NGS: An Application Layer Network Game Simulator

Steven Daniel Webb
webbsd@cs.curtin.edu.au

William Lau
lauhow@cs.curtin.edu.au

Sieteng Soh
soh@cs.curtin.edu.au

Department of Computing
Curtin University of Technology
Perth, Western Australia

ABSTRACT

In the last five years the popularity of Massively Multiplayer Online Games (MMOGs) has exploded. Unfortunately, the demand has far outweighed the resources developers can provide. Many MMOGs are suffering from scalability issues, resulting in sharding, down time, and server crashes. To solve these problems, the research community is investigating peer-to-peer (P2P) overlay networks to support MMOGs, as P2P networks are theoretically and practically scalable. The majority of analysis of P2P gaming architectures has been qualitative, making it difficult to understand the strengths and weaknesses of each system. This is partially due to the lack of appropriate simulation tools. To address this problem we have developed an application layer network game simulator - *NGS* - for modelling network game architectures. *NGS* includes mechanisms to collect quantitative metrics, which may then be used to perform comparisons with other architectures. *NGS* is flexible enough to model Client/Server, Region based, Neighbour based, and hybrid architectures. It is extensible and modular, and will enable the research community to evaluate the benefits and weaknesses of existing and new network gaming architectures. Results demonstrating the extensibility and performance of NGS, and comparisons of the performance of several different architectures are included.

Categories and Subject Descriptors

I.6.8 [**Simulation and Modelling**]: Types of Simulation—*Gaming*

General Terms

Measurement

Keywords

Simulation, Peer-to-peer, MMOG

1. INTRODUCTION

Massively Multiplayer Online Games (MMOGs) differ from traditional network games as they present a single universe in which thousands or tens of thousands of players interact simultaneously [17]. In the last five years the popularity of MMOGs has increased dramatically; enabled by the explosive growth of the Internet and the availability of broadband connections for home users. Unfortunately, current architectures have limited scalability [5], requiring developers to provision large amounts of hardware and bandwidth. Furthermore, as more resources cannot be deployed rapidly, developers often over-provision resources, to allow for a rapid grow in the number of players [6]. This prevents small companies from developing MMOGs, and even some large developers struggle to provision enough resources [25].

The vast majority of commercial MMOGs available today use a centralised architecture. While more servers can be added, the demand placed upon the system by the users has polynomial grow, resulting in a bottleneck [1]. Centralised architectures are not scalable beyond tens of thousands of players, far below the market potential. A scalable architecture can support a large number of concurrent players, and can tolerate a rapid increase in the number of players without dramatically increasing the usage of centralised resources.

The most common approach used by game developers to solve scalability issues is *sharding* [3]. A shard is a complete and independent copy of the game world. The maximum number of concurrent players in a shard is bounded. Players in different shards cannot interact. By adding more shards the developer can accommodate more players; however, every shard remains independent.

World of Warcraft (WoW) is arguably the most popular MMOG to-date, with over 6 million players world wide [25], and peak loads of several hundred thousand concurrent players [22]. The WoW universe is sharded into many mutually exclusive worlds. Each shard is limited to several thousand concurrent players. Despite the massive success of WoW and the huge revenue it is generating, WoW has been plagued with scalability issues [25]. Shards rapidly reach their player limits, resulting in long queues of players waiting to join the shard. Several quests that result in a large number of players generating a large number of events have been known to crash the server.

In recent years Peer-to-Peer (P2P) overlay networks have become a hot research topic [21]. Their primary advantage over centralised architectures is their scalability. For every node that joins the system and makes requests, the node also provides resources to the system to handle the requests of other nodes. There are real-world P2P systems that can scale up to millions of concurrent users [21].

Researchers are developing P2P overlays specifically to handle the requirements of network games, such as: SimMud [17], Zone Fed [14], P2P-MES [15], MIP [7], Solipsis [16], VON [12], Federated P2P [20], MOPAR [26], etc. In the context of network games, each player can be considered a P2P node, providing resources to the system. Unfortunately, there is very little comparison between architectures and it is difficult to evaluate the strengths and weaknesses of each architecture, such as bandwidth, processing requirements, and the latency introduced by the system.

To allow quantitative comparisons of new architectures, we have developed an application layer Network Gaming Simulator (NGS). This simulator was designed to allow rapid prototyping of architectures, with metric collection for evaluation. To reduce the development time of new architectures NGS is written in Java, enabling object orientation and garbage collection. NGS does not simulate the network stack, as it would add unnecessary processing time and memory usage, and only provide an insignificant increase in accuracy.

