model adapts based on the network resource requirements of the events such as bandwidth and latency. Quake II was chosen for a case study, measuring bandwidth, latency and packet loss requirements of a multiplayer game. Individual event streams, which are different interactions within the virtual world, were measured against these metrics. The overall results of these can be seen in the Table 2.2.

Table 2.2: The results of Bandwidth, Latency and Packet Loss research when playing Quake II

Measurement	**Result**
Bandwidth	The majority of the bandwidth for an individual stream was taken up by the header, statistics and entities.
Latency	As the latency increases, the playability of the game decreases. This being in respect to both the overall and individual event streams.
Packet Loss	Similar to latency, with an increase in packet loss there is a decrease in game playability.

The results of the work in this paper show that when adapting to the variation in wireless networks, individual event streams having different resource requirements can be taken into consideration.

2.4.3 MOBILE GAMES WITH MICRO CLOUDS

The use of Gamelets in cloud gaming to reduce its drawbacks of latency, server scalability and lack of client side game data for latency hiding and synchronisation techniques is proposed in [39]. A Gamelet

is an extra piece of hardware placed within the network local to the client and is designed to take over the task of rendering from the cloud servers. A major advantage of this system is that video is streamed only within the local network allowing each user to receive larger bandwidth and not experience greater costs.

The challenges for such a system, along with a proposed solution are explained in Table 2.3.

Table 2.3: Challenges with implementing Gamelets

Challenge	**Solution**
Zone Distribution	With the size of modern games, this is addressed by dividing the world and resources into zones. A Gamelet can share the rendered data of its zone with another Gamelet to improve efficiency.
Distributed Rendering	This helps slow the inevitable obsoletion of hardware as game graphics improve quickly over time.
Security	Due to the addition of new hardware, there is a loss of centralised control. This can be overcome by handling problems only up to Gamelet level, and not client level.
Content-Based Adaptive Streaming	To reduce bandwidth consumption, the properties of game content are exploited. An example of this is “static game regions are streamed at a lower frame rate.”

The results of the experiment carried out in this paper, in which a test game was created on the system, presented two limitations on distributed rendering:

1. With the constant adding and removing of game zones there is a large zone handling overhead. By altering zone size, this can be reduced.
2. A predictive method is used to download game zones. A player moving close to zone boundaries triggers the download of the zones and they could potentially fill up the memory of the Gamelet quick and possibly for nothing.

This existing work has shown that with the addition of more hardware to an architecture, the user can experience a higher QoS. With this finding it can be argued that by having access to existing nodes that are local to the client, instead of adding more hardware locally, the user will experience a higher QoS without the need of additional hardware.

2.4.4 AI PARTITIONING

The process of AI partitioning in which the AI is divided into a high-frequency (high transmission rate) component and low-frequency (low transmission rate) component is detailed in [40]. The high-frequency component, being computationally simple, is placed on the server with the low-frequency component being placed on the client as it is more intensive.

One issue, as seen in many MMOG's, is that the AI used is simple, and this is because the servers being used have insufficient power to handle a more complex one. The solution employed in [40] is that the extra processing power of the client machines can be utilised. Therefore, the AI can be partitioned, and some processing offloaded to the client machine.

Experimental results show that even in a high latency environment this technique of partitioning AI is effective as it can tolerate a high round trip time.

2.5 DISTRIBUTED MANAGEMENT

The distributed environments seen in the section above can be managed via load balancing. Load balancing is the process of distributing the load over separate systems thereby increasing the processing power available [41]. In [42] load balancing is examined within Grid Computing, with a discussion on six different load balancing algorithms, these are as follows:

1. Fuzzy Based Approach: A rule base is created, and then these rules take the form of IF-THEN statements.
2. Genetic Algorithm Based Approach: Solutions from a population are taken, and a new population is created. Over several generations of solutions, an optimal solution is formed.
3. Agent-Based Approach: With this approach, computer programs known as agents act on behalf of the user. These Agents can communicate with each other to determine where a task is to be executed.
4. Hybrid Approach: This approach focuses on nodes swapping between a static state, in which there is no need for continuous monitoring, and dynamic state which requires continuous monitoring.
5. Policy-Based Approach: The computation time of a job is calculated on some nodes; the average time is then taken and is updated in an iterative scheduling approach.
6. History-Based Approach: A job execution history is used to estimate a job's start time.

The approach we will focus on here is the Agent-Based Approach. For gaming, Agents can be utilised to distribute game services and therefore improve a user's gaming experience

2.5.1 AGENT PROPERTIES

A software agent is a computer program that acts on behalf of the user. Agents are seen to have two important qualities, first is to be able to satisfy the objective assigned to them by deciding which actions need to be taken and second is to be able to communicate with other agents [43].

The properties of mobile agents can be found within [42] and [43]. The majority of these are similar within both papers, with 3 in total being different.

The main properties which all agents will have are being reactive meaning it responds to changes in the environment and being autonomous meaning it controls its own actions. Agents are also goal-oriented/pro-active meaning that the agent doesn't simply react but acts towards a goal and finally, to be temporally continuous, meaning the agent is always running to perform what it has been assigned to do.

Other properties which some agents have are that they are communicative/social meaning that the agent will communicate with other agents and people. The agent is learning to allow it to adapt its behaviour to fit its environment and it can be mobile allowing it to travel to different nodes. The agent can be flexible, meaning that its actions are not pre-determined, truthful, meaning it will not communicate false information and finally rational, meaning it should never prevent its goals from being achieved.

With agents being autonomous, they have the capability to decide for themselves how best to go about achieving their delegated goal. The agent can be thought of as being in a close-coupled continual interaction with its environment leading it into the Perceive Think Act Cycle [46] also known as the Sense Decide Act Loop [43], this is presented in Figure 2.4 below.

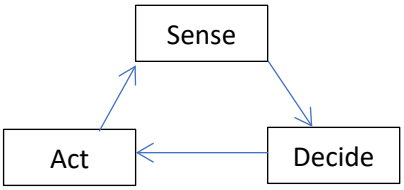


Figure 2.4: Sense, Decide, Act Cycle

The agent continually senses the environment, on the basis of this information they decide which action to perform next in pursuit of the delegated goal, and they then perform the action which will change the environment. Once having acted, the agent will sense the environment again.

2.5.2 THE NEED FOR AGENTS

A distributed system allows for users to offload processes to other more powerful nodes on the network. These systems are flexible and scalable. However with this area there comes challenges, including the following [44], [47]:

1. Addition and removal of nodes.
2. The distributed system should be adaptable when environment conditions change.
3. A variety of hardware and software will be utilised.
4. A guaranteed minimum QoS.

Limitations are also present in client/server approaches. The authors of [48] present five major issues: Centralised Management, Scalability, Bandwidth Wastage, Response Time and Fault Tolerance.

The challenges and limitations identified above have generated more research in the area of software agents.

2.5.3 BENEFITS

There are several benefits to employing agents in a system. These include: network load reduction, network latency overcome and dynamic adaptation [37] [39].

With these benefits, it is seen that the best way to manage a distributed system is with agents. These benefits have also been recorded in our paper [3], beside each of them is the benefit in terms of a networked game.

Table 2.4: Agent Benefits as found in [3]

Benefit	In relation to a Networked Game
Network load reduction	Instead of all the calculations for AI being transferred to another device or node, the agent can travel to the device and then start transmitting results.
Network latency overcome	The agent will be stored on a node and, when required, can travel to the device to carry out the task.
Dynamic adaptation	Users join and disconnect from games constantly; this ever-changing topology requires software that will react to the changing environment.

2.5.4 MOBILE CODE PARADIGMS

The research in [50] describes three mobile code paradigms, that extend from the client-server paradigm, for designing a distributed application, namely: Remote Evaluation, Code on Demand and Mobile Agent. Table 2.5 explains these in terms of the location of components before and after the execution of a service.

Table 2.5: Mobile code paradigms

Paradigm	**Before**		**After**	
	Site A	**Site B**	**Site A**	**Site B**
Client-Server	A	know-how resources B	A	know-how resources B
Remote Evaluation	know-how A	resources B	A	know-how resources B
Code on Demand	resources A	know-how B	resources know-how A	B
Mobile Agent	know-how A	resources		know-how resources A

As seen from the row with the mobile agent paradigm, before execution it will have the know-how but not the resources, the agent will then travel to the site (Site B) that has the resources and execute the service there. Unlike the other paradigms, the agent is the only entity involved.

2.5.5 AGENT COMMUNICATION AND EXPLORATION

The authors of [51] provide three different models of agent communication. These models are detailed in Table 2.6.

Table 2.6: Communication methods of agents

Communication Method	**Explanation**
Face to Face	When two agents are on the same node, information can be exchanged between both.
Pebble Model	When an agent visits a node, they can leave a pebble (data packet) which is visible to all other agents and can be picked up by any other agent visiting the node.
Whiteboard Model	Like the pebble model, the agent can leave information on a public whiteboard of the current node. When another agent visits this node the information is visible to it and can be modified by it.

Another topic addressed in [51] is that of agent exploration. When the agents are operating on a network in which all nodes are labelled, exploration can be executed via depth-first search (DFS) or breadth-first search (BFS). Figure 2.5 below presents a graphical representation of these search methods.

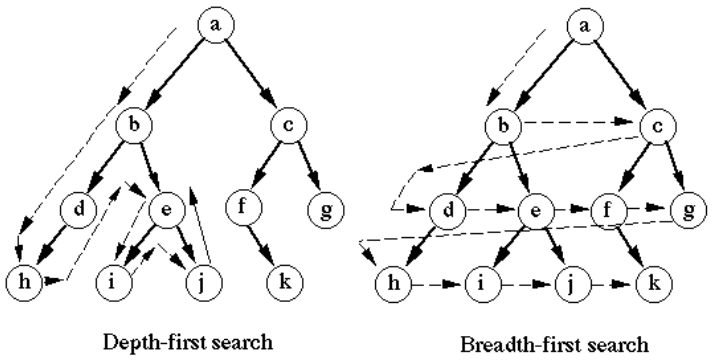


Figure 2.5: Graphical Representation of Depth-first and Breadth-first searches [51]

An alternative method of exploration on a labelled network is a piecemeal exploration in which the agent returns home periodically during exploration.

2.5.6 POLLING METHODS

As software agents are separate computer programs, they can operate on their own without the interaction of the node that sent them, hence allowing them to operate offline as well as online. This ability provides two approaches to data acquisition known as Polling Modes. The work in [52] proposes two polling modes allowing mobile agents to retrieve real time and offline data. These modes are known as: Get 'n' Go and Go 'n' Stay. Table 2.7 outlines these methods.

Table 2.7: Polling Modes

Mode	Explanation
Get 'n' Go	This mode is used for the collection of real-time data. The network is divided into several areas with a mobile agent assigned to each. This agent visits all nodes in its area and then returns to its originating node.
Go 'n' Stay	This mode is used for the collection of data that can be analysed offline. An agent is sent to each node on the network; the agent can stay for an allotted amount of time and then return.

The above polling modes help back up the adaptability of agents as the node they are currently on is not required to be constantly available on the network, therefore, reducing the power consumption of the node.

This is relevant as agents will be required to travel. This travel is realised in Chapters 4 and 5 which detail the experimental method and system setup aswell as the results of the experiments.

2.5.7 AGENTS FOR OVERLAY NETWORKS

An overlay network is a computer network which is built on top of one or more existing networks. Overlay networks are created by the hosts and servers and can "enhance end-to-end application performance and availability" [53]. These overlay networks monitor themselves, however, as this was not their intended purpose it is a burden. This waste of resources can be prevented with the introduction of multi-agent technology, allowing for the network to focus on other things. The research proposes introducing a main test agent and an assistant test agent. Each agent has a variety of components as experiments were carried out with results showing that the agent framework can provide a service to measure parameters of the overlay network.

From this research, it can be argued that the addition of agents to a network can improve its efficiency as resources are not spent on monitoring the network.

2.5.8 SYSTEM AGENT RESEARCH

In [54] an architecture is proposed that employs mobile agents for resource transactions. The proposed has three parts: game server, encapsulated game group (EGG) and game group manager. There is a peer to peer connection within the EGG and a client server connection between the server and the managers of the EGG's, with a manager being chosen when the EGG is being set up. The agents within the system set up in game transactions between players by travelling within EGG's and further

afield. This is relevant to the work in our project as agents will be employed to travel distances in-house and further afield.

Another model of note is that found in [55]. In this research, mobile agents are used to report on changes of routing in grid systems. Along with execution agents who carry out the tasks of the user, there are routing agents which are categorised into short distance agents and long-distance agents. Short distance agents can only travel to local nodes while long distance agents can travel much further. This research is similar to that in [54] as agents travel short and long distances however the work here categorises these into short and long distance agents while [54] uses the same agents to travel short and long distances.

The frameworks designed here have a significant influence on this project as a variety of agents can be used to access the required resources for a game. Short-distance agents can operate in-house, medium-distance agents can operate within the area of the Fog and the long-distance agents can operate outside the Fog to the Cloud. These agents will travel to their destination and open a channel for resources to be streamed along [3].

2.5.9 METRICS

To measure the effectiveness of agents in a distributed system, we require metrics. Metrics are standards of measurement that, with regards to this project, can be used to measure the QoS a player is receiving. Metrics can be found throughout the literature helping to compare old systems with new within two categories: QoS and QoE. QoS metrics are focused on the network such as the current latency whereas QoE metrics which are usually centred on the user such as their current mood.

TYPE OF GAME BEING PLAYED IS KEY

When defining the minimum network conditions required to play a game, it can be seen that these vary based on the game genre. Both [54] and [55] highlight this and agree that a first person shooter will require a much lower latency than that of a real-time strategy game. Background work in [56] found a study that concluded with the user's QoE being significantly impacted by the type of game they were playing [58]. Experimental results have shown that cloud-based first-person shooter players are less accepting of poor network conditions than those of other game genres. An example of this is found in [59] as a latency of more than 100ms can affect a gamers experience in the first person shooter Call of Duty. Simulation games such as The Sims are less effected by latency due to their less time-sensitive nature.

DIRECT AND INDIRECT METRICS

QoS measurements focus on the network level metrics such as packet loss, jitter and one-way delay [60]. With the volume of multimedia content, emerging and being consumed, increasing rapidly, these measures are seen to no longer be sufficient with ways to assess user satisfaction via QoE methods being researched. QoE measurements are very user-centric involving measures not related to the network such as the current mood of the user. An example of a QoE metric is the mean opinion score (MOS) [13] in which having played a game, the player's rate certain aspects of it such as responsiveness.

When it comes to assessment, two types of metrics have been defined:

- Direct Metrics
    - These are metrics that directly affect user perception. They can be obtained from data such as variations in delay and packet loss.
- Indirect Metrics
    - These metrics are not directly related to the quality of the delivered content but still affect the experience.

Examples of Direct metrics include:

- Peak Signal to Noise Ratio (PSNR): Compares each pixel of the original images to those of the received.
- Structural Similarity (SSIM): Compares structure contrast and luminance of the original and received images.
- Video Quality Metric: Compares colour and blurring of the original and received images.

Examples of Indirect metrics include:

- Start-up time: Time from the user querying the system until they receive the requested content.
- Response Time: Time for a specific action to occur on screen.
- Delivery Synchronisation: Time difference in the delivery of content to different users.
- Freshness: Time between content being created and received by the user.
- Blocking: Describes the irregularity of the video.

With games being a form of multimedia, these metrics may be utilised when measuring the QoS and QoE of a framework.

RESPONSIVENESS AS THE MOST CRITICAL OF MEASURES

The authors of [57] claim that the responsiveness of a system is the most critical metric in respect to QoS. The responsiveness in this paper, defined as Response Delay, has the same definition as the response time of [60].

Response Delay has been divided into four cloud based game components [57]:

1. Network Delay: The round-trip time of the network or more specifically the time to transmit a command to receiving the game screen.
2. Processing Delay: The time from receiving the command to sending the result.
3. Game Delay: The time the software of the game takes to process a command and render the result.
4. Playout delay: The time it takes for a client to receive its results from the server and display them.

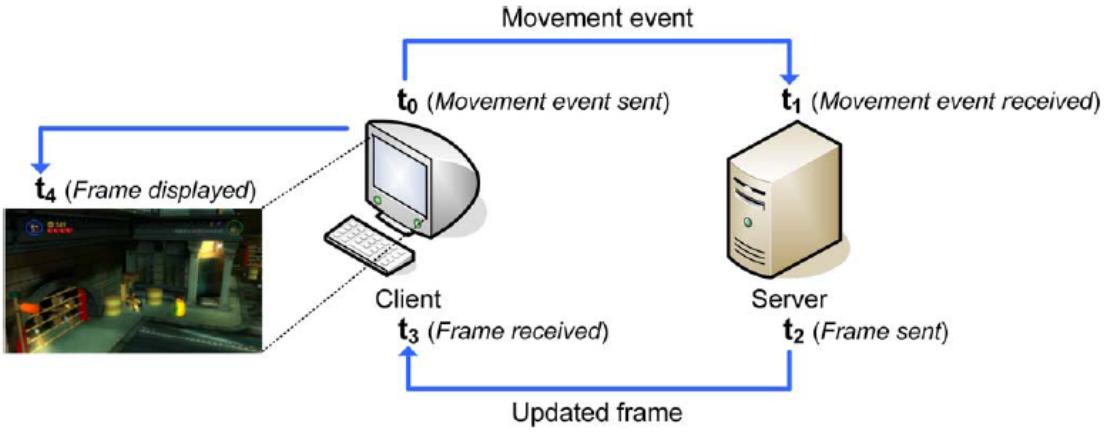


Figure 2.6: Time frames in measuring response delay [57]

From Figure 2.6 above it can be seen that each of the components can be assigned a time frame equation. The "t" at each point represents an amount of time.

- Network delay is equal to (t_1-t_0) and (t_3-t_2)
- Processing Delay and Game Delay are equal to (t_2-t_1)
- Playout Delay is equal to (t_4-t_3)

The total Response Delay can be calculated with $t_4 – t_0$. From [57] it can be seen that Response Delay can be considered a crucial metric as it directly affects a user's experience and performance.

The authors of [61] also refer to Response Delay. The Response Delay here has three components with one component split further:

1. Tclient: This is a combination of the first part of the network delay and playout delay seen above. It is the time it takes to send information, e.g. player movement, to receiving and playing the video.
2. Tnetwork: The delays described below can be placed in between t_0 and t_1 in the diagram above.
    a. Taccess: The time for data to move from the client to the first internet connected router.
    b. Tisp: The time for data to move from the router to the "peering point connecting the ISP to the next hop transit network."
    c. Ttransit: The time for the data to move from the "peering point" to the front-end server at the datacentre.
    d. Tdatacenter: The time for the data to move from the front-end server to the hosting server
3. Tserver: This is similar to the processing delay and game delay. This is the time spent processing the received information, generating video and transmitting this back to the client.

The research here has broken down response delay into smaller delays through which calculations can be carried out to determine in which part of the network the most delay is occurring. From this, we can determine responsiveness to be the most critical of measures.

CONTINUOUS ANALYTICS

The work in [62] presents CAMEO: Continuous Analytics for Massively multiplayEr Online games on the cloud. With MMOs there is an abundance of communities that inform and entertain players. Analytics are used by these communities to produce reports such as the best overall character class for damage [63]. The authors believe that the analysis process can benefit from the cloud. CAMEO can mine information and present results based on the mined information.

The system was tested by analysing the game RuneScape. At the time of publication, this was the largest data collection and analysis of RuneScape to date. The behaviour of almost three million characters was analysed along with the progress of half a million characters being followed closely for a week. With CAMEO being connected to cloud services, Amazon EC2 was compared against a local cloud provider with the Amazon service able to carry out the task of collecting player identities much

quicker. Regardless of cloud service provider, the results of testing with CAMEO prove that continuous data analysis of MMOGs can be carried out with cloud resources.

In this project, the concept of continuous analysis can be utilised to assess the network and have it adapt, then assess again and so on.

EVALUATING AN EXPERIENCE

Not only does the type of game being played have an impact on a player's experience, but there are other aspects that influence the overall experience of a game playing session. As discussed earlier, metrics can be divided into those which affect QoS and those which affect QoE. The research in [64] focuses on the QoE providing detail on three categories: Influencing Factors, Interaction Performance and Quality Features. The following tables detail the subcategories of these.

Table 2.8 shows the subcategories of Influencing factors: User factors with four measures, System factors with five measures and Context Factors with four measures.

Table 2.9 shows the subcategories of the Interaction Performance factors: System Performance with four measures and User Performance with three measures.

Table 2.10 details the subcategories of Quality Features. All of these, except for Player Experience, have no further division. Player Experience has three measures.

Table 2.8: The subdivisions of Influencing Factors [64]

User Factors	**System Factors**	**Context Factors**
Experience: Based on the gaming experience of the player.	*Game Genre*: Game genres include shooter, real-time strategy, etc.	*Physical Environment Factors*: Where the game is being played.
Playing Style: Players have many styles, these including achievement hunters or a socializer.	*Game Structure*: Is the game a single player, co-op against the game, player v player, etc.	*Social Context*: Playing with other players and relationships with them.
Intrinsic Motivation: Behaving in a way which is personally rewarding.	*Game Mechanics and Rules*: Individual to each game.	*Extrinsic Motivation*: Behaving in a way to achieve an award.
Static and Dynamic User factors: Static factors include age and gender while dynamic includes the current emotional status and how distracted the player is.	*Technical System Set-up*: Characteristics of the server and network requirements.	*Service factors*: Running costs etc.
	Design Characteristics: The design of the system and the design of the game. Individual to each system and game.	

Table 2.9: The subdivisions of Interaction Performance [64]

System Performance	**User Performance**
User Interface Performance: Input and output performance of the interface.	*Perceptual Effort*: Understanding the system and its outputs.
Backend Platform Performance: Input and output performance of the backend of the platform	*Cognitive Workload*: The cost of achieving a task, the task here being obtaining an outcome.
Game Performance: The success of the game.	*Physical Response Effort*: The physical effort required to play the game.
Communication Channel Performance: Performance of carrying input and output from and to the user.	

Table 2.10: The subdivisions of Quality Features [64]

Interaction Quality	The quality of the input and output and the behaviour of the player interacting with the game.
Playing Quality	This is the level at which a player can learn, control and understand the game. [65]
Aesthetic Aspects	The user's perception of the look and feel of the system.
Player Experience	*Flow, Challenge, Control*: How the game flows, the challenges it presents and the control the user has over these.
	Immersion: How much the user feels to be in the game and not simply playing a game.
	Positive and Negative Affect: How the game affects the player.
Acceptability	How acceptable the user is towards the system.

From Tables 2.8, 2.9 and 2.10, it can be seen that measuring the QoE can be a long and difficult task. In our approach, a strong focus is placed on QoS metrics as it may be argued that when a good QoS is achieved then the QoE depends only on the player and their condition when playing, such as their mood.

USING LOD TO IMPROVE QOS

In computer graphics, level of detail (LoD) is the rendering of more or fewer polygons of an object depending on its distance from the viewport of a camera. The further away the object is, the less detail is required. Figure 2.7 below shows that even with significantly fewer triangles used to render an object it still keeps its shape. The fewer triangles used the further away the object will be.

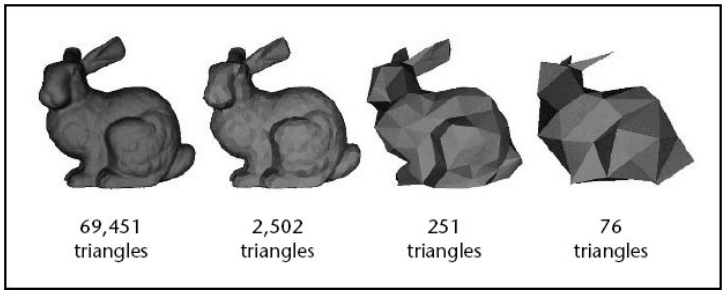


Figure 2.7: The LoD of a rabbit as found in [66].

An adaptation framework inspired by the LoD mechanism to improve a player's experience is proposed in [66]. With this approach, the updates of some entities are prioritised over others. Two types of messages are used for the message passing protocol: Asynchronous and Synchronous Messages. Three types of components are used in the model: Entity, Group and Mode. An entity represents a game object, with each belonging to a group, each group has a role and each group has a communication rate assigned to them. A drastic change in network settings results in the reassignment of groups to an appropriate mode based on the new communication capabilities.

The experiment proved that the difference between the game experience with the adaptation framework and without is significant. With the LoD inspired framework the most important entities were given a high priority and so were updated first, without this the messages for updates were sent at pre-determined intervals and therefore gameplay suffered.

This level of adaptability can be utilised within the proposed framework. If there are no resources available for the client device to pass off processing to, then the device itself could lower the LoD to help improve performance.

2.6 FOG COMPUTING

Fog computing is very similar to cloud computing with one of the main differences being that distribution occurs closer to the user, for example from another client or local edge device. In [67] we see the benefits of Fog computing for a business focusing on the Internet of Things. Of these, the benefit that stands out is the "Lower operating expense" benefit. As data is processed locally instead of on the cloud, network bandwidth is conserved. Having data processed closer to where it is required

"solves the challenges of exploding data volume, variety and velocity." Figure 2.8 is adapted from [68] and shows the processing of distributed data in an environment which employs Fog computing.

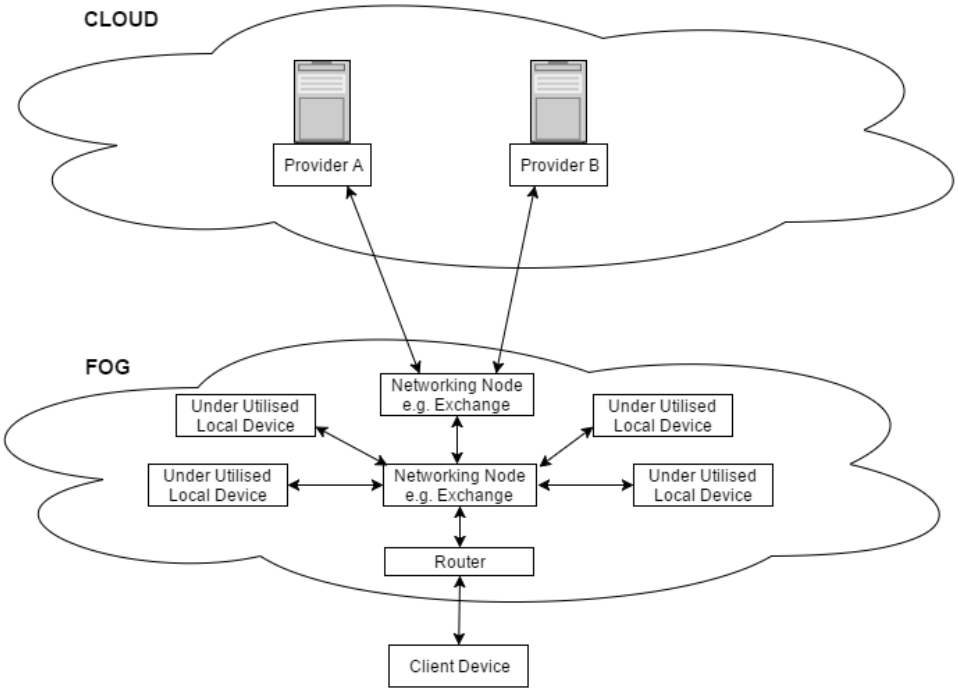


Figure 2.8. Distributed Data Processing in an Environment Utilising Fog Computing (adapted from [68])

In November 2015 ARM, Cisco, Dell, Intel, Microsoft and Princeton University Edge Computing Laboratory founded the OpenFog Consortium. The purpose of this group is to assist with the implementation of fog computing in "advanced concepts in the digitised world" to alleviate issues from latency and bandwidth to the challenge of communication [69]. Two different architectures currently utilise Fog Computing in a game context, EdgeCloud and CloudFog.

2.6.1 EDGECLOUD

The authors of [70] propose the augmentation of existing cloud infrastructure with an EdgeCloud to improve on-demand gaming. An EdgeCloud can be argued to be Fog computing as network nodes, referred to as participating peers are local to the user and house specialised hardware. An example of a participating peer is a games console. With the millions of games consoles sold annually the argument is made that the service provided can be improved by including these within the architecture. Figure 2.9 details the architecture of the EdgeCloud which consists of a data centre, peers and a co-ordinator for assigning peers to clients.

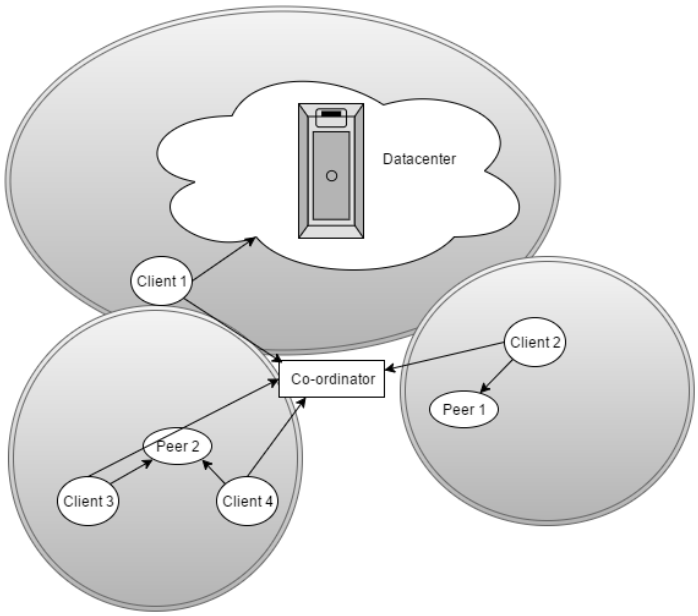


Figure 2.9. The EdgeCloud Architecture (adapted from [70])

The EdgeCloud focuses on game streaming through remote access. A co-ordinator is used in this architecture to decide which clients are to be served by which peers. The peer that the client is connected to is given an application which it serves to the clients in the form of a video stream.

This architecture focuses on how to meet the strict latency requirements of cloud gaming. The game itself is brought much closer to the client via their games console or another suitable node. Each suitable node (participating peer) will have specialized resources necessary for gaming. The client connects to the node, and a video stream of the game is transmitted to them.

The results of this architecture show it to be an improvement over the currently-in-use cloud gaming architecture. Evaluating the EdgeCloud in [70], with the utilisation of peers, the EdgeCloud was able to serve roughly 90% of the connected population with an 80-millisecond latency in comparison to Amazon's EC2 which was only able to serve 70% with the same latency.

The problem with this architecture is within the participating peers itself. Each of these peers hosts all of a game's data. With the wide variety of video games available this can become a problem as each peer can only hold a certain number of games. As well as this, by adding more games to the peer pool, there is potential to increase bandwidth usage as large files are transmitted across a network.

2.6.2 CLOUDFOG

Yuhua Lin and Haiying Shen authored two papers in 2015 both of which introduce a system known as CloudFog [71], [72]. This system proposes the use of supernodes which are close to the end users. These supernodes hold the responsibility of rendering game videos and streaming them to the clients.

The cloud still has a very important role in this system which is to handle the intensive computation of the virtual world's new game state. From the cloud, updates are transmitted to the supernodes which then renders and streams video to the connected clients. This system also accounts for client nodes that may not be able to form connections with supernodes. In this case, the clients connect directly to the cloud. Figure 2.10 below details the CloudFog Architecture.

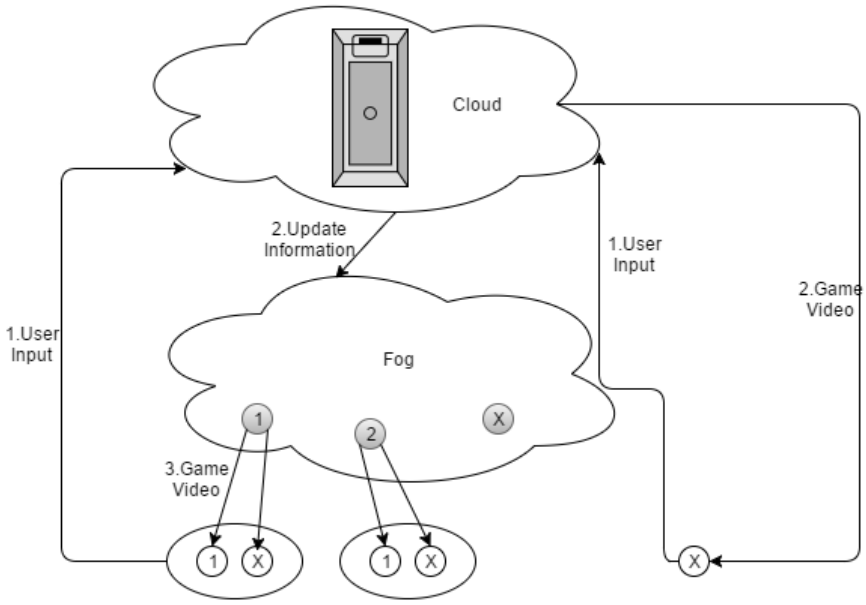


Figure 2.10. The CloudFog Architecture as adapted from [72]

CloudFog focuses on video streaming to the clients. Each client will connect to its local supernode and through this connection receive a game video stream. As seen in Figure 10, each client will provide input which is transmitted to the cloud, the cloud then updates each supernode on changes, and then each supernode sends game video to the clients connected to it.

The gap filled by this research is similar to that of EdgeCloud in that it focuses on the reduction of latency in cloud gaming. With the introduction of extra servers closer to the end users, which have up-to-date information on the state of the game world, there is less distance for the data to cover to be received by the client and therefore a lower latency.

The results of this architecture have shown an increase in user coverage due to the added servers, a reduction in response latency as the data is much closer to the clients and a reduction in bandwidth consumption as supernodes are updated with the game state and video is streamed to the clients.

The problem with this architecture lies within the supernodes. New hardware is added to a network which will already have underutilised nodes. Instead of adding new hardware, a similar approach can be taken to EdgeCloud by utilising specialised hardware (games consoles) already in existence.

2.6.3 EDGECLOUD VS CLOUDFOG

With both of these architectures it can be seen that they are very similar:

- Both focus on the reduction of latency in cloud gaming.
- Both utilise nodes close to clients.
- Both stream video from Fog nodes to clients.

However, in some areas they are different:

- EdgeCloud stores an application on its Fog node while CloudFog sends updates of game state to its Fog node.
- EdgeCloud utilises existing hardware such as game consoles to become Fog nodes while Cloud Fog introduces new hardware to the network.

With these similarities and differences, it can be argued that a combination of these two architectures would lead to a more beneficial system. With EdgeCloud the downloading of applications to participating peers can be potentially both time and bandwidth consuming. With CloudFog, the addition of new hardware to a network in which there is potentially a wide variety of under-utilised network nodes seems to be a waste. A combination of these two architectures would result in a system in which smaller data packets, i.e. only information updates instead of large application files, would be sent to the fog nodes. This combination would also employ underutilised nodes as its fog nodes in comparison to purchasing, setting up and maintaining new server nodes.

The architecture proposed in this thesis is very similar to the combination of these architectures. By updating nodes close to the clients, information can be transmitted to the clients in a much shorter time in comparison to the information coming directly from the cloud. The nodes close to the clients will take the form of under-utilised network nodes, for example a games console that is on but not being used or even the router supplying the Wi-Fi connection.

2.7 SAVING ENERGY

The aim of this work is not to save energy but to improve a client's QoE through the utilisation of under-utilised network nodes. However, it may be argued that client energy is saved using this architecture. A diagram from [73] details that with an increase in CPU Utilisation there is also an increase in power consumption. Figure 2.11 below shows the correlation.

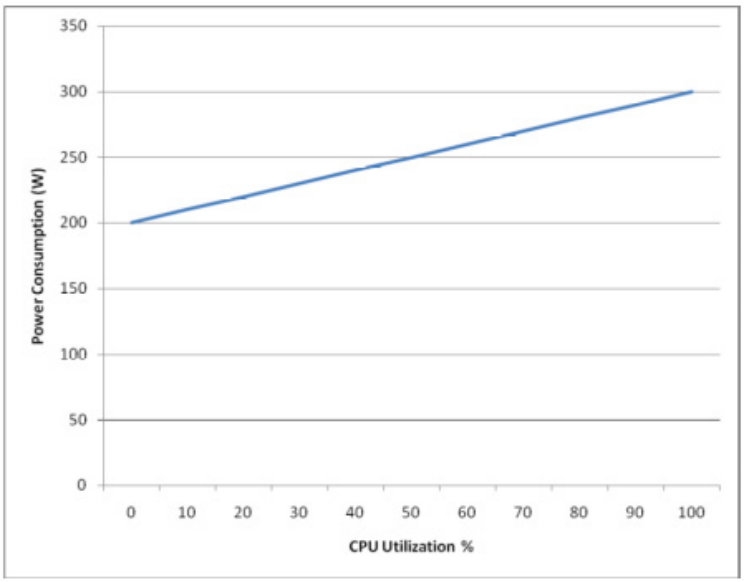


Figure 2.11. CPU Utilisation to Power Consumption as found in [73]

The area of green computing is still relatively new. The focus of the research found has been to utilise distributed resources to save energy. In [74] the role of computer virtualization to save energy is discussed. By replacing a standalone server with a virtual server which runs on a variety of computers, PC resources are used more efficiently. This paper discusses the challenges of virtualization as well as providing an example in NComputing [75]. NComputing is a desktop virtualisation solution provider in business from 2003. As PC's have become more powerful over the years; applications begin to use less of the processing power of the machine. NComputing introduces additional hardware in the form of a small box through which each client can connect to a single machine. Many clients can be connected to one machine. Patil [74] also goes on to discuss other methods of saving energy in this paper including the user of a computer's sleep and hibernate modes.

Sarkar and Misra discuss the theoretical modelling of fog computing in [76]. An investigation compares energy consumption and service latency. In Cloud and Fog approaches it is found that as requests from the client to the cloud increase there is a close to linear increase in processing energy required. It is also claimed that, with Internet of Things applications, if around 25% of client requests require real-time services then an improvement in energy consumption of 40.48% is observed in the fog computing architecture.

Other articles of note which support the case for distributing load to save power are [77]–[79]. In [77] Cao, Zhu and Wu design and develop an "energy-aware scientific workflow scheduling algorithm to minimize energy consumption." One step of the algorithm is to choose a best fit data center based on a variety of factors such as energy cost and time to complete workflow. Experimentation results had shown a reduced energy consumption, energy cost and CO2 emissions. In [78] a three-part

architecture that manages the sharing of resources in Mobile Cloud Computing. The simulation results of the proposed architecture have shown that it assists in reducing “handover delay, packet loss, average queuing delay and device lifetime. Energy is saved through the selection of a virtual machine based on which machine allows for the minimum amount of energy required for communication. In [79] Hassan et al address the issue of energy expenditure, which is a key issue faced by cloud providers, by proposing a “capacity-sharing mechanism in a federated cloud environment.” The overall goal of this mechanism was to maximize the social welfare of the cloud providers and reduce the energy cost. The model used in testing had shown that different cloud providers were motivated to cooperate within a federation based on their evaluation of profit and energy cost.

2.7.1 GREEN GAMING

The authors of [80] discuss cloud gaming from the perspective of green computing. The green features of gaming on the cloud are presented and these include easier software maintenance, no client compatibility issues and higher utilisation of hardware. A novel architecture is presented which improves upon mobile cloud gaming. The authors build upon the knowledge that as more sophisticated graphics rendering becomes available on mobile devices, this can be utilised to “reduce the transmission bandwidth of game images.” Figure 2.12 presents the system.

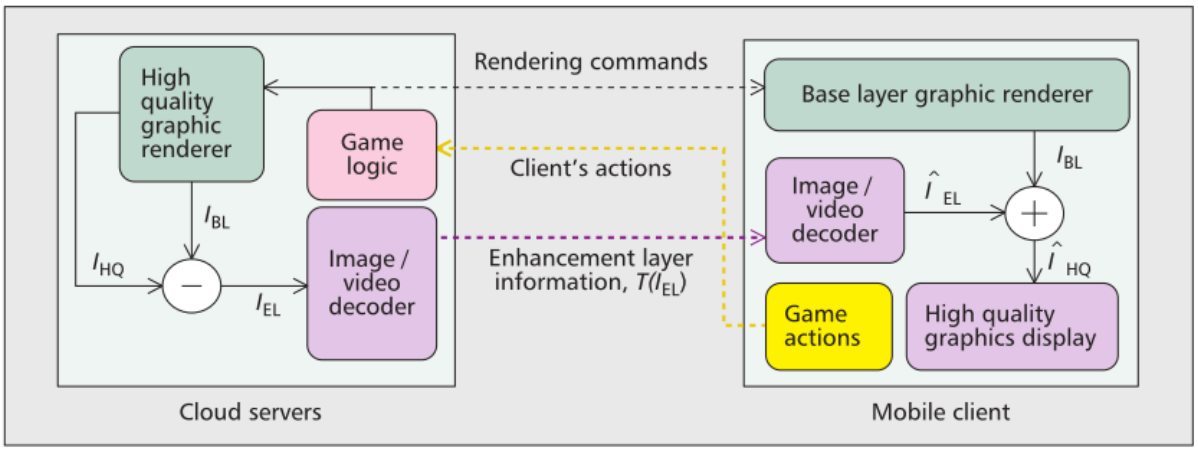


Figure 2.12. A mobile cloud gaming system that uses layered coding ([80]).

This process of sending rendering commands to the client is very similar to previously discussed architectures in which graphics commands are transmitted from server to client. However as seen from Figure 2.12, enhancement layer information is also transmitted to the client. This is the layered coding of the system. The base layer render of the image, carried out on the client, combined with the enhancement layer information leads to a high-quality image displayed on the client device. The results of this layered coding are compared to the direct coding used in the cloud. The layered coding is seen to achieve a lower bit rate at a similar video quality in comparison to a direct coding approach.

2.8 SUMMARY

The focus of this thesis is on the utilisation of distributed resources to improve the QoE of network aware games. The background research areas considered in this thesis are: Streaming Games, Distributed Environments, Distributed Management, Fog Computing and Energy Saving.

A key trend with research focused on streaming game technologies is to consider mobile devices as a core client device even though these devices are much less capable than the latest games consoles. The game itself is not stored on the client but rather in the cloud. With its resources, the cloud can run the game, dealing with its processes and stream the game down to the mobile device. Improvements in mobile networks and the rise of 4G allows for faster streaming.

Research into distributed systems provided us with an insight into architectures other than client-server and peer-to-peer. The disadvantages of client server approaches have been identified, and it is evident that many systems incorporate multiple servers to overcome these disadvantages such as the potential for server overload which can occur if many players are within the same area in the virtual world. Other systems may not directly employ multiple servers, but use the concept of distributing resources over several nodes. Systems of note in this category are those which use micro clouds in mobile games with the micro clouds housing a section of the game world and that which uses the client's machine to assist with AI calculations to improve the overall AI in a game.

As distributed resources will be utilised to improve the overall QoE, some form of management of these resources is required. One possible solution to providing this control is via the use of agents. Agents can be used to inform the cloud that resources are required and to assist in bringing these resources to the user's device. These resources can reside anywhere, from in the cloud itself, the local exchange or even a neighbour's PC. Research has shown the many benefits of Agents and how they operate, and they appear to be a suitable solution in assisting with the distributed resources.

QoS and QoE metrics are the predominant measurement mechanisms to see if there has been an improvement between old and new systems. In the proposed project, more of a focus will be placed on the QoS metrics as it can be argued that if a good QoS is provided then a good QoE should follow.

Building on the process of utilising the cloud, another area of note is that of Fog Computing. The Fog is very similar to the cloud except for its location which is much closer to the end user. With a more local option to use for extra processing and storage, the major benefit is the lower transmission time from the Fog to the client. 'Lag' is at the forefront of an online gamer's mind and the utilisation of the Fog can help improve the QoS provided to them and therefore improve their QoE.

Another area of note in this research is that of energy saving. While this is not the overall aim of this thesis it is important to keep this in mind as 'Going Green' is becoming more popular and becoming more of a necessity.

Table 2.11 summarises the main research areas covered in this chapter.

Table 2.11: Summary of research areas for this thesis.

Research		**Relevant Models**
Streaming Games		Mobile Cloud Gaming ([11], [25]) Games@Large ([26], [13]–[16], [16], [28]) Remote Visualisation ([29], [30]) Asymmetric Graphics Rendering ([31], [32], [33])
Distributed Environments		Client Server Distribution ([35], [36]) Distributed Virtual Environments ([37], [38]) Mobile Games with Micro Clouds ([39]) AI Partitioning ([40])
Distributed Management	Agents	System Agent Research: Use of agents for resource transactions, categorisation of agents based on the distance they can travel ([3], [37], [43]-[55])
	Metrics	Direct and Indirect Metrics, Responsiveness, Continuous Analytics, Improvement of Quality via Level of Detail ([13], [52] -[66])
Fog Computing		EdgeCloud ([70]) CloudFog ([71], [72])
Energy Saving		Green Gaming ([73]-[76])

With the areas in Table 2.11 researched, we plan to maintain a game player's quality of experience, regardless of which device is being utilised, through the use of a system of distributed game asset streaming and code execution.

Streaming Games research proved the validity of an architecture which would be adaptable to the end user's device as the systems could choose between video streaming and graphics streaming. Research into Distributed Environments has shown that the distribution of processing benefits the end user and it can be argued that by being able to access nodes more local to the client device, instead of adding additional hardware to the network, that the benefits will be seen without the cost of added

hardware. An adaptable, distributed environment requires management and the most suitable way to manage this environment is through the use of agents. Agents are best suited to monitoring the environment and making decisions based on the wide variety of metrics available. The architecture proposed in this thesis can be seen as a combination of both Cloud and Fog Computing. By utilising nodes local to the end user (Fog), information can be transmitted to the client much faster while the nodes further away (Cloud) can be relieved of some processing. It is not a goal of this work to save energy, however, the research into Energy Saving has shown that, with the reduction in utilisation of resources, there is a reduction in energy usage.

CHAPTER 3 ARCHITECTURE

3.1 OVERVIEW

Within this chapter, an architecture is presented that can be implemented to improve a user's QoS in playing an online game through the utilisation of distributed resources. Firstly, the overall architecture is detailed and discussed. Then a range of possible scenarios for client support is explored i.e. server assistance, local client assistance or self-adaptation. Finally, the nature of databases and architecture intelligence is explained along with events and the role of RPCs and the network and its effect on the architecture.

3.2 INTRODUCTION

QoS attributes such as FPS, and the CPU and GPU usages can be used as indicators of how well a system is doing and therefore should be monitored within an intelligent system that might adapt to the situation. The most important QoS measure, in a single player game, is arguably the FPS count. It can be seen that this is the most significant factor in providing a high QoS as it affects the regularity of the game [81] and it is a relatively straightforward attribute to measure. The proposed architecture takes into consideration the state of the client device, the network over which game data is transmitted, the server which supplies game data and its current state and the game itself. By considering all the available data, the architecture will be able to improve a user's QoE by focusing on the QoS provided. The QoS can be enhanced by utilising resources that are potentially available on the server or another network node. A fail-safe is implemented in this architecture in the form of self-adaptation in which the client will adapt to its current situation if there are no resources available elsewhere.

Based on the relevant research areas such as Cloud and Fog Computing, and the QoS requirements of a client such as to maintain an FPS above a certain threshold i.e. the game must run above 30FPS at all times, the architecture comprises of adaptive software that distributes processing when the QoS provided falls. This architecture combines three different configurations that can be utilised to maintain a good QoS: Given good quality remote connections (Cloud Computing), the availability of local resources (Fog Computing), and when there are no network resources available (Self-Adaptation). Cloud Computing is represented in the architectural diagrams by the server which can carry out processing for the client and transmit results to them from a distance. Fog Computing is represented by the network nodes which are close to the client and can also carry out processing tasks for the client. Cloud and Fog Computing are very similar, the only difference being a matter of distance. Self-Adaptation is contained within the client itself; this is the fail-safe of the architecture and executes only when there are no resources available from the network for assistance. This approach executes only when the others have failed and will attempt to maintain a good QoS. Self-Adaptation will remove

unnecessary objects from the game-world such as ambient AI e.g. fish in a lake. Objects are removed based on their priority. Higher priority objects can be removed but this will only occur if there is no other option. This process applies to both single-player and multi-player scenarios in which the priority of objects can change from one to the other.

The three elements of this architecture are as follows:

1. The Client: This comprises a game and the device that it runs on. Each game will have a different resource requirement and each device will have a different level of capability.
2. The Network: This includes everything between the Client and the Server, from routers to exchanges and other clients and devices.
3. The Server: This is very similar to the client in that it is also divided into the game (content and resource requirements) and the device (capability).

This architecture requires a wide range of information to be able to improve a user's QoE effectively. The game data, device data and network data are all values which are measured, such as frame rate, CPU usage and packet loss, with all being stored in a local database (DB) on each node. The data utilised by the architecture is explored further within the Databases and Intelligence sub-section. It is based on this stored data that decisions will be made that will lead to the improvement of a user's QoE.

The remainder of this chapter goes on to explain the architecture in detail. Firstly, it begins with a high-level explanation of the architecture and then moving on to an explanation of the possible scenarios that would be encountered by the architecture and how it would adapt to these scenarios. The chapter then moves on to explaining the DBs and Intelligence, Events and RPC's and finally focusing on the Network and its effect on the Architecture.

3.3 THE ARCHITECTURE

Figure 3.1 shows the proposed architecture. The purpose of this architecture is to provide a client with a high QoS, regardless of client and network conditions, through the utilisation of distributed resources. The distributed resources are represented by Node 0 and Node X; these nodes can have resources available to be able to provide assistance to Node 1 and therefore deliver a high QoS. The three elements (the Client, Network and Server) are illustrated along with the DBs through which distribution and assistance decisions can be made by intelligence nodes located in all devices running the game. Elements of a basic Simple Network Management Protocol (SNMP) are at work within this architecture [82], [83]. In the same way, SNMP gathers information from a diverse range of systems

and acts upon that information, this architecture collects data from agent-like Remote Procedure Calls (RPCs) and stores it in local databases (DBs) upon which the local intelligence will act. The RPC allows a node to call a function on a remote node, for example, a client could send an RPC to the Server in the same way an Agent would be sent asking for assistance and this would cause the server to check its resource availability.

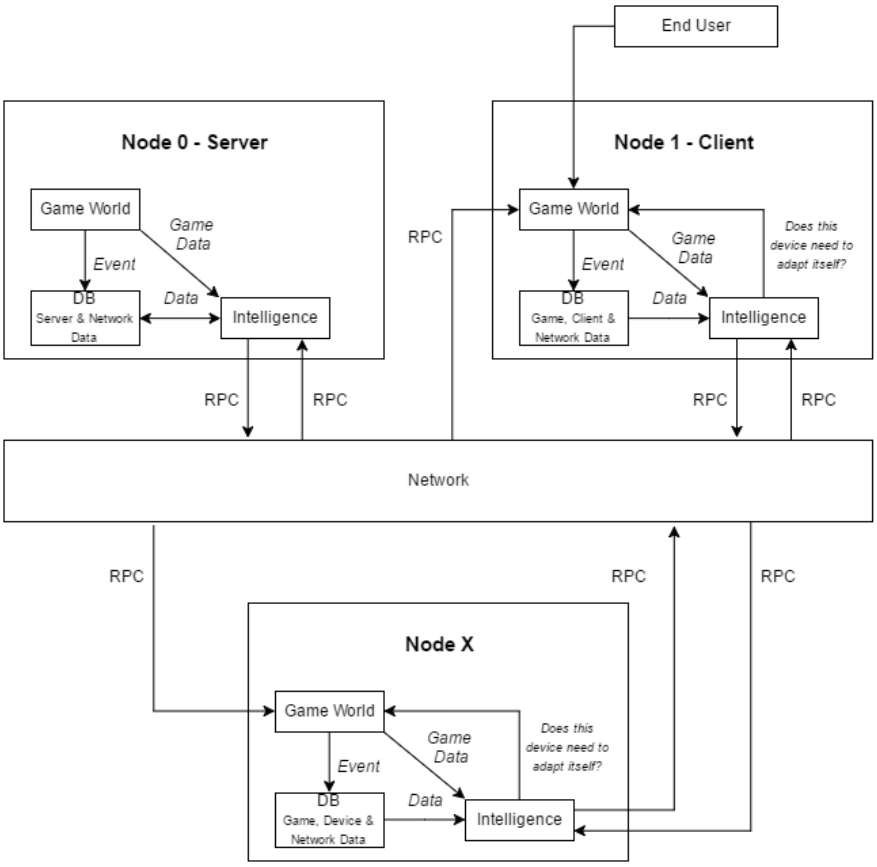


Figure 3.1 The proposed architecture which combines Cloud assistance (represented by the server), with Fog Assistance (represented by Node X) and Self-Adaptation (occurs within Node 1 – Client)

The server maintains the state of the game of all connected clients and will have data stored relating to the condition of the server and the network. The server's intelligence, a component which can make a variety of decisions based on information fed to it from the DB, will decide whether to assist a node itself, pass it off to another more capable node or inform a client device that it will have to self-adapt. For example, the server receives an RPC from a client detailing a declining framerate. The server will then send an RPC to the client asking for its game and client data and from this it can deduce which game component from graphics, physics and AI is causing the decline. When the server has this data, it can then decide where distribution will take place based on the information within the server DB - the data stored within this is explained within the sub-chapter 'Explanation of DBs and Intelligence. If the server has the available resources, it can take on the extra processing and assist the client. If this

is not possible, then the task can be passed to a network node (Node X in Figure 3.1) which can assist the client and if this is not possible then the client must adapt to its current situation. Node X represents local network nodes such as PCs, games consoles and other video game ready devices.

The network covers all devices between the client and the server. Each device will have different capabilities and differing connection types and speeds. Network device data consisting of capability and connection data will feed into the DB on the server. The network device will send its capability data (CPU usage etc.) to the server and along with its connection status to the struggling client (round trip time and connection type). If the server does not have the available resources to provide assistance to a client requiring aid, then processing can be passed off to a node on the network which has the required resources and a reliable connection.

Within the client, data relevant to the game, such as FPS, will feed into the Game DB. Simultaneously data relating to the client device, such as CPU percentage usage, will feed into the Device DB. Client decisions, made by the intelligence portion, include which element to ask for assistance with and which element to reduce if an RPC has been received relaying the message to self-adapt. All decisions are made based on data in the Game and Device DBs.

From Figure 3.1 it can be seen that, via the network, a single client is connected to many network nodes. These network nodes encompass everything, from the server supplying game data to the local network exchange as well as devices local to the user. This proposed architecture combines the Cloud with the Fog and an added form of assistance, dubbed Self-Adaptation, created for this architecture. The Cloud element of this architecture takes the form of all nodes that are a large distance away from the client device, represented by Node 0 – Server within Figure 3.1. The Fog element takes the form of all nodes that are local to the client, for example, devices in the same building, nearby building or local town. The Fog is very similar to the Cloud with a difference being distance as devices that can assist with processing are found closer to the client device. The Self-Adaptation element is contained within the client itself and is represented by the Intelligence node within Node 1 – Client in Figure 3.1. If all else fails with regards to acquiring assistance from other sources, then the client can adapt to help maintain a good QoS. This element will begin executing if a signal is received notifying the client that no assistance can be provided. The Self-Adaptation will then begin to reduce non-essential game elements, such as the number of particles within a particle effect, which will have a positive effect on the QoS and possibly the QoE had by the user.

3.4 DATABASES AND INTELLIGENCE

In this section, there is a discussion of the role of the databases and the intelligent components found in each network node. The role of the databases is to store information about the device itself, the video game and the network. The role of the intelligence is more complex in that it must make decisions based on the information within the database as well as the information it receives from outside sources such as a server or other node.

The databases store a lot of information relevant to the architecture. The table below summarises the data required and where it is stored.

Table 3.1. The databases and information required for this Architecture:

Client/Network Node			**Server**
Game	**Device**	**Network**	**Device/Network**
1. FPS 2. Current Level of Detail (LOD) 3. Resource Requirement of Tasks 4. Number of objects on screen (AI, Physics, Graphics) 5. Priority of on-screen objects	1. CPU usage 2. GPU usage 3. RAM usage 4. Device Type 5. Receiving/Providing Assistance	1. Latency from server 2. Packet Loss from server 3. Connection Type 4. Latency from device 5. Packet Loss from device	1. CPU usage 2. GPU usage 3. RAM usage 4. Number of devices connected

Each of these database entries are explained below. The values of each will determine which game element from AI, Physics and Graphics gets processed and where.

Within the Game section on the Client/Network Node:

1. *FPS*: The frames per second count that the game is running at. If this value falls below a certain threshold then some form of assistance is required.

2. *Current LOD*: A game utilises LOD techniques to improve or reduce the quality of all objects. A lower LOD will provide a higher FPS count as objects become more pixelated in appearance and therefore take less time to draw.

3. *Resource Requirement of Tasks*: Each task, for example an explosion moving many objects, in a game has a resource requirement; if the resources to run the task are not available then the task will be run elsewhere. For example, a task could require 10% of the CPU processing power and 20% of the GPU.

4. *Number of Objects on screen*: The number of objects of each element on the computer screen.

5. *Priority of on-screen objects*: Each object or group of objects on-screen will have a priority. Objects that are core to the gameplay, such as a non-player character that is crucial for furthering the game, will have a high priority whereas ambient AI, such as fish in a pond or a flock of birds, will be a low priority.

Within the Device section of the Client/Network Node:

1. *CPU Usage*: The percentage of CPU power in use, if this value increases too much, then the computer will slow, affecting game performance.

2. *GPU Usage*: The percentage of GPU power in use, if this value increases too much, then the computer will slow affecting in-game performance.

3. *RAM Usage*: The amount of RAM in use. The less RAM available the slower the computer is.

4. *Device Type*: The devices on the network will vary, knowing what they are will give some indication to their abilities and therefore it can be determined if they can assist or not. Highly capable PCs can aid whereas mobile devices and tablets cannot.
5. *Receiving/Providing Assistance*: This is a note on the device which states whether it is receiving or providing assistance or if it is available. This will allow the server to determine whether the device can be used to assist another.

Within the Network section of the Client/Network Node:

1. *Latency from server*: This value is the time it takes for a signal to be sent from the server. If this value is too high and assistance is required, then it will be necessary to distribute processing to somewhere other than the server or for local changes to occur.

2. *Packet Loss from server*: This is the rate at which data packets are transmitted from the server but do not reach the destination. An experiment in which volunteers played a first-person

shooter over a varied network conditions found that a higher latency (around 200ms) and a lossless connection was preferred over a lower latency with just 0.75% packet loss [56].

3. *Connection Type*: Clients can connect to the network through either a 3G/4G connection wireless connection, a wireless connection in-house to a hub or via a wired connection.

4. *Latency from device*: This is similar to Latency from server. In this case, this is the latency from an assisting node to a client that requires assistance. If this value is too high, then it will be necessary to distribute processing to somewhere other than this device.

5. *Packet Loss from device*: This is similar to Packet Loss from server. In this case, this is the packet loss from an assisting node to a client that requires assistance. If this value is too high, then it will be necessary to distribute processing to somewhere other than this device.

Within the Device/Network section of the Server:

1. *CPU usage, GPU Usage, RAM Usage*: These are the same as the headings under the Device of the Client/Network Node.
2. *Number of Devices Connected*: This is the total number of devices connected to the server. This value can help determine whether there are other nodes available to potentially provide assistance.

These values can vary, some only slightly as there is a limited number of entries such as the Priority of on-screen objects can be either High or Low, others such as FPS can vary greatly. Depending on how much they deviate from an optimum value, the intelligence will act. These values will affect what is distributed and where. The most important of these values is the FPS value, once this begins to fall then action will be taken, also if it increases dramatically with a change in other values, then action will be taken. Regardless of if there is distribution occurring or not the DB will be monitored frequently by the intelligence.

With this architecture, the first intelligence to act is the clients. The client will make a decision as to whether it needs assistance or not. The first metric to be checked is the FPS. If this begins to drop over a set period of time, then action needs to be taken. Likewise, if there is an increase then action may need to be taken. If the FPS drops then a change in the number of objects on the screen is more than likely the cause, this can be checked as it is a metric stored in the DB. The priority of these objects will then also be checked, if they are of a very high priority then ideally they should be kept to the local device and if not then they can be cleared for distribution or reduction in self-adaptation. A decision tree for the client/network node intelligence is detailed in Figure 3.2.

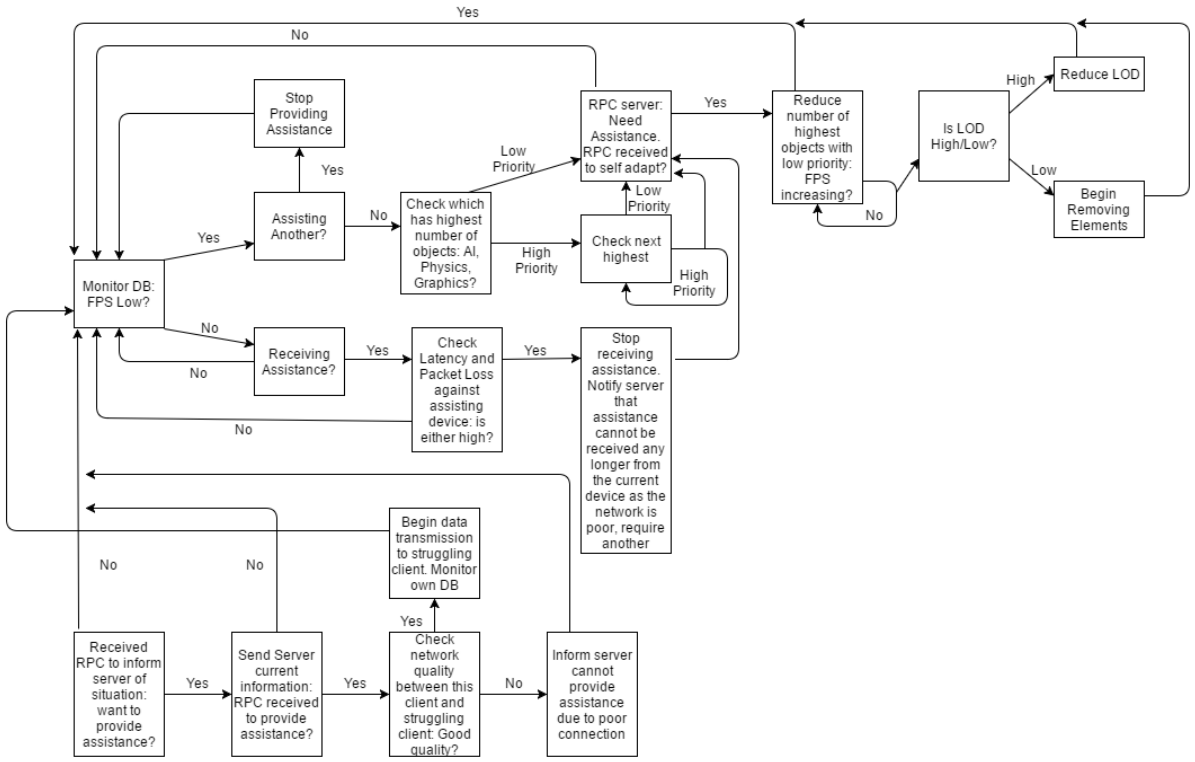


Figure 3.2. The decision tree of the client/network node intelligence.

All values in Table 3.1, underneath the heading of Client/Network Node, fit into the above decision tree. The first measure being looked at in each cycle of the intelligence, as it checks the database periodically, is the current FPS. Regardless of whether the FPS is above a certain threshold or below, decision making will progress through the tree. If the FPS is low, then it needs to be determined if this is caused by assisting another machine. If assistance is being provided, then this must be reduced to facilitate the maintenance of a higher QoS. If assistance is not being provided then the database must be examined to determine which element between AI, Physics and Graphics has the highest number of objects on screen and being processed. For each of these three in-game elements, there can be several sub-elements. The priority of this group is then checked, if it is a high priority, then it would be ideal to have this group processed locally and so the next highest number of objects is checked until a low priority is found that has a high resource requirement. If a low priority group is found, then an RPC is sent to the server asking for assistance with this group. If no low priority group is found that has a high impact on resources, then the group with the highest number of objects becomes the topic of the RPS sent to the server.

The RPC that is sent to the server will contain the following information:

- Whether it is an AI, Physics or Graphics element.
- A more specific description of what sub-element it is, such as its name. For example, AI Birds which could be a flock of birds in-game.

- The number of objects within the sub-element. For example, 5 objects contained within AI Birds.
- The resource requirement of the task.

Once this RPC is sent to the server, the client may or may not receive an RPC in return informing it to self-adapt. If this RPC is not received, then the client will be receiving assistance either from the server or another network node and can continue monitoring its database. If this RPC is received, then the sub-element that was chosen by the client will begin to be reduced. After this, there is a check on the FPS and if it increases then the client can continue monitoring its database. If not, then the client will continue to reduce the same element to a point where the core of it remains. If there is still no change having reduced a sub-element, then the client will move on to reduce the next sub-element which has both high volume and a high resource impact. If this reduction continues until only a fraction of the objects remain then the current LOD of the game will be examined. If it is high, then it can be lowered until there is a positive change in the current FPS. If the LOD is reduced to its lowest possible value, then game sub-elements will begin to be removed until the FPS improves. These will be removed on a low priority basis.

If the FPS within the database is found to be high, then it must be noted if there are external factors causing this. The DB is checked to see if the client is currently receiving assistance. If not then the client can continue monitoring itself, if assistance is being received then the current latency and packet loss between the client and the assisting node must be checked. If either of latency or packet loss are high, then the assistance must be stopped as these can prevent data arriving at the destination which can potentially lead to a decrease in processing and eventual increase in the current FPS. The server will receive a message from the client asking for help from another node as the network condition between the current node and itself is too poor to transmit data across.

Another step attached to this decision tree is if an RPC is received that asks the client to inform the server of its current situation. This branch focuses on the providing assistance ability of the client. The end user can state whether they would like to provide assistance if asked, they may want to for some form of reward and may not if they are within a competitive gaming environment in which carrying out additional processing may affect their performance. If they choose not to assist, then the client will continue its monitoring and if they wish to assist, the client will update the server on its current information. The server will then make a decision as to whether the client is required or not. In this case, if an RPC is not returned then the client will continue monitoring itself, however, if an RPC is returned then the client will be required to assist another. Once this is confirmed, the network quality between the assisting node and struggling node will need to be determined. If both the latency and packet loss are low, then data transmission between the two nodes can begin. Once this occurs, the

assisting node will continue to monitor its situation and can cancel its provision of assistance if its FPS falls. If the network quality between the assisting and struggling node is poor then the assisting node will inform the server that it cannot assist, and the server will find another.

This decision tree allows the client to adapt appropriately to dynamic situations. More intelligence can be found operating on the server. A decision tree for the server intelligence is detailed in Figure 3.3.

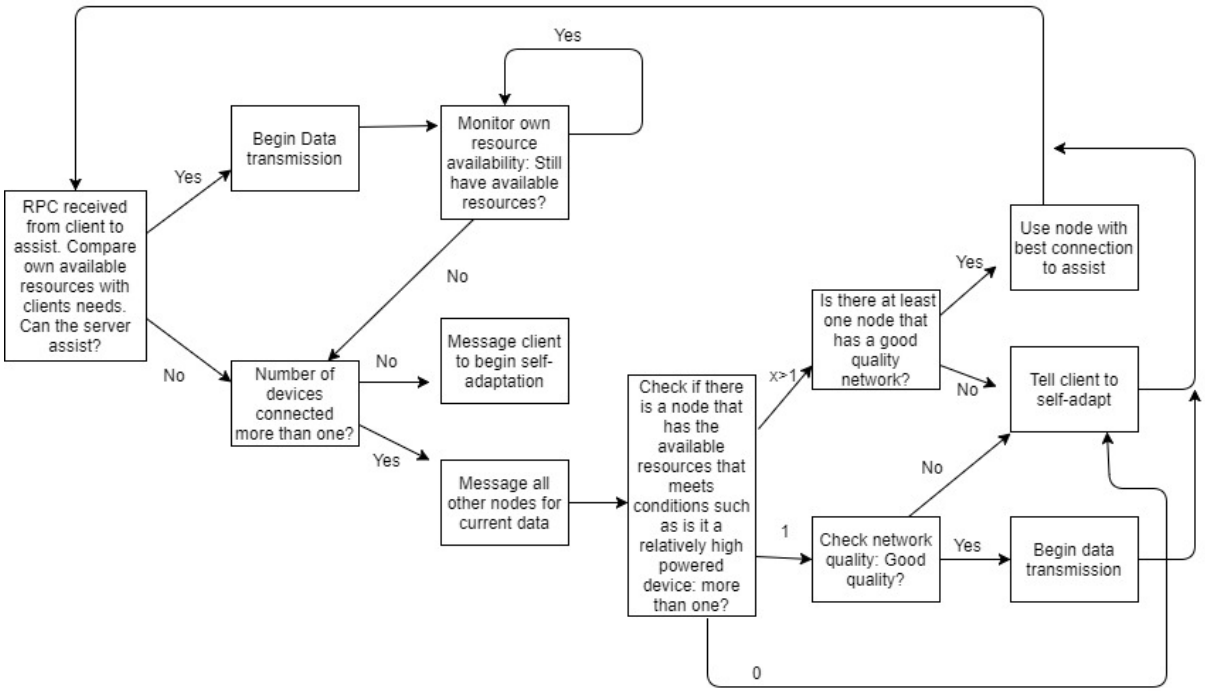


Figure 3.3. The decision tree of the server intelligence.

The server intelligence begins when an RPC is received from the client asking for assistance. The first step taken here is to compare the server's available resources against the resources required by the client. If the server has the available resources, then it will begin transmitting game data to the client and then monitor its resource availability. If the required resources are not available, then the server will look to another device on the network to assist the struggling client. If there are no other devices on the network besides the server and the struggling client, then the server will send an RPC to the client informing it to begin its self-adaptation process. If there are other nodes on the network, then the server will RPC them all asking for their resource availability information, and once it receives all of this information, then it will compare the values of each network node against the struggling client's requirements. If there are no nodes suitable for the task, then the server will RPC the struggling client to self-adapt. If there is only one node that can assist, then the server will inform the node to check the connection between itself and the struggling client. If the connection is good, then the node can

assist and if the connection is poor then the server will be told by the node that it cannot assist, and the server will then RPC the client informing it to self-adapt. If there is more than one node that is capable of assisting and at least one has a good quality connection with the struggling client, then the node with the best connection will be chosen to assist. There is a possibility that even though many nodes can assist, the network is too poor to transmit data over, if this is the case, then the server will RPC the client to begin self-adaptation.

The decision tree's in Figures 3.2 and 3.3 show an adaptable architecture. If any of the conditions change at any time, for example, an assisting nodes FPS drops then the node will halt all assistance provision. The struggling client that just had its assistance cut off will notice a drop in its own FPS through which it continues along its series of decisions to then RPC the server for assistance. From here the server will either help the client itself, find another suitable candidate node or RPC the client back to begin self-adaptation.

3.5 CLIENT SCENARIOS

As explained previously in this chapter, this architecture combines cloud assistance, fog assistance and self-adaptation to provide a high QoS, which in turn will help provide a high QoE. With these three approaches, each client can have their QoS improved three different ways:

1. Assisted by the server (Cloud)
2. Assisted by a local network node (Fog)
3. Limited access to network resources or high priority processing is required (Self-Adapt)

The first and second of these are similar, with the only difference being which node is assisting the client. Firstly, it will be determined if the server can aid, if not then a local node will be messaged for assistance and then finally if not then self-adaptation will begin.

The first of these architectural scenarios, in which the server assists the client, is shown in Figure 3.4 with components explained in Table 3.2.

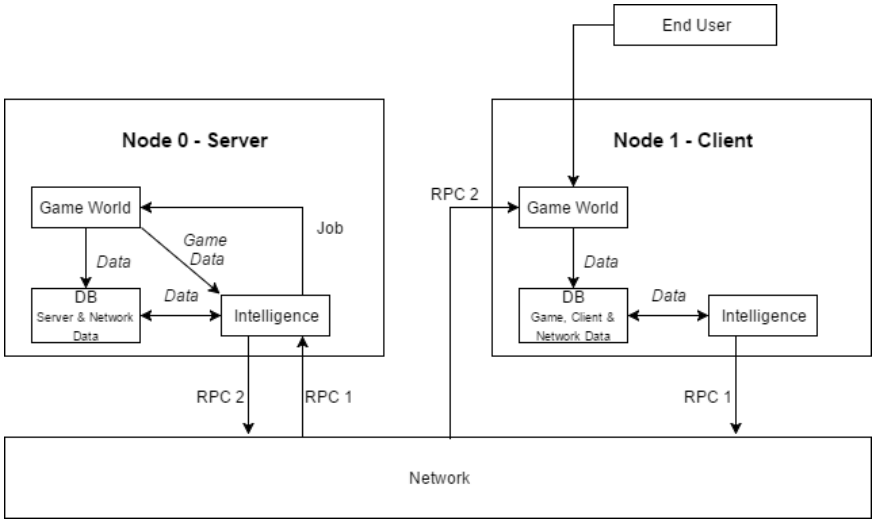


Figure 3.4. The Server Assisting the Client

Table 3.2. Items in Figure 3.4 explained

Item	**Explanation**
Node 1 – Client: Game World	Player actions, such as shooting, cause events within the game world. Each event will have a variety of Graphics, AI and Physics jobs. Each job can then be split into many tasks. For example, a Physics job can have a task to roll rocks down a hill and another task cause a sheet to move in response to wind. Data from the Game World such as the number of objects on screen feeds into the DB.
Node 1 – Client: DB	Data from the Game, Client Device and Network feed into here.
Node 1 – Client: Intelligence	This component reads data from the DB and makes decisions based on that data. The decision here being to ask for assistance with a group of jobs.
RPC 1	A message for the Server asking for assistance with an element from Graphics, AI or Physics.
Node 0 – Server: Intelligence	This component will receive RPCs asking for assistance. It will check its resource availability, found in the DB on the Server, against the needs of the client and find it can assist. The relevant job is then executed in the game world.
RPC 2	Contains information relevant to the element which Node 1 – Client is struggling with. For example, positional data of game objects. This is passed directly into the Game World of the client.

The starting point of this architecture is when the client begins to struggle under the current load. As the end user continues to move their in-game character around the game world, the DB on the client is updated. Data in this DB includes the current FPS, GPU percentage usage and CPU percentage usage. This data is accessed by the intelligence component (Figure 3.4) on the client device and it is here that critical decisions, with regards to the end users QoE, are made. If the intelligence deems that the client is struggling and requires assistance, then a message in the form of an RPC is transmitted to the server. This RPC is received by the server intelligence and informs it that the transmitting client requires assistance and what it requires assistance with for example a CPU intensive task such as AI pathfinding. This RPC will trigger the server to compare its resource availability with the needs of the client, and if there are the resources available then it will begin data transmission. From the server, the relevant data is transmitted via RPC's along the network and to the struggling client. For example, they can contain positional data of in-game AI objects and a message telling the client to stop local processing of those AI objects. As the client is now relieved of some processing, it will check its DB and compare the values against minimum QoS values. Provided the values within the DB are higher than the minimum values then no further assistance will be asked for.

The second of the scenarios, in which a local network node such as another client provides assistance, is detailed in Figure 3.5 with Figure items explained in Table 3.3.

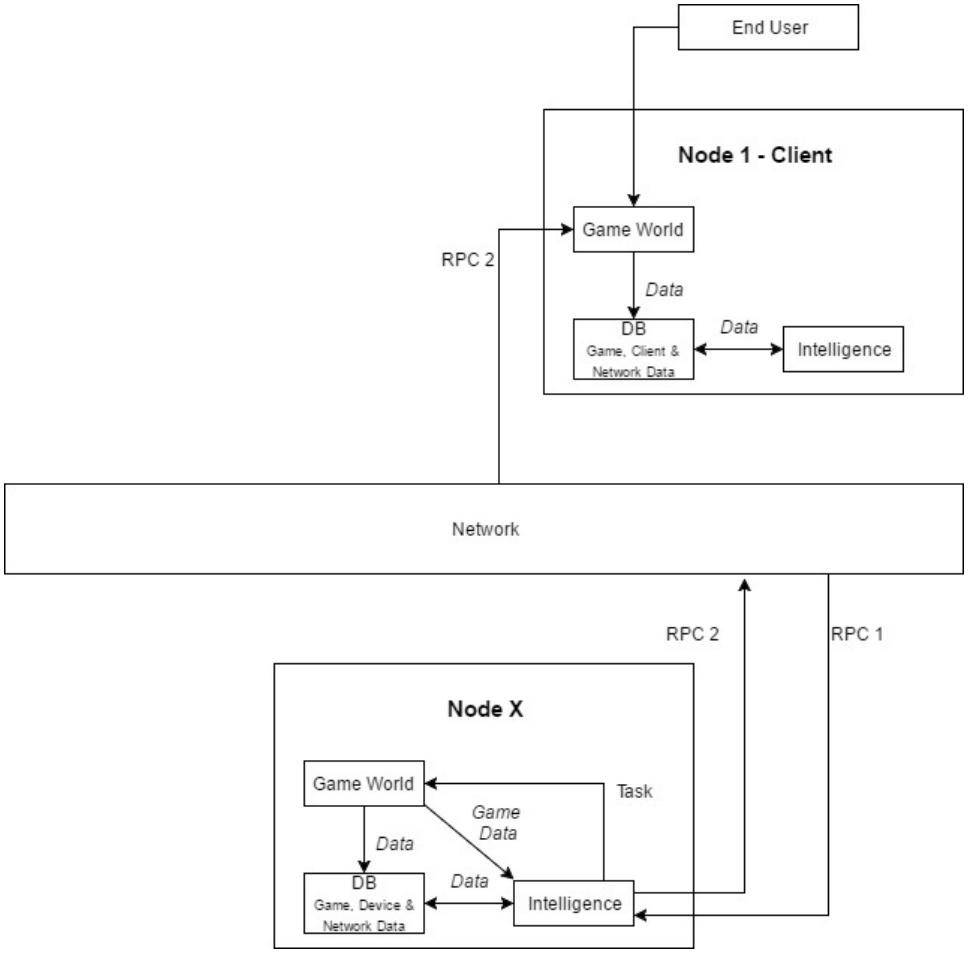


Figure 3.5. A local network node assisting the Client

Table 3.3. Items in Figure 3.5 explained

Item	Explanation
RPC 1	The local network node receives an RPC informing it to begin assisting a client with a task. This node will have been chosen due to its best fit in terms of resources available and connection health.
Node X: Intelligence	This component will receive the RPC to assist and begin execution of the required task within the Game World. Once this begins the Intelligence will receive the game data and pack it into an RPC.
Node X: Game World	The task will be executed here and then the necessary game data will be sent to the intelligence.
RPC 2	Contains Information relevant to the element which Node 1 – Client is struggling with. For example, positional data of game objects. This information is passed directly into the Game World of the client.

This scenario will only occur if the first scenario, where the server assists the client, is not feasible. The first scenario may not be feasible if, for example, the condition of the connection between the client and the server is poor or the server may not have the available resources to assist the client with their task. Classifying a poor connection depends on the type of game being played. For example, a simple turn based game will not require the same network speeds as that of a competitive first person shooter as data is not required instantly. So, as in the first scenario, the client messages the server asking for assistance and the server checks to see if it has the resources available and that the network is stable. If for any reason there are no resources, or the network is unstable then this scenario begins. The server will send an RPC out to all network nodes asking for information about them such as their current CPU and GPU usage, their current FPS if they are a client playing a game and their latency and packet loss to the struggling client. This information is then sent back to the server from each node. From this information, the server will be able to determine which network node is the best fit for assisting the struggling client. The struggling client will want assistance from a node that is close, has a good connection and low CPU and GPU usage values. At the same time, if the node assisting is another game player then the end user will only want to assist a client whose processing needs will not affect their experience. Bearing this in mind, only game clients that inform the server that they allow their resources to be utilised by others will be considered.

The third of the scenarios, Self-Adaptation, is detailed in Figure 3.6 with Figure items explained in Table 3.4.

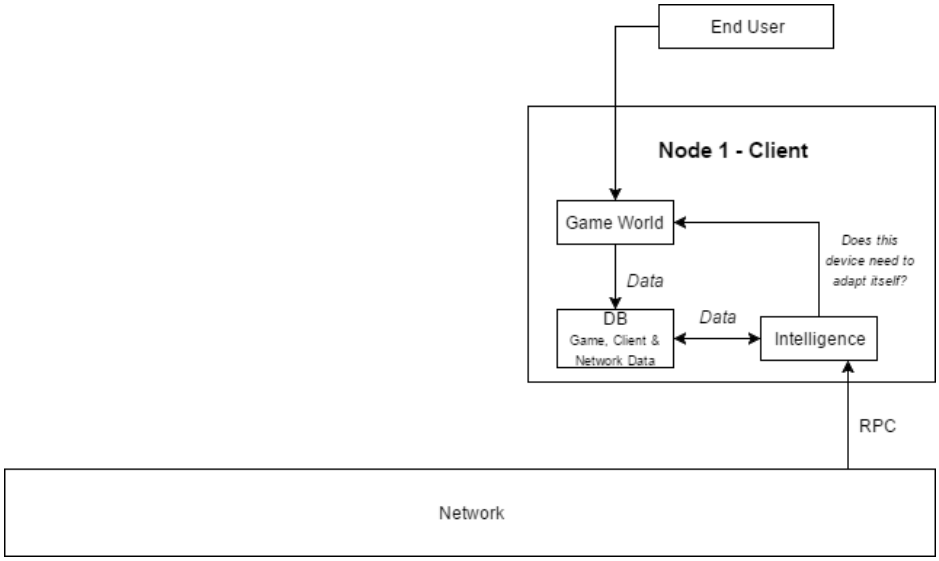


Figure 3.6. Self-Adaptation occurs within the Client

Table 3.4. Items in Figure 3.6 explained

Item	Explanation
RPC	This is a message from the Server telling the Client to begin Self Adaptation. This will be because there are no resources available on the network to assist.
Node 1 – Client: Intelligence	This component will receive the RPC and look at the data available in the DB to see which element it can begin to reduce in number based on the number of each element and its priority. It will then reduce this element within the Game World. The Intelligence will closely monitor the DB, when an improvement in QoS is seen then the Self-Adaptation process will stop.
Node 1 – Client: Game World	Player actions such as shooting cause events within the game world. Each event can have a variety of Graphics, AI and Physics jobs. Each job can then be split into many tasks. Data from the Game World such as the number of objects on screen feeds into the DB. The Game World will be affected as elements are removed in an attempt to improve the current QoE. This will be reflected in the data entered into the DB.
Node 1 – Client: DB	Data from the Game, Client Device and Network feed into here.

This scenario is a fail-safe in that if the server cannot assist and neither can any other network node then the client must carry out its own improvements. This scenario begins when the server receives all network node data and finds that there are no nodes available to provide assistance, another case may be that the condition of the client's connection is so poor that to provide assistance would be a waste of resources. An RPC will be received by the intelligence component on the client informing it that it must adapt to its current situation itself. Based on the data received from the DB, the intelligence component will make a decision to reduce a game element that is causing it to struggle. For example, the DB could show that there is a very high concentration of particles in the current scene, these could be from a fire animation. This information could be backed up by a very high GPU usage value and low FPS. With this data, it would be decided to begin reducing the current number of particles within the fire animation while monitoring the FPS and GPU usage. By reducing the particles in the particle effect and monitoring the DB, the intelligence will be able to find an optimum number of particles that can remain in the game world and still have a high FPS and low GPU usage. Removing the fire animation from the scene completely would see a very sudden improvement in values. However, this would then take away from the immersion aspect of the game [84]. By simply reducing the number of particles, the game element remains, and the FPS increases and the immersion of the

game is not affected as much. Other elements that can be reduced include the number of physics objects or artificial intelligence (AI) objects in the scene. For example, as shown in Figure 3.7, a building that is collapsing can have fewer objects falling as some can be grouped together or a flock of birds in the background, which are not affecting gameplay, can be reduced in number.

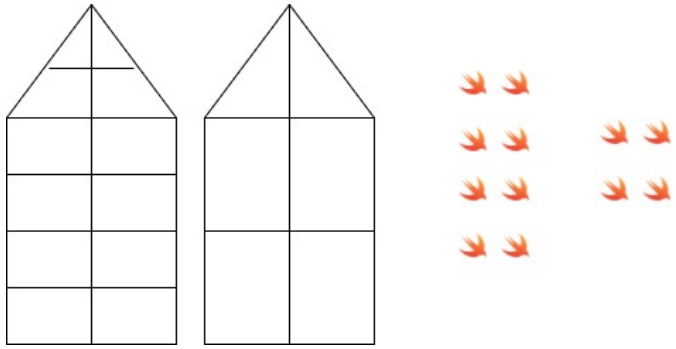


Figure 3.7. The grouping of building objects and reduction in a flock of birds due to Self-Adaptation

With these three scenarios (Server Assistance, Client Assistance and Self-Adaptation) there is always a way for the client device to improve upon the QoS if it begins to fall. The server that provides game data is the first node to provide assistance. If this is unavailable for any reason, then all available network nodes are examined for the best fit for the client and finally, if there are no other network nodes available then the client itself will adapt to its situation through the reduction of in-game elements.

3.6 EVENTS AND RPCS

Each video game can be broken down into core tasks that ultimately occur due to the actions of the end user. Figure 3.8 shows the breakdown of these actions into eventual tasks.

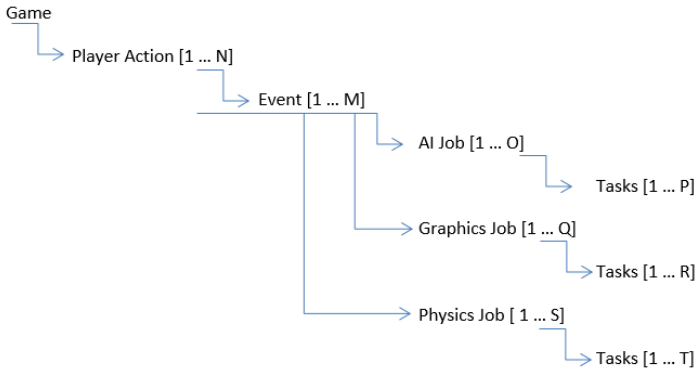


Figure 3.8. The breakdown of a game into many tasks.

As a game is running, the player will execute actions through pressing a button. This button press can cause an event such as the destruction of a building. This event can then go on to run any number of

AI, Physics and Graphics Jobs. For example, AI characters running away, Graphics animations in the form of fires and Physics in the form of parts of the building falling. Each job is made up of many tasks. For example, the AI running away will need to run to a specific location and not visit a previous one. Other AI could be characters running towards the building. Many graphics tasks could be multiple fire animations as well as water animations. Many physics tasks would be separate parts of the building collapsing at different times. Each of these tasks will have a priority aswell as a resource requirement. It is with this requirement that it can be determined whether a task will be executed locally, on the server or elsewhere on the network. The priority of the task can determine whether the task is run at full capacity on the client device, distributed via the network or the client can self-adapt in which the number of objects can be reduced.

Detailed on the Architecture Figures (Figures 3.1, 3.4, 3.5, 3.6) are RPCs. The RPCs are similar to agents as they carry information within this architecture and execute it. The RPC from a client to a server will carry information either asking for assistance and for which problem or informing the server of its resource availability. The RPC from a server to a client can be a message saying to self-adapt or game data to assist with processing on the client's issue. Similarly, the client to client RPC will be game data such as object position being transmitted from one client to another.

3.7 THE NETWORK AND ITS EFFECT

Even though there have been vast improvements made to network infrastructure, there are still unreliable areas and complications that can occur. The three metrics considered in this architecture are the Latency, Packet Loss Percentage and the Connection Type. All three can greatly affect the network. The Latency is the time data takes to get from one location to another, if this is too high then data is taking too long to be received and processed by the destination device. The Packet Loss Percentage is the percentage of data packets dropped between the transmission point and the destination. If this is too high, then no data is being received to be processed. With Latency and Packet Loss, if either of these is high then there will not be an accurate read on the FPS of the client device if processing is being distributed. If data is taking too long to be received or not being received at all then little or no processing is occurring at the destination. If less data is processed, then FPS will be high as only some local processing will be taking place. This is a false positive as objects on screen that should be performing some action are not, causing a reduction in the immersion of the game and possible reduction in the QoE provided. Therefore, while distribution is taking place, the Latency and Packet Loss must be monitored closely. The connection type comes into focus more when looking for assistance, a device may be performing well, but this could be over a mobile network which could run out of data at any time or drop signal completely.

3.8 SUMMARY

This chapter has discussed the proposed architecture that will aim to improve the QoE of an end user by focusing on the QoS provided. The analysis of techniques currently in use, shown in chapter two, have shown that each of the Cloud and the Fog have their benefits and drawbacks. By combining the cloud and fog approaches along with a fail-safe in the form of self-adaptation, an architecture has been created that will help to provide a high QoS which in turn will provide a high QoE for the end user. In this architecture chapter, three client scenarios are explored and show that whatever the situation of all components, the architecture will adapt to benefit the client.

CHAPTER 4 EXPERIMENTAL METHOD AND SYSTEM SETUP

4.1 OVERVIEW

This chapter presents the method and system setup designed to test the architecture proposed in Chapter 3. There is a focus on the decision-making ability of the architecture. The decisions made within the Architecture are: which game element (AI, Graphics, Physics) to ask for assistance with and which network node will assist which client based on a wide variety of metrics such as device type, capability and connection condition.

4.2 INTRODUCTION

In Chapter 3 an architecture was proposed that intends to deliver a high QoS for an end user. A high QoS can be supplied via the utilisation of distributed resources. As technology improves, in relation to both hardware and software, and the number of devices connected to a network increases, then there is an increased amount of resources available. Instead of adding additional resources to an already highly resourced network as some research suggests [39], this architecture utilises the resources already available. This distributed architecture can find available resources in the game server which can be providing game content or on other nodes discovered along the network such as at an exchange local to the end user or on another end user's device. The client will react to its current situation, if the need arises, and will inform the server of its requirement for assistance. The server can provide assistance, or it can instruct another network node to assist. This architecture also contains a fail-safe mechanism which is required due to the unpredictable nature of a computing environment.

Within Chapter 3, three scenarios were identified, namely:

1. The client is assisted by the server.
2. The client is assisted by a local network node.
3. The client can self-adapt.

This chapter focuses on developing these scenarios into experiments and explaining the findings. Firstly, Section 4.3 describes a paper based illustration of the decision-making process. The decisions illustrated here are the client choosing which element to ask for assistance with and the decisions made by the server to improve a client's low QoS. Section 4.4 describes the design of the Distribution experiments of Section 4.5 and the Self-Adaptation experiments of Section 4.6. As seen from the above list, the client can receive assistance from either the server or another network node. Section 4.5 explores the improvement in QoS of a client through assistance from a node that has the resources available. The Self-Adaptation experimental scenario in Section 4.6 shows how Self-Adaptation will improve the QoS when all other avenues are exhausted.

As explained in Chapter 3, the most important measure in the architecture is the FPS as it can be seen to be the most significant factor in providing a high QoS. Therefore, the focus of the experiments within this chapter is to provide a high FPS for the end user.

4.3 DECISION-MAKING ILLUSTRATION

The core of the proposed architecture is the decision-making processes that it follows were both the client and the server have decisions to make. For example, the client needs to decide which game element to ask the server for assistance with, or the server has to determine whether it can assist a struggling client or not.

As highlighted in Section 3.4 of Chapter 3, there is a decision tree for each of the client and the server. This section details experiments that show the decision-making ability of this architecture for both the client and the server.

4.3.1 CLIENT DECISION: WHICH ELEMENT TO ASK FOR ASSISTANCE WITH

The first illustration demonstrates how a client decides which element to ask for assistance with between AI, Physics and Graphics. Each of these elements affects the client device in different ways, and therefore different amounts of each are required to cause a client to struggle with its performance. For example, a client may be able to handle 1300 AI objects navigating throughout a game world before it begins to show signs of struggling performance. The same client may begin to struggle when the total number of physics objects increases beyond 1800 objects, and finally, it may take 40,000 particles in particle effects to cause the same client to show poor performance. Clients with different capabilities will be able to run different amounts of these elements. Therefore, the client must be able to determine which of the three is causing the drop in performance. Some benchmark software can be run on the client before playing a game, with this the client will be able to determine how it performs when running high numbers of these elements. Once benchmarking is complete the client can store the amounts it could run and compare with these whenever the performance falls when playing the game.

Figure 4.1 is adapted from Figure 3.4 found in the Architecture Chapter. It details the part of the decision tree that this illustration focuses on which is how a client would decide which element to ask for assistance with out of the three focused in this thesis: AI, Physics and Graphics. The remainder of Figure 3.4 focuses on whether the client receives an RPC to assist another node or if an RPC is received to begin the Self-Adaptation feature of the Architecture.

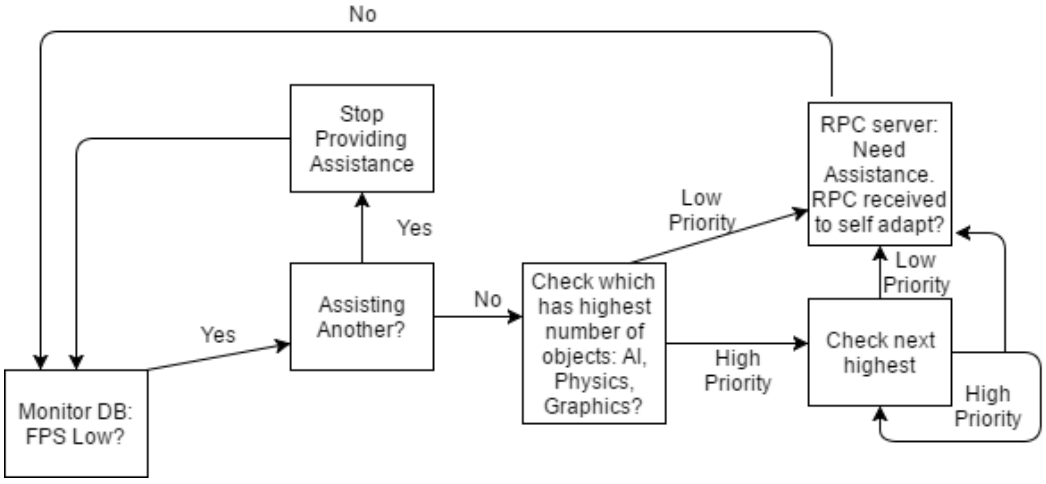


Figure 4.1. Deciding which element to ask for assistance with between AI, Physics and Graphics.

The purpose of this illustration is to show the client deciding which game element to ask for assistance with based on Figure 4.1. To begin with, a low FPS will have been detected by the client. Firstly, the client will check if they are providing assistance to another as this can lower the QoS. If they are aiding another, then this will be stopped, and the FPS checked again. If the FPS is low and they are not assisting then the element, between AI, Physics and Graphics, with the highest number of objects and lowest priority will be selected as the element to be processed elsewhere. If there is a case where all elements are of a high priority within the current game scene, then the one chosen is that which has the highest number of objects as it will have the largest impact on QoS. Once an element has been chosen, then an RPC will be sent to the server asking for assistance. If an RPC is not received in return informing the client to self-adapt, then it can continue monitoring its FPS. It must be kept in mind that, between these three game aspects, that there will be a ratio involved as the number of particles within a particle effect will be far greater than the number of AI objects within an AI task. The ratio observed for this illustration is 1 AI object, to 1 Physics object to 20 particles. In a real-world scenario, it would be ideal, before playing a game, for a client to be tested to find its ratio of AI : Physics : Graphics. This ratio can then be used to find which element has the highest number of objects and can be processed elsewhere on the network. Table 4.1 show the results of this illustration.

Table 4.1. The results in which a client decides which element to reduce based on varying object counts and priorities.

Client	AI Object Count	AI Priority	Physics Object Count	Physics Priority	Graphics Particle Count	Graphics Priority	Element to distribute
1	1200	Low	500	High	20,000	High	AI
2	200	High	1500	Low	20,000	High	Physics
3	200	High	500	High	60,000	Low	Graphics
4	1000	Low	800	Low	20,000	High	AI
5	800	Low	1000	Low	20,000	High	Physics
6	600	High	800	Low	60,000	Low	Graphics
7	1000	High	1500	High	20,000	High	Physics
8	1000	Low	2000	Low	20,000	Low	Physics

The results in Table 4.1 show which game element the client would ask for assistance with based on object count and priority. The values for the object count are based on empirical testing carried out during research. A ratio of 1:1:20 was observed. An object count would be considered to be the highest if it was above the ratio. The priority of the elements varies from test to test as there is a variety of each element in games today with each having a different effect on overall gameplay. For example, a high priority AI could be the enemy shooting back at your character while a low priority AI could be a flock of birds in the sky. A set of decisions can be observed in these results:

1. If there is a high number of low priority elements, then ask for assistance with this element.
2. If there is more than one low priority element, then choose the element with the most objects.
3. If all elements have a high priority then, to have the largest impact on QoE, the element with the largest number of objects must be chosen. This decision will be made based on the ratio decided.

By asking for assistance with the element with the highest number of objects and lowest priority then the client will experience the best change it can. The purpose of this illustration was to show which element a client would choose to be assisted with based on two variables: the object count and the priority.

4.3.2 SERVER DECISION: MULTIPLE CLIENTS CONNECTED

The second illustration details the decision-making ability of the server. Many clients can be connected to the server with some requiring assistance, some able to provide assistance and others that are able to maintain a high QoS without receiving or providing assistance.

A network is comprised of a dynamically varying number of devices with a range of specifications. Each device may be considered a node, and each node may require assistance, may be able to provide it or may be on the borderline of requiring it in which case they can be more closely monitored.

Figure 4.2 is also found in Chapter 3 as Figure 3.3. It details the decision-making process of the server within this Architecture and is the focus of this Server Decision Illustration.

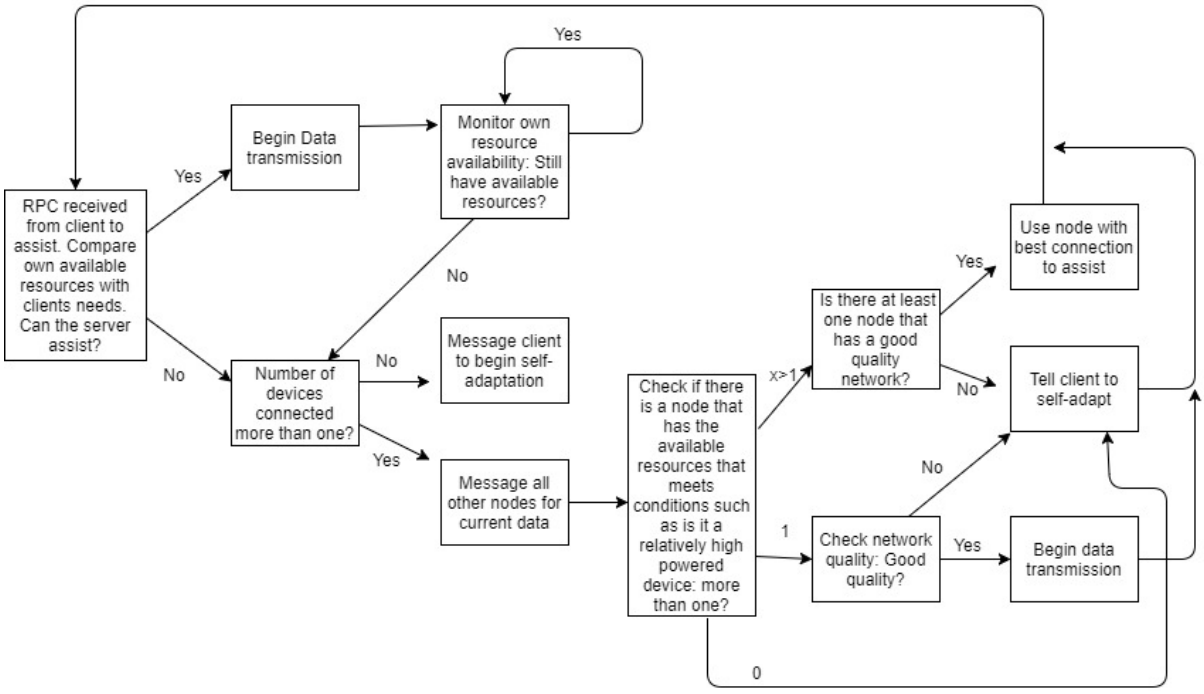


Figure 4.2. The decision tree of the server intelligence.

The server monitors communication from the client agents and decides how to best allocate resources to improve the client QoS. The server needs to decide where a client will receive assistance from or if the client must adapt to its current situation. This decision is made based on data from the nodes such as its current FPS, CPU and GPU usage, latency and packet loss percentage to the server and the type of device.

The first level of the decision made is based on the FPS of the device as this will determine whether a device will be able to provide assistance or not. A device is marked as being able to assist if its FPS is above 45 FPS, this value has been chosen as a midway point between the lowest expected FPS of

games today which is 30 FPS and the sought after 60 FPS. If the FPS is lower than this, then the device will not be considered as its own QoS comes first. If this measure is met, then the remaining data collected can be compared to the needs of the client.

The next values measured on the nodes that can assist are their CPU and GPU usage values. If one of these values is less than 40% usage, then this node will be marked as being able to provide assistance with this type of processes. Based on experimental testing the 40% boundary usage was chosen, this could potentially change in a real-world scenario. If a node has 30% usage on its GPU and 60% usage on its CPU, then the node is marked as being able to aid with GPU processes. A node can aid with both types of processes (CPU and GPU processes) if both usage values are above 40% then the node will no longer be marked as being able to help. For the purpose of this paper-based illustration, a node will need help with CPU or GPU processes if either components percentage usage is above 70%, this is an example percentage and could potentially change in a real-world scenario.

If the device requiring assistance has a good connection and the server has enough resources available, then the server can help. In this illustration, a good connection is considered as having less than 100ms latency and less than 2% packet loss as testing found this to be so. The boundary values for latency and packet loss have been chosen as examples; these would be subject to change based on game type i.e. a first-person shooter will require lower boundary values in comparison to a turn-based card game. If the server is unable to help, then a network node with resources available can assist. However, if the connection is poor then the device must help itself as, regardless of resource availability, the results of calculations must arrive quickly.

Device type is the least important regarding this experiment; it is included as the variety of devices able to play high-quality games is increasing. In this experiment, a device can be a PC, Laptop or Mobile. This variable only comes into play when finding nodes that can potentially aid a client. If a mobile is a network node found to be having the necessary resources to be able to provide assistance, it will not be used to provide assistance as a mobile will more than likely not have the same idle amount of CPU and GPU power as a laptop or PC. Other issues with mobile devices include battery power and connection reliability.

Table 4.2 shows the metrics of six example nodes connected to the server which have transmitted an RPC asking for assistance.

Table 4.2. The metrics of the six nodes connected to the server which are in need of assistance in an illustration to show the decision-making process of the server.

Node	FPS	CPU %	GPU %	Latency to Server	Packet loss % to server	Device Type
1	15	95	95	60	0.2	PC
2	25	85	85	200	3	PC
3	20	90	90	75	0.1	Laptop
4	23	88	65	50	0.5	PC
5	25	65	90	35	0.3	PC
6	19	90	90	70	0.3	Laptop

Going back and examining Figure 4.2, it can be seen that the next step in this process is to contact other nodes to find out their current resource availability and to determine whether they can assist those in need. Table 4.3 below provides example data of four nodes which have the possibility of providing assistance.

Table 4.3. The metrics of the four nodes which the server was able to contact with regards to potentially assisting those in need.

Node	FPS	CPU %	GPU %	Latency to Server	Packet loss % to server	Device Type
7	75	20	20	45	0.4	PC
8	65	20	45	85	0.1	Laptop
9	70	45	20	90	0.1	Laptop
10	46	39	39	70	0.2	Mobile

The values found in Tables 4.2 and 4.3 show some of the possible variation of device capabilities. Table 4.4 is the result of the server asking the nodes that can assist to find the latency and packet loss to each of the nodes that require assistance. This is example data. The results here will then be used to determine if and where a node gets assistance from.

Table 4.4. The latency and packet loss between the nodes of Tables 4.3 and 4.2.

Node That Can Assist	Node Requiring Assistance	Latency	Packet Loss %
7	1	100	0.4
	2	200	1
	3	40	0.05
	4	90	0.3
	5	80	0.2
	6	120	0.3
8	1	150	0.4
	2	200	1.2
	3	110	0.5
	4	50	0.1
	5	80	0.2
	6	90	0.3
9	1	300	2
	2	220	1.5
	3	90	0.15
	4	80	0.3
	5	50	0.1
	6	120	0.4
10	1	500	1
	2	250	2
	3	180	0.4
	4	100	0.2
	5	120	0.2
	6	140	0.3

The values found in Tables 4.2 to 4.4 show some of the possible variations of device capability and connection health. In this paper-based illustration, the server will see this information and make decisions. With the clients and connection having these example values, the results of this decision-making are found in Figure 4.3.

The following is a list of clients that require assistance:
Client 1: PC with a FPS count of 15. This device requires CPU&GPU assistance.
Client 2: PC with a FPS count of 25. This device requires CPU&GPU assistance.
Client 3: Laptop with a FPS count of 20. This device requires CPU&GPU assistance.
Client 4: PC with a FPS count of 23. This device requires CPU assistance.
Client 5: PC with a FPS count of 25. This device requires GPU assistance.
Client 6: PC with a FPS count of 19. This device requires CPU&GPU assistance.

The following is a list of clients that can provide assistance (A Mobile cannot):
Client 7: PC, it can provide help with CPU&GPU tasks.
Client 8: Laptop, it can provide help with CPU tasks.
Client 9: Laptop, it can provide help with GPU tasks.
Client 10: Mobile, it can provide help with CPU&GPU tasks.

The following client will be assisted by the server as it has the lowest FPS with an ideal network connection and the server has the available resources: Client 1

Client 2's Lag and Packet Loss Percentage are too high to send information to (from both the server and other nodes), therefore this client will begin reducing the number of objects in its current scene.

Client 3 needs help from another who can offer CPU&GPU resources. The client that will assist is Client 7 as the Latency and Packet Loss % are the lowest between these two nodes.

Client 4 needs help from another who can offer CPU resources. The client that will assist is Client 8 as the Latency and Packet Loss % are the lowest between these two nodes.

Client 5 needs help from another who can offer GPU resources. The client that will assist is Client 9 as the Latency and Packet Loss % are the lowest between these two nodes.

Client 6 needs help from another who can offer CPU&GPU resources. As there are no nodes available to provide assistance at this time, Client 6 will receive an RPC to begin Self-Adaptation.

Figure 4.3. A paper-based analysis of the Server Decision-Making.

As can be seen from Figure 4.3, each client that requires assistance either receives it or is messaged to begin Self-Adaptation. Client 1 is assisted by the server as it has the lowest FPS with a good connection to the server and the server has the available resources. Client 2 is struggling with a weak network connection to all network nodes and therefore would benefit more from Self-Adaptation. Client 3 requires assistance with both CPU and GPU intensive processes. Therefore, a network node that can provide these is searched for and found in Client 7. Client 4 needs assistance with CPU intensive tasks, and the node that can assist is Client 8. Client 5 requires assistance with GPU intensive tasks, and the node that can support with these is Client 9. Client 6 also requires help with both CPU and GPU processes, however as there are no more available nodes for this it must self-adapt. To reflect the wide variety of game-ready devices, each client also has a device type with a rule in place that

even though the CPU and GPU usage are low, if the device is a mobile then it cannot help. The most important reason for this rule is due to the limited battery life of a mobile to the large variation in mobile networks.

4.4 SUMMARY OF DECISION MAKING

These illustrations have shown how the decision-making ability of the architecture works with example values. In 4.3.1 the client decides which game element to ask for assistance with based on the element object count and priority. A low priority will be chosen over a high priority and a high count will be chosen over a low count in order to benefit the client as soon as possible. Secondly the decision making of the server is explored in 4.3.2. In this illustration ten clients are connected to the server, six of these are marked as requiring assistance with the remained marked as being able to assist. Each node has been provided with example values which vary from node to node in order to replicate a real-world scenario. The server goes through a set of rules in order to determine where each node will receive/provide assistance. Each of these experiments shows the core of this architecture which is its ability to make decisions which will result in an improved QoE for all.

4.5 DISCUSSION

Chapter 4 focuses on the experimental method and system setup for the architecture proposed in Chapter 3 and paves the way for the results covered in Chapter 5.

The illustrations of section 4.3 show how the decision-making ability of the architecture works. In 4.3.1 a decision is made by the client as to which game element to ask for assistance with, based on the object count of the element and its priority. A low priority will be chosen over a high priority, and a high object count will be chosen over a low object count in order to benefit the client as soon as possible. Section 4.3.2 explores the decisions made by the server. There are many clients connected to the server, some requiring assistance and others able to provide it. The server then makes decisions to improve the QoS of each client requiring assistance based on the metrics of all network nodes connected to the server. In this illustration, the architecture displays the three ways in which a client's QoS can be improved: Server Assistance, Node/Client Assistance and Self-Adaptation. Through this architecture, a client will be able to have its QoS improved.

CHAPTER 5 RESULTS

5.1 OVERVIEW

In this results chapter, the distribution aspect of the architecture is investigated. AI and Physics processes are distributed in different ways along varying network conditions. There are many ways of distributing data; the two focused on for these experiments are constant positional updates of objects, which is split into full and partial distribution for both AI and Physics, and waypoint data for objects which is solely for AI. The self-adaptation ability of the client is also investigated. This component of the architecture executes only when there are no resources available on the network.

Following the presentation of the findings of the experimental scenarios, there is a discussion of the results before concluding the chapter.

5.2 EXPERIMENTAL DESIGN

The following sections, 5.3 and 5.4, are experiments run within Unity 3D to show the Distribution (5.3) and Self-Adaptation (5.4) capabilities of the architecture. The Distribution section focuses on the distribution of data as a client is being assisted. AI data and Physics data are distributed separately and in various ways. Each method of distribution is tested against varying network conditions. The Self-adaptation section focuses on the fail-safe of the architecture which executes when there are no resources available to provide assistance and the client receives an RPC informing it to begin the Self-Adaptation process. This section shows each game element separately building up, in object count, over time and its effect on a client, the client would then receive an RPC to self-adapt and reduce that element. Table 5.1 shows the experiments within each section.

Table 5.1. The Experiments as found in Sections 5.3 and 5.4.

Scenario	Experiment	Section
The Distribution of Data Section 5.3	Full Distribution of AI Objects	5.3.2
	Partial Distribution of AI Objects	5.3.3
	Sending Path Data of AI Objects	5.3.4
	Full Distribution of Physics Objects	5.3.5
	Partial Distribution of Physics Objects	5.3.6
Self-Adaptation Section 5.4	AI	5.4.2
	Physics	5.4.3
	Graphics	5.4.4

5.3 THE DISTRIBUTION OF DATA

The aim of this experimental scenario is to show the effect distributing processing has on the client device and to show which method of distribution is best. The processing distributed here is AI and Physics tasks. The AI and Physics tasks are distributed in different ways along varying network conditions. Three forms of processing distribution are investigated for the AI task; full distribution, partial distribution and sending path data, and two forms for the Physics task; full distribution and partial distribution. Full distribution involves the entirety of a task being handed over to another network node. With partial distribution, a portion of the task is handed over. The sending of path data, which is AI specific, has the assisting node calculating a route for each object, placing the points to visit into an array and then transmitting the arrays to the client device. To find the best form of distribution each for AI and Physics, the network over which the distribution takes place will vary in quality. For each form, there are seven different network variations used: an unaffected network, packet loss variations of 5%, 10% and 15% and latency variations of 100ms, 200ms and 300ms. Testing found these variations to yield the greatest difference in results.

5.3.1 EXPERIMENTAL SETUP

An experimental testbed was constructed using the Unity3D game engine to find the best form of distribution for AI and Physics processes. Both the client and the assisting node are running an instance of the game with groups, of the game object being tested, spawning at set intervals on both. Once the client decides to ask for assistance, it will then stop some or all the local processing of that game element and begin to receive data for that element. The data it receives is positional data which, depending on the experiment, is either a single position to visit or an array of positions. Only a network node with enough resources will be selected to provide assistance. Therefore, the assisting node in this situation must have more resources than the client. To create a gap in resources between the client and the assisting node, the client was a deliberately under-resourced Virtual Machine (VM). The specification of the assisting network node is as follows:

- 8GB Memory
- I7-3770/3.4Ghz
- 4 CPU
- Intel HD Graphics

The VM (Client) was created through Virtual Box with the following specification:

- 1024MB Memory
- 1 CPU
- Execution Cap: 50%
- 128MB Video Memory

Testing had shown that by reducing the execution cap to 50% provided enough of a resource gap between the client and the assisting node and therefore the assisting node will have enough resources to help when required. Having this percentage, any higher resulted in the client running high counts of objects before asking for assistance. For example, in the case of full distribution, the assisting node struggled to update the client device with the immediate high number of objects. Having a low execution cap allowed for the client to ask for assistance with a lower number of objects which the node acting as the server could handle.

These experiments contain results for FPS and CPU percentage. FPS is worked out in-game (see Appendix A) while CPU percentage is recorded via MSI Afterburner [85]. GPU percentage usage is

absent from these final results as there was no way to get a reliable result, this could be due to the VM being under-resourced (see Appendix B). There are also test results for varying network conditions so as to have a better understanding of how these distribution methods would perform in the real world. Network quality was varied using the software package Clumsy 0.2 [86] as follows (see Appendix C):

- Packet loss percentage at 5%, 10% and 15%
- Latency at 100ms, 200ms and 300ms

These experiments transmit relatively small amounts of simple data in comparison to modern triple-A games, and so high packet loss percentages and latencies were used to compensate for this. The data transmitted is the position of the object and only a maximum of 500 objects is used.

The three AI distribution methods explored are explained below:

1. Full Distribution (Section 5.2.2): In this case, the server takes over all currently existing AI objects and those still to be created. The client does no AI processing. The server executes the AI calculations and moves the objects accordingly. Through the use of Unity's built-in networking, more specifically the Network Transform Script [87], the movement of the AI object is synced between the assisting node and the client. The server is in full control of the AI. Therefore, any time an object moves on the server its movement will be copied by the client. Instead of the client processing a path for each object to follow, the client simply has each move to a position fed to it by the server (see Appendix D).
2. Partial Distribution (Section 5.2.3): The same approach as above is used, however only objects that are spawned after the distribution is requested will be handled by the server. All AI objects that exist prior to asking for assistance will still be controlled by the client. This method allows for less dependency on the network as some objects are being processed locally, it also produces less strain on the server as fewer objects need to be processed by it.
3. Transmitting Path Data (Section 5.2.4): In this case, once assistance is requested, the client prepares to receive an array of positions to visit for each AI object, both existing and those yet to be created. The assisting node will calculate a path for each AI object currently on screen and then send that path to the client in the form of an array of positions. The client receives this array and has the corresponding object visit each position in the array. In this method the client will be moving exactly like the assisting node, however just slightly behind as the array takes some time to be transmitted to the client (see Appendix E).

The sending of the pathfinding data is similar to the full and half distribution methods. The only difference being the frequency of data transmission. The full and half distribution methods involve constant positional updates which will have an object move only a small number of pixels at a time whereas the sending of pathfinding data involves updating a list of scattered points in the game world that each object must visit. The constant positional updates will be small yet frequent in comparison to the larger but infrequent update to the list of points that each object must visit.

The two Physics distribution methods are as follows:

1. Full distribution (Section 5.2.5): this works in the same way as the AI full distribution experiment. The assisting node handles all processing of all currently existing physics objects and those still to be spawned.

2. Partial Distribution (Section 5.2.6): this works in the same way as the AI half distribution experiment. The assisting node handles all processing of physics objects created after the receiving of the message to provide assistance for the client.

Both AI and Physics have a Full Distribution and Half Distribution approach of distributing data. The Full distribution approach is based on most client-server games in that the server handles the processing of the objects and updates the client on the objects' positions in the game world. With Partial Distribution, the client continues to handle some of the processing. The reasoning for this approach is that it may perform better in comparison to Full Distribution if the network is strained as less information needs to be transmitted from the assisting node to the client, it also reduces the number of resources required on the node. The Transmission of Path Data is another approach for the AI as each AI object will have a path calculated for it by a node before it moves within the game world on the client. This method also provides positional updates for the objects; these are less frequently transmitted however they are larger as each packet will contain a list of multiple points to visit within the game world. The reason for this approach is that it may perform very differently across different network conditions.

5.3.2 FULL DISTRIBUTION OF AI OBJECTS (EXPERIMENT 1)

In this experiment, ten AI objects are spawned every second up to a maximum amount of 500 objects. Empirical testing had shown this spawn rate to place enough pressure on the client VM to affect its performance and not too much as to create periods of "hanging" where nothing is happening. The maximum object count is set to 500 as experimentation had shown the assisting node, when distribution occurs, to be able to handle this maximum. Increasing this further showed a decline in results as the node struggled to process the objects and transmit the relevant data.

HOW A SYSTEM REACTS WITH NO DISTRIBUTION

A baseline was established for reference in this experiment for FPS and CPU usage. Figure 5.1 shows the FPS on the client processing in-game data itself with no assistance. As can be seen, the FPS falls before beginning to level out to an average of 50FPS. The FPS is seen to vary quite a bit due to the low resource availability of the system not being able to cope immediately with the spawning objects.

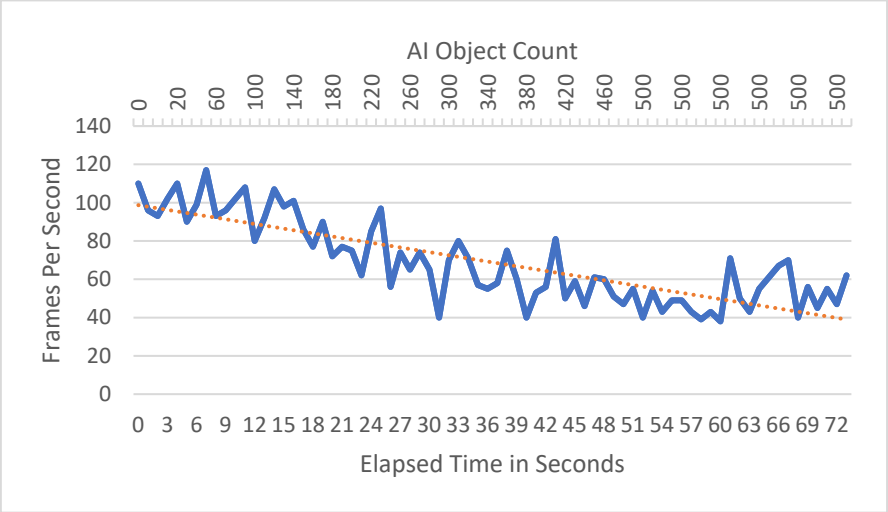


Figure 5.1. FPS against an increasing number of AI objects and increasing elapsed time (No Distribution)

Figure 5.2 shows the CPU percentage usage against the number of AI objects.

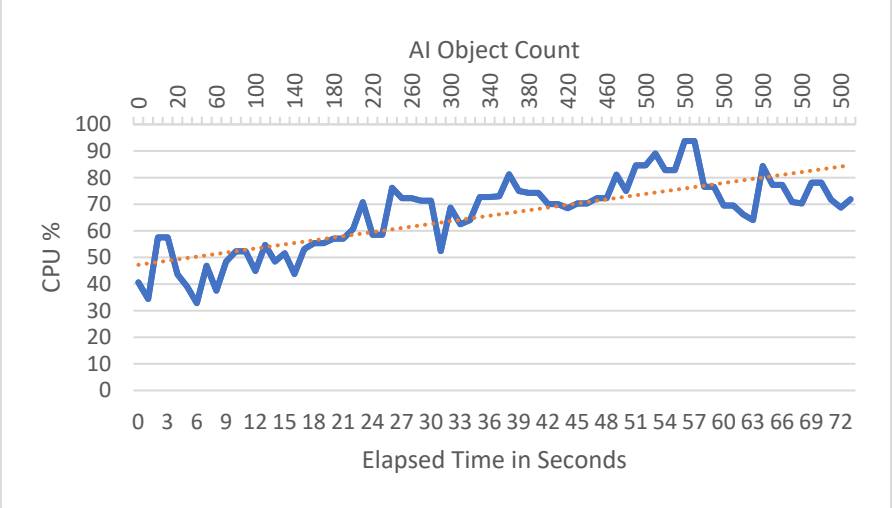


Figure 5.2. CPU against an increasing number of AI objects and increasing elapsed time (No Distribution).

In both Figure 5.1 and Figure 5.2 it can be seen that if the AI object count were to increase any further than 500 then both the FPS and CPU usage would suffer. Both data series level out as the experiment remains at 500 objects, however the linear regression trendline predicts both will continue to decline.

EFFECT OF DISTRIBUTION ON AN UNAFFECTED NETWORK

It can be expected that by distributing AI processing, there should be an improvement in both the FPS and the CPU percentage usage. Figure 5.3 shows the FPS when processing is distributed over an unaffected network. For the client to ask for assistance the average FPS of the previous 3 seconds must be below 70FPS. 70FPS was chosen as the limit based on Figure 5.1, if a lower limit were chosen there would be less likelihood of assistance being requested. A moving average is taken over 3 seconds to smooth out the variations in the graph, the window allows enough time for a spike in FPS to settle. At times, there are random drops in FPS for only a second, taking an average allows for these to occur without penalty. If the action was taken based on one FPS value, then there is the possibility that an assisting node is being employed for no worthwhile reason. Figure 5.3 shows that with distribution, even though the number of AI objects on the client's screen is increasing, the FPS can remain at an acceptable level. Unfortunately, as the number of objects increases the FPS does not remain above 70FPS. However, it remains higher than a client which does not receive assistance (Figure 5.1). In this case, distribution occurred at 20 seconds where the average FPS over a 3 second period fell below the limit of 70FPS.

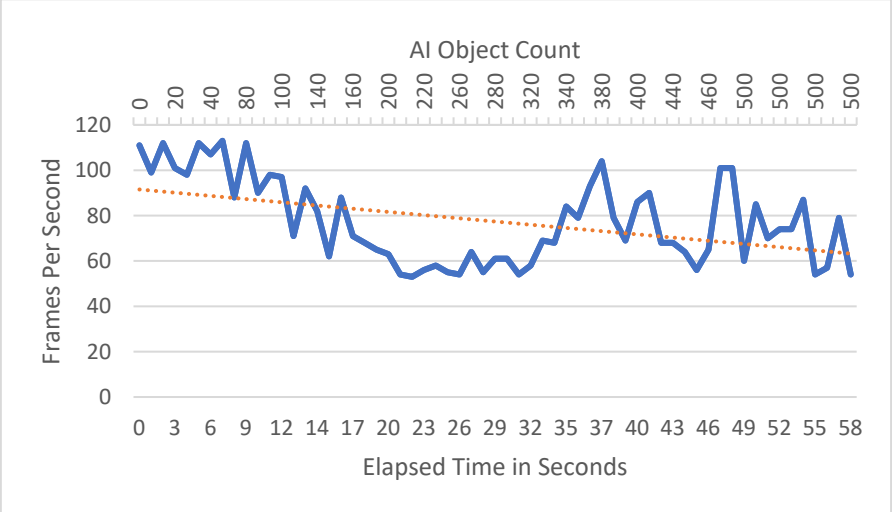


Figure 5.3. FPS against an increasing amount of AI objects. Distribution occurs at 20 seconds.

Figure 5.4 shows the CPU percentage when distribution occurs. As can be seen, around the time of distribution there is no immediate improvement, however, the CPU peaks at 100% for around 10 seconds. This plateau in CPU usage could be due to the handover in processing occurring and the under-resourced VM taking time to deal with the change. The CPU percentage usage then begins to

drop and remains around 60%. This is in no way a dramatic improvement over no distribution, but an improvement nonetheless as the CPU remains at a lower percentage usage.

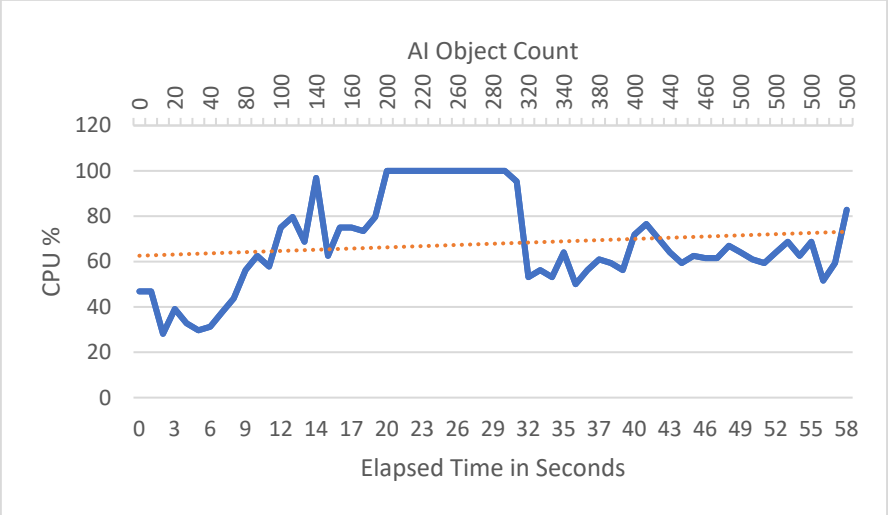


Figure 5.4. CPU against an increasing amount of AI objects. Distribution occurs at 20 seconds.

From comparing Figures 5.1 and 5.2 and Figures 5.3 and 5.4, it can be seen that by distributing processing to another node there is a marked improvement on the client. The data series of Figures 5.3 and 5.4 show the improvement that can be made when processing is handled by another network node. The trendlines on each are also an improvement as they do not predict as large a decline.

EFFECT OF DISTRIBUTION ON FPS OVER AN INCREASING PACKET LOSS

This method of distribution was also tested over six different network variations: three different packet loss percentages (5%, 10% and 15%) and three different latencies (100ms, 200ms and 300ms). Figures 5.5, 5.6 and 5.7 show the effect an increasing packet loss percentage has on the clients FPS.

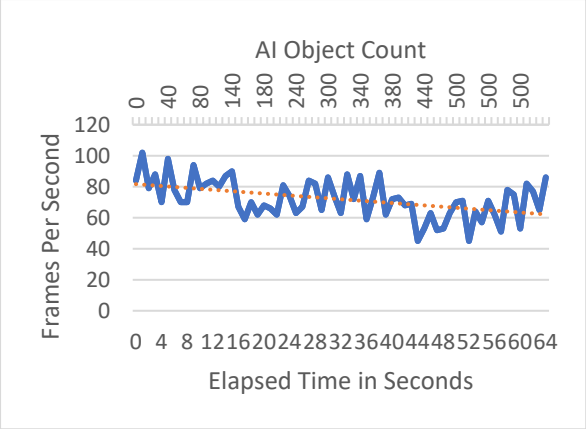


Figure 5.5. FPS when distributing over a network with 5% Packet Loss. Distribution occurred at 18 seconds.

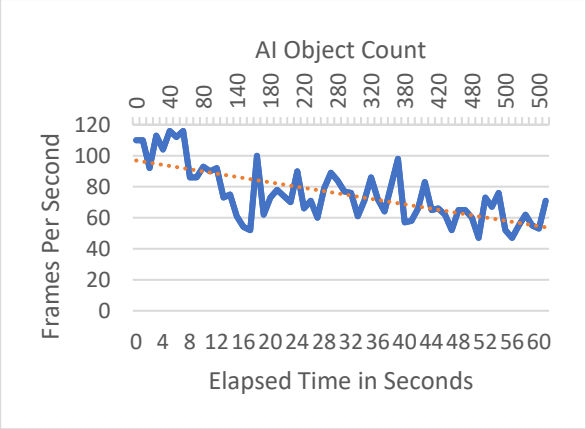


Figure 5.6. FPS when distributing over a network with 10% Packet Loss. Distribution occurred at 16 seconds.

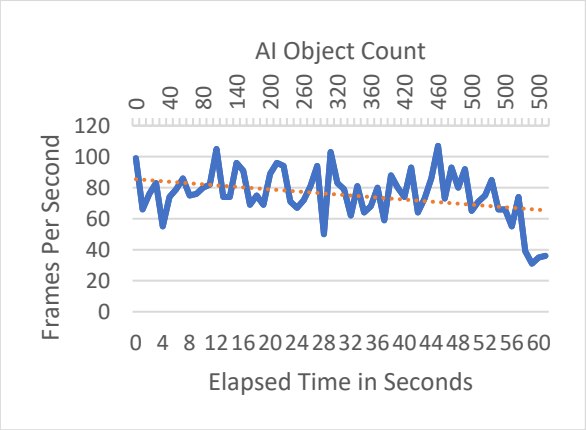


Figure 5.7. FPS when distributing over a network with 15% Packet Loss. Distribution occurred at 7 seconds.

In Figure 5.5, with a 5% packet loss, distribution occurs at 18 seconds as the average FPS over 3 seconds falls below the 70FPS limit. It can be seen that there is a slight improvement in FPS, for a time after 18 seconds. However, it seems to dip and then improve again around 50 seconds. This second dip could be due to data being received after being lost and the improvement is due to the maximum number of objects having been spawned. In Figure 5.6, with a 10% packet loss, distribution occurs at 16 seconds. Similarly, to the 5%, there is a slight improvement of just under 10FPS initially. Comparing this to the 5% FPS graph, no second improvement is viewed. The FPS towards the end of this experiment lies flatter on average in comparison to its 5% counterpart. Figure 5.7 shows the FPS with a 15% packet loss, in this case, distribution occurs at 7 seconds. Assistance begins a lot earlier here as a sharp drop in FPS brought the 3 second average below 70FPS, this drop could have been due to the VM struggling with the spawn of AI objects. The FPS remains higher than its counterparts at 5% and

10% Packet Loss as the trendline is more centred at roughly 80FPS. This high FPS could be due to the game not having to receiving as much data data. Therefore, it is not having to move as many objects which may be causing a higher FPS than normal. The fall in FPS at the end of the graph signifies the end of the experiment as programs are shut down. Similarly, to Figures 5.5 and 5.6, an initial improvement is seen once distribution occurs which then declines again after a short time.

These trends show that by increasing the packet loss percentage of the network, the average overall FPS will increase, however, this is due to data not being received and the client not moving as many objects. Less processing will lead to a greater FPS value. Regardless of the packet loss, the results show that once distribution occurs each client instance is seen to have an improvement in FPS. However, it must be taken into consideration that if packet loss is too high, then inaccurate results will follow. The decision-making ability of this architecture will ensure that if the packet loss is too high, then distribution will need to occur elsewhere (Figure 3.2). Packet Loss percentage is one of the metrics utilised in this architecture and helps form a decision on where distribution will take place.

EFFECT OF DISTRIBUTION ON CPU USAGE OVER AN INCREASING PACKET LOSS

Figures 5.8, 5.9 and 5.10 below show the effect an increasing packet loss percentage has on the clients' CPU usage.

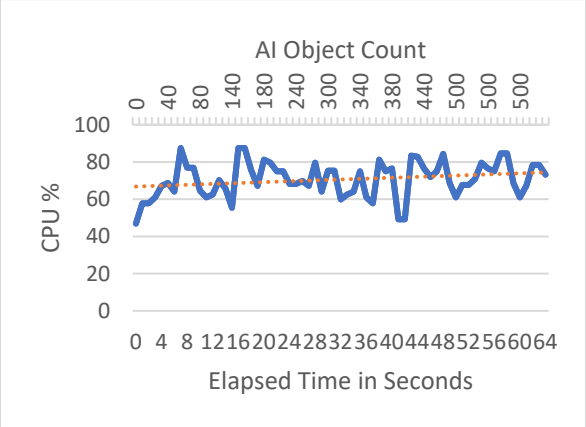


Figure 5.8. CPU % when distributing over a network with 5% Packet Loss. Distribution occurred at 18 seconds.

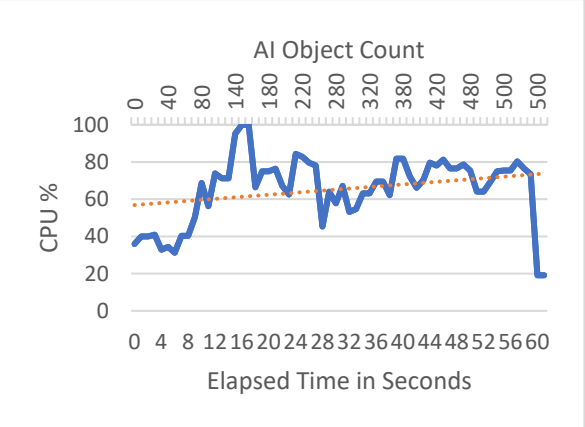


Figure 5.9. CPU % when distributing over a network with 10% Packet Loss. Distribution occurred at 16 seconds.

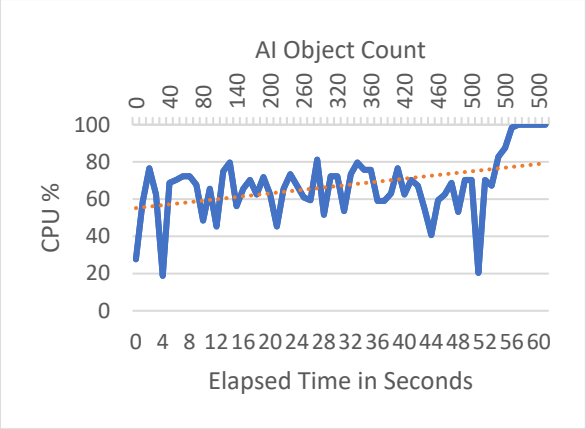


Figure 5.10. CPU % when distributing over a network with 15% Packet Loss. Distribution occurred at 7 seconds.

In Figure 5.8, with a 5% packet loss, distribution occurs at 18 seconds. The CPU, a few seconds after distribution appears to reduce in usage. Shortly before the second dip in FPS (Figure 5.5 at around 40 seconds), the CPU usage appears to increase slightly again. The trend shows that shortly after distribution there is an improvement in CPU usage which is then, after another short period, cancelled out at the usage increases again. The average CPU usage here is around 70%. In Figure 5.9, with a 10% packet loss, distribution occurs at 16 seconds. The CPU peaks at 100% around the time of distribution, due to the handover in processing, and then falls to below 80% for the remainder of the experiment, the fall at the end of the graph is the end of the experiment. The CPU peak in usage causes the large drop in FPS in Figure 5.6. Shortly after distribution occurs, the CPU usage improves in a similar fashion to the 5% packet loss. However, shortly after this improvement, there is an increase in the usage again between 28 seconds and 40 seconds with the usage then remaining around 80% for the rest of the experiment. Figure 5.10, with a 15% packet loss, begins distribution at 7 seconds. The CPU varies mostly between 50% and 80% for this experiment and roughly mirrors the FPS graph as in some cases when the FPS dips the CPU peaks as expected. There is a peak of 100% usage at the end of the graph signifying the end of the experiment and programs being shut down. Comparing these results to the 5% and 10% packet loss instances, there is an improvement once distribution occurs, however, the usage is slightly lower on average in Figure 5.10 due to the lesser amount of data received by the client, this gives a false positive.

These trends in CPU usage show that by increasing packet loss percentage, there will be an increasing variation in CPU usage as seen by the difference between Figure 5.8 and 5.10. There is a general increase in usage between the 5% and 10% results as the client struggles with the start stop nature of the object data and attempts to maintain a high FPS. This usage then decreases as the packet loss percentage increases as less data is being received and therefore less processing is occurring client-

side. The overall increase in packet loss sees an increase in the variation of CPU percentages; this could be due to the start-stop nature of the objects as sometimes data is received to move them and others it is not. Regardless of the packet loss, once distribution occurs each client instance is seen to have an improvement in CPU usage. The decision-making ability of this architecture will ensure that if the packet loss is too high, then distribution will need to occur elsewhere as packet loss is a metric which helps make the decision of where distribution will take place.

EFFECT OF DISTRIBUTION ON FPS OVER AN INCREASING LATENCY

Another network condition tested is latency. Latency is the length of time information takes to get from point A to B along a network. In relation to QoE, this delay would reduce the experience for the player as it increases as actions take longer to perform. For example, a player hits a wall to knock it down. If there is a low latency, then the wall will fall immediately, yet if there is a high latency then the wall will fall after a delay. With the reduction in latency comes an increase in game immersion. The latency variations experimented on were 100ms, 200ms and 300ms as empirical testing found this variation to provide the greatest difference in results. Figures 5.11, 5.12 and 5.13 show the effect an increasing latency has on the client's FPS.

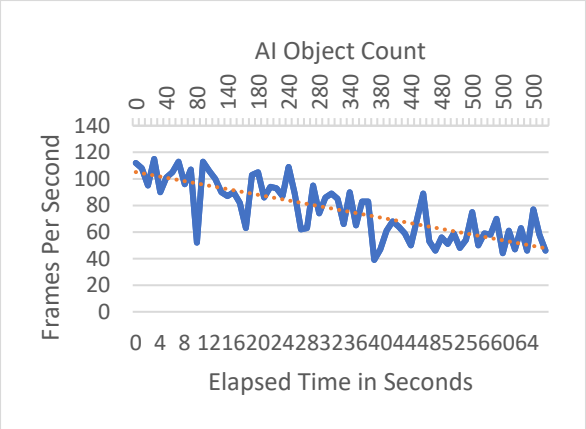


Figure 5.11. FPS when distributing over a network with 100ms latency. Distribution occurred at 40 seconds.

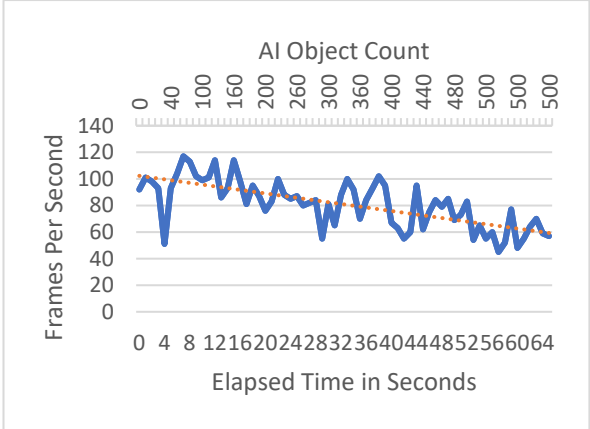


Figure 5.12. FPS when distributing over a network with 200ms latency. Distribution occurred at 32 seconds.

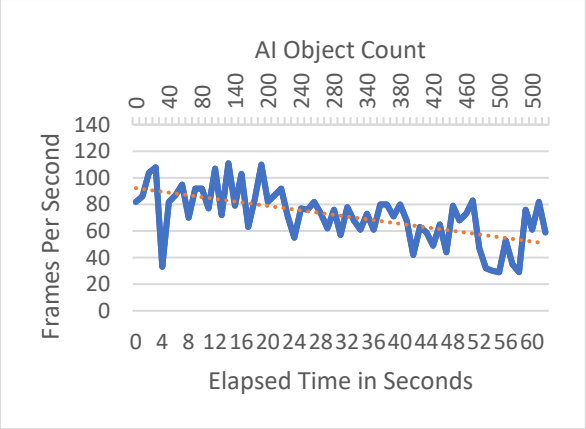


Figure 5.13. FPS when distributing over a network with 300ms latency. Distribution occurred at 7 seconds.

In Figure 5.11, with a 100ms latency, distribution occurred at 40 seconds. In this case, there is no improvement in the average FPS of the client device. Around the 45 second mark, there is a peak improvement over 80FPS which is followed shortly after by smaller peaks, however overall the distribution halts the decline in FPS. In Figure 5.12, with a 200ms latency, distribution occurred at 32 seconds. In this case, distribution causes the overall decline in FPS to slow for a short period of time as it peaks above 100 FPS on two occasions, however, eventually the FPS begins to decline further. Towards the end of the experiment there is a slight increase in the average FPS, again this could be due to the end of the spawning of the AI objects. Figure 5.13 shows how the FPS reacted to a 300ms latency when distribution occurred at 7 seconds. Once distribution occurred there is a slight improvement in the FPS as seen from the trend. However, this then drops before levelling out and finally dropping again.

Similar to packet loss, an increasing latency during the distribution process will eventually lead to a high FPS value. This value, when seen with a high latency or packet loss, is a false positive. Lesser amounts of data are being processed by the client device leading the device to perform better and therefore provide a higher framerate. The immersion of the game would be affected as some objects would not be moving.

EFFECT OF DISTRIBUTION ON CPU USAGE OVER AN INCREASING LATENCY

Figures 5.14, 5.15 and 5.16 show the effect an increasing packet loss percentage has on the clients' CPU usage.

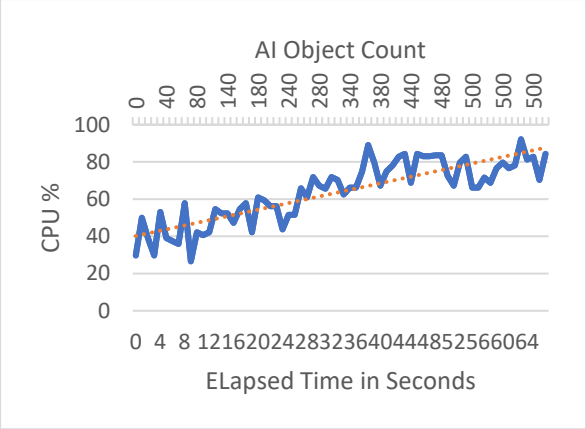


Figure 5.14. CPU % when distributing over a network with 100ms latency. Distribution occurred at 40 seconds.

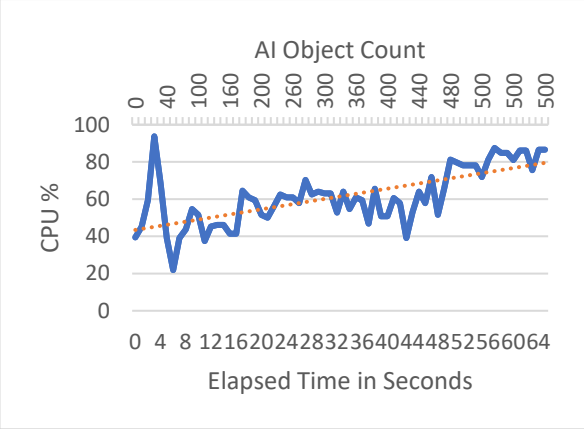


Figure 5.15. CPU % when distributing over a network with 100ms latency. Distribution occurred at 32 seconds.

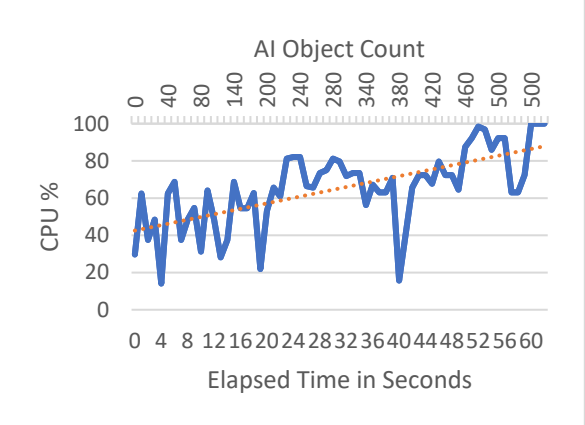


Figure 5.16. CPU % when distributing over a network with 100ms latency. Distribution occurred at 7 seconds.

In Figure 5.14, with a 100ms latency, distribution took place at 40 seconds. As with the FPS counterpart of this experiment, Figure 5.11, distribution halts the increase in CPU usage. In Figure 5.15, with a 200ms latency, distribution occurred at 32 seconds. The CPU usage here is roughly level for a time before increasing at around 45 seconds. This increase is mirrored in the FPS graph of Figure 5.12 as the FPS decreases. Figure 5.16 shows how the CPU reacted to a 300ms latency and distribution occurred at 7 seconds. CPU usage does not improve with the distribution of the AI processing as it increases over time to peak at 100% usage at 52 seconds. The final peak of 100% usage at the end of the graph signifies the end of the experiment and programs being shut down.

These trends show that by increasing the latency of a network, the variation in CPU usage will rise significantly as seen by comparing Figure 5.14 and 5.16. Overall there is a general increase in the CPU

usage, this could be due to the nature of the data transmission as objects may or may not receive information on where to move to next.

DISCUSSION

By fully distributing the AI, there is a good increase in FPS viewed along a good network connection. The CPU usage improves too. By creating poorer network conditions, it can be seen that these improvements diminish. With the poorest of network conditions, the FPS hits low peaks unacceptable to users as does the CPU. Both packet loss and latency are metrics used within the proposed architecture with both helping to make the decision of where to distribute processing. In the case of these experiments, the 5% packet loss and 100ms could, separately, be the upper limit of what is allowed of a connection to distribute processing over. A combination of these would yield worse results as when sending information over a network affected separately with these, there was a slight improvement on the client.

5.3.3 PARTIAL DISTRIBUTION OF AI OBJECTS (EXPERIMENT 2)

In this experiment 10 AI objects are spawned every second up to a maximum amount of 500 objects. Empirical testing had shown this spawn rate to place enough pressure on the client VM to affect its performance and not too much as to create periods of "hanging" where nothing is happening. The maximum object count is set to 500 as testing had shown the assisting node, when distribution occurs, to be able to handle this maximum. Increasing this further showed a decline in results as the assisting client struggled to process the objects and transmit the relevant data.

The difference between this experiment and experiment 1 lies in the fact that a smaller amount of data is distributed. In this case, only the objects spawned after assistance is requested will be handled by the assisting node. It is believed that this method of distribution will perform better under poorer network conditions as less information is being transmitted. As seen from the full distribution of AI objects the network affected the results in a significant way. It is predicted that with less information to send, the FPS and CPU levels will not be affected as much. However, it also must be taken into consideration that some objects will still be handled locally which will influence the FPS and CPU of the client in comparison to if those objects were processed elsewhere.

EFFECT OF PARTIAL DISTRIBUTION ON AN UNAFFECTED NETWORK

Figures 5.17 and 5.18 shows the half distribution of the AI objects over an unaffected network. In this instance, distribution occurred at 39 seconds. The FPS in Figure 5.17 shows no increasing improvement with the assistance from another node and, in comparison to the FPS of the Full Distribution over a good network (Figure 5.3), the partial distribution performs poorly. It could be argued, based on Figure

5.17, that there is a slowing down in the rate at which FPS falls once distribution occurs. For the first 39 seconds of the experiment, the FPS falls by roughly 50FPS and over the following 10-15 seconds, which are when distribution occurred, the FPS falls by around 5-10FPS. There is no increasing improvement, however, it can be seen that partial distribution has a smaller positive effect. The CPU also shows no improvement after distribution as from this point the CPU usage increases. This increase could be due to the multitasking that the client now comes under as some AI objects have a path processed for them locally while others are moved via positional updates from another node.

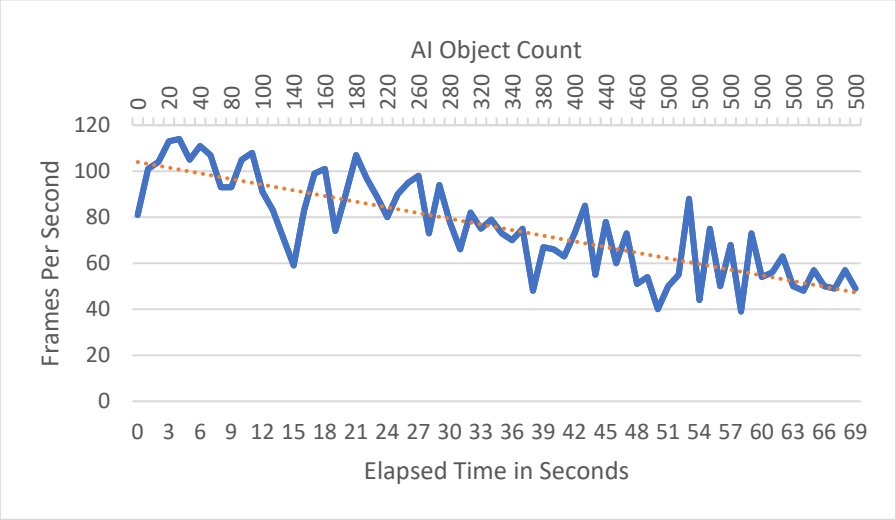


Figure 5.17. FPS with Partial Distribution over an unaffected network. Distribution occurred at 39 seconds.

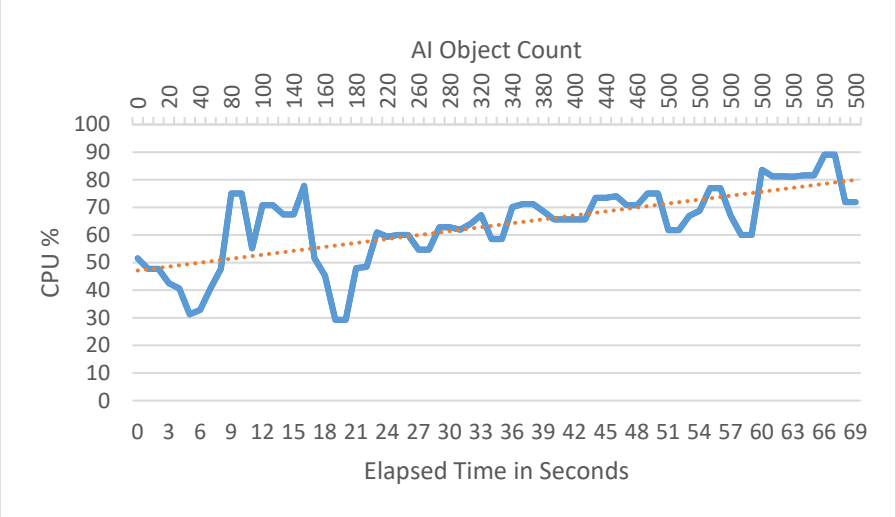


Figure 5.18. CPU % with Partial Distribution over an unaffected network. Distribution occurred at 39 seconds.

This method of distribution was also tested over six different network variations: three different packet loss percentages (5%, 10% and 15%) and three different latencies (100ms, 200ms and 300ms). Figures 5.19, 5.20 and 5.21 show the effect an increasing packet loss percentage has on the clients FPS. Although partial distribution was seen to have no increasing improvement over an unaffected network, there is a small positive effect in the form of a slower decline. It is predicted that, over poorer network conditions, this method's slower decline may outperform the full distribution method.

EFFECT OF PARTIAL DISTRIBUTION ON FPS OVER AN INCREASING PACKET LOSS

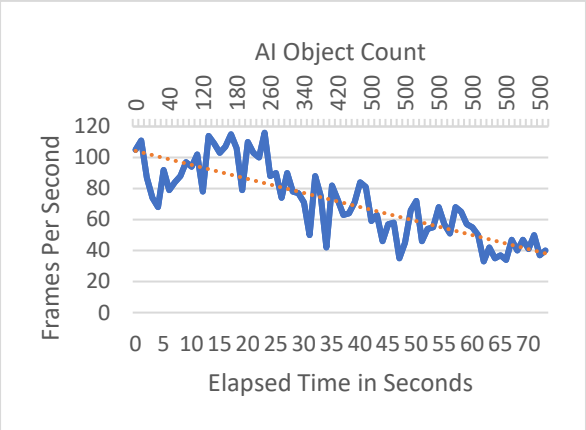


Figure 5.19. FPS with Partial Distribution over a network with 5% Packet Loss. Distribution occurred at 32 seconds.

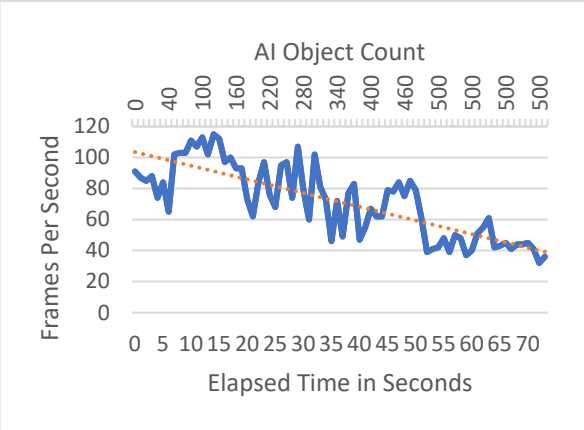


Figure 5.20. FPS with Partial Distribution over a network with 10% Packet Loss. Distribution occurred at 36 seconds.

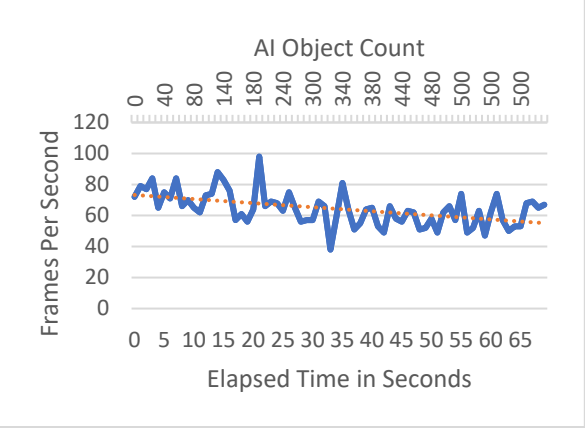


Figure 5.21. FPS with Partial Distribution over a network with 15% Packet Loss. Distribution occurred at 11 seconds.

In Figure 5.19, with a 5% packet loss, distribution took place at 32 seconds. Once distribution occurred, on average, the FPS began to level out before dropping a final time to around 40 FPS. For this distribution method, it was expected that, based on the unaffected network results, there would not

be much improvement due to the variety of information having to be processed by the client device. In Figure 5.20, with a 10% packet loss, distribution occurred slightly later at 36 seconds. With this packet loss variant, there is no improvement or even levelling out of the FPS count once distribution occurs as seen by the trend. In Figure 5.21, with a 15% packet loss, distribution happened at 11 seconds which is much earlier in comparison to the other packet loss percentages. This variation sees the FPS remain around 60 FPS shortly after distribution occurs. Similar to previous experiments at the highest end of the packet loss or latency spectrums tested, Figure 5.21 shows a better FPS overall, this is a false positive as less data is being received by the client in comparison to the results of Figures 5.19 and 5.20. As less data is being received the client device is having to process less information and therefore providing a higher FPS. Evidence of this in-game was that few of the AI objects were moving at all as distribution occurred early on only some were being processed locally.

Comparing Figures 5.19 and 5.20 with Figures 5.5 and 5.6, which are results from the full distribution experiment, the full distribution method is seen to perform better for FPS. Once AI objects have stopped spawning, the full distribution method shows a levelling out of the FPS in both cases whereas partial distribution shows a continuing decline. Both Figures 5.7 and 5.21 show that 15% packet loss has too great an effect.

EFFECT OF PARTIAL DISTRIBUTION ON CPU USAGE OVER AN INCREASING PACKET LOSS

Figures 5.22, 5.23 and 5.24 show the effect an increasing packet loss percentage has on the client's CPU usage.

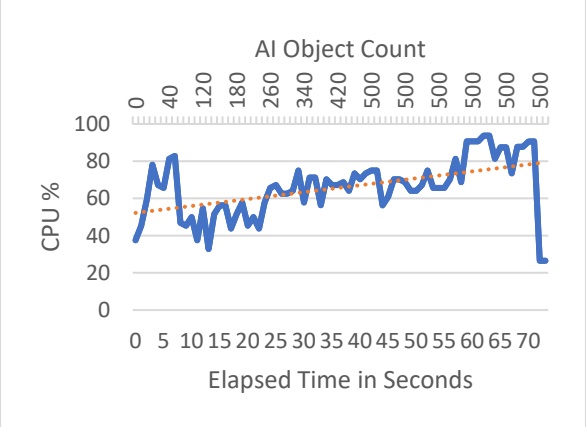


Figure 5.22. CPU % with Partial Distribution over a network with 5% Packet Loss. Distribution occurred at 32 seconds.

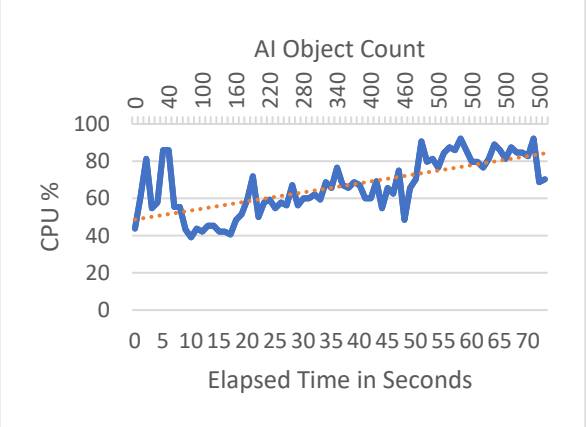


Figure 5.23. CPU % with Partial Distribution over a network with 10% Packet Loss. Distribution occurred at 36 seconds.

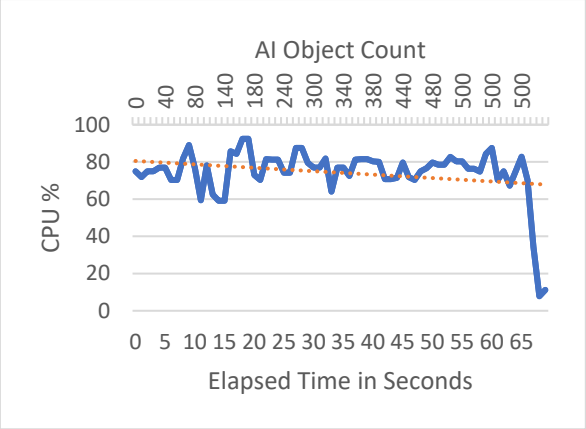


Figure 5.24. CPU % with Partial Distribution over a network with 15% Packet Loss. Distribution occurred at 11 seconds.

In Figure 5.22, with a 5% packet loss, distribution occurred at 32 seconds. Once distribution occurred the data series shows the CPU usage levelling out, this is reflected in Figure 5.19 as the FPS levels out aswell. However, this usage then increases to over 90% on average and the FPS drops again as seen in Figure 5.19 due to the increase in the number of objects requiring positional data from the assisting client. The sharp drop at the end of the graph is the game client being stopped and therefore the CPU not being required for game processing. In Figure 5.23, with a 10% packet loss, distribution occurred at 36 seconds. The CPU usage is seen to level out for around 15 seconds after distribution before increasing similarly to Figure 5.22; this would also be due to the increased number of objects receiving information from the assisting client. In Figure 5.24, with a 15% packet loss, distribution occurred at 11 seconds. Similarly, to the FPS, the CPU usage varies little throughout this experiment. Only a small amount of AI objects are processed locally and a lot of data is being lost due to the high packet loss percentage. Therefore, there is little variation.

The trends in these experiments show that by increasing the packet loss percentage there is a decreasing chance of distribution improving the FPS or CPU usage. As seen in the 5% and 10% packet loss experiments, the distribution of some of the AI objects slowed both the decrease in FPS and increase in CPU usage. Nonetheless, this effect is lessened as the packet loss percentage increases as the FPS continues to decrease and CPU usage increase regardless of distribution.

Comparing these results to their full distribution counterparts, Figures 5.8-5.10, there is similar performance for a period of time. The full distribution causes the CPU usage to level out for the remainder of the experiment, however, the partial distribution levels out the CPU usage for a short amount of time before it increases towards the end of the experiment as seen by the data series. With regards to an increasing packet loss, the full distribution method performs better.

EFFECT OF PARTIAL DISTRIBUTION ON FPS OVER AN INCREASING LATENCY

Another network condition tested is latency. The latency variations used were 100ms, 200ms and 300ms. Figures 5.25, 5.26 and 5.27 show the effect an increasing latency has on the clients FPS.

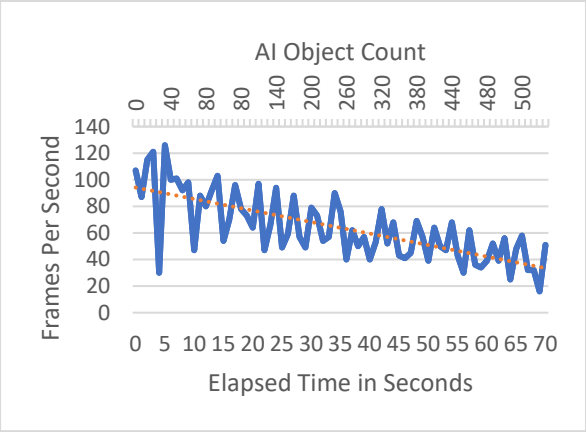


Figure 5.25. FPS when distributing over a network with 100ms latency. Distribution occurred at 23 seconds.

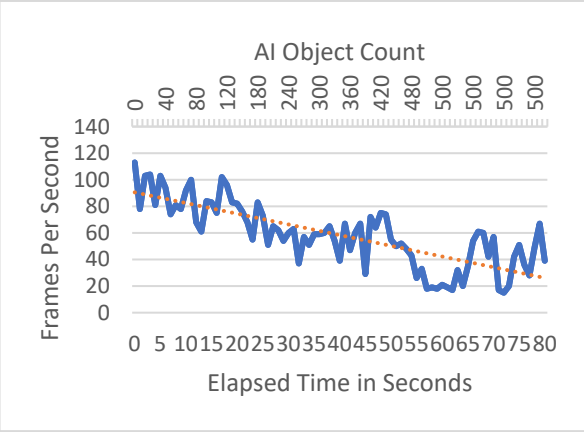


Figure 5.26. FPS when distributing over a network with 200ms latency. Distribution occurred at 25 seconds.

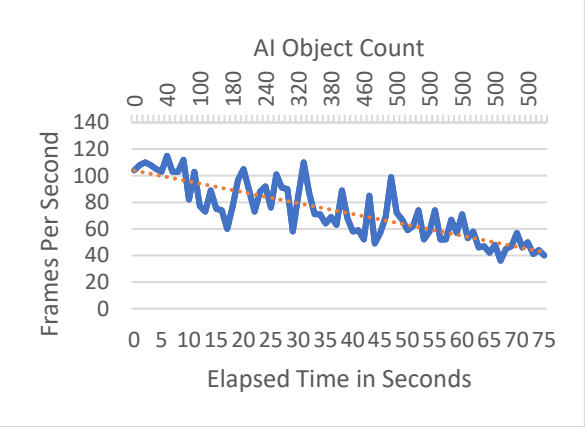


Figure 5.27. FPS when distributing over a network with 300ms latency. Distribution occurred at 18 seconds.

In Figure 5.25, with a latency of 100ms, distribution occurred at 23 seconds. As seen by the trend, distributing the data caused no change in the declining FPS rate. This graph shows a very varied FPS and is a good example of why the FPS is calculated from an average. At the 5 second mark, there is a large drop in FPS for only a second, if distribution occurred based on a single second's value, then distribution would have occurred here. In Figure 5.26, with a latency of 200ms, distribution occurred at 25 seconds. As with the previous experiment in Figure 5.25, distribution had no effect on the FPS as it continued to decline. Towards the end of the experiment the FPS, on average, show signs of some

improvement. In comparison to Figure 5.25, the results of Figure 5.26 are over a longer period, and therefore the FPS may have improved as there was no more extra incoming information. In Figure 5.27, with a latency of 300ms, distribution occurred at 18 seconds. Similarly, to the previous experiments, distribution had no effect on the declining FPS. Even though distribution occurred earlier, there were larger amounts of data being received which had an adverse effect on FPS.

These trends show that partial distribution does not improve the FPS over a network affected by latency. Even the lowest latency tested showed that this distribution method has no effect on the FPS as it continues to decline. The latency will have affected the time taken for data to arrive at the client. Interestingly, had the first tested latency experiment been left to run longer, there may have been an improvement seen at the end as the client had no more extra data incoming. This conclusion can only be drawn as with the 200ms latency experiment there was an improvement in FPS at the end as the maximum number of AI objects had been spawned for some time.

Comparing these results to the full distribution method, Figures 5.11 – 5.13, overall the partial distribution method is out performed. Interestingly, Figure 5.26 sees a more level FPS for a period of time before falling again which is similar to the full distribution result in Figure 5.12. However, the following sharp decline in FPS in Figure 5.26 at around 50 seconds shows that partial distribution performs at a lower standard in comparison to full distribution.

EFFECT OF PARTIAL DISTRIBUTION ON CPU USAGE OVER AN INCREASING LATENCY

Figures 5.28, 5.29 and 5.30 show the effect an increasing latency has on the clients' CPU usage.

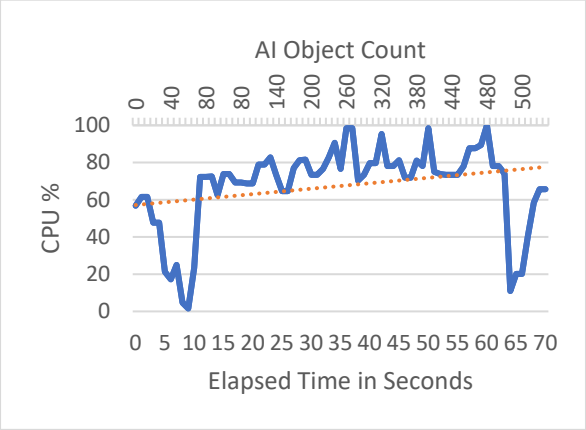


Figure 5.28. CPU % with Partial Distribution over a network with 100ms latency. Distribution occurred at 23 seconds.

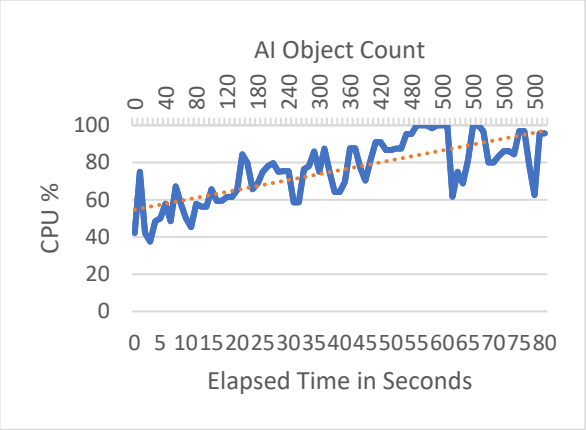


Figure 5.29. CPU % with Partial Distribution over a network with 200ms latency. Distribution occurred at 25 seconds.

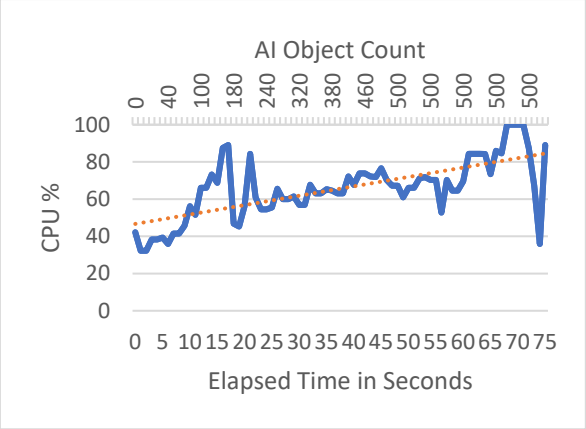


Figure 5.30. CPU % with Partial Distribution over a network with 300ms latency. Distribution occurred at 18 seconds.

In Figure 5.28, with a 100ms latency, distribution occurred at 23 seconds. The CPU usage falls to just above 0% on two occasions, none of which have any correlation with the FPS values. The majority of the experiment sees the CPU remain above 60% usage with several occasions being almost 100%. The drops in CPU usage could be a process outside of the game executable affecting the under-resourced VM. In Figure 5.29, with a 200ms latency, distribution occurred at 25 seconds. Distribution of processing was ineffective in halting the increase in CPU usage in this experiment. On some occasions, there are dramatic drops in CPU usage which, when compared to Figure 5.26, show increases in FPS. However, overall the CPU usage increases to numbers that are not acceptable due to a combination of low resources, latency and the method of partial distribution which, as seen in Figures 5.17 and 5.18, only slows the decline in QoS. In Figure 5.30, with a 300ms latency, distribution occurs earlier than the other experiments at 18 seconds. In this instance, the CPU usage seems to perform its best which results in a better FPS rate aswell. However, as distribution occurred relatively early on there are fewer objects to process locally. This, combined with the poor network conditions resulting in positional information arriving very late leads to a better processing ability as less data is being processed. Towards the end of the experiment it is seen that the CPU usage increases again overall leading to the conclusion that even though less data is being processed locally, the network is affecting the arrival of data and therefore affecting the overall CPU usage.

Comparing these results to the full distribution experiment, Figures 5.14 – 5.16, the full distribution method provides better results. This is especially evident in Figure 5.14 and Figure 5.15 in which the CPU usage is more consistent and lower towards the end of the experiment.

DISCUSSION

By only distributing some of the AI objects there is no increasing improvement to overall FPS or CPU usage even over an unaffected network. The local processing of some objects and the distribution of others is a combination that had little effect on the performance of the client as the decline in QoS was slowed. When compared with Figures 5.3 and 5.4, distributing all the objects will produce better results in comparison to partial distribution. When creating poor network conditions, the best situation that could be hoped for with this method of distribution is that the decrease in FPS and increase in CPU usage is slowed. The lowest packet loss percentage was the only experiment to show a levelling out of the metrics when distribution occurred. As either packet loss or latency increases both the FPS and CPU will drop in performance. At the highest packet loss and latency both FPS and CPU usage seemed to show improvement, however, this was due to data either taking too long to be received and therefore processed or it not being received at all. It is easier for the system to process less data therefore providing better results on paper, however, the immersion of the game would be affected as some objects fail to move.

5.3.4 SENDING PATH DATA OF AI OBJECTS (EXPERIMENT 3)

This experiment is very different to the previous two AI experiments. The previous experiments focused on constant positional updates from the assisting node. This experiment has the assisting node sending the calculated path each AI object has to follow to the client. It was believed here that a less frequent data transmission, although a bigger packet, would perhaps perform better in comparison to a high-frequency transmission of smaller data packets. The assisting node created a path for each AI object to follow and stored each point to go to in an array. The array for each object was then transmitted to the client for it to append it to the bottom of an ever-growing array of points to visit for each AI object.

In this experiment, 10 AI objects were spawned every second, increasing to a maximum of 500 objects. This is the same spawn rate and maximum as the previous experiments. Similar to the full distribution experiment, this experiment also took all processing from the client and handed it off to an assisting node.

Figures 5.31 and 5.32 show the effect the sending of path data has on FPS and CPU usage over a good quality network. In this instance, distribution occurred at 21 seconds. Both graphs show an approach that does not benefit the client. Figure 5.31 presents the FPS of the client. Once distribution occurs, the FPS falls to around 5FPS showing no signs of improving. Similarly, the CPU usage of the client remains very high once distribution occurs, with no sign of improvement. For a long period, the CPU

percentage usage is at 100%. The sudden drop at the end of Figure 5.32 is the end of the experiment and the game executable being closed. This method of distributing data has proven to be too much for the under-resourced client due to the size of the data packets being received with each packet being an array of positions that an AI object must visit. When comparing this to Figures 5.3 and 5.4, it can be seen that small constant positional updates are much easier for an under-resourced client to process than large and less frequent positional arrays which result in low framerates and high CPU usage.

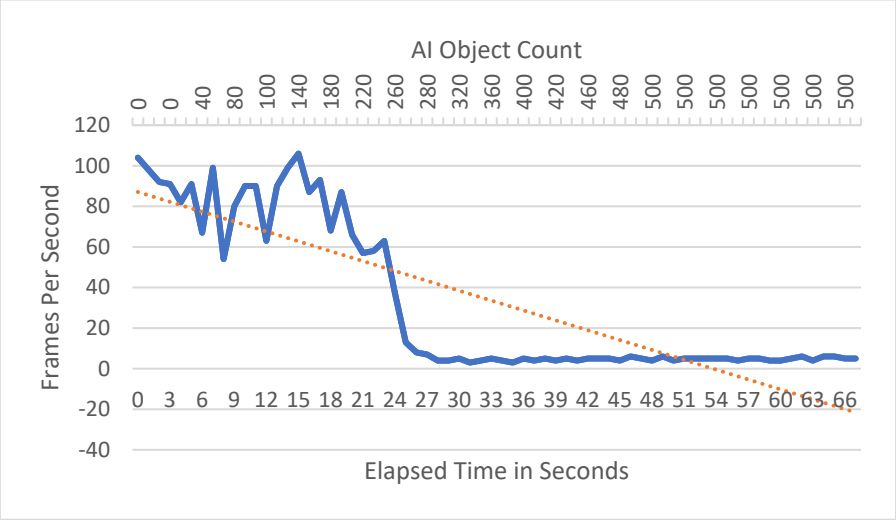


Figure 5.31. FPS when transmitting path data over an unaffected network. Distribution occurred at 21 seconds.

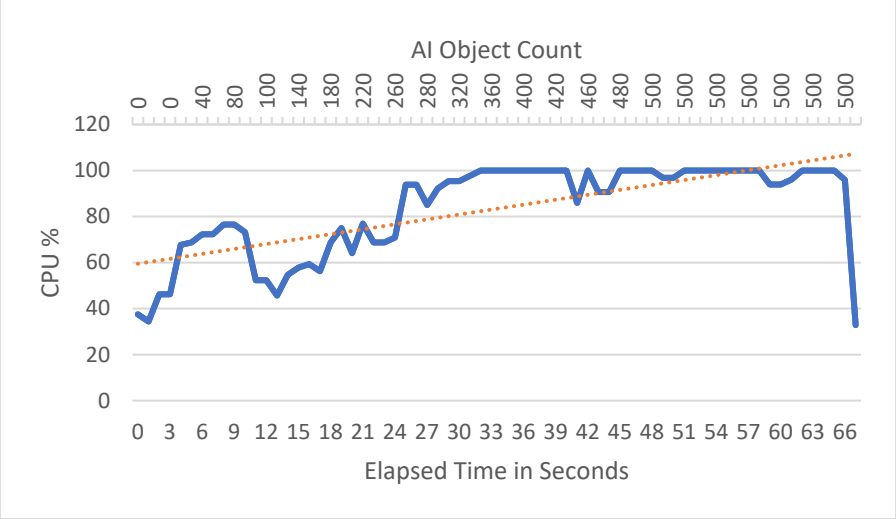


Figure 5.32. CPU % when transmitting path data over an unaffected network. Distribution occurred at 21 seconds.

Based on the results in Figures 5.31 and 5.32, there was no reason to proceed with testing this method over various network conditions.

5.3.5 FULL DISTRIBUTION OF PHYSICS OBJECTS (EXPERIMENT 4)

The approach in this experiment is similar to that of the full distribution of AI Objects, the only difference here being that it is Physics objects that are being distributed. In this experiment, 10 Physics objects are spawned every second up to a maximum amount of 500 objects. Empirical testing had shown this spawn rate to place enough pressure on the client VM to affect its performance and not too much as to create periods of "hanging" where nothing is happening. The maximum object count is set to 500 as testing had shown the client acting as the server, when distribution occurs, to be able to handle this maximum. Increasing this further showed a decline in results as the assisting client struggled to process the objects and transmit the relevant data.

HOW A SYSTEM REACTS WITH NO DISTRIBUTION

Before showing the results of distribution there must first be a good baseline to compare to. Figure 5.33 is the FPS of the client if there was no distribution. As can be seen, the FPS is on a slow decline throughout the experiment. The line itself is full of spikes and dips, these are due to the low resource availability of the system not being able to cope immediately with the spawn of the objects. Figure 5.34 shows the CPU usage during this experiment. As the FPS slowly falls, the CPU usage slowly increases. The CPU graph mirrors the FPS graph, this is expected as when the FPS falls, for example at around 57 seconds, there is a CPU spike that hits 100% usage roughly two seconds before. Distribution is required here to prevent the increasing reduction in FPS and increase in CPU usage.

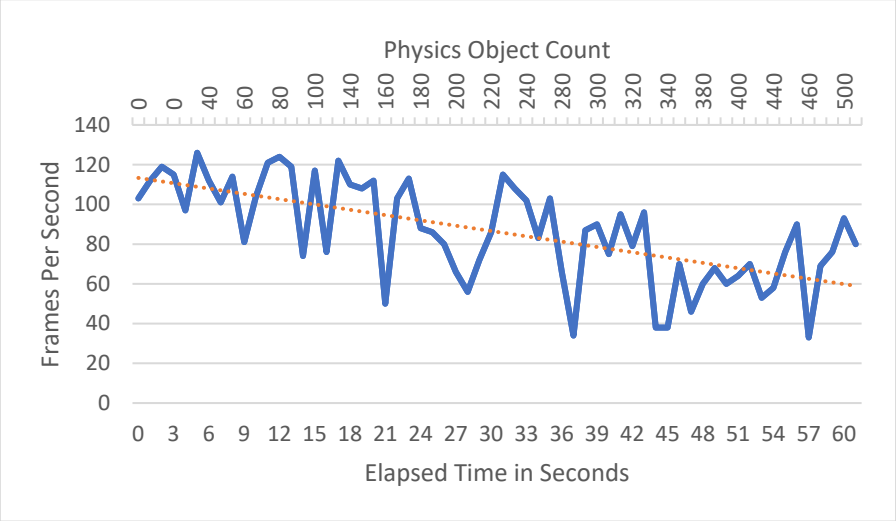


Figure 5.33. FPS against an increasing number of Physics objects (No Distribution)

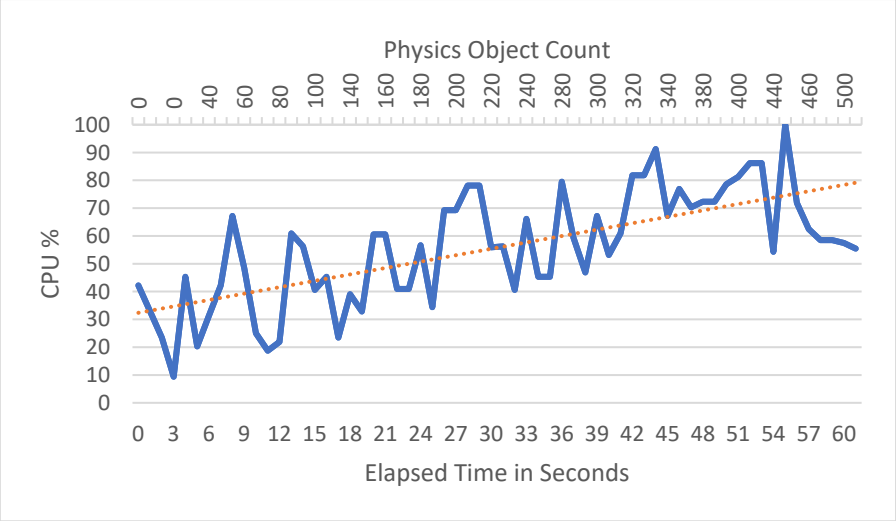


Figure 5.34. CPU % against an increasing number of Physics objects (No Distribution)

Now having seen the condition of the FPS and CPU percentage it can be predicted that by passing off the processing of the Physics there will be an improvement in both the FPS and the CPU percentage usage. This prediction is based on the results from Experiment 1, the Full Distribution of AI in which an improvement was seen when distributing AI processes (Figures 5.3 and 5.4). Figure 5.35 shows the FPS when distribution can occur. This is tested over a university network through which no software is used to affect the network. For the client to ask for assistance, the average FPS of the previous 3 seconds must be below 70FPS.

EFFECT OF DISTRIBUTION ON AN UNAFFECTED NETWORK

Figure 5.35 shows that with distribution, even though the number of Physics objects on the client's screen is increasing, the FPS can remain at an acceptable level. In this case, distribution occurred at 38 seconds where the average FPS fell below 70FPS. With the distribution of Physics data in place, the FPS became less varied and remained between 60-80FPS.

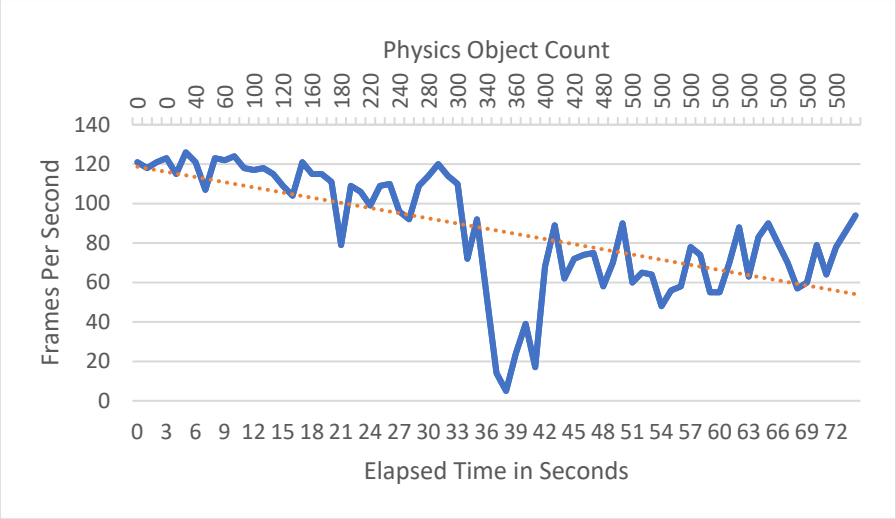


Figure 5.35. FPS against an increasing number of Physics objects. Distribution occurred at 38 seconds.

Figure 5.36 shows the CPU percentage when distribution occurs. As can be seen, once distribution occurs the CPU usage begins to drop to between 60% and 80% usage for the majority of the time. When comparing this to Figure 5.34, the CPU when there is no distribution, the CPU usage is more stable with less variation.

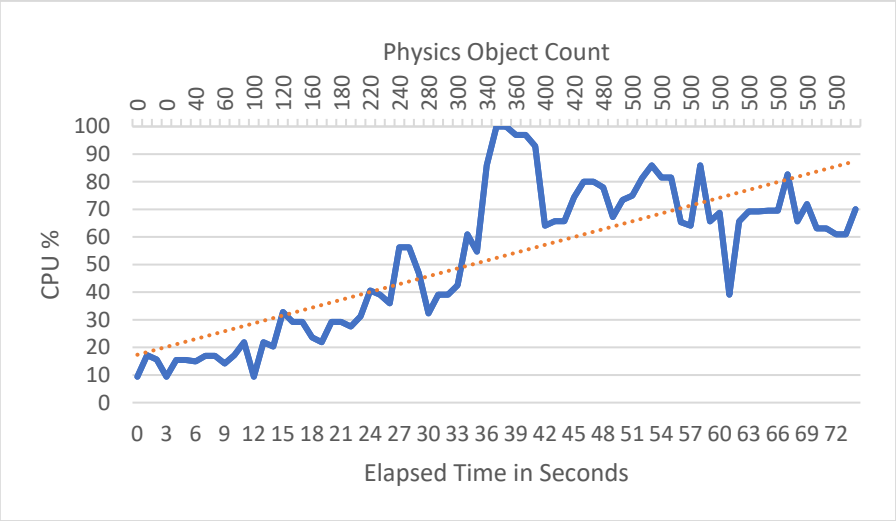


Figure 5.36. CPU % against an increasing number of Physics objects. Distribution occurred at 38 seconds.

Comparing Figures 5.35 and 5.36 and Figures 5.33 and 5.34, there is a noticeable difference when processing of these objects is passed off to another node. As predicted, the client begins to improve on both its frame rate and CPU usage.

EFFECT OF DISTRIBUTION ON FPS OVER AN INCREASING PACKET LOSS

This method of game processing distribution was also tested over six different network variations: three different packet loss percentages (5%, 10% and 15%) and three different latencies (100ms, 200ms and 300ms). Figures 5.37, 5.38 and 5.39 show the effect an increasing packet loss percentage has on the clients FPS.

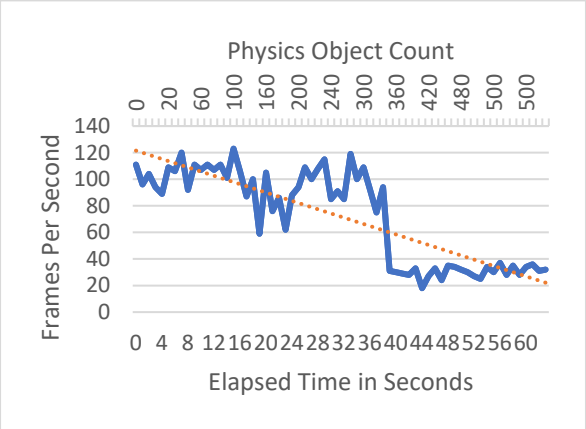


Figure 5.37. FPS when distributing over a network with 5% Packet Loss. Distribution occurred at 40 seconds.

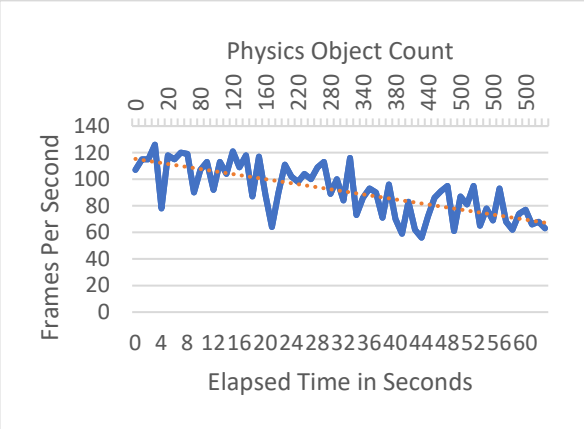


Figure 5.38. FPS when distributing over a network with 10% Packet Loss. Distribution occurred at 44 seconds.

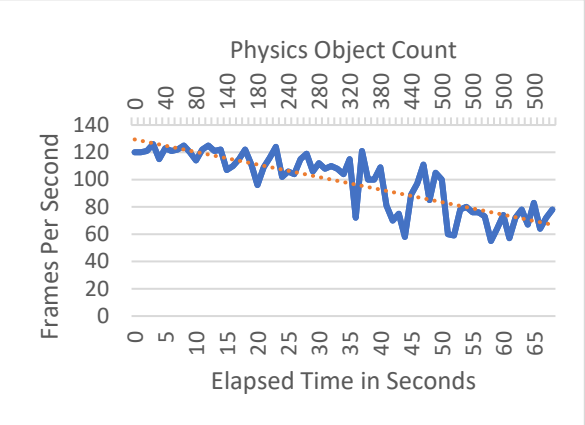


Figure 5.39. FPS when distributing over a network with 15% Packet Loss. Distribution occurred at 45 seconds.

In Figure 5.37, with a 5% packet loss, distribution occurred at 40 seconds. Distribution occurred after a sharp decline in FPS brought the average of the past three seconds under the 70FPS limit. In this instance, distribution did not result in an increase in FPS. Instead, the FPS remained between 20FPS and 40FPS. Changing from local processing to receiving positional data about each object over poor network conditions will have placed too much strain on the client device in this instance, leading to a

low FPS rate. This is a good example of an instance in which there was an unexpected result, which can occur regardless of set-up. Having such a low FPS would negatively affect the QoS and therefore the QoE, therefore the proposed architecture would decide to distribute the data elsewhere provided there are enough resources available and network conditions allow. In Figure 5.38, with a 10% packet loss, distribution occurred at 44 seconds. This variation presents much better results in comparison to Figure 5.37. However, this will be due to the increased packet loss leading to less data being received by the client device which has less processing to do and therefore can produce a higher FPS. A similar effect is shown in Figure 5.39, with a 15% packet loss distribution occurred at 45 seconds. Even less data is received by the client device eventually leading to an increased FPS. Packet Loss is measured within the proposed architecture due to potential results such as these, a high FPS would provide a high QoS and therefore high QoE, however, if there is also a high Packet Loss then this will result in a high FPS. By making sure that the Packet Loss is low, situations such as these false positives can be avoided.

These trends show that even a low packet loss percentage can have a large effect on the client device when receiving data from another network node. Increasing this percentage further leads to a false positive as even though they have higher FPS rates, less data is received by the client device.

EFFECT OF DISTRIBUTION ON CPU USAGE OVER AN INCREASING PACKET LOSS

Figures 5.40, 5.41 and 5.42 show the effect an increasing packet loss percentage has on the clients' CPU usage.

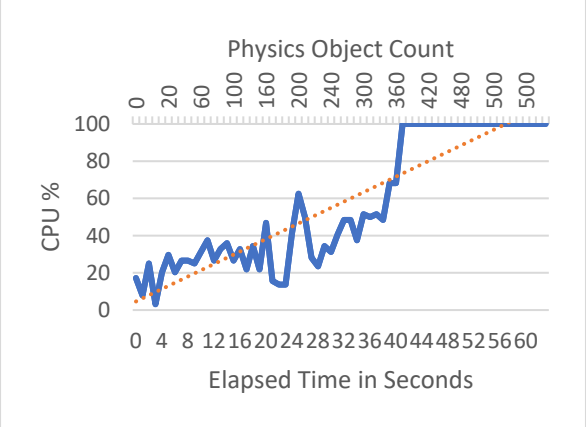


Figure 5.40. CPU % when distributing over a network with 5% Packet Loss. Distribution occurred at 40 seconds.

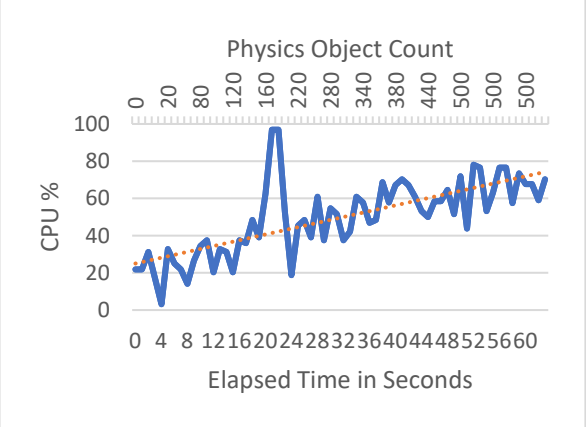


Figure 5.41. CPU % when distributing over a network with 10% Packet Loss. Distribution occurred at 44 seconds.

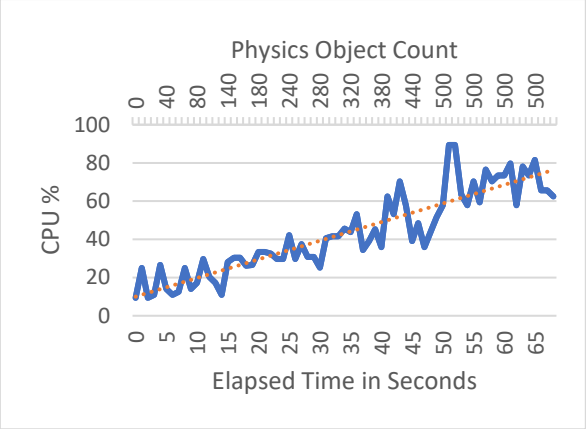


Figure 5.42. CPU % when distributing over a network with 15% Packet Loss. Distribution occurred at 45 seconds.

In Figure 5.40, with a 5% packet loss, distribution occurred at 40 seconds. When there is a single sharp decline in FPS for only a second, this can sometimes reflect in the CPU usage as a small peak. Distribution occurred here because the FPS dropped dramatically enough to cause the average FPS to be below 70FPS. At 41 seconds the CPU usage peaks to 100% and does not move from here. When comparing Figure 5.37 and Figure 5.40 it is seen that the increase in CPU usage is slightly after the drop in FPS. A sudden and large drop in FPS caused distribution which then caused an increase in CPU usage. Similar to Figure 5.37, this is a good example of an unexpected result, and should this occur, the proposed architecture would seek assistance for this client elsewhere. In Figure 5.41, with a 10% packet loss, distribution occurred at 44 seconds. Similarly, to the FPS, these are much better results when compared to the lower packet loss percentage. This is due to the lesser amount of data being received by the client which leads to better processing values. Figure 5.42, with a 15% packet loss, sees distribution at 45 seconds and as in Figure 4.45, there is a better CPU usage presented when comparing these to Figure 5.40.

These trends draw the same conclusion as the FPS graphs as even a low packet loss percentage can have a large effect on the client. By receiving less data each time, there is more of a likelihood of obtaining better results. These better results are not a true reflection of the type of experience that would be had by the user and why packet loss is measured within the proposed architecture. If it were not measured, there would be a high QoS but low QoE as the FPS would remain high but the objects on screen would not be moving as positional data for them is not received. By having packet loss in as a QoS measure, there is a greater assurance of providing a high QoE.

EFFECT OF DISTRIBUTION ON FPS OVER AN INCREASING LATENCY

Another network condition tested is latency. The latency variations used were 100ms, 200ms and 300ms. Figures 5.43, 5.44 and 5.45 show the effect an increasing latency has on the clients FPS.

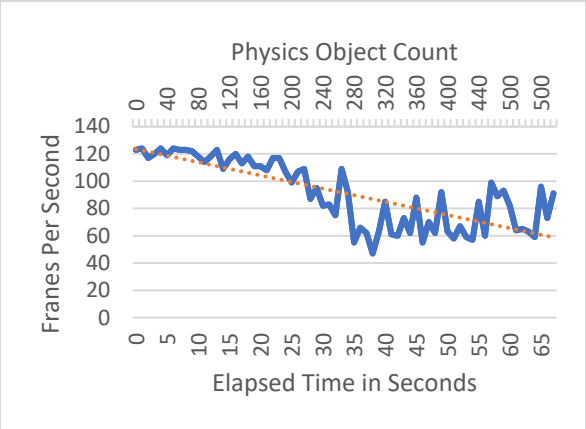


Figure 5.43. FPS when distributing over a network with 100ms latency. Distribution occurred at 38 seconds.

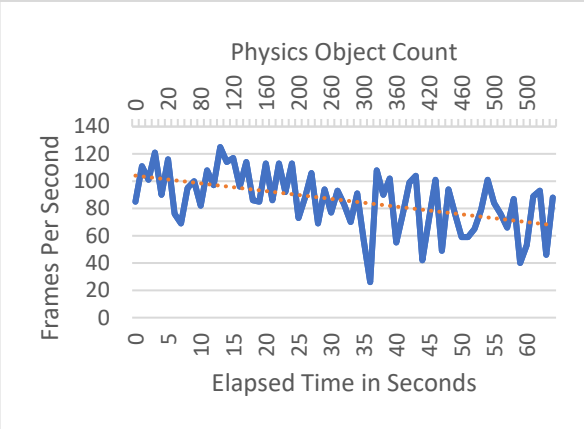


Figure 5.44. FPS when distributing over a network with 200ms latency. Distribution occurred at 37 seconds.

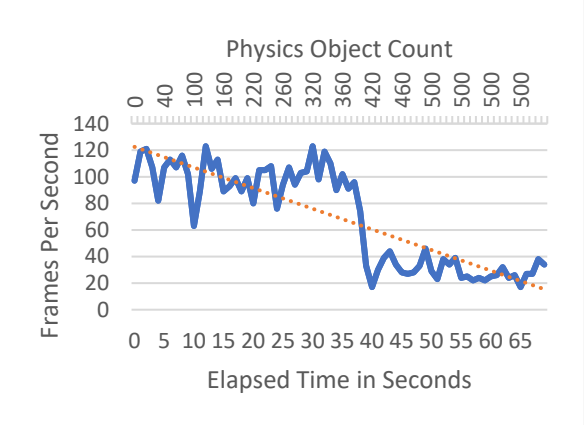


Figure 5.45. FPS when distributing over a network with 300ms latency. Distribution occurred at 40 seconds.

In Figure 5.43, with a 100ms latency, distribution occurred at 38 seconds. In this instance, it can been seen that distribution of processing helps to increase the average FPS again. By having another network node handle the physics processes and update the client with positional data, the FPS of the client can remain above 60FPS even with an increasing number of objects. In Figure 5.44, with a 200ms latency, distribution occurred at 37 seconds. In this instance, distribution of processing did not help increase the FPS, however, it did help to keep it from reducing further. Increasing the latency has led

to a larger variation in FPS after distribution, this is due to the later arrival of the positional data. In Figure 5.45, with a 300ms latency, distribution occurred at 40 seconds. A large drop in FPS, which brought down the average below 70FPS, caused distribution to occur. By distributing data, there is no improvement in FPS, and in this case, the architecture would seek to distribute the data elsewhere to improve the QoS provided.

These trends show that latency will have an increasingly worse effect on FPS when distributing data as seen by the difference in results in Figures 5.43 and 5.44.

Comparing these results to the packet loss variation (Figures 5.37 – 5.39), this method of physics distribution is more tolerable of a varying latency than packet loss. The same conclusion of data not arriving affecting the FPS and providing a false positive cannot be drawn here. From these results it can be seen that increasing latency has the predicted effect of lowering FPS. Further comparison of these results to the full distribution of AI over an increasing latency (Figures 5.11 – 5.13) show that a different ruleset will be needed for physics as the FPS declines in comparison to an increase for AI over an increasing latency.

EFFECT OF DISTRIBUTION ON CPU USAGE OVER AN INCREASING LATENCY

Figures 5.46, 5.47 and 5.48 show the effect an increasing latency has on the clients' CPU usage.

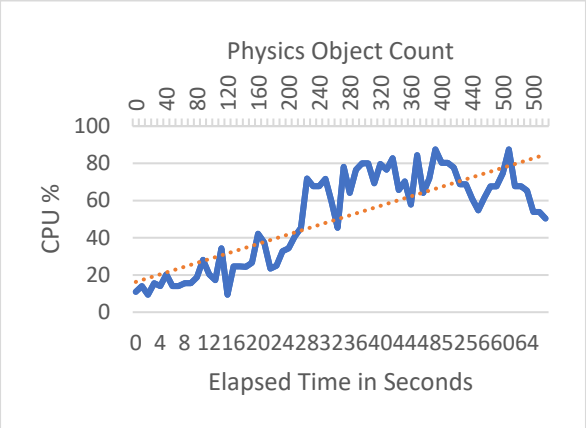


Figure 5.46. CPU % when distributing over a network with 100ms latency. Distribution occurred at 38 seconds.

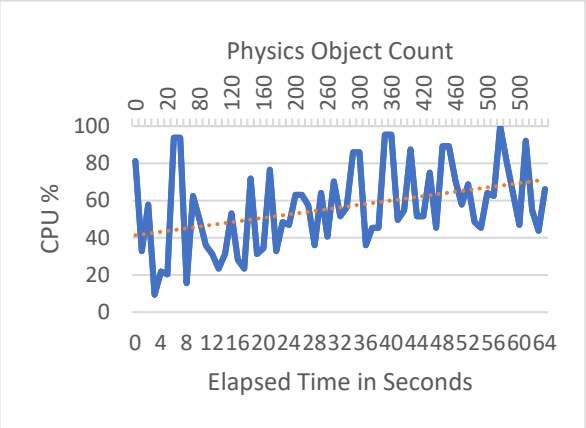


Figure 5.47. CPU % when distributing over a network with 200ms latency. Distribution occurred at 37 seconds.

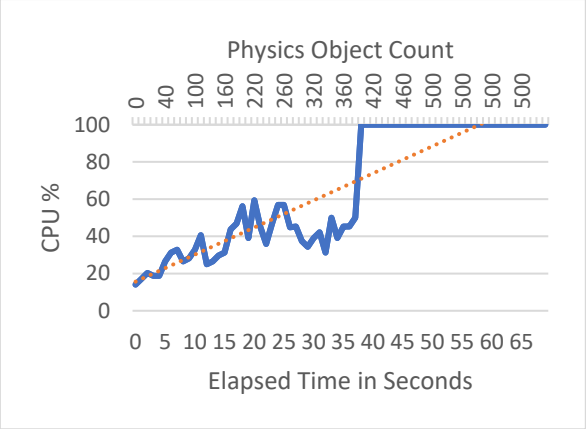


Figure 5.48. CPU % when distributing over a network with 300ms latency. Distribution occurred at 40 seconds.

In Figure 5.46, with a 100ms latency, distribution occurred at 38 seconds. Once distribution occurs, there is an improvement in the CPU usage as its average begins to decrease. This latency seems to have little effect on the CPU as the graph is a similar shape to Figure 5.36 in which no distribution took place. In Figure 5.47, with a 200ms latency, distribution occurred at 37 seconds. Similar to the difference in FPS graphs of the same latencies, a 200ms latency leads to no improvement in CPU usage however the CPU usage, on average, no longer increases. In Figure 5.48, with a 300ms latency, distribution occurred at 40 seconds. Shortly before distribution, the CPU usage peaks at 100% usage and once distribution begins there is no change in the CPU. This constant 100% usage results in the low FPS found in Figure 5.45 and will have been caused by a combination of an under-resourced VM, poor network conditions and local and distributed processing of physics objects.

These trends show that latency has a more traditional effect on CPU usage in comparison to packet loss in the sense that as the network conditions worsen so too does the CPU % usage.

With the full distribution of the physics objects, when data is transmitted over a good quality network, there is an improvement in both FPS and CPU usage. However, once the network conditions begin to change, there is an increasingly deteriorating effect on both the FPS and CPU. Increasing the packet loss percentage of the network produces the worst results as less data is being received by the client and giving false-positive results as a high FPS is recorded but fewer objects are moving on screen. Increasing latency has a better effect in comparison to packet loss as at the lowest latency tested there is an improvement in FPS as well as the CPU usage. Increasing latency further leads to a halt in the decline of FPS and increase in CPU. If data could be distributed over two different networks, one affected by packet loss and the other affected by latency then the network with the latency issue may be chosen provided the latency value is not too high. The proposed architecture measures both

latency and packet loss to avoid a poor performance and false high performance due to network issues. Decisions would be made to avoid this and to assist clients found in this situation.

DISCUSSION

By fully distributing the physics objects, there is a good increase in FPS and CPU usage viewed along a good network connection as seen in Figures 5.35 and 5.36. Like the full distribution of AI objects, Experiment 1, by creating poor network conditions, these improvements diminish. An increasing latency is seen to cause both the FPS and CPU to decline in performance. An increasing packet loss has the opposite effect as both the FPS and CPU improve; this is a false positive. Data is either received at a slower rate or not at all leading to less processing being carried out and therefore higher performance. In the case of these experiments, a 100ms latency could be the maximum latency accepted by a user as both the FPS and CPU usage improved slightly and remained constant once distribution occurred. With regards to packet loss, as seen from the results, a percentage of less than 5% could be accepted as to have 5% or above would greatly affect performance. This could be explored in future work.

5.3.6 PARTIAL DISTRIBUTION OF PHYSICS OBJECTS (EXPERIMENT 5)

This experiment is very like that of the Partial Distribution of AI Objects experiment, the only difference being that Physics processing is distributed here. In this experiment, 10 Physics objects are spawned every second up to a maximum amount of 500 objects.

It is believed with this experiment, that although over a good network the results may not be as high, over a poor network, performance should be better. As seen from the full distribution of Physics objects the network affected the results in a big way, it is predicted that with less information to send, the FPS and CPU levels will not be affected as much. However, it also must be taken into consideration that some objects will still be handled locally which will have a bigger effect on the FPS and CPU in comparison to if they were distributed.

EFFECT OF PARTIAL DISTRIBUTION ON AN UNAFFECTED NETWORK

Figures 5.49 and 5.50 show the partial distribution of Physics objects over a good network. In this instance, distribution occurred at 42 seconds. A large drop in FPS brought the average below 70FPS resulting in the call for distribution. A few seconds after distribution there is another large drop in FPS. This drop is reflected in the CPU usage in Figure 5.50 which sees a usage of 100% for a short period. Soon after this drop, the FPS begins to recover again however it becomes highly varied, the CPU is similarly affected. This drop in FPS, peak in CPU usage and high variation after distribution is due to the local processing of the Physics objects combined with the processing of incoming data and

applying it to the corresponding Physics objects causing the FPS to fluctuate between 40-80FPS at times.

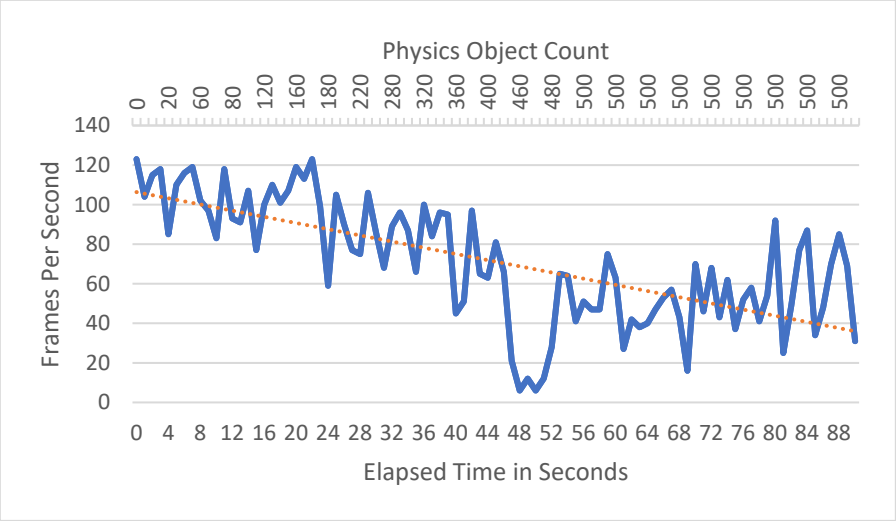


Figure 5.49. FPS against an increasing number of Physics objects. Distribution occurred at 42 seconds.

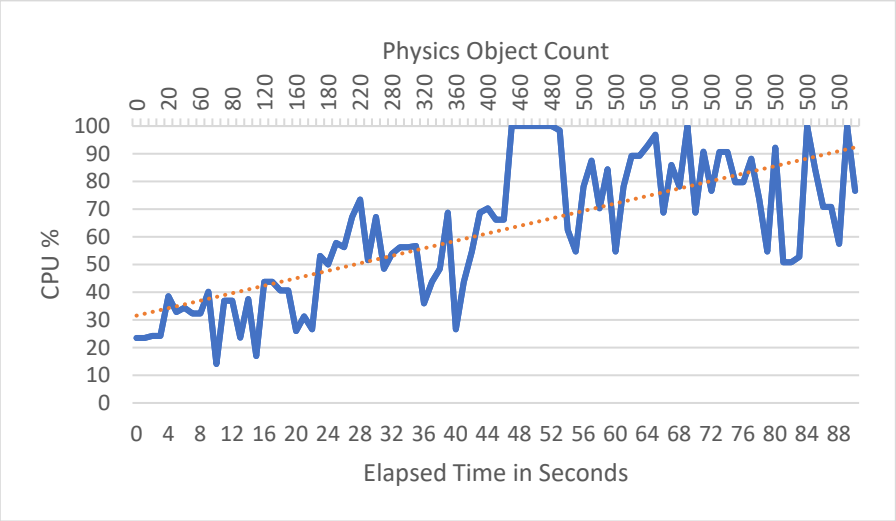


Figure 5.50. CPU % against an increasing number of Physics objects. Distribution occurred at 42 seconds.

EFFECT OF PARTIAL DISTRIBUTION ON FPS OVER AN INCREASING PACKET LOSS

This method of distribution was also tested over six different network variations: three different packet loss percentages (5%, 10% and 15%) and three different latencies (100ms, 200ms and 300ms). Figures 5.51, 5.52 and 5.53 show the effect an increasing packet loss percentage has on the clients FPS.

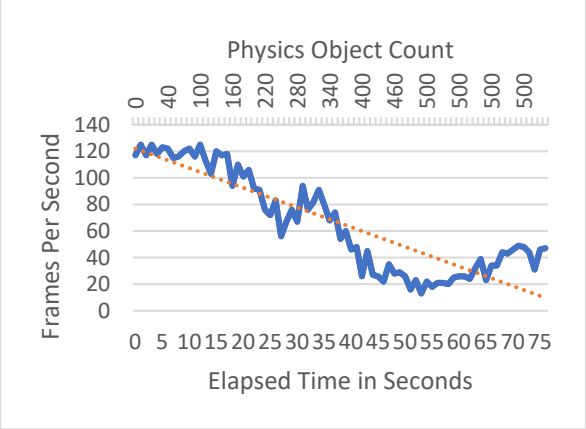


Figure 5.51. FPS when distributing over a network with 5% Packet Loss. Distribution occurred at 29 seconds.

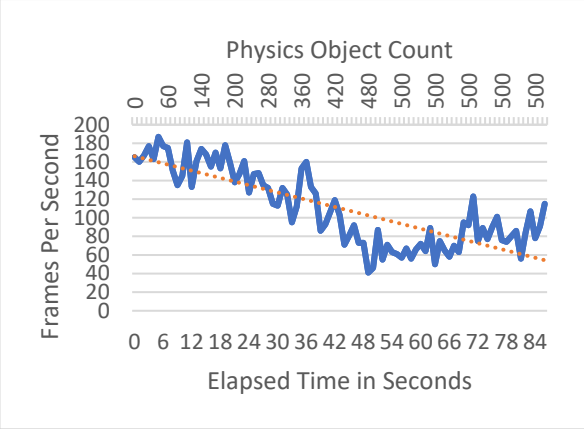


Figure 5.52. FPS when distributing over a network with 10% Packet Loss. Distribution occurred at 50 seconds.

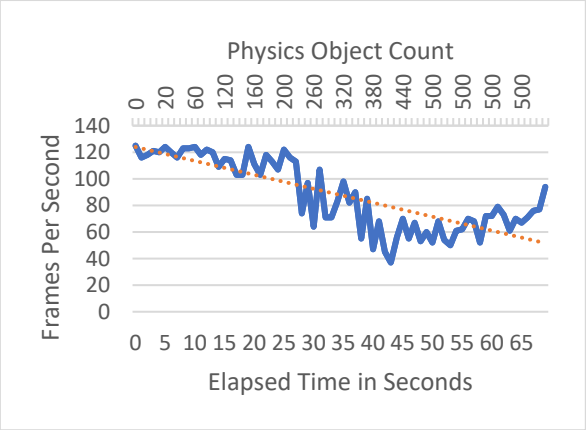


Figure 5.53. FPS when distributing over a network with 15% Packet Loss. Distribution occurred at 41 seconds.

In Figure 5.51, with a 5% packet loss, distribution occurred at 29 seconds. The trend shows that even with distribution the FPS continues to fall but then improves by 20FPS around 20-25 seconds later as no more objects are spawned. In Figure 5.52, with a 10% packet loss, distribution occurred at 50 seconds. Shortly after assistance is provided, the FPS improves. However, in this instance, as distribution occurred so late there were no physics objects passed off for processing elsewhere so here all of the objects were processed locally. The improvement in FPS could be due to the end of the spawning of the objects. In Figure 5.53, with a 15% packet loss, distribution occurred at 41 seconds. The need for distribution was after 380 physics objects had spawned. However, there will still be objects passed off for processing. In this case, there is an improvement in FPS a short time after distribution.

The trends here show the effect packet loss has on FPS when partially distributing data. An improvement is seen in each of the results. However, this only occurs whenever the spawning of objects has been stopped. Distribution occurred later than expected in Figures 5.52 and 5.53 and the distribution in Figure 5.51 had no effect on the FPS which only improved when there were no longer any more objects to spawn.

EFFECT OF PARTIAL DISTRIBUTION ON CPU USAGE OVER AN INCREASING PACKET LOSS

Figures 5.54, 5.55 and 5.56 show the effect an increasing packet loss percentage has on the clients' CPU usage.

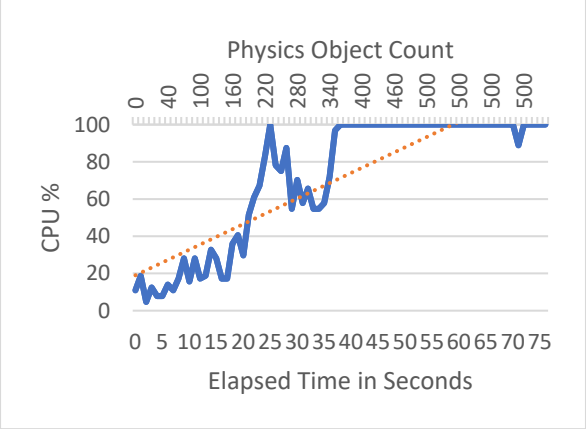


Figure 5.54. CPU % when distributing over a network with 5% Packet Loss. Distribution occurred at 29 seconds.

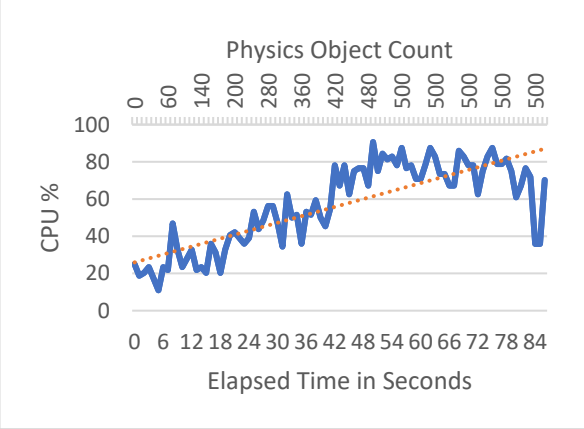


Figure 5.55. CPU % when distributing over a network with 10% Packet Loss. Distribution occurred at 50 seconds.

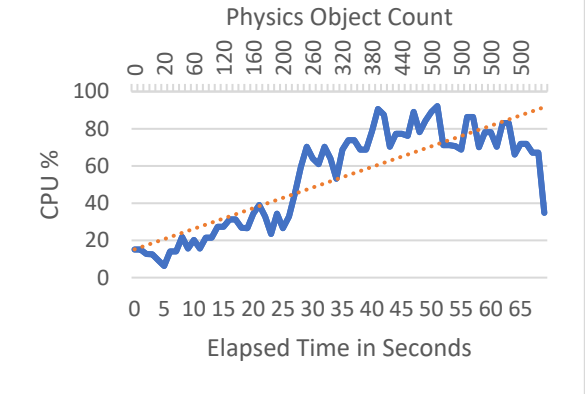


Figure 5.56. CPU % when distributing over a network with 15% Packet Loss. Distribution occurred at 41 seconds.

In Figure 5.54, with a 5% packet loss, distribution occurred at 29 seconds. Similar to the counterpart FPS graph, distribution had little to no effect on the CPU usage as the trend shows it continuing to

increase to its maximum of 100% usage. There seems to a small effect in that for a few seconds after assistance is provided the CPU usage remains around 60% before increasing. The low FPS of Figure 5.51 is reflected here as the CPU usage is very high. In Figure 5.55, with a 10% packet loss, distribution occurred at 50 seconds. In this instance, no physics objects were passed off as distribution was asked for when 500 (the maximum used) objects had spawned therefore the CPU usage remains high. The packet loss of the network had no effect on this experiment as no data was transmitted over it. In Figure 5.56, with a 15% packet loss, distribution occurred at 41 seconds. Distribution was requested for later than expected. However, some 120 objects were handled away from the client meaning 380 were processed locally. The high packet loss working on the 120 objects combined with the local processing of 380 objects leads to a better result in comparison to Figure 5.54. However, as with previous experiments, it is seen that a high packet loss will provide better results on paper. On screen, objects will not move as data is not received meaning no processing is taking place leading to better overall results.

With this method of distribution, it performs poorly over a low packet loss percentage. This is evident in Figures 5.51 and 5.54 as, with a rough 50/50 split in processing distribution, there is a constant low FPS and high CPU usage. Comparing the full distribution results to these, Figures 5.37 and 5.40 (5% packet loss), neither method is effective at a low percentage. With either, the most that can be attained is a levelling out of FPS and CPU usage.

EFFECT OF PARTIAL DISTRIBUTION ON FPS OVER AN INCREASING LATENCY

Another network condition tested is latency. The latency variations used were 100ms, 200ms and 300ms. Figures 5.57, 5.58 and 5.59 show the effect an increasing latency has on the clients FPS.

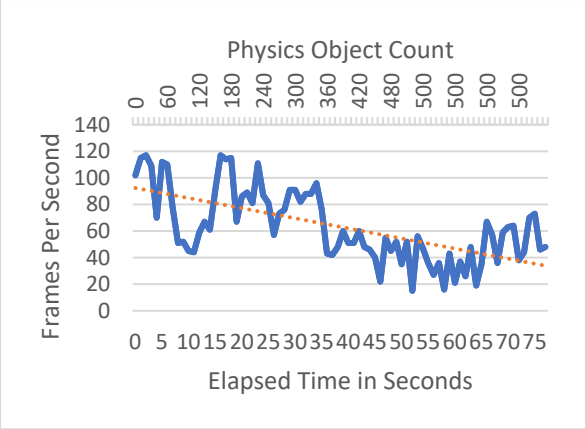


Figure 5.57. FPS when distributing over a network with 100ms latency. Distribution occurred at 10 seconds.

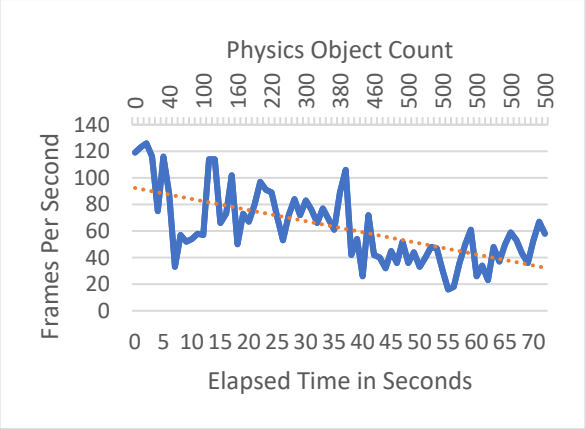


Figure 5.58. FPS when distributing over a network with 200ms latency. Distribution occurred at 9 seconds.

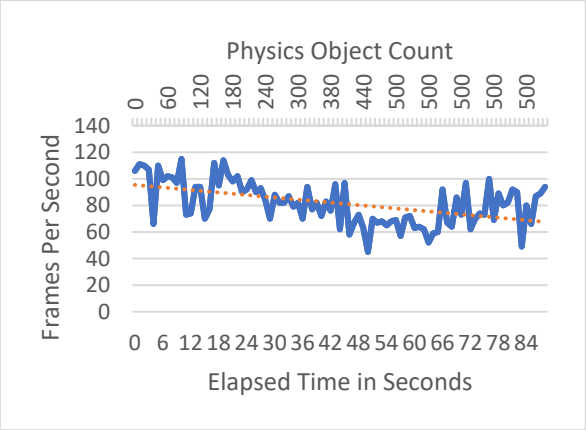


Figure 5.59. FPS when distributing over a network with 300ms latency. Distribution occurred at 49 seconds.

In Figure 5.57, with a 100ms latency, distribution occurred at 10 seconds. The average value of the previous 3 seconds fell below the 70FPS limit put in place, therefore, assistance was required. Based on the data series, the average FPS improves until around 35 seconds and then fluctuates between 20FPS and 40FPS due to the increased amount of data being received. The last 20 seconds of this experiment see the FPS improve again as there are no more physics objects created. In Figure 5.58, with a 200ms latency, distribution occurred at 9 seconds. The data series and trendline here are almost the same as Figure 5.57 as distribution sees an improvement to FPS for a while before falling and then increasing towards the end of the experiment. In Figure 5.59, with a 300ms latency, distribution occurred at 49 seconds. Before this point, the FPS was on a slow decline which then improved slightly once assistance was provided.

The trends here show that there is an improvement in FPS once objects stop spawning as the creation of 10 physics objects every second consumes the resources of the poorly resourced VM. The most noticeable improvement occurs in Figure 5.57 with the lowest latency and with the worst results shown in Figure 5.59, this was expected. In the 300ms latency experiment, distribution occurred very late, and only 60 of the objects were passed off to another node while the other 440 were processed locally, this lead to a better FPS over the other experiments.

EFFECT OF PARTIAL DISTRIBUTION ON CPU USAGE OVER AN INCREASING LATENCY

Figures 5.60, 5.61 and 5.62 show the effect an increasing latency has on the clients' CPU usage.

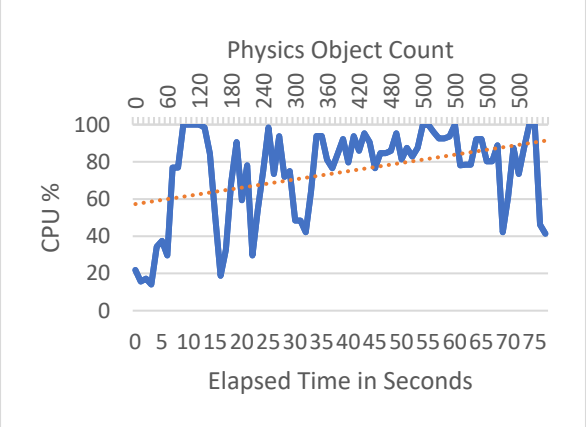


Figure 5.60. CPU % when distributing over a network with 100ms latency. Distribution occurred at 10 seconds.

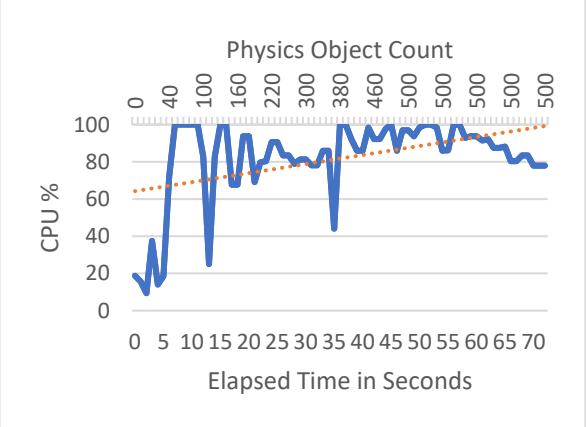


Figure 5.61. CPU % when distributing over a network with 200ms latency. Distribution occurred at 9 seconds.

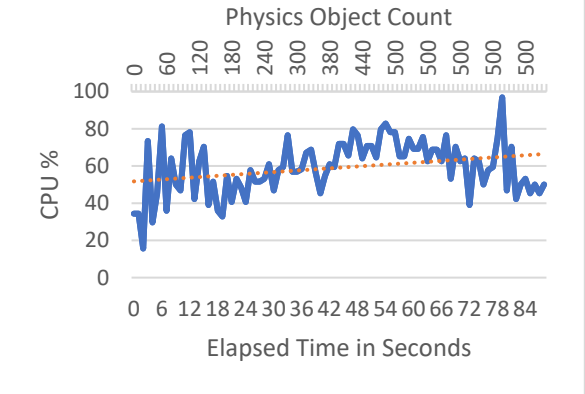


Figure 5.62. CPU % when distributing over a network with 300ms latency. Distribution occurred at 49 seconds.

In Figure 5.60, with a 100ms latency, distribution occurred at 10 seconds. This graph is very varied as there are many peaks and drops. The trend shows that when assistance is provided, there is a small

improvement, on average, of the CPU usage which then increases to almost 100% usage. As seen from the FPS results of this experiment there is a small improvement in FPS towards the end; the same happens with the CPU usage. This is due to the end of the spawning of physics objects. In Figure 5.61, with a 200ms latency, distribution occurred at 9 seconds. In this instance, it takes almost 10 seconds before there is a small improvement in the CPU usage which is not maintained as the average almost peaks at 100% usage similar to Figure 5.60. The trend of this graph also shows an improvement in usage towards the end of the experiment again due to the end of object creation. In Figure 5.62, with a 300ms latency, distribution occurred at 49 seconds. Assistance is provided late here and the majority of the objects, 440 of them, are processed locally while only 60 are passed off. Once assistance is provided there is a drop of roughly 20% in CPU usage. This drop is a combination of the end of object creation and not having to process as much incoming data.

The trends here show that, similar to the FPS graphs, once the objects stop spawning, CPU usage begins to improve again as resources are freed up. It is better to compare Figures 5.60 and 5.61 here as the results of Figure 5.62 are very different. Increasing the latency increases the likelihood of poorer performance in relation to CPU usage. As more objects continue to spawn the CPU usage will suffer, however, once this subsides the CPU will begin to improve again.

DISCUSSION

Comparing the results of this experiment to experiment 4, it seems that complete distribution of all objects performs better than partial distribution. It was predicted that, with this method, there would be better client performance when distributing over poor network conditions. This prediction was false as the partial distribution of objects leads to a greater variation in client performance regardless of network condition. At the highest packet loss and latency each of FPS and CPU usage seemed to show improvement. However, this was due to data either taking too long to be received and therefore processed or it not being received at all.

5.3.7 SUMMARY OF DISTRIBUTION

The purpose of these distribution experiments is not to show that data can be distributed but how it can be distributed and how these methods perform over varying network conditions. Based on these results it can then be determined which method would be best utilised within the architecture. Only AI and Physics processes have been experimented with.

AI distribution was tested over three variations, and these were a full distribution in which all objects were handled off-client, partial distribution in which only some objects were handled off client and the sending of path data in which all objects were handled off-client but in a different way. The full

and half distribution methods involved updating the client with constant positional data on each object while the sending of path data involved updating the client on a list of waypoints for each object to visit, this data was much less frequent in transmission.

Physics distribution was tested over two variations, and these were a full distribution and half distribution. These approaches were the same as the AI examples as the client was updated with positional data on each object that was being processed off-client.

Each of these five experiments were then tested over seven network variations: An unaffected network, packet loss percentages of 5%, 10% and 15% and latencies of 100ms, 200ms and 300ms.

Empirical testing found these variations in packet loss and latency to provide the greatest variation in results. Each experiment has the FPS and CPU usage recorded.

By far the best approach used, regardless of AI or Physics distribution is the full distribution method. With this method, when the average FPS fell below 70FPS, all objects currently in the game world and those still to be spawned would be handled by another network node. This other network node then constantly updated the client on the current position of each object. Over a good network, the improvement was very visible as seen from the results as both the FPS and CPU improves. With the other approaches of half distribution and the sending of data for AI, the best-case scenario that was viewed was the decline of FPS and increase in CPU usage to stop and for each to level out. Both packet loss and latency are metrics used within the proposed architecture with both helping to make the decision as to where processing can be distributed.

By fully distributing the AI in experiment 1, there is a good increase in FPS and CPU usage when distributed along a good network connection. By creating poor network conditions, these improvements diminish. With the poorest of network conditions, the FPS hits low peaks unacceptable to users as does the CPU.

Partially distributing the AI objects, experiment 2, saw no improvement to overall FPS or CPU usage even over an unaffected network. The local processing of some objects and the distribution of others is a combination that had a negative effect on the performance of the client. When compared with experiment 1, distributing all the objects produced better results in comparison to partial distribution. When creating poor network conditions, the best situation that could be hoped for with this method of distribution is that the decrease in FPS and increase in CPU usage is slowed and levels out. As either packet loss or latency increases both the FPS and CPU will drop in performance. At the highest packet loss and latency tested, each of FPS and CPU usage seemed to show improvement, however, this was

due to data either taking too long to be received and therefore processed or it not being received at all.

The distribution of AI processing by transmitting path data from assisting node to client produced results poor enough that it was concluded not to test over a network affected by packet loss or latency.

By fully distributing the physics objects, there is a good increase in FPS and CPU usage viewed along a good network connection as seen in Figures 5.35 and 5.36. Similar to the full distribution of AI objects, experiment 1, by creating poor network conditions, it can be seen that these improvements diminish.

Similar to the comparison of experiments 1 and 2, comparing the results of experiment 4 to experiment 5 sees that complete distribution of all physics objects performs better than partial distribution. It was predicted that, with partial distribution, there would be better client performance when distributing over poor network conditions. This prediction was false as the partial distribution of objects leads to a greater variation in client performance regardless of network condition.

The poorest of network conditions tested in these experiments see both the FPS and CPU usage perform well; this is a false positive as data is either received at a slower rate or not at all leading to less processing being carried out and therefore higher performance. It is for this reason that both latency and packet loss percentage are monitored within the proposed architecture.

5.4 CLIENT ADAPTATION

This is the last resort of the proposed architecture in that if a client requires assistance and there are no available network resources then it must reduce its processing. Regarding AI, Physics and Graphics, this takes the form of reducing the number of AI objects, Physics Objects or particles in a particle effect. For this experimental scenario, there are three sets of results: AI, Physics and Graphics. For each of the experiments of this scenario there are results on the FPS, CPU and GPU of the client device. It is possible to use GPU results for this scenario as there is no VM involved, these experiments were run on a PC.

5.4.1 EXPERIMENTAL SETUP

This set of experiments were run on a PC with:

- 8GB Memory
- I7-3770/3.4Ghz
- 4 Core CPU

- Intel HD Graphics

The experiments were created in Unity3D version 5. The FPS is calculated within the game while the CPU and GPU percentages are retrieved from MSI's Afterburner. Figure 5.63 is adapted from Figure 3.2 found in the Architecture Chapter. It details the part of the decision tree that this scenario focuses on which is Client Adaptation. The Client will be forced to reduce the AI, Physics or Graphics presence on screen to improve the QoE provided as, for these experiments, there are no resources available on the network to assist with and therefore the client must adapt.

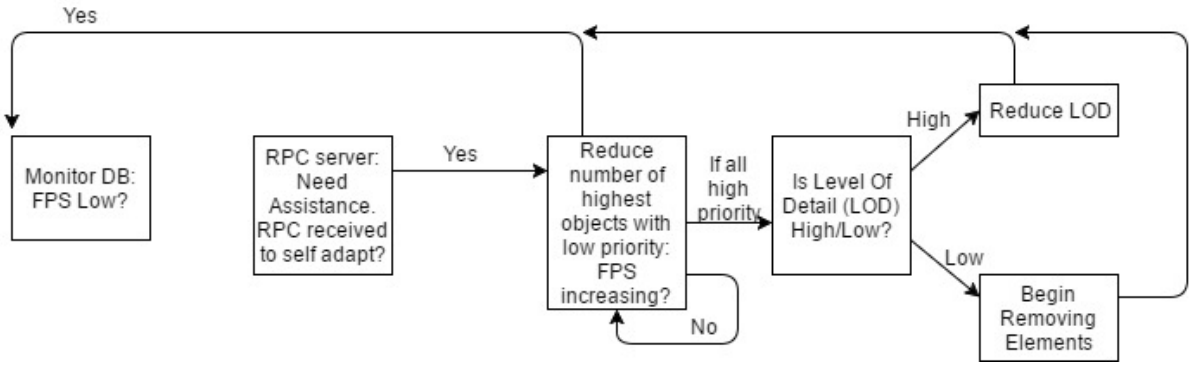


Figure 5.63. Client Adaptation

Figure 5.63 begins with the client receiving an RPC informing it to begin the self-adaptation process. The client will then begin to reduce the element, from AI, Physics and Graphics, which has the highest number and lowest priority to improve the QoS. This process continues until there is an improvement in the FPS, after which the architecture continues to monitor the FPS. If the FPS is low, then the architecture begins again from the start of the decision tree found in Chapter 3, Figure 3.2. If there is a game scene in which all objects within are of a high priority, then the client will reduce the current level of detail, however, if this is already low or the reduction has no effect on the FPS then objects must be removed. The paper-based analysis in section 4.3.1 presents how an element would be chosen.

The experiments in this section show the improvement in a client when each element is reduced. As the paper-based analysis covers the selection of the element, this scenario focuses on each element increasing over time and then being reduced, showing the improvement to the system regarding performance. Each element in its experiment is seen as having the lowest priority with the highest number of objects. Therefore, it will be reduced as seen from the results in Table 4.1. Due to the higher resource availability of the PC that these experiments were run on, 30FPS was chosen as the lowest limit for the frame rate. 30 FPS has been chosen as it is the framerate which all companies set as the minimum target for their games. Once the average FPS, over 3 seconds, falls below 30 FPS,

then the reduction in the element will begin. The reduction is slow to find the optimum number of objects for 60 FPS. A rate of 60 frames per second is the main target of games companies.

5.4.2 CLIENT REDUCTION OF AI OBJECTS (EXPERIMENT 6)

In this experiment, the client has an increasing number of AI objects. Twenty AI objects are created every second to have a visible effect on the frame rate. As seen from the FPS in Figure 5.64, as the number of objects increases, the FPS decreases. The FPS is monitored every second with the average being taken of the previous 3 seconds. Once the average FPS falls below 30, then the reduction in the number of AI objects begins.

Figure's 5.64, 5.65 and 5.66 show the FPS, CPU usage and GPU usage as the client self-adapts to a situation focused entirely on AI objects.

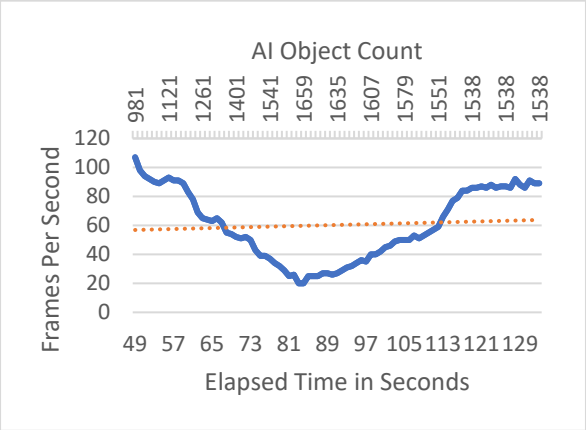


Figure 5.64. FPS of a client self-adapting to AI. The process began at 83 seconds.

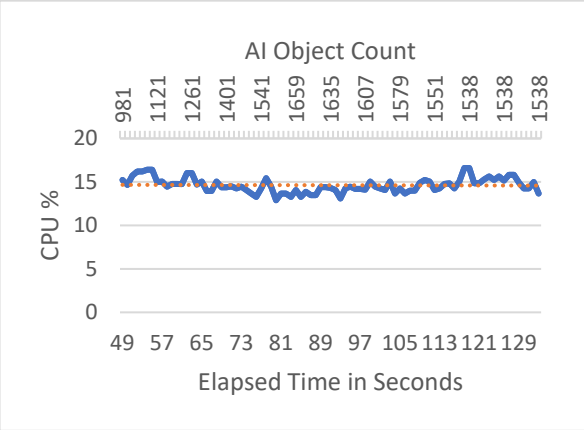


Figure 5.65. CPU usage of a client self-adapting to AI. The process began at 83 seconds.

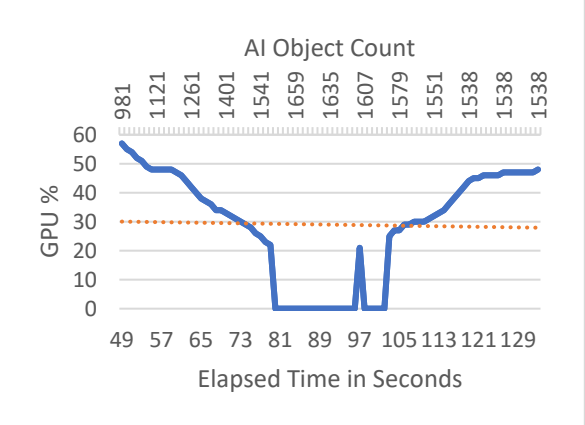


Figure 5.66. GPU usage of a client self-adapting to AI. The process began at 83 seconds.

Figure 5.64 shows a declining FPS rate as the number of AI objects increases. At 83 seconds the average FPS is low enough for self-adaptation to begin and as AI objects are removed from the scene, the FPS improves. Figure 5.65 shows the CPU percentage usage during this experiment. The most it can be seen to vary is around 4%. This shows that this experiment had little to no effect on the CPU usage of the PC. Figure 5.66 shows a declining and then improving GPU usage. The GPU percentage usage remains at 0% for a considerable time. This is interesting as it would be expected that with a declining FPS the GPU usage would increase as frames become harder to render. Looking at forums such as Tom's Hardware [88] and Reddit [89], this seems to be quite a common issue with PC games. Fixes include purchasing a new CPU cooler as the CPU can overheat causing the GPU to crash and plugging the monitor into the graphics card instead of the motherboard. A crash is a worst-case scenario, as seen from Figure 5.66, by reducing the number of AI objects on screen the GPU begins to recover. The spike along the 0% line is the GPU attempting to recover, only when the object count is low enough can the GPU begin to perform again.

5.4.3 CLIENT REDUCTION OF PHYSICS OBJECTS (EXPERIMENT 7)

In this experiment, the client has an increasing number of physics objects. Fifty physics objects are created every second to have a visible effect on the frame rate. As seen from the FPS graph, as the number of physics objects increases, the FPS decreases. The FPS is monitored every second with the average being taken of the previous 3 seconds. Once the average FPS falls below 30 then the reduction in the number of physics objects can begin.

Figure's 5.67, 5.68 and 5.69 show the FPS, CPU usage and GPU usage as the client self-adapts to a situation focused entirely on Physics objects.

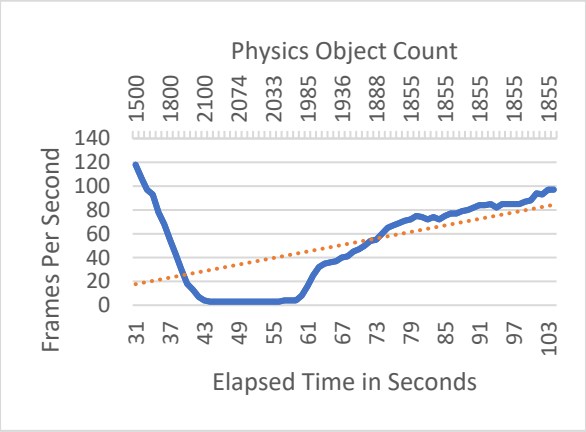


Figure 5.67. FPS of a client self-adapting to Physics. The process began at 43 seconds.

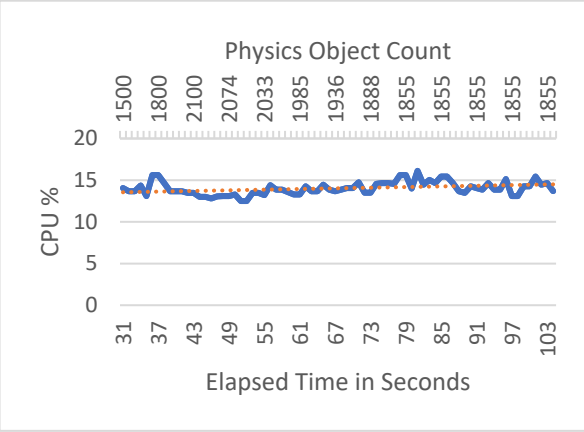


Figure 5.68. CPU usage of a client self-adapting to Physics. The process began at 43 seconds.

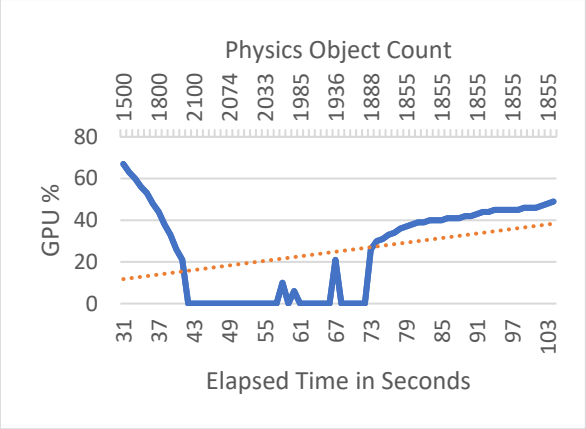


Figure 5.69. GPU usage of a client self-adapting to Physics. The process began at 43 seconds.

Figure 5.67 shows a declining FPS rate as the number of Physics objects increases. At 43 seconds the average FPS is low enough for self-adaptation to begin and as Physics objects are removed from the scene, the FPS improves. However, as the reduction rate is slow, the FPS remains under 30 for almost 20 seconds. Once the number of objects falls under 2000, then the FPS begins to improve again. In this situation, the rate of reduction would need to increase over time for the client to be able to continue gaming at an acceptable FPS sooner. Figure 5.68 shows the CPU percentage usage during this experiment. The most it can be seen to vary is around 4%. This shows that this experiment had little to no effect on the CPU usage of the PC. Figure 5.69 shows a declining and then improving GPU usage. This is similar to the GPU result of the AI portion of this experiment as the GPU remained at 0% usage for a considerable amount of time due to the high number of physics objects. However, as the number of Physics objects decreases, the GPU usage recovers. The three spikes along the 0% line are the GPU attempting to recover, only when the number of physics objects is low enough can the GPU recover again.

5.4.4 CLIENT REDUCTION OF GRAPHICS (EXPERIMENT 8)

In this experiment, the client has three particle systems. A particle system is a component of many games that utilises a large number of small images to simulate certain kinds of “fuzzy phenomena” such as a fire [90]. Testing showed that a huge number of particles was required to slow the system down. Therefore, the number of particles was increased per update cycle. Each update cycle saw each particle system increase the number of particles it emits by 100 (total of 300 each cycle). As seen from the FPS graph, as the number of particles increases, the FPS decreases. The FPS is monitored every second with the average being taken of the previous 3 seconds. Once the average falls below 30 FPS, then the reduction in the number of particles onscreen can begin.

Figure's 5.70, 5.71 and 5.72 show the FPS, CPU usage and GPU usage as the client self-adapts to a situation focused entirely on Graphics in the form of particle effects.

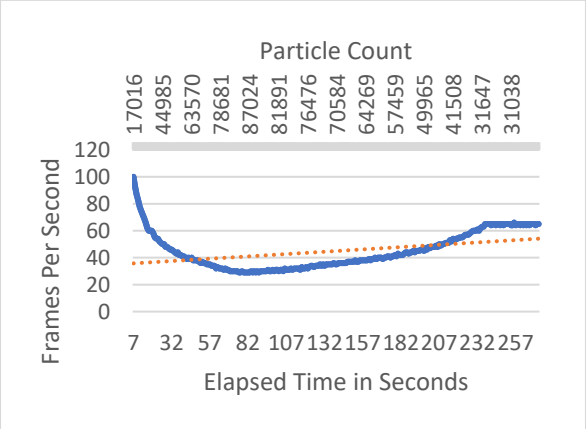


Figure 5.70. FPS of a client self-adapting to Graphics. The process began at 78 seconds.

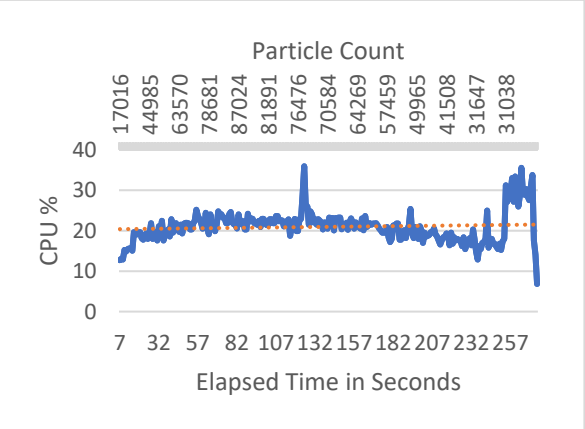


Figure 5.71. CPU usage of a client self-adapting to Graphics. The process began at 78 seconds.

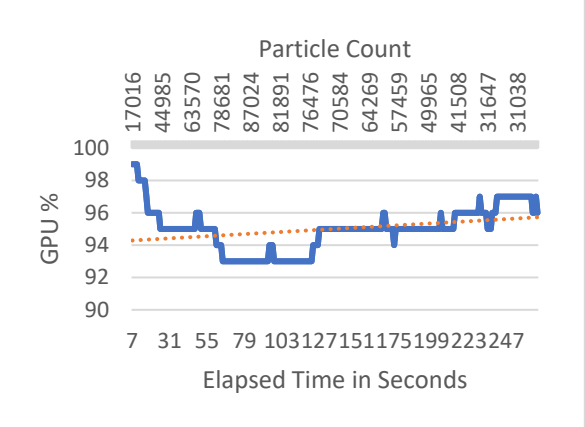


Figure 5.72. GPU usage of a client self-adapting to Graphics. The process began at 78 seconds.

Figure 5.70 shows a declining FPS rate as the number of particles increases. At 78 seconds the average FPS is low enough for self-adaptation to begin and as Graphics objects are removed from the scene, the FPS improves. Similar to experiment 7, the reduction rate is slow however the FPS climbs steadily and does not remain low for an extended period such as in Figure 5.64. In this situation, the reduction rate may be increased slightly to allow for a higher framerate sooner. Figure 5.71 shows the CPU percentage usage during this experiment. For the majority of the time, the CPU percentage does not vary much between 15% and 25%. At around 124 seconds there is a spike sending the percentage above 35%, this then settles as soon as it appears, this could be due to a background process. The same can be said of the increase to just below 35% towards the end of the experiment. However, another suggestion is that the high functioning GPU is the cause. Figure 5.72 shows a GPU usage that does not vary much as it is constantly above 93%. Some correlation can be seen between this and the

FPS in Figure 5.70. The result is very different from the previous two experiments in which the GPU usage fell to 0% in both. This may be because the particle systems were easier for the GPU to maintain in comparison to AI and Physics processes.

5.4.5 SUMMARY OF SELF-ADAPTATION

The purpose of the self-adaptation experiment set was to show how a client would react when under pressure from an increasing amount of AI objects, Physics objects and Graphics particles. Self-adaptation began when the average FPS fell below 30 FPS, and the number of objects on screen was reduced. The reduction of these objects was stopped once the average FPS was 60 FPS or above.

The AI self-adaptation had 20 AI objects spawned every second to place an increasing amount of pressure on the system. Each of these objects would be pathfinding their way throughout the game world. Once the average FPS fell below 30 FPS, the self-adaptation process began in which the number of objects began to reduce at a rate of between 2 to 5 objects per second. The improvement in FPS was immediate and the CPU usage was unaffected during this experiment as the usage percentage varied by around 3%. The GPU usage shows a period of 0% usage as the GPU crashes for a time as the AI objects increase. Once the number of objects reaches a more acceptable level the GPU begins to function again.

The Physics self-adaptation had 50 Physics objects spawned per second. Each of these objects had force applied to them upon creation causing them to immediately interact with the game world. Once self-adaptation began, the rate of reduction was on average eight objects per second. In this instance, the FPS did not immediately improve. With a larger number of objects, the rate of reduction would need to be higher to see a better turn-around. The CPU usage of this experiment reacted similarly to the AI in that it remained unaffected. The GPU usage also reacted in a similar fashion to the AI experiment as it too crashed for a period. Both the FPS and GPU usage drop quickly here, therefore, a slower spawn rate would have benefitted the results.

The Graphics self-adaptation had on average 1100 particles created each second until self-adaptation was required at which point the rate of reduction was around 360 particles per second. The FPS here does not show an immediate improvement but one over time. The large spawn rate caused the FPS to drop quickly, however, the rate of reduction was much slower causing a slow return to an acceptable FPS level. The CPU usage here provided the most varied results as well as an on average higher usage in comparison to the previous self-adaptation experiments. Reduction in the number of particles shows a small drop in the CPU usage with a slight increase of around 10% towards the end.

The self-adaptation process is the fail-safe in the proposed architecture. This process is only required if the client needs assistance and cannot receive it from a server or other network node. The objects are reduced at a low rate so that the client can find the maximum number of objects for a high framerate as a fast reduction would mean a lesser number of objects and possible reduction in the immersion of the game. It would be ideal to have as many objects as possible on the screen to help with game immersion for example if a client cannot process 100 birds in the sky but can process 50 then a reduction of either 5 or 10 objects per second would leave 50 in the game. If the number of objects were reduced at a faster rate, for example, 20 objects per second, the number of birds would be much lower, potentially affecting the game experience. Self-adaptation allows the user to continue playing their game even when there are no resources available for the distribution of data.

5.5 DISCUSSION

Chapter 5 has focused on the results of experiments created to support the architecture proposed in Chapter 3. Section 5.3 contains five experiments which focus on several methods of distribution for both AI and Physics tasks; these are also tested against various network conditions to find the best distribution method for the game element. Section 5.4 contains three experiments which explore the self-adaptation component of the proposed architecture. This component only executes when there are no resources available on the network. Therefore, the client will begin reducing an element. The decision of which element to reduce is the same as which element to ask for assistance with in section 4.3.

The aim of the experiments in section 5.3 is to explore methods of distribution for both AI tasks and Physics tasks and to determine which would perform best under a range of network conditions. Three forms of processing distribution were investigated for AI; full distribution in which all AI objects are handled elsewhere, partial distribution in which a portion of the AI objects are handled elsewhere while others are handled locally, and the sending of path data in which the path each object had to follow was calculated elsewhere and transmitted to the client. Two forms of processing distribution were investigated for physics and these were full distribution and partial distribution, both of these methods operated in the same way as the AI methods of the same name. For each form, there are seven different network variations used: an unaffected network, packet loss variations of 5%, 10% and 15% and latency variations of 100ms, 200ms and 300ms. It was discovered that the best approach for distribution of both the AI and Physics tasks was the full distribution method. With this method, when the average FPS fell below 70FPS then all objects currently in the game world and those still to be spawned would be handled by another network node. This other network node then constantly

updated the client on the current position of each object. This approach to distribution had the best performance regardless of network condition.

Section 5.4 saw the exploration of the self-adaptation component and the benefit that this would have to the client in a situation where there would be no possibility of distribution. This process will begin if a client receives an RPC from the server informing it to self-adapt as explored in section 4.3. Each game element from AI, Graphics and Physics was self-adapted separately, as it would occur within this architecture. Each experiment has shown an improvement in the overall performance of the client once the process begins. There is a slow rate of reduction in the number of objects to find the highest number of objects that can still be on screen for a high framerate. By keeping as many of the original objects as possible, there is less likelihood of reducing the overall game immersion. However as seen from experiments 7 and 8, the rate of reduction would benefit from increasing over time instead of staying at a flat rate, this would lead to a quicker improvement in the performance of the client.

Altogether these experiments prove the validity of the architecture proposed in chapter 3. This architecture utilises resources widely available on the network to improve the QoS and therefore the QoE of a client's device. Decisions are made by both the struggling client and server to facilitate this improvement in QoE. A fail-safe has been included in the architecture in the form of self-adaptation which will help the client improve its QoS when the distribution of processing is not possible.

CHAPTER 6 CONCLUSION

6.1 INTRODUCTION

The overall aim of this work was to research and develop an architecture that would improve a user's QoE of a network aware game by improving the QoS provided through the utilisation of distributed resources. The architecture developed combined cloud and fog computing as well as a self-adaptation component. The background research of the surrounding areas describes the benefits and drawbacks of both cloud and fog computing. Much of the research focused on adding hardware to a network to facilitate either the cloud or fog, therefore providing justification for this research which focuses on utilising hardware that is already present within the network.

The testing of this unique combination of cloud computing, fog computing and a self-adaptation component fell into three areas:

1. Decision-Making
2. Distribution
3. Client Adaptation

The Decision-Making area focused on the decision-making ability of the architecture. Firstly, it was demonstrated how a client would decide which element to ask for assistance with between AI, Physics and Graphics. Then the servers process was focused on with how it would react with multiple clients. In this paper-based analysis, many clients were connected to the server with some requiring assistance, some able to provide and others that are able to maintain a high QoS without receiving or providing assistance. Here the server decides which client will receive assistance from where, based on values including CPU %, GPU % and device type.

The Distribution area focused on the AI and Physics elements specifically and a variety of ways that these could be distributed along a network. In order to find the best method of distribution for these elements, each was tested against various network conditions. The results from these experiments show that the best method of updating the struggling client is with constant positional updates on objects as this performed best under all network conditions.

The Client-Adaptation area focused on the fail-safe implemented into this architecture which will only execute if there are no resources available either locally or globally to aid a client. The reason this is a fail-safe is due to the removal of unessential game objects from the game world. In order to maintain a high QoS this may be unavoidable.

6.2 EVALUATION OF OBJECTIVES

As established in the introductory chapter of this thesis, the research objectives were as follows:

1. To review existing methods of distribution in video games. How they distribute data and which data they choose to distribute.
2. To determine which metrics can be used in a decision-making architecture, in which the outcome of all decisions is to improve the QoS provided. These metrics can then be prioritised.
3. To develop an adaptable architecture which will utilise the cloud and fog resources available to improve QoS. This architecture can then be tested against differing data types and connection variations. Self-adaptation will be included as a last resort. In the unlikely event that no resources are available from either the cloud or fog, the client device can adapt the game itself.

The existing research was split into five areas: Streaming Games, Distributed Environments, Distributed Management, Fog Computing and Energy Saving. The area of Streaming Games contained the most relevant models. Although there was little detail in the data that was distributed within these models, there was much more detail in the process used to distribute the data with the main benefit gleaned from this area being the adaptable nature of the models. This adaptability transferred over into the final architecture. The Distributed Environments area yielded interesting details with regards to Distributed Virtual Environments in which a model of event communication was proposed, the results for which show that when adapting to the variation in wireless networks, individual event streams have different resource requirements. Other research in this area shows that the introduction of additional hardware to a network can improve upon the service provided aswell as partitioning AI processes into high and low frequency components. The area of Distributed Management focused on Agents and their benefits aswell as metrics. The RPCs found within the proposed architecture represent the Agents found in research as they have many of their properties and therefore benefits. Fog Computing yielded two models: EdgeCloud and CloudFog. The architecture proposed can be seen as a combination of these two models as smaller data packets are transmitted, and under-utilised nodes are employed to assist. The final area of energy saving is small as the aim of this thesis was not to save energy but to improve a client's QOE through the utilisation of under-utilised network nodes. However, it can be argued that the client's energy is being saved via this architecture.

Within the proposed architecture are metrics which are core to the overall proper function, these are found in Chapter 3. Each metric is involved within a decision which will ultimately result in the improvement of QoS provided and therefore QoE. The values utilised are as follows:

- FPS
- Current Level of Detail (LOD)
- Resource Requirement of Tasks
- Number of objects on screen (AI, Physics, Graphics)
- Priority of on-screen objects
- CPU usage
- GPU usage
- RAM usage
- Device Type
- Number of devices connected
- Latency
- Packet Loss Percentage
- Connection Type

The main metric through which the decision-making process begins is the FPS metric as it is arguably the most significant factor in providing a high QoS as it affects the smoothness of the game as discussed in Chapter 3.

The final objective focused on the creation of the architecture. The proposed architecture combined Cloud computing with Fog Computing and an added component referred to as Self-Adaptation. The Cloud represents all network nodes not local to the client such as a server in another country and the Fog represents all network nodes local to the client such as the local network exchange. When a Client's FPS begins to drop it messages the server informing it of the situation. The Server (Cloud) will then see if it can help, if not then it will examine network nodes closer to the client (Fog) in relation to their resource availability and finally if there are not resources available locally then the Client will be informed that it must adapt to its situation through the reduction in the number of unessential game objects (Self-Adaptation). At each stage, there is a decision to be made such as:

1. Does the client require assistance?
2. Can the server assist?
3. Which network node can assist?
4. Which element can be reduced?

The metrics established within Chapter 3 fuel these decisions.

6.3 FUTURE WORK

The following are suggestions for future work to improve upon the completed work and extend the scope of the research:

- With the utilisation of distributed resources and spreading load, there is an argument for the possible energy saving implications of this architecture. As a client is not running at full capacity to support a low FPS and server is not expected to provide assistance to all struggling nodes there is potential to save energy.
- The Graphics element is only focused on within the Self-Adaptation component of the architecture. This could be explored further by identifying Graphics processes which can be passed off to another node. These methods can then be tested against varying network conditions.
- The proposed architecture can be explored further regarding adaptability as many clients can connect or disconnect at any given time. The local networks of assisting nodes can be strained at any given time as devices connect and disconnect to them. It is unlikely that as a client Self-Adapts that there will be no resources available at a later stage on the network, therefore the server could become aware of connecting clients and immediately assign them to assist provided they have the available resources and an acceptable connection.
- The methods developed could be integrated into a generic platform for games development. Various game types could be covered by the platform and it would adapt to different processing demands. A first-person shooter may be more focused on delivering a more intelligent AI while a third person sand-box game may be more focused on physics and graphics as the game world surrounding the player changes over time.

6.4 CONCLUDING STATEMENT

The research presented in this thesis has shown that it is possible to improve a user's QoE by focusing on improving the QoS provided through the utilisation of already present network resources.

REFERENCES

[1] Mirror, “Pokémon GO servers down AGAIN as gamers around the world vent fury on social media - Mirror Online,” 2016. [Online]. Available: http://www.mirror.co.uk/news/world-news/pokmon-go-servers-down-again-8437414.

[2] M. Claypool, K. Claypool, and F. Damma, “The effects of frame rate and resolution on users playing First Person Shooter games,” *Multimed. Comput. Netw.*, vol. 6071, pp. 607101-1-607101–11, 2006.

[3] C. Hull, D. Charles, P. Morrow, and G. Parr, “FRAGED : A Framework for Adaptive Game Execution and Delivery to Improve the Quality of Experience in Network Aware Games,” *PGNet*, 2014.

[4] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems: Concepts and Design*, 5th ed., vol. 4. 2012.

[5] I. Englander, *The Architecture of Computer hardware, System Software, and Networking*, Fourth Edi. John Wiley & Sons, Inc, 2009.

[6] “PS Now on PC | PS Now | PlayStation,” 2016. [Online]. Available: https://www.playstation.com/en-gb/explore/playstation-now/ps-now-on-pc/. [Accessed: 01-Sep-2016].

[7] W. Cai and V. Leung, “Multiplayer cloud gaming system with cooperative video sharing,” *Cloud Comput. Technol. Sci. (CloudCom), 2012 IEEE 4th Int. Conf.*, pp. 640–645, 2012.

[8] F. Lu, H. Wang, X. Ji, and G. Er, “Quality assessment of 3D asymmetric view coding using spatial frequency dominance model,” *3DTV-CON 2009 - 3rd 3DTV-Conference True Vis. - Capture, Transm. Disp. 3D Video, Proc.*, no. 60772046, pp. 1–4, 2009.

[9] G. Saygili, C. G. Gurler, and A. M. Tekalp, “Evaluation of asymmetric stereo video coding and rate scaling for adaptive 3D video streaming,” *IEEE Trans. Broadcast.*, vol. 57, no. 2 PART 2, pp. 593–601, 2011.

[10] G. Saygili, C. Gurler, and A. Murat Tekalp, “Quality assessment of asymmetric stereo video coding,” *Proc. - Int. Conf. Image Process. ICIP*, vol. 675, pp. 4009–4012, 2010.

[11] W. Cai, V. Leung, and M. Chen, “Next Generation Mobile Cloud Gaming,” *Serv. Oriented Syst. Eng. (SOSE), 2013 IEEE 7th Int. Symp.*, pp. 551–560, Mar. 2013.

[12] “Games @ Large.” [Online]. Available: http://www.cti.gr/en/activities-en/research-projects-en/item/92-games-large/92-games-large. [Accessed: 01-Sep-2016].

[13] A. Jurgelionis and F. Bellotti, “Testing cross-platform streaming of video games over wired and wireless LANs,” *Adv. Inf. Netw. Appl. Work. (WAINA), 2010 IEEE 24th Int. Conf.*, pp. 1053–1058, 2010.

[14] A. Laikari, P. Fechteler, P. Eisert, A. Jurgelionis, F. Bellotti, and a. De Gloria, “Games@ Large Distributed Gaming System,” *Proc. Networked Electron. Media Summit*, 2009.

[15] P. Eisert and P. Fechteler, “Remote Rendering of Computer Games.,” *SIGMAP*, pp. 438–443, 2007.

[16] A. Laikari, P. Fechteler, B. Prestele, P. Eisert, and J. Laulajainen, “Accelerated Video Streaming for Gaming Architecture,” *3DTV-Conference True Vision-Capture, Transm. Disp. 3D Video (3DTV-CON), 2010. IEEE*, pp. 1–4, 2010.

[17] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, Sixth Edit. Pearson Education, Inc, 2013.

[18] “Streaming media - Wikipedia, the free encyclopedia.” [Online]. Available: http://en.wikipedia.org/wiki/Streaming_media. [Accessed: 20-May-2014].

[19] “Wii U GamePad display latency is less than many HDTVs.” [Online]. Available: http://www.eurogamer.net/articles/2012-10-16-wii-u-gamepad-wireless-latency-is-less-than-many-hdtvs. [Accessed: 18-Mar-2014].

[20] “SteamOS.” [Online]. Available: http://store.steampowered.com/livingroom/SteamOS/. [Accessed: 18-Mar-2014].

[21] “NVIDIA SHIELD | Ultimate Gaming and Portable Entertainment.” [Online]. Available: http://shield.nvidia.com/. [Accessed: 18-Mar-2014].

[22] “OnLive.” [Online]. Available: http://onlive.com/.

[23] “Gaikai.com.” [Online]. Available: http://www.gaikai.com/. [Accessed: 13-Mar-2014].

[24] “Google Glass.” [Online]. Available: http://www.google.co.uk/glass/start/. [Accessed: 02-Jun-2014].

[25] J. Brandt and L. Wolf, “Adaptive video streaming for mobile clients,” *Proc. 18th Int. Work. Netw. Oper. Syst. Support Digit. Audio Video - NOSSDAV ’08*, p. 113, 2008.

[26] A. Shani, “Games@Large.” 2006.

[27] “JPerf.” [Online]. Available: http://www.techrepublic.com/blog/linux-and-open-source/using-jperf-to-check-network-performance/. [Accessed: 30-May-2014].

[28] P. Fechteler and P. Eisert, “Depth map enhanced macroblock partitioning for H. 264 video coding of computer graphics content,” *Image Process. (ICIP), 2009 16th IEEE Int. Conf.*, pp. 3441–3444, 2009.

[29] S. Stegmaier, J. Diepstraten, M. Weiler, and T. Ertl, “Widening the remote visualization bottleneck,” *3rd Int. Symp. Image Signal Process. Anal. 2003. ISPA 2003. Proc.*, vol. 1, pp. 174–179, 2003.

[30] L. Cheng, A. Bhushan, R. Pajarola, and M. El Zarki, “REAL-TIME 3D GRAPHICS STREAMING USING MPEG-4,” pp. 1–16, 2004.

[31] B. J. Gaudiosi, “Future of Cloud Gaming : Industry Leaders ’ Thoughts Future of Cloud Gaming :,” 2011.

[32] A. Ojala and P. Tyrväinen, “Developing Cloud Business Models : A Case Study,” pp. 42–47, 2011.

[33] Y. Liu and S. Dey, “Enhancing Cloud Mobile 3D display gaming user experience by asymmetric graphics rendering,” *2014 Int. Conf. Comput. Netw. Commun.*, pp. 368–374, Feb. 2014.

[34] “Most Played MMORPG Games of 2016.” [Online]. Available: http://igcritic.com/most-played-mmorpg-games-of-2016/. [Accessed: 20-Sep-2016].

[35] G. Wang and K. Wang, “An efficient hybrid P2P MMOG cloud architecture for dynamic load management,” *Inf. Netw. (ICOIN), 2012 Int. Conf.*, pp. 5–10, 2012.

[36] W. Cai, P. Xavier, S. Turner, and B. Lee, “A scalable architecture for supporting interactive games on the internet,” *PADS ’02 Proc. Sixt. Work. Parallel Distrib. Simul.*, pp. 60–67, 2002.

[37] “What is Distributed Virtual Environment (DVE) | IGI Global.” [Online]. Available: http://www.igi-global.com/dictionary/distributed-virtual-environment-dve/8097. [Accessed: 20-Sep-2016].

[38] S. Workman and G. Parr, “Modelling Event Communication to Enable Adaptive Behaviour in Resource-Constrained Distributed Virtual Environments,” *Auton. Auton. Syst. 2006. ICAS ’06. 2006 Int. Conf.*, vol. 0, no. c, 2006.

[39] B. Anand, H. Edwin, and A. Jia, “Gamelets—Multiplayer mobile games with distributed micro-clouds,” *Mob. Comput. Ubiquitous Netw. (ICMU), 2014 Seventh Int. Conf.*, pp. 14–20, 2014.

[40] J. R. Douceur and J. R. Lorch, “Enhancing game-server AI with distributed client computation,” *Proc. 17th Int. Work. Netw. Oper. Syst. Support Digit. Audio Video*, 2007.

[41] G. Soni and M. Kalra, “A novel approach for load balancing in cloud data center,” *2014 IEEE Int. Adv. Comput. Conf.*, pp. 807–812, Feb. 2014.

[42] N. Pandey, S. Verma, and V. Tamta, “Load Balancing Approaches in Grid Computing Environment.,” *Int. J. Comput. Appl.*, vol. 72, no. 12, pp. 42–49, 2013.

[43] M. Wooldridge, *An Introduction to MultiAgent Systems*. Cambridge: Cambridge University Press, 2002.

[44] W. J. Buchanan, M. Naylor, and a. V. Scott, “Enhancing network management using mobile agents,” *Proc. Seventh IEEE Int. Conf. Work. Eng. Comput. Based Syst. (ECBS 2000)*, pp. 218–226.

[45] A. Poggi and M. Tomaiuolo, “Chapter 22 Mobile Agents: Concepts and Technologies,” in *Handbook of Research on Mobility and Computing: Evolving Technologies and Ubiquitous Impacts*, 2011, pp. 343–355.

[46] M. Van Lent, J. Laird, and J. Buckman, “Intelligent agents in computer games,” *AAAI/IAAI*, 1999.

[47] C. Bobed, S. Ilarri, and E. Mena, “Distributed Mobile Computing: Development of Distributed Applications Using Mobile Agents.,” *PDPTA*, 2010.

[48] A. Mishra and a. K. Sharma, “Application of Mobile Agent in Distributed Network Management,” *2012 Int. Conf. Commun. Syst. Netw. Technol.*, pp. 930–935, May 2012.

[49] D. Lange and M. Oshima, “Seven good reasons for mobile agents,” *Commun. ACM*, vol. 42, no. 3, pp. 88–89, 1999.

[50] A. Carzaniga, G. Picco, and G. Vigna, “Designing distributed applications with mobile code paradigms,” *ICSE ’97 Proc. 19th Int. Conf. Softw. Eng.*, pp. 22–32, 1997.

[51] S. Das, “Mobile agents in distributed computing: Network exploration,” *Bull. EATCS*, no. 109, 2013.

[52] D. Gavalas and D. Greenwood, “Using mobile agents for distributed network performance management,” *Intell. Agents Telecommun. Appl.*, vol. Lecture No, pp. 96–112, 1999.

[53] Y. Jianren, H. Ruiming, C. Jun, and Z. Jianbo, “A Service-Oriented Framework of Distributed QoS Measurement Based on Multi-Agent for Overlay Network,” *2009 Int. Conf. Commun. Softw. Networks*, pp. 158–162, 2009.

[54] J. Ke, Y. Kao, and T. Lu, “MOG Platform Using Mobile Agents for Resource Transactions,” *2009 Ninth Int. Conf. Hybrid Intell. Syst.*, pp. 129–134, 2009.

[55] Y. Jin, W. Qu, Y. Zhang, and Y. Wang, “A mobile agent-based routing model for grid computing,” *J. Supercomput.*, vol. 63, no. 2, pp. 431–442, May 2011.

[56] V. Clincy and B. Wilgor, “Subjective Evaluation of Latency and Packet Loss in a Cloud-Based Game,” *Inf. Technol. New Gener. (ITNG), 2013 Tenth Int. Conf.*, pp. 473–476, Apr. 2013.

[57] K. Chen, Y. Chang, H. Hsu, and D. Chen, “On the Quality of Service of Cloud Gaming Systems,” *IEEE Trans. Multimed.*, vol. 16, no. 2, pp. 480–495, Feb. 2014.

[58] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hoßfeld, “Gaming in the clouds: QoE and the users’ perspective,” *Math. Comput. Model.*, vol. 57, no. 11–12, pp. 2883–2894, Jun. 2013.

[59] P. Mastin, “How latency is killing online gaming.” 2016.

[60] R. Serral-Gracià and E. Cerqueira, “An overview of quality of experience measurement challenges for video applications in IP networks,” *Wired/Wireless Internet Commun.*, pp. 252–263, 2010.

[61] S. Choy, B. Wong, G. Simon, and C. Rosenberg, “The brewing storm in cloud gaming: A measurement study on cloud to end-user latency,” *NetGames ’12 Proc. 11th Annu. Work. Netw. Syst. Support Games*, pp. 1–6, 2012.

[62] A. Iosup, A. Lăscăteu, and N. Ţăpuş, “Cameo: enabling social networks for massively multiplayer online games through continuous analytics and cloud computing,” *Netw. Syst. Support Games (NetGames), 2010 9th Annu. Work.*, pp. 1–6, 2010.

[63] “WoW DPS Rankings (Realistic) in Patch 5.” [Online]. Available: http://www.noxxic.com/wow/dps-rankings/realistic#BiS. [Accessed: 04-Jun-2014].

[64] S. Moller, S. Schmidt, and J. Beyer, “Gaming taxonomy: An overview of concepts and evaluation methods for computer gaming QoE,” *Qual. Multimed. Exp. (QoMEX), 2013 Fifth Int. Work.*, pp. 236–241, 2013.

[65] D. Pinelle, N. Wong, and T. Stach, “Heuristic evaluation for games: usability principles for video game design,” *CHI ’08 Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, pp. 1453–1462, 2008.

[66] R. Ewelle and Y. Francillette, “Level of detail based network adapted synchronization for cloud gaming,” *Comput. Games AI, Animat. Mobile, Interact. Multimedia, Educ. Serious Games (CGAMES), 2013 18th Int. Conf.*, pp. 111–118, 2013.

[67] Cisco, “Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are,” 2015.

[68] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, “iFogSim : A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things , Edge and Fog,” pp. 1–22, 2016.

[69] “About Us : OpenFog Consortium.” [Online]. Available: https://www.openfogconsortium.org/about-us/#introduction. [Accessed: 22-Sep-2016].

[70] S. Choy, B. Wong, G. Simon, and C. Rosenberg, “EdgeCloud: A New Hybrid Platform for On-Demand Gaming.”

[71] Y. Lin and H. Shen, “Leveraging Fog to Extend Cloud Gaming for Thin-Client MMOG with High Quality of Experience,” pp. 3–4, 2015.

[72] Y. Lin and H. Shen, “CloudFog : Towards High Quality of Experience in Cloud Gaming,” 2015.

[73] A. G. Blackburn M., “Five Ways To Reduce Data Center Server Power Consumption,” *Int. J. Prod. Res.*, vol. 45, no. 18–19, pp. 4143–4162, 1989.

[74] P. Patil, “Green Computing : To Saving Energy by Computer Virtualization,” vol. 4, pp. 6–10, 2016.

[75] “NComputing thin clients & desktop virtualization software.” [Online]. Available: https://www.ncomputing.com/. [Accessed: 29-Sep-2016].

[76] S. Sarkar and S. Misra, “Theoretical modelling of fog computing : a green computing paradigm to support IoT applications,” vol. 5, pp. 23–29, 2016.

[77] F. Cao, M. M. Zhu, and C. Q. Wu, “Energy-Efficient Resource Management for Scientific Workflows in Clouds,” *2014 IEEE World Congr. Serv.*, pp. 402–409, 2014.

[78] A. Ahmad, A. Paul, S. Member, M. Khan, S. Jabbar, M. Mazhar, U. Rathore, N. Chilamkurti, S. Member, and N. Min-allah, “Energy Efficient Hierarchical Resource Management for Mobile Cloud Computing,” vol. 2, no. 2, pp. 100–112, 2017.

[79] M. M. Hassan, M. Abdullah-Al-Wadud, A. Almogren, B. Song, and A. Alamri, “Energy-Aware Resource and Revenue Management in Federated Cloud: A Game-Theoretic Approach,” *IEEE Syst. J.*, vol. 11, no. 2, pp. 951–961, 2017.

[80] S. P. Chuah, C. Yuen, and N. M. Cheung, “Cloud gaming: A green solution to massive multiplayer online games,” *IEEE Wirel. Commun.*, vol. 21, no. 4, pp. 78–87, 2014.

[81] K. T. Claypool and M. Claypool, “On frame rate and player performance in first person shooter games,” *Multimed. Syst.*, vol. 13, no. 1, pp. 3–17, 2007.

[82] A. Leskiw, “What is SNMP? A Simple Network Management Protocol Tutorial.” [Online]. Available: http://www.networkmanagementsoftware.com/snmp-tutorial/. [Accessed: 01-Nov-2016].

[83] J. Ellingwood, “An Introduction to SNMP (Simple Network Management Protocol) | DigitalOcean.” [Online]. Available: https://www.digitalocean.com/community/tutorials/an-introduction-to-snmp-simple-network-management-protocol. [Accessed: 01-Nov-2016].

[84] J. Madigan, *Getting Gamers: The Psychology of Video Games Book*. Rowman & Littlefield, 2015.

[85] “Afterburner.” [Online]. Available: http://gaming.msi.com/features/afterburner. [Accessed: 01-Sep-2015].

[86] “Clumsy.” [Online]. Available: https://jagt.github.io/clumsy/. [Accessed: 28-Jan-2018].

[87] U. Technologies, “NetworkTransform,” 2017. [Online]. Available: https://docs.unity3d.com/Manual/class-NetworkTransform.html. [Accessed: 11-Mar-2018].

[88] “random gpu usage drops (which kill fps) (nvidia GTX 970) - [Solved] - Systems.” [Online]. Available: http://www.tomshardware.co.uk/answers/id-2504043/random-gpu-usage-drops-kill-fps-nvidia-gtx-970.html. [Accessed: 10-Oct-2016].

[89] “[Troubleshooting] GPU usage drops down to 0-10 and causes frame drops in every game. Also it’s changing all the time. Tried almost anything please suggest any fix! : buildapc.” [Online]. Available: https://www.reddit.com/r/buildapc/comments/3m265w/troubleshooting_gpu_usage_drops_down_to_010_and/?st=iypzlhd5&sh=2b3a643c. [Accessed: 10-Oct-2016].

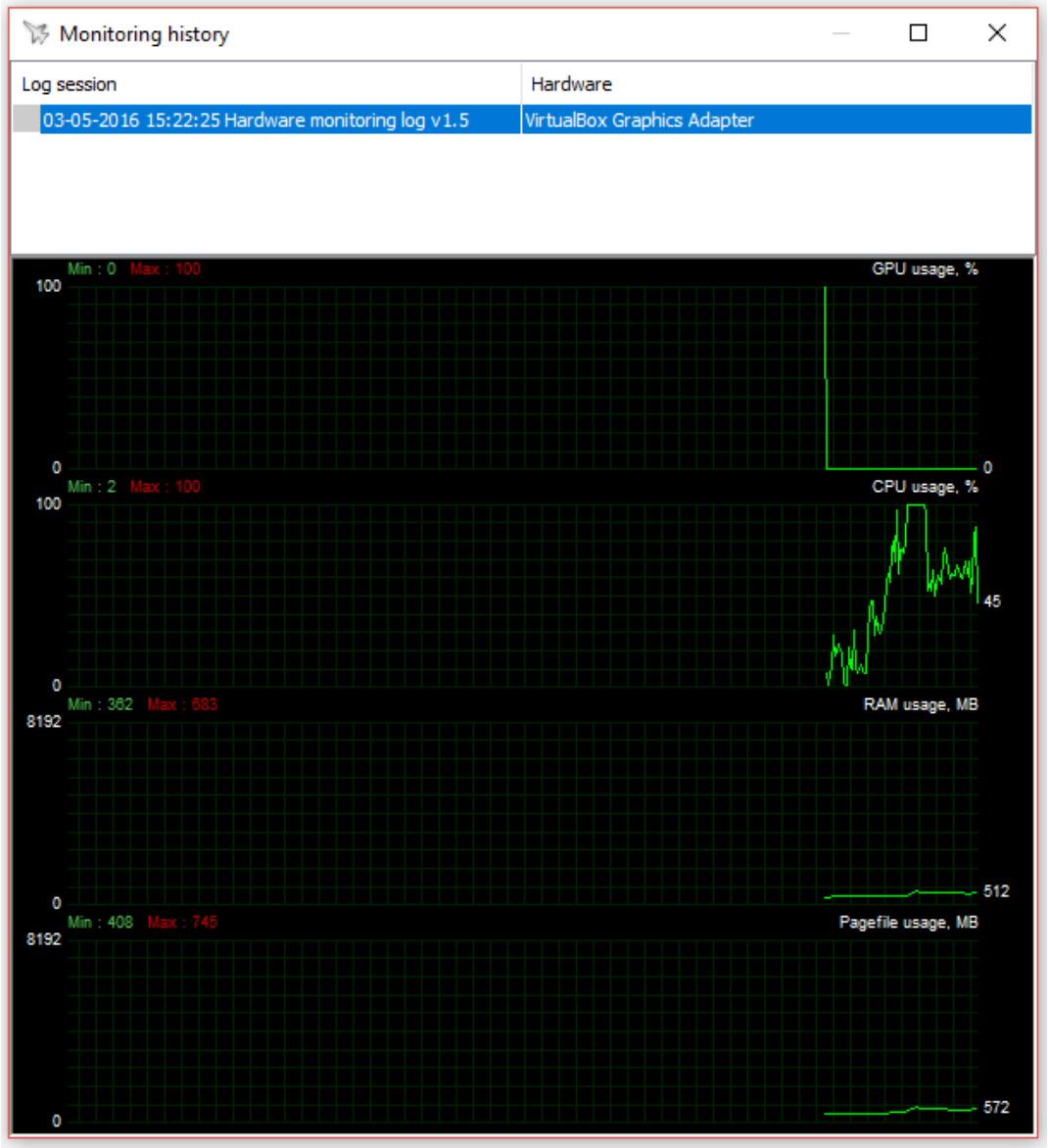
[90] “Particle system - Wikipedia.” [Online]. Available: https://en.wikipedia.org/wiki/Particle_system. [Accessed: 15-Oct-2016].

APPENDIX

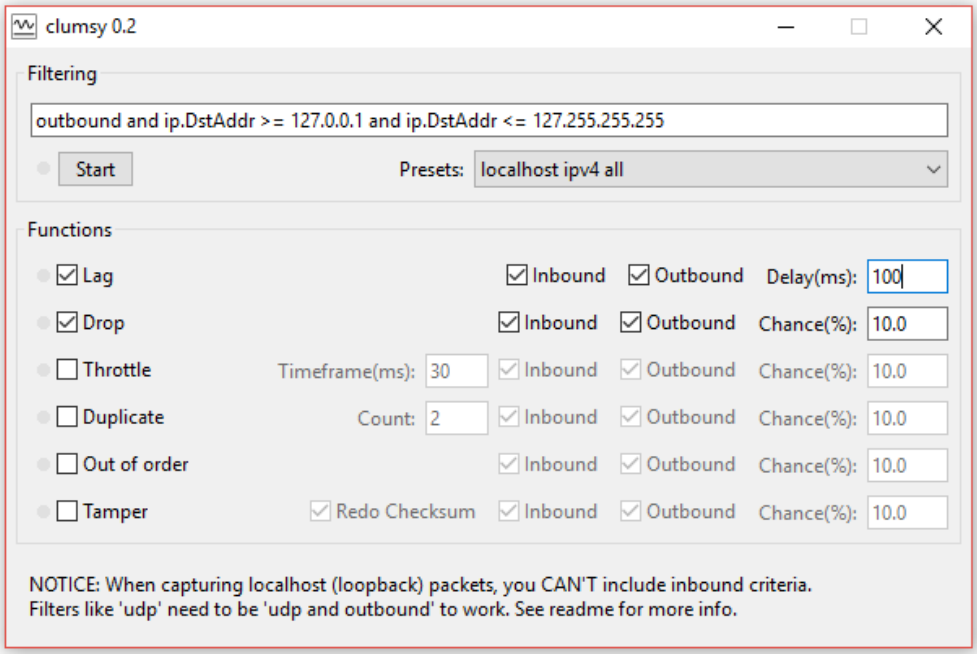
A. The code used within Unity to calculate the applications current FPS. Each update method cycle a integer value called count is incremented by 1. Every second a method called SaveCount is run which saves the current total of update cycles and resets the count to 0. After 3 seconds and then every second after that the ReadFPS method is run which takes the previous three count values and averages them. If the average is below 70 then more code will execute depending on the method of distribution.

```
void Start ()
    {

        if (isServer)
        {
            UnityEngine.Debug.Log ("I am the Server");
        }
        if (!isServer)
        {
            InvokeRepeating ("SaveCount", 1, 1);
            InvokeRepeating ("ReadFPS", 3, 1);
        }

    }

    void Update()
    {
        count++;
    }

    void SaveCount()
    {
        UnityEngine.Debug.Log (count);
        listTime.Add (""+DateTime.Now);
        listFPS.Add (count);
        count = 0;
    }

    void ReadFPS()
    {
        int total = listFPS.Count;
        int first = listFPS [total-1];
        int second = listFPS [total-2];
        int third = listFPS [total-3];

        int average = (first + second + third) / 3;

        if (average < 70)
        {

        }

    }
```

B. An example of an MSI Afterburner Output. This result is from the experiment showing the Effect of Distribution on an Unaffected Network under Full Distribution of AI Objects (Experiment 1). As can be seen the GPU Usage results are meaningless aswell as the RAM and Pagefile Usage.



C. The software package Clumsy 0.2 was used to vary network quality. The only functions used out of the six available were Lag (Latency) and Drop (Packet Loss Percentage)



D. The code used within Unity to contact the server to take control of the objects and send positional updates via the built in NetworkTransform component. If the average FPS falls below 70 then a command is sent to the server commanding it to switch on the NetworkTransform component on each object. With this on, the position of each object will be updated on the client constantly.

```
void ReadFPS()
{
    int total = listFPS.Count;
    int first = listFPS [total-1];
    int second = listFPS [total-2];
    int third = listFPS [total-3];

    int average = (first + second + third) / 3;

    if (average < 70)
    {
        //UnityEngine.Debug.Log ("less than 300");
        GameObject [] gameobjects = GameObject.FindGameObjectsWithTag("Barrel");
        File.AppendAllText(path2, "Distribution occurred: " + DateTime.Now);

        for (int i = 0; i<gameobjects.Length; i++)
        {
            Destroy(gameobjects[i].GetComponent<Rigidbody2D>());
        }

        deleteRigidbodies = true;
        sendCommand = true;
        UnityEngine.Debug.Log ("Command Sent");
    }
}

void Update () {

    if (isLocalPlayer)
    {

        if (GameObject.Find ("Main Camera").GetComponent<CameraFollow> ().sendCommand)
        {
            UnityEngine.Debug.Log ("Sending Command...");
            CmdTurnOnNetworkTransformOnServer ();
            cameraFollowScript.sendCommand = false;
        }
    }
}

[Command]
void CmdTurnOnNetworkTransformOnServer()
{
    UnityEngine.Debug.Log ("Received");

    GameObject[] allBarrels;

    allBarrels = GameObject.FindGameObjectsWithTag("Barrel");

    foreach(GameObject i in allBarrels)
    {
        i.GetComponent<NetworkTransform>().enabled = true;
    }

}
```

E. The code used within Unity to enable the transmission of Path Data from the Assisting Node to the Client. Once the FPS falls below 70, two Boolean values are changed to true. This then enables the sending of data from the assisting node to the client which enables a script called ClientAiMoveToPosArray. On the Assisting Node a Path is created for each object to follow and through a ClientRPC this path is transmitted to the object and appended to the end of a list of positions for it to visit.

```
void ReadFPS()
{
    int total = listFPS.Count;
    int first = listFPS [total-1];
    int second = listFPS [total-2];
    int third = listFPS [total-3];

    int average = (first + second + third) / 3;

    if (average < 70)
    {
        //UnityEngine.Debug.Log ("less than 300");
        GameObject [] gameobjects = GameObject.FindGameObjectsWithTag("Seeker");
        File.AppendAllText(path2, "Distribution occurred: " + DateTime.Now + Environment.NewLine);

        for (int i = 0; i<gameobjects.Length; i++)
        {

            Destroy(gameobjects[i].GetComponent<Rigidbody2D>());
            Destroy(gameobjects[i].GetComponent<Unit>());
            gameobjects[i].GetComponent<ClientAiMoveToPosArray>().enabled = true;
        }

        deleteUnitAndRigidbodies = true;
        changeUnitVariable = true;
    }

}

public void SpawnGroup(string name)
{
    File.AppendAllText(path2, name +" "+ DateTime.Now + Environment.NewLine);
    foreach (Transform child in GameObject.Find (name).transform)
    {
        GameObject.Find(child.name).GetComponent<SpriteRenderer>().enabled = true;
        GameObject.Find(child.name).GetComponent<CircleCollider2D>().enabled = true;
        if(isServer || !isServer && deleteUnitAndRigidbodies == false)
        {
            GameObject.Find(child.name).AddComponent<Rigidbody2D>();
            GameObject.Find(child.name).GetComponent<Rigidbody2D>().gravityScale = 0;
            GameObject.Find (child.name).GetComponent<Unit>().enabled = true;
        }
        else if (isServer && deleteUnitAndRigidbodies == true)
        {
            GameObject.Find(child.name).GetComponent<Unit>().sendData = true;
        }
        else if (!isServer && deleteUnitAndRigidbodies == true)
        {
            GameObject.Find(child.name).GetComponent<ClientAiMoveToPosArray>().enabled = true;
        }
    }
}
```

```
void Update () {

    if (isLocalPlayer)
    {
        if (GameObject.Find ("Main Camera").GetComponent<CameraFollow> ().changeUnitVariable)
        {
            UnityEngine.Debug.Log ("Sending Command...");
            CmdTurnOnUnitVariable ();
            cameraFollowScript.changeUnitVariable = false;
        }
    }
}


[Command]
void CmdTurnOnUnitVariable()
{
    UnityEngine.Debug.Log ("Received");
    cameraFollowScript.deleteUnitAndRigidbodies = true;
    GameObject[] allSeekers;
    allSeekers = GameObject.FindGameObjectsWithTag("Seeker");

    foreach(GameObject i in allSeekers)
    {
        i.GetComponent<Unit>().sendData = true;
    }
}

if(sendData)
{
    aiManager.PrintPath(path, transform.name);
}

public void PrintPath(Vector3 [] path, String name)
{
    if (isServer)
    {
        UnityEngine.Debug.Log("Send Array ..." + name);
        RpcPosFromServerToGoTo (path, name);
        UnityEngine.Debug.Log("Sent!!");
    }
}

[ClientRpc]
void RpcPosFromServerToGoTo(Vector3 [] path, String name)
{

    clientAiMoveToPosArray = GameObject.Find (name).GetComponent<ClientAiMoveToPosArray> ();
    clientAiMoveToPosArray.receivedArrays.AddRange(path);
    clientAiMoveToPosArray.goodToGo = true;
}
```