Game network traffic simulation by a custom bot

Trevor Alstad, J. Riley Dunkin, Simon Detlor, Brad French, Heath Caswell, Zane Ouimet, Youry Khmelevsky, Gaétan Hains

2015 Annual IEEE Systems Conference (SysCon) Proceedings. IEEE, 2015. p. 675-680.

Game Network Traffic Simulation by a Custom Bot

Trevor Alstad, J. Riley Dunkin, Simon Detlor,
Brad French, Heath Caswell, Zane Ouimet,
and Youry Khmelevsky
Computer Science Department, Okanagan College
Kelowna, BC V1Y 9L2, Canada
Emails: trevor.alstad@myokanagan.bc.ca
john.dunkin13@gmail.com, simon.detlor@myokanagan.bc.ca
brad.french@myokanagan.bc.ca, heath.caswell@gmail.com
zane.ouimet@outlook.com, and khmelevsky@gmail.com

Gaétan Hains*
Lab. d'Algorithmique, Complexité et Logique
Université Paris-Est Créteil
Paris, France
Email: gaetan.hains@gmail.com
*Also Affiliated with LIFO, Université d'Orléans, France

***Abstract*—Minecraft is a popular video game played worldwide, and is built simply enough to be used for network analysis and research. This paper describes an automated software agent created to simulate player traffic within the game. Realistic network traffic simulation was the goal that inspired the creation of our "Minecraft bot": an automatic program or *bot* that could act in similar ways to a real player, and be able to be mass produced to saturate a local area network. This will facilitate network research by allowing users to have a more scalable testing environment and thus enable controlled laboratory experiments that are impossible to set up in live online gaming environments. The basic commands in Minecraft consist of moving, placing and breaking blocks (pieces of environment) and a realistic bot needs to replicate these actions. Another important objective was to have the ability to create hundreds or thousands of bots doing the same actions, to be able to create artificial latency on the network. This paper will go through the entire lifecycle of our project, starting with some information on existing research about the subject, and how it relates to ours. Following that we describe our bot requirements, the work that was done to find a pre-built solution, the solution we ended up using and how it was modified to fit our requirements. We then have a section showing performance experiments we ran, which compared the packet count and traffic volume between players and bots, as well as cpu usage statistics as more connections were made to the server to ensure that our server hardware was not a factor in our network testing. The final section is the conclusion which talks about the outcome of our project in relation to our original goals, and how it will impact future research in this area.**

I. Introduction

Minecraft is a popular video game played worldwide, and is built simply enough to be used for network analysis and research. This paper describes an automated software agent created to simulate player traffic within the game, which was developed to investigate a simulated virtual infrastructure of the game private networks project.

The project was the first step in the design, construction and test of a new layer of game server software than can optimize and monitor in real-time game services. It stems from the observation that game servers place demands on computing resources - hardware and network - that can vary with user behaviour and whose optimization is the key to customer satisfaction. Virtualized servers provide new flexibility in hardware reservation and allocation but their use can make resource optimization difficult by making it context sensitive i.e. dependent on the allocation of virtual machines (VMs) to hardware.

The project's objective was to study predictive monitoring and optimization for game server clusters. The first phase of the project was to gather performance data about game servers, then analyze its time behaviour to allow the creation of a performance-prediction software module. The initial module version applied virtualized game servers in various configurations, and later versions were tested with physical servers as well as parallel (cluster) game servers. Later, the project will investigate performance optimization based on short-term predictions.

We have built an experimental setup to demonstrate the reproduction of realistic game network traffic by artificial players called bots. The method was to first measure the effect of a few human players, on traffic volume and number of packets. This was tested under many controled situations and the resulting data scritinized to extract and explain regular patterns. The same controled situations were then systematically applied to a growing number of bots, to verify that their behaviour was similar to that of human players. The larger number of bots served as extrapolations for as many human players, something that is either too expensive or impractical to test in such a rigorous environment. The main results of this work are a set of detailed correlations between experiments and traffic, plus a general confirmation that bots can be used to recreate realistic game traffic. Future work will be able to use them is large numbers and much more complex situations leading to finer analyses.