The remainder of this paper is organised as follows: Section 2 provides a background into the different types of P2P architectures currently available, and highlights the difficulties in making comparisons between them. This section also provides an overview of current simulators for network architectures. Section 3 outlines the design of NGS. Section 4 demonstrates the extensibility of NGS, provides performance metrics, and presents initial results of simulating several different architectures. Section 5 concludes the paper and outlines areas of future development for the simulator.

2. BACKGROUND

2.1 Network Gaming Architectures

Table 1 provides a taxonomy of current network game architectures. All architectures are classified as either Centralised, Distributed (P2P), or hybrid - depending on where the game state is stored. Centralised and distributed architectures store the game state on centralised servers and client machines respectively. Hybrid architectures store the game state on both centralised and client machines.

A few early network games utilised P2P architectures; however, the use of flooding to distribute updates made them unscalable [17]. Client/Server has been the most common architecture for network games for over a decade - generally supporting between 8 and 64 players. It will continue to be the most common architecture for games with small numbers of players such as First Person Shooters (FPS); however, this architecture is not scalable and cannot handle more than a few hundred players.

To handle large numbers of concurrent players developers use clusters of computers, also known as Federated Client/-Server. This is the approach used by EVE Online [3], which holds the record for the greatest number of concurrent players in one shard: 23000. Each computer in the cluster is responsible for managing a different portion of the virtual world. Load balancing transfers players between computers in the cluster, keeping every machine's processing and bandwidth requirements from reaching saturation. Federated Client/Server is the most common architecture for commercial MMOGs; however, hardware resources can only grow linearly due to economic reasons, while the required resource growth is polynomial to the number of simultaneous players. This limits the maximum number of concurrent players per shard, typically less than 10000.

P2P systems are both theoretically and practically scalable, as every node that joins the network and starts making requests also provides resources to the system - processing power, bandwidth, storage, etc. If the resource requirements of every node are bounded the system is infinitely scalable [12]. P2P network game architectures are categorised as either: *neighbour based*, *region based*, or *hybrid* [9].

Neighbour based schemes are true P2P systems where every node has the same responsibilities. Every node is responsible for sending updates about events to its neighbours. Neighbour based architecture use either *neighbour-list exchange* - such as P2P-MES [15] and MIP [7] - or *mutual notification* - such as Solipsis [16] and VON [12]. The bandwidth required for neighbour based schemes is a function of the density of avatars, rather than the nodes in the system. Neighbour based systems can achieve excellent scaling; however, they may suffer from network partitioning. The effect of network partitioning is that a node moves within another node's Area of Interest (AoI), but neither node is made aware of the other. If partitioning occurs the game mechanics will not execute correctly.

Region based techniques divide the virtual world into geometric shapes - usually rectangles or hexagons [26]. The regions may be dynamic or static; however, most approaches use static regions for simplicity, even though dynamic regions offer superior load balancing. For every region a coordinator or super-peer is assigned. This node is responsible for managing players entering and leaving the region. The regions are used to control the AoI of players. Events generated by avatars in a region will only be propagated to other nodes within that region - except when a node is moving between regions, in which case the event is propagated to both regions. The coordinator's avatar does not need to be a member of the region, allowing high-capacity nodes to be selected as coordinators.

Region based P2P systems often use a lookup mechanism such as a Distributed Hash Table (DHT) [9]. This makes maintaining connectivity far easier than in neighbour based systems, but requires additional overhead to run the DHT.

Region size is an important factor when developing a region based architecture. The smaller the region the tighter the match to a players AoI; therefore, fewer unnecessary updates will be sent to the player. However, this results in avatars moving between regions rapidly, which incurs a large overhead as transitions require using the lookup mechanism of

Table 1: Taxonomy of current network game architectures.

Classification	Type	Examples
Centralised	Client/Server	Quake [13], Torque [11]
	Federated Array (cluster)	World of Warcraft (WoW) [2], EVE Online [3]
Distributed (P2P)	Neighbour Based	VON [12], Solipsis [16], P2P-MES [15], MIP [7]
	Region Based	SimMud [17], Zone-Fed [14]
	Hybrid	MOPAR [26]
Hybrid		Federated P2P [20]

the DHT. The region size should approximate the players AoI, without producing excessive DHT overhead.

2.2 Problems

Most analysis of architectures is qualitative, making direct comparisons very difficult. Furthermore, fundamental differences pose questions about the validity of any comparison. Three examples are: when comparing neighbour based architectures with region based architectures how does AoI compare with region size? End System Multicast (ESM) allows high capacity nodes to forward messages onto multiple receivers for low capacity nodes, reducing the required capacity for the sender [8]. By reducing the required bandwidth, larger regions can be used, reducing the overlay overhead; however, ESM introduces latency into the system. Most region based architecture use ESM, while most neighbour based architecture do not. Can a meaningful comparisons be made between an architectures using ESM and one without? Does the mobility model favour one architecture over another? NGS was developed to help answer these questions.

2.3 Previous work on simulation

There are several excellent network simulators already available to the research community [4]. These simulators accurately model all layers of the protocol stack, and are designed to aid the development of new protocols, and model the interaction of protocols between layers. However, the processing and memory requirements of simulating the entire protocol stack are considerable, greatly reducing the scalability of the simulator. Furthermore, developing network gaming architectures using current network simulators is difficult due to the need for careful memory management, and the complexity of debugging a protocol stack. Moreover, network game architecture developers are only interested in application layer events, and are not concerned with the details of the underlying protocols.

Most network games researchers creating new architectures develop a specific simulation prototype, rather than using existing tools, which shows that current simulators are inappropriate. Simulation prototypes are designed specifically for the intended application, making them very inflexible and unsuitable to simulate other architectures. There are several open-source game networking libraries such as the Torque Network Library (TNL) [11], and the Quake source code [13]. However, these libraries are very complex to use and debug, and not suitable for simulating all architectures. To effectively model thousands of nodes requires a *flexible* and *light-weight* application layer simulator.

3. NGS: NETWORK GAME SIMULATOR

3.1 Goals

Our objective is to develop a single framework/simulator that captures quantitative metrics, such as bandwidth and latency, which can be used to compare different architectures. NGS can be used to evaluate new architectures and features. To meet this objective, NGS must be: (1) Flexible - able to model centralised, distributed, and hybrid architectures; (2) Extensible - allow new architectures & features to be easily incorporated; (3) Modular - allow components to be interchanged for comparison; (4) Simple - allow researchers to rapidly prototype without concern for details; (5) Scalable - able to run simulations with thousands of nodes.

3.2 Design

A simulation consists of a series of nodes that interact. Every node contains several modules that determine its functionality and behaviour. The modules loaded are specified by command line arguments, allowing multiple simulations to be run in parallel without re-compilation or consistency issues. Figure 1 shows the components of a node. The core of a node is the Manager, which is responsible for loading the required modules and passing information between them. Every type of module - Application Layer Router, Mobility Model, Compressions, etc. - has a corresponding Factory class to construct objects of the correct type. To develop a new module the appropriate abstract class is extended and 6 lines of code are added to the Factory.

Every node requires an Application Layer Router (ALR) module, to communicate with the other nodes in the system. The ALR is specific to the architecture being simulated, and may provide different functionality depending on the responsibility of the node - client, server, peer, etc. Clients send events to the server and receive updates, servers receive events and propagate them to interested nodes, and peer nodes broadcast events and maintain the network overlay.

Every avatar requires a mobility model to generate movement; while nodes without a virtual presence - such as servers - do not. Different compression modules may also be loaded such as Delta Encoding (Dead Reckoning) and Message Aggregation. This design allows new modules to be developed without affecting the system, and new types of modules to be incorporated into the system (Modules A, B, ..., Z).

3.3 Implementation

The current modules implemented are: Client/Server, VON, DHT, DHTMultiRegion, Random Walk, and NGMM.

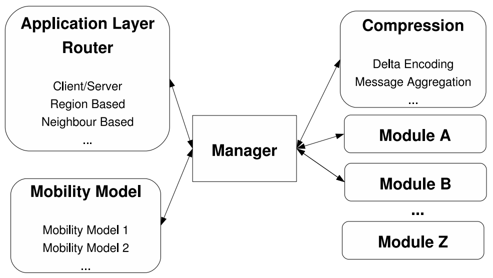


Figure 1: The components that provide the node functionality

The Client/Server module implements the traditional client server architecture. The first node constructed acts as the server and all other nodes are clients. The clients update the server every time they perform an action. When the server receives an update it uses Euclidean Distance to calculate the AoI and only update interested clients.

We selected *VON* as a representative neighbour based architecture. *VON* has a strong mathematical base by using voronoi diagrams [12] to dynamically divide the world into regions. The voronoi diagram is used to determine which nodes must be sent event updates. We selected *VON* as it does not suffer from partitioning, which is disastrous for MMOGs. The authors have developed an open-source implementation of *VON* called *VAST* [24], which we have modified to make it compatible with *NGS*. *VAST* supports a dynamic AoI; however, we disabled this for our comparison as none of the other architectures support a dynamic AoI. In addition to this, dynamic AoI is only allowable for certain types of games, based upon the game mechanics/play.

When developing a region based architecture we use a DHT to maintain global connectivity. Although there are several open-source implementations of DHTs [17], we decided that using one would require considerable effort compared to the benefits. Instead we made an assumption that a DHT mechanism is available, and it requires $log_2(n)$ messages to reach any other node in the overlay, where n is the number of nodes in the overlay. The $log_2(n)$ messages are distributed randomly to all nodes in the DHT. This assumption is reasonable as there are several DHTs that offer $log_B(n)$ worst case performance - where B is a constant - such as Tapestry and Pastry [19].

The world is divided into rectangular regions of fixed size. The region that a node is occupying is its AoI. For every region there is a coordinator which is responsible for sending and receiving notifications about nodes entering and leaving the region. Each region uses End System Multicast (ESM) [8] to propagate updates to all interested nodes. The coordinator is responsible for building and maintaining the multicast graph for nodes joining and leaving the region. The coordinator's avatar does not need to be located within the region, making it possible to select high bandwidth nodes to be coordinators; however, the simulator currently assumes that all nodes have equal bandwidth.

When a node joins the simulation it selects a random region and contacts the coordinator. If this is the first node in the simulation it becomes the coordinator for every region in the simulation. If the new node is not the first node, the coordinator responds by dividing the number of regions it controls in half and assigning responsibility of half to the newly joined node, balancing the load across both nodes. The greater the number of regions a node is controlling the greater the probability that a node entering the system will take some of the workload. If the existing coordinator only controls one region the new node does not become the coordinator for any regions. Only coordinators are members of the DHT to reduce the hop count of queries.

DHTMultiRegion is an extension to DHT, allowing nodes to subscribe to multiple regions. This allows nodes to receive updates from adjacent regions, presenting the player with a seamless world.

Previous studies have found that over 80% of events generated by network games are movement updates [23]; therefore, when developing a simulator we are primarily interested in movement updates. *NGS* currently has two mobility models: *Random Walk* and *NGMM*. Every time *Random Walk* is executed the current (x, y) coordinates are updated by two randomly generated distances - Δx & Δy to create the new location of the avatar (x', y'). Random Walk can be considered a sequence of random steps. *NGMM* is a statistical mobility model based on 100 Quake game traces. A comprehensive discussion is found in [23].

To keep the comparison between architectures simple, compression modules for delta encoding and message aggregation have not been implemented for *NGS*.

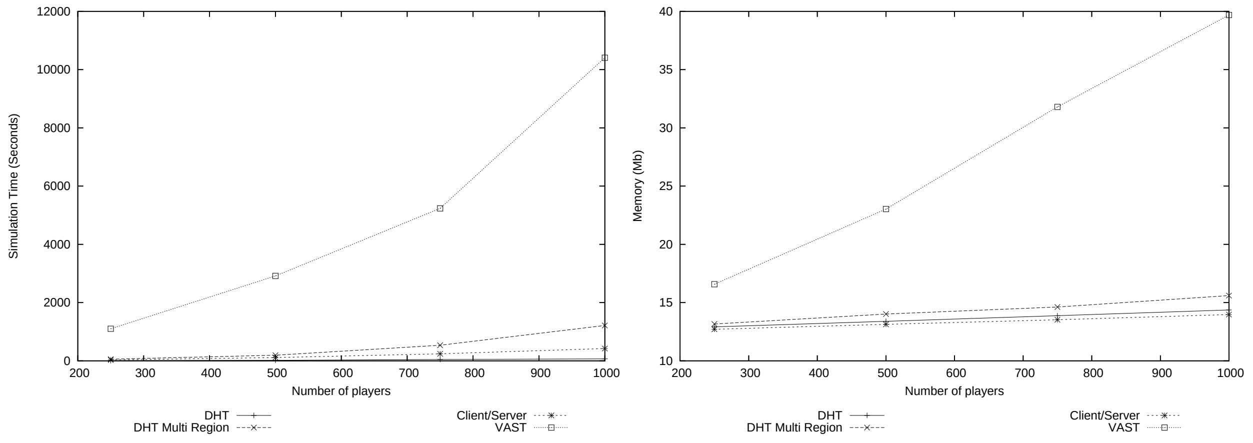
4. RESULTS AND DISCUSSION

The results presented in this section use the following constants. All units are relative. The world size is 1000 × 1000 units. The total time simulated was 1000 seconds. The *Random Walk* mobility model was used. Every node is capable of up to five connections, as the average bandwidth required per node is 40Kbps [10], and the average available upstream bandwidth is 212Kbps [18]. The AoI has a 50 unit radius. For the remainder of this paper, n is the number of players in the simulation. Except for the server node in the Client/Server architecture, every node corresponds to one player; therefore, when discussing the complexity of each game architecture, the terms player and node are interchangeable.

4.1 Extensibility of the Simulator

The DHT architecture presented in Section 3.3 is unsuitable for some network games as it does not present the user with a seamless world. When a player is crossing into a new region they would not be aware of avatars that are within their AoI, but located in an adjacent region. If region transitions should not influence game mechanics a seamless world is necessary.

To achieve a seamless world players must subscribe to regions adjacent to the one their avatar is in [17] (DHTMultiRegion). By subscribing to the eight adjacent regions, players receive updates about events in neighbouring regions,



(a) Simulation time (b) Memory

Figure 2: Simulation times and memory requirements for NGS architectures

and transitions between regions will not cause avatars to magically appear and vanish on the players screen.

Due to the extensibility of NGS only trivial changes were required to implement DHTMultiRegion. The DHT class is extended to allow nodes to join multiple regions and the ApplicationLayerRouterFactory is updated for the new class.

As there is no advantage in implementing the mobility model directly in Java, we implemented a trace driven mobility model that reads a trace file for every avatar and moves it accordingly. This allows other mobility models, such as *NGMM* [23], to be used by generating new traces, and also allows different avatars to use different mobility models within a simulation. This extension was trivial to implement due to the use of inheritance in the simulator.

We have shown that adding new architectures, mobility models, and features is a trivial task. This is supported by the ease in which the VAST source code was incorporated into NGS.

4.2 Efficiency of the Simulator

To demonstrate the efficiency of NGS we ran simulations with varying numbers of players. The real-world time taken to simulate each architecture for 1000 seconds is shown in Figure 2(a). The system was an Intel Pentium 4 - 3.4GHz, with 512MB of RAM, running Fedora Core 4. The processing required per node is dependent on the number of avatars within its AoI; therefore, the time growth for the simulator is polynomial compared to the number of nodes. Figure 2(a) demonstrates that NGS is very light weight - able to simulate the DHT architecture with 1000 nodes in only 69 seconds - and that the time to run a simulation is directly dependent on the complexity of the architecture. We have successfully run simulations of 10000 nodes using the DHT architecture.

Figure 2(b) shows the memory requirements for simulating each architecture. The DHT, DHTMultiRegion, and Client/Server architecture scale extremely well as they were specifically developed for NGS. VAST requires considerably more memory for its internal data structures; however, it still scales approximately linearly with respect to the number of nodes.

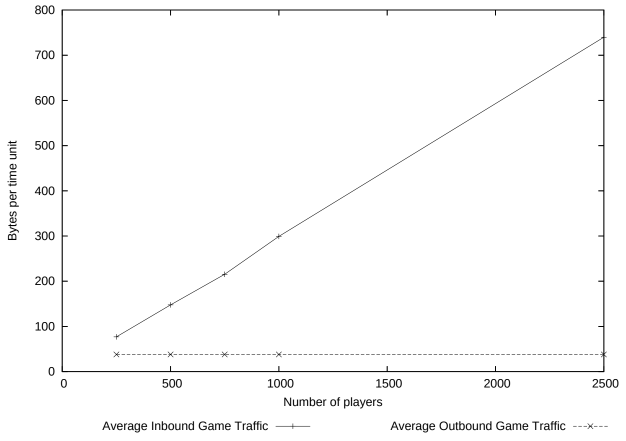
4.3 Architecture Comparison

In this section we present some initial results from using the simulator. It should be noted that the results should not be considered as exact values, as there would be unforeseen additional overhead in a real application. The shape of each graph is of greater interest, as it should accurately demonstrate the scalability of a system.

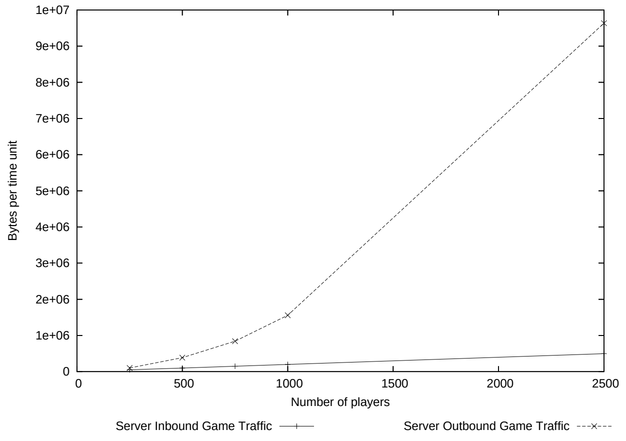
Figure 3 shows the bandwidth requirements for the client and server game traffic in a Client/Server architecture. As noted by other researchers [1], the bandwidth requirement for clients is $O(n)$; however, the server bandwidth increases polynomial to the number of players. Even when using AoI filtering, every update must be forwarded to all other nodes within the AoI. As the number of players grows, so does the number of players that must receive updates; hence, the bandwidth is $O(n^2)$. Using a federated client/server architecture only provides a temporary solution, as adding hardware only provides a linear increase in resources - insufficient for hundreds of thousands of players.

Examining the results for the DHT architecture in Figure 4, it is clear that game state propagation requires far more bandwidth than signalling traffic. ESM successfully limits the number of connections per node to five; however, it also introduces considerable delay for propagating updates. This is compounded by the absence of a self-improvement algorithm for the multicast graph, resulting in $O(m)$ worst case performance, where m is the number of nodes in a region. The *Optimal Hops* line is the theoretical minimum number of hops for a region with the corresponding number of nodes. While the DHT system is scalable, the long propagation delays do not make it practical for most real-time applications such as games.

Figure 5 compares the bandwidth and latency for DHT and

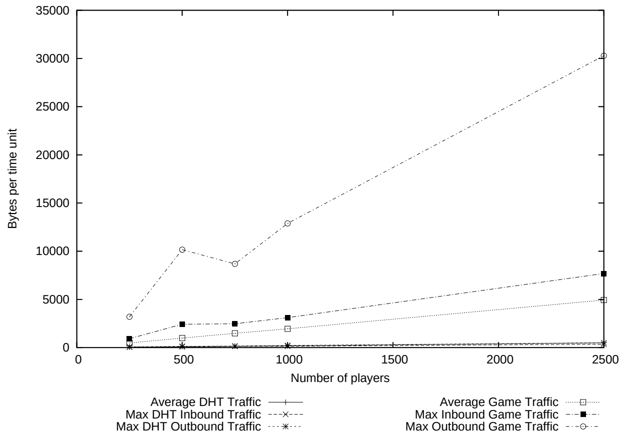


(a) Client Bandwidth

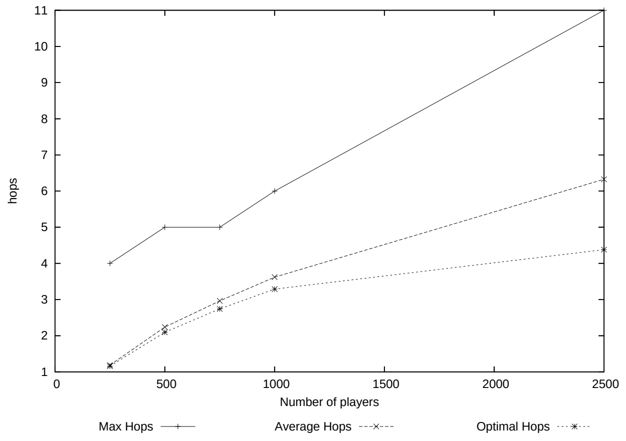


(b) Server Bandwidth

Figure 3: The game traffic requirements for a Client/Server architecture

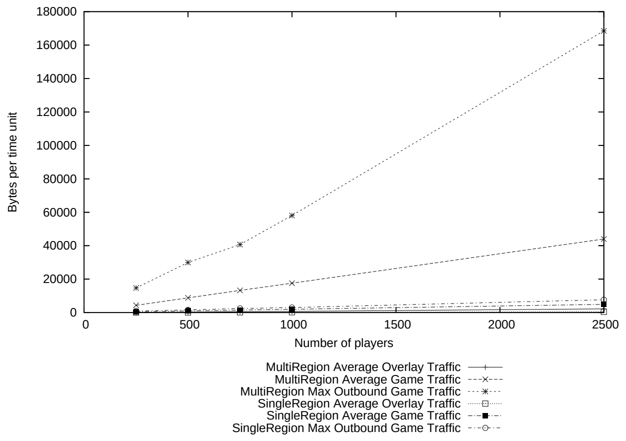


(a) Overlay and Game Traffic

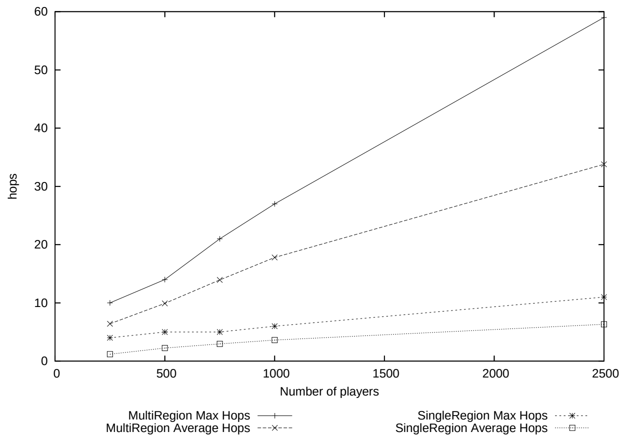


(b) Multicast propagation delay

Figure 4: Bandwidth requirements and latency for the DHT architecture

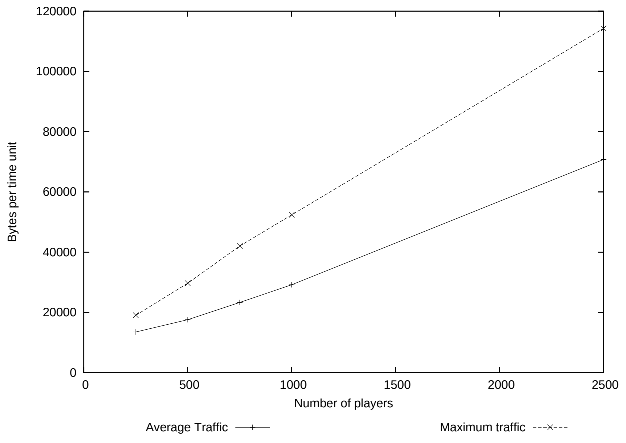


(a) Overlay and Game Traffic

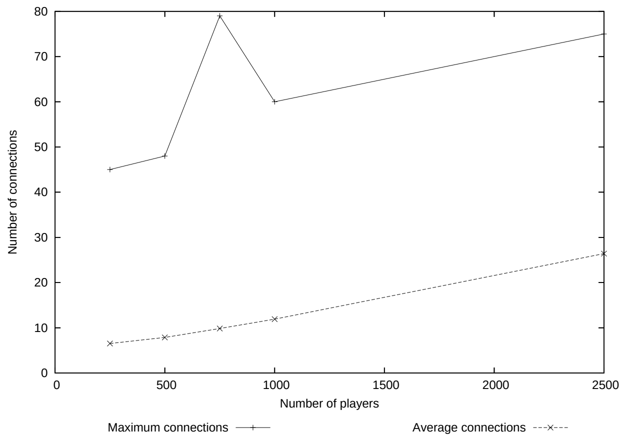


(b) Multicast propagation delay

Figure 5: Bandwidth requirements and latency comparison for single and multi-region architectures



(a) Game Traffic



(b) Connections made

Figure 6: Bandwidth requirements and the number of connections made for the VAST Architecture

DHTMultiRegion. As expected, the DHTMultiRegion architecture generates nine times the game traffic generated by the DHT architecture; and the average latency increases by approximately 1.55, as every node propagates updates received to four of its neighbours.

VAST does not support any form of ESM or message aggregation. Every node must connect to every other node within its AoI. This results is very low latency (1 hop), but unrealistically high connection rates. The connection rates in Figure 6 far exceed the capabilities of typical broadband hosts [18], making VAST unscalable. The authors [12] attempted to address this by using a variable AoI; however, this is not a suitable solution for many games.

5. CONCLUSIONS

We have developed *NGS*, an application layer Network Game Simulator, and have successfully simulated *Client/Server*, *Region based*, and *Neighbour based* architectures. The simulator is very flexible and can easily be extended to include new features and architectures. This is primarily due to the use of modules and Object Orientation, as shown by the extension of the DHT architecture, the development of the trace driven mobility model, and the incorporation of the VAST source code with only minor modifications. As NGS does not simulate the protocol stack it is very simple. There are no complex routing algorithms or APIs to learn, and debugging is easy. Finally, we have run simulations of up to 10000 nodes, and there is no technical reason preventing simulations with more than 10000 nodes.

We are currently performing a full comparison of network game architectures. This will involve determining a set of metrics that can be used for a fair comparison across architectures. This is not a trivial task due to the fundamental differences between architectures.

Currently processing overhead is not captured by the simulation. This is not important for current desktop machines; however, it may be critical for other devices such as mobile phones and wireless sensor networks.

To encourage applicable research we intend to implement topology features to more accurately measure latency, and promote the development of architectures that minimise the latency between nodes, rather than just the end system hop count. This will also include modelling nodes with different link qualities, an important consideration for ESM. The topology will be read in from a file generated by a topology generator.

While we have only performed a preliminary evaluation it appears that the greatest factor effecting scalability is the multicast mechanism used to propagate updates. The Client / Server model places $O(n^2)$ requirements on the server bandwidth and processing power, making it fundamentally unscalable. For P2P systems - whether they are region based or neighbour based - an ESM method must be developed that has very low latency, while remaining within the bandwidth requirements of the nodes. Until this is achieved P2P systems may not be scalable enough for many types of real-time applications and games.

6. ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their helpful suggestions.

7. REFERENCES

[1] T. Alexander, editor. *Massively Multiplayer Game Development 2*. Charles River Media, Inc, Hingham, Massachusetts, 2005.

[2] Blizzard Entertainment. World of Warcraft. *http://www.worldofwarcraft.com/*.

[3] D. Brandt. Networking and scalability in EVE Online. Slide Show, Oct 2005. *http://www.research.ibm.com/-netgames2005/papers/brandt.pdf*.

[4] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. Advances in network simulation. *IEEE Computer*, 33(5):59–67, May 2000.

[5] F. R. Cecin, R. Real, R. de Oliveira Jannone, C. F. R. Geyer, M. G. Martins, and J. L. V. Barbosa. FreeMMG: A scalable and cheat-resistant distribution model for Internet games. In *IEEE Int. Sym. on Distributed Simulation and Real-Time Applications*, pages 83–90, Oct 2004.

[6] C. Chambers, W. Feng, S. Sahu, and D. Saha. Measurement-based characterization of a collection of on-line games. In *Internet Measurement Conf.*, 2005.

[7] J.-F. Chen, W.-C. Lin, H.-S. Bai, and S.-Y. Dai. A message interchange protocol based on routing information protocol in a virtual world. In *Int. Conf. on Advanced Information Networking and Applications*, pages 377–384, Mar 2005.

[8] Y. Chu, S. G. Rao, and H. Zhang. A case for end system multicast (keynote address). In *Proc. ACM SIGMETRICS int. conf. on Measurement and modeling of computer systems*, pages 1–12, 2000.

[9] S. Douglas, E. Tanin, A. Harwood, and S. Karunasekera. Enabling massively multi-player online gaming applications on a P2P architecture. In *Proc. IEEE Int. Conf. on Information and Automation*, pages 7–12, 2005.

[10] W. Feng, F. Chang, W. Feng, and J. Walpole. Provisioning on-line games: a traffic analysis of a busy counter-strike server. In *Proc. ACM SIGCOMM Workshop on Internet measurment*, pages 151–156, 2002.

[11] GarageGames. Torque Network Library (TNL). *http://www.opentnl.org/.*

[12] S.-Y. Hu, J.-F. Chen, and T.-H. Chen. VON: A scalable peer-to-peer network for virtual environments. *IEEE Network*, 20(4), July/August 2006.

[13] id Software. *http://www.idsoftware.com/.*

[14] T. Iimura, H. Hazeyama, and Y. Kadobayashi. Zoned federation of game servers: a peer-to-peer approach to scalable multi-player online games. In *Proc. ACM SIGCOMM workshop on Network and system support for games*, pages 116–120, 2004.

[15] Y. Kawahara, T. Aoyama, and H. Morikawa. A peer-to-peer message exchange scheme for large-scale networked virtual environments. *Telecommunication Systems*, 25(3):353–370, Mar 2004.

[16] J. Keller and G. Simon. Solipsis: A massively multi-participant virtual world. In *Parallel and Distributed Processing Techniques and Applications*, pages 262–268, 2003.

[17] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. In *Joint Conf. IEEE Computer and Communications Societies*, volume 1, page 107, March 2004.

[18] K. Lakshminarayanan and V. N. Padmanabhan. Some findings on the network performance of broadband hosts. In *Proc. ACM SIGCOMM conf. on Internet measurement*, pages 45–50, 2003.

[19] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys & Tutorials, IEEE*, pages 72–93, 2005.

[20] S. Rooney, D. Bauer, and R. Deydier. A federated peer-to-peer network game architecture. Research report, IBM Research GmbH, Zurich Research Laboratory 8803 Ruschlikon Switzerland, Jan 2004.

[21] D. Stutzbach, R. Rejaie, and S. Sen. Characterizing unstructured overlay topologies in modern P2P file-sharing systems. In *Internet Measurement Conf.*, pages 49–62, 2005.

[22] T. Surette. World of Warcraft sells more than 600,000 units, Jan 2005. *http://www.gamespot.com/pc/rpg/-worldofwarcraft/news.html?sid=6116075.*

[23] S. A. Tan, W. Lau, and A. Loh. Networked game mobility model for first-person-shooter games. In *Proc. ACM SIGCOMM workshop on Network and system support for games*, pages 1–9, 2005.

[24] V. D. Team. VAST: VON-based Adaptive Scalable Transfer. *http://vast.sourceforge.net/.*

[25] D. Terdiman. World of Warcraft battles server problems, Apr 2006. *http://news.com.com/World+of+Warcraft+battles-+server+problems/2100-1043_3-6063990.html.*

[26] A. P. Yu and S. T. Vuong. MOPAR: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games. In *Proc. int. workshop on Network and operating systems support for digital audio and video*, pages 99–104, 2005.