Our main contributions are: (1) a unique network and gaming servers infrastructure created for the simulation experiments, which is also being used to perform stress testing and data analysis of network game applications, as well as to monitor the performance of game servers; (2) the simulation gaming bot that is described in this paper, which was developed as a tool for use in our test infrastructure. It allows the generation of game traffic for thousands of players in a predictable and controllable way for our experiments, without the need for recruiting that many actual people to play. (3) Using this new tool we are able to design performance-prediction models for game servers optimization as well as for networking optimization research, and for use in industry (this is our current ongoing research). This research potentially

has far reaching impacts not only in reducing game latency but also in optimizing other types of network traffic via the prioritizing of the most important data packets sent over the network.

II. Background and Theory

Predictable and sub-second response time has long been a key concern for interactive computer systems [5]. For a majority of video games this is an obvious requirement that modern hardware has satisfied, despite a continuous rise in graphics and interaction quality. A *video game network* is a distributed set of "apparatus which are capable of exhibiting an interactive single identity game", as defined in a patent dated 1986 [13]. The requirements for response time are even more stringent in this context and in addition to inevitable network latencies, "the on-line service's computers themselves introduce latencies, typically increasing as the number of active users increases" [12]. The work described here is an experimental analysis of the conditions for satisfying this key requirement, namely low and predictable response time for a game network faced with a scalable number of players.

The last decade had seen a growing interest in tackling this problem. Some researchers like Iimura, Jardine and co-authors have proposed peer-to-peer architectures for multiplayer online video games [9], [10], this with the intention of reducing the bandwidth and processing requirements on servers. This can in theory provide better scaling but "opens the game to additional cheating, since players are responsible for distributing events and storing state". Pellegrino et al. [11] have then proposed a hybrid architecture called P2P with central arbiter. The bandwidth requirements on the arbiter are lower than the server of a centralized architecture. Like many non-functional properties of online services (security, scalability, reliability etc.) the choice between centralization and distribution is not one that can be given a definitive answer. Our work concentrates on a logically centralized architecture, its potential for predictability and scalability of the server and router ("arbiter") performance. Other work [7] has studied the same performance problems in the presence of mobile player nodes. Despite its clear importance for the future, this line of study appears even less mature than the P2P approach.

Zhou, Miller, and Bassilious [15] have made the obvious but central observation that "Internet delay is important for FPS games because it can determine who wins or loses a game." Many game mechanics are time sensitive, but it is the time the information reaches the server that matters, not the time the player actually pushes the button. Our experiments measure packet size and inter-packet times or traffic volume as they have in their statistical model. Those authors' investigation also took into account the effects of other Internet traffic. But our study will exclude those effects precisely because we wish to isolate the scalability and load-resistance of the server and routing modules. Claypool and Claypool [3] have observed that Internet latency's effect is strongest for games with a first-person perspective and a changing model. The work we describe here takes this into account by experimenting with the game Minecraft, which is first-person and has changing game environments.

In a study of a different first-person game shows that "client traffic is characterized by an almost constant packet and data rate" [6]. The bot we developed for our experiments (architecture of our simulation infrastructure is on Fig. 1) follows this pattern, as it was shown in our initial experiments [1]. Additionally, that study found the average interpacket time for client to server traffic to be 51ms for the game being studied. While this value is game dependant, it is more beneficial for us to use a common value for our tests so that the results are more generalizable. To that end, we designed the bot to send its action packets at 50ms intervals.

More recent studies [2], [4] of first-person shooter games have modeled time series behaviour of game traffic and tested the model on up to eight different games. According to our previous comment, such a comparative study would not have allowed us to get very stable load measurements, hence our choice of a single first-person game. Indeed the study of Wu, Huang and Zhang [14] shows that "the server-generated traffic has a tight relationship with specific game design", again from our point of view confirming the need for precise measurements of a given architecture on a single game. Hariri et al. go even further in this line of thought by designing a model of the player's activity to extract traffic patterns [8]. Such a representation is beyond the scope of this paper but is certainly relevant and its combination with our conclusions should be the object of future work.

III. Results

We began our project with the search for an existing program that would meet our criteria. We had access to a bot that could simply connect to a server and send chat commands from a previous project, but that no longer met our needs. In order to accurately simulate a player we needed a bot that would move in and interact with the game world. We found one project, DarkBot [**?**], but it was not compatible with the newest version of Minecraft, which we intended to use. No bot was found that was suitable for our experiments so we developed our own bot program. The solution we were able to use for our project came out of a tool we found online called MCProtocolLib [**?**].

A. The solution

MCProtocolLib is an open source library that was created to allow simple communication between Minecraft clients and servers. We were able to use this to replicate the creation of player packets and communicate between our bot and a server on our internal network. The program was written in java, as is the Minecraft source code. We started with a test program included within the library, and then expanded it to create our fully functioning automaton. The code to connect and disconnect from servers was provided as part of the library as well. Bots are created through XML scripts, which are then parsed, and the appropriate method is called based on the action requested. That method then uses the MCProtocolLib library to create packets in the same way that the Minecraft code would, and then send them to the desired server. The code that we had to create was the parsing of the script and the gathering of the correct arguments to create each packet. Once the program was able to duplicate the actions required, it was refactored to allow for multiple bots to be created simultaneously, and made into an executable jar file for ease of use. Many scripts were created and used to test the different

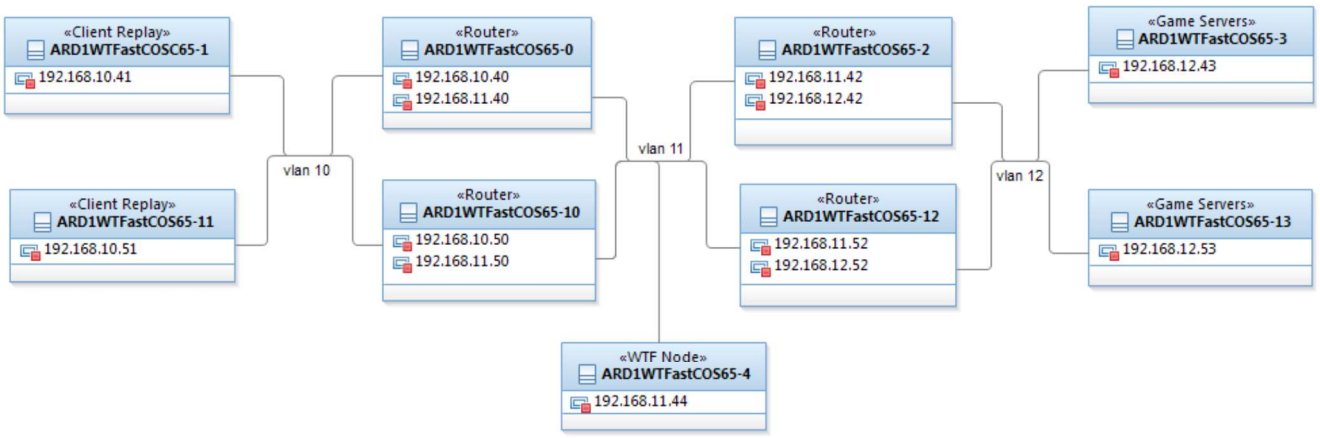

Fig. 1. Architecture of Our Simulation Infrastructure

Fig. 2. Running simulation bots in a square

bot actions, and all were successful. A short video clip with 50 bots running in a square is published on YouTube for the demonstration purposes only [**?**]. A screenshot of running bots is shown on Fig. 2.

IV. Validation

At the beginning of our project we were provided with test results of the chat bot, which showed the difference in packets created between real players and bots when running idle on the server [1] namely that the bot only sent chat and keepalive packets. We wanted to reproduce the same results for a bot that did more complex actions, such as run in a pattern. With our completed bot, we had the opportunity to do just that. Experiments were all done using the original chat bot, our newly created bot, and a real player. The experiments run were as follows:

A. 1 min capture idle

The first test captured the packets created by each of the 2 bots and the player while sitting connected to the server but without any movement or other actions. We performed separate captures, each with only one connection. We expected that the new bot will have closer results to the player than the old bot did. In all cases, the traffic volume is heavily dependent on the game state. A typical game world includes hills, trees, animals and other environmental objects. To help narrow our focus to the traffic related to player actions we used a much simpler game environment for these experiments. We used a flat world with no enemies, animals or even weather effects.

Hypothesis: We expect that the old idle bot and the new idle bot will produce similar amounts of traffic, as there is very little external interference. It is expected that an idle player will produce more traffic than the bots due to overhead packets being transmitted that is not done by the bot.

Observations: Reducing the complexity of the bot's environment resulted in a large decrease in the traffic generated by the bot, from an average of 647 packets/min to 245 packets/min. Fig. 3 shows that both the old and new bot produce much less traffic than the player under these circumstances, the player however, still shows similar levels of traffic whether the environment is simple or complex. We can see that the player exchanges different packet types than the bot and this is the reason for the discrepancy. Further research is planned to identify the packets that the bot does not use and to add methods of sending these packets to the bot program. For the purpose of our experiments, the exact type of game packet is not as important as the number of packets and the volume of traffic. Unfortunately, the simplified environment changed the traffic of the idle bot to the point where it cannot be used to represent an idle player.

B. 1 min capture square

The next experiment utilizes the improvements made to the bot since our previous work. Now that we have a bot that can be active in the game world we can compare the traffic generated by its actions to that of a real player taking similar action. For this experiment the active bot and the player run in a square pattern in the game world while we capture the network traffic (two captures, one for each individually).

Hypothesis: Our new bot program should produce traffic that is more similar to a real player than the previous idle bot which produced between 70% and 180% as much traffic as a player in our initial experiments with the chat bot (see Section III).

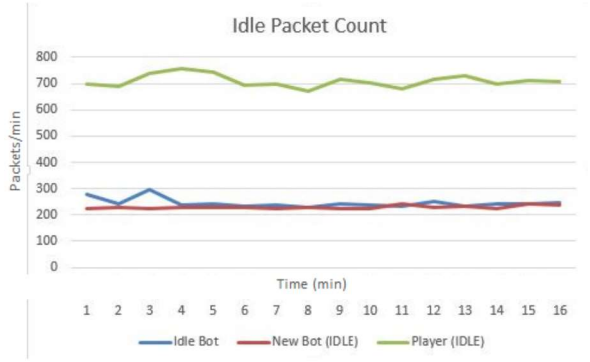

Fig. 3. Packet count of idle connections over time (min)

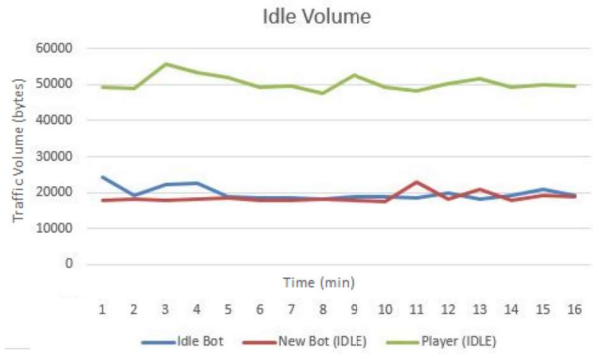

Fig. 4. Traffic volume of idle connections over time (min)

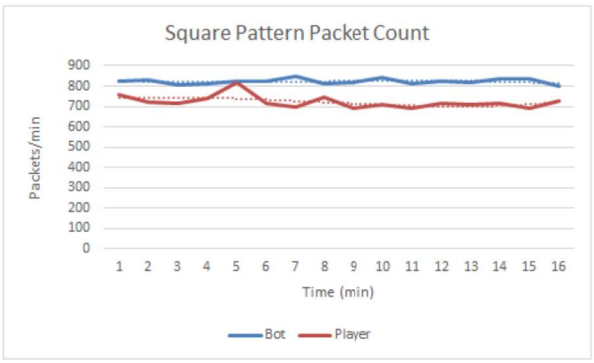

Fig. 5. Square pattern experiment

Observations: Fig. 5 shows the comparison where the bot produces approximately 114% the traffic the player does. This confirms that the new bot does represent an active player more accurately than our previous idle bot. Especially so since the idle bot produces so much less traffic in the simplified environment.

C. 1 minute square with surrounding bots

In a modified version of the last experiment we included other bots running in a square along with the bot/player we are analyzing.

Hypothesis: We expect that these nearby bots will increase the amount of packets being sent across the network, due to information updates about nearby entities.

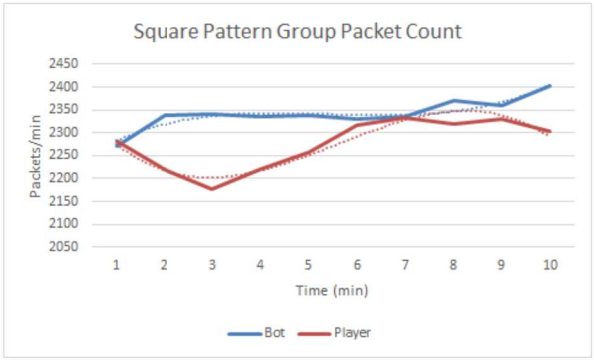

Fig. 6. Square pattern group experiment

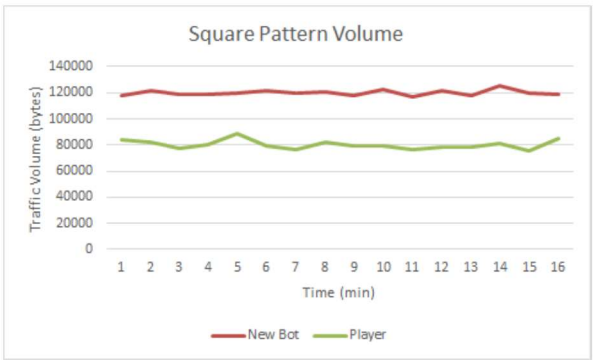

Fig. 7. Square pattern volume experiment

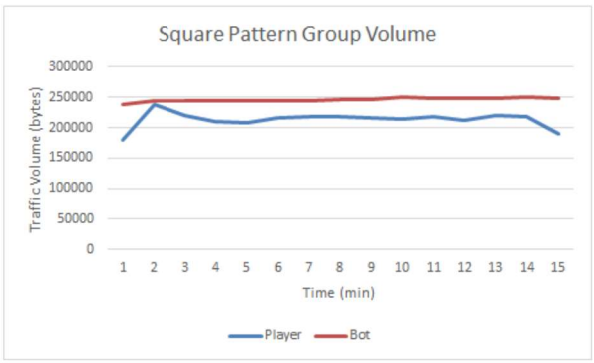

Fig. 8. Square pattern group experiment

Observations: Fig. 6 shows the comparison between player and bot traffic, with both the bot and player traffic having increased by an average of 266% for the player and 205% for the bot. The fact that adding the additional bots decreases the difference between bot and player packet counts, shows that as more bots are added, a single user being a bot or a player becomes less distinguishable.

Fig. 7 and Fig. 8 show experimental results for the square pattern running by the bot and the player, but represented by volume of packets as opposed to number of packets. This was done to show that packet size has little impact on the overall analysis, and in this case actually makes for more stable data points.

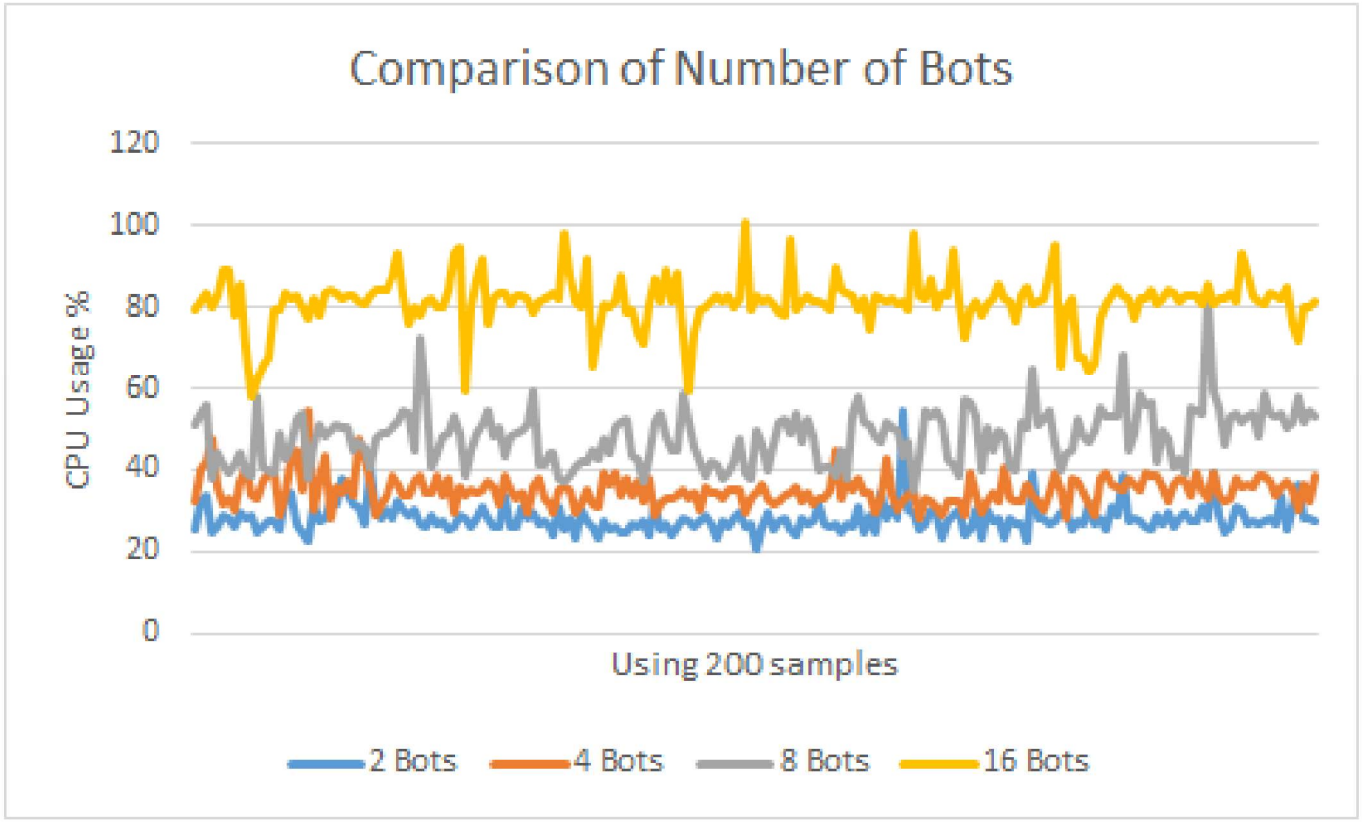

Fig. 10. CPU usage of the game server with various numbers of bots connected over time

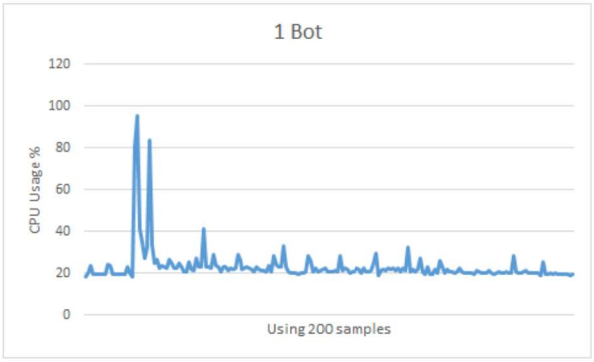

Fig. 9. CPU usage of the game server with 1 bot connected over time

D. CPU usage

In another experiment we measured the CPU usage of the game server while under load from a varying number of bots. This experiment shows the capacity of the game servers and can be used to determine how many servers we would need in future experiments in order to actually overload the network software with game traffic.

Hypothesis: As the number of bots connected increases so should the computer resources required to run the game server. However the game itself requires some resources to run even with no connections, so the CPU usage should not be strictly proportional to the number of bots.

Observations: Fig. 9 shows spikes in the CPU usage that are most likely related to saving the game state or some other server-side game process, the spikes did not occur when testing with other numbers of bots and overall are fairly rare. Tests in future experiments covering an extended period of time should provide more information about the cause of these spikes.

The CPU usage of the game server can be broken down into three basic components. First, there is a base level of CPU usage to run the server, even with no players connected. Next, a small amount of extra processing is required for each player that is connected. The third part is the processing related to interactions between players. In terms of our experiments, this interaction is simply the game server providing information about each bot to the ones around it. These three factors fit into a quadratic regression of the comparison in Fig. 10, which yields the result CPU% $\approx 0.0351n^2 + 3.2169n + 20.7962$, where n is the number of bots. The virtual machine hosting the game server was not able to keep up with higher numbers of bots, as it was designed as a scaled down version of a real server machine.

V. Future Work

Our future research will be related to utilizing the infrastructure created in the first phase of the project as the framework on which to test and build game server software that can optimize and monitor WTFast game services in real time. This infrastructure will allow new servers to be automatically deployed and configured for use as private game servers, while

also monitoring their performance and usage statistics. By using predictive models, new servers may be automatically added when traffic levels require more resources to maintain optimal performance.

Testing of the network infrastructure and the network software will be done using a program that simulates many players connecting to multiple game servers on several virtual machines in the network. The goal of these network tests is to identify the point at which the network software can no longer keep up to the flow of traffic, i.e. the point where the network software becomes a cause of latency. The tests will also serve to identify the capacity of the game servers and their host virtual machines in terms of the number of players they can support. Initially the tests will be performed under ideal conditions, that is, on a local network with almost no wire latency. Later, network factors such as latency and jitter can be artificially added to the network to simulate real conditions of the game being played over the Internet on a geographically remote server in order to confirm our tests in a more realistic environment.

VI. Conclusion

Network analysis is a broad spectrum of research, that can be studied in many different ways. Our focus on latency within video games led us to experimentation with Minecraft clients and servers. The need for a controlled environment to ensure consistent and accurate results led to the requirement of multiple automated bots that could interact with the servers in a predictable way. This project was successful in creating just that, meeting all originally set criteria. Although research is not yet complete, we anticipate that the bot will greatly help out future work in this field. Based on our experiments, the bot accurately recreates real player traffic, which gives to us a suitable tool for the network performance investigation and to generate a stress test for the game servers.
Future research will investigate the performance of very large bot populations, for example with parallel hardware, and time-series models of such populations for ultra-large-scale simulations of game environments. The resulting methods will constitute a full laboratory toolbox to explore latency and general online-gaming performance in an offline, controlled environment. Such studies will then allow intelligent, dynamic and online methods for game network optimization.

Acknowledgment

This work has been funded partially by NSERC's College and Community Innovation Program - Applied Research and Development Grant Level-1 in 2014 (Canada): "GPN-Perf: Investigating performance of game private networks".

The development of the bot program was done by computer science students at Okanagan College as an XP Spike project within COSC 470 SW Engineering Project course in 2014. Thanks to Marc Schroth, Mike Adkins, Marc-Andrew Dunwell for their work on the bot application.

References

[1] Trevor Alstad, J. Riley Dunkin, Rob Bartlett, Alex Needham, Gaétan Hains, and Youry Khmelevsky. Minecraft computer game simulation and network performance analysis. In *Second International Conferences on Computer Graphics, Visualization, Computer Vision, and Game Technology (VisioGame 2014)*, Bandung, Indonesia, November 2014. Accepted for publication.

[2] Philip A Branch, Antonio L Cricenti, and Grenville J Armitage. An arma (1, 1) prediction model of first person shooter game traffic. In *Multimedia Signal Processing, 2008 IEEE 10th Workshop on*, pages 736–741. IEEE, 2008.

[3] Mark Claypool and Kajal Claypool. Latency and player actions in online games. *Commun. ACM*, 49(11):40–45, November 2006.

[4] Antonio L Cricenti and Philip A Branch. A generalised prediction model of first person shooter game traffic. In *Local Computer Networks, 2009. LCN 2009. IEEE 34th Conference on*, pages 213–216. IEEE, 2009.

[5] WJ Doherty and AJ Thadhani. The economic value of rapid response time (ibm technical report ge20-0752-0). *Zugriff via* `http://www.vm.ibm.com/devpages/jelliott`, 1982.

[6] Johannes Färber. Traffic modelling for fast action network games. *Multimedia Tools and Applications*, 23(1):31–46, 2004.

[7] Preetam Ghosh, Kalyan Basu, and Sajal K Das. Improving end-to-end quality-of-service in online multi-player wireless gaming networks. *Computer Communications*, 31(11):2685–2698, 2008.

[8] Behnoosh Hariri, Shervin Shirmohammadi, and Mohammad Reza Pakravan. A hierarchical HMM model for online gaming traffic patterns. In *Instrumentation and Measurement Technology Conference Proceedings, 2008. IMTC 2008. IEEE*, pages 2195–2200. IEEE, 2008.

[9] Takuji Iimura, Hiroaki Hazeyama, and Youki Kadobayashi. Zoned federation of game servers: A peer-to-peer approach to scalable multi-player online games. In *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '04, pages 116–120, New York, NY, USA, 2004. ACM.

[10] Jared Jardine and Daniel Zappala. A hybrid architecture for massively multiplayer online games. In *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '08, pages 60–65, New York, NY, USA, 2008. ACM.

[11] Joseph D. Pellegrino and Constantinos Dovrolis. Bandwidth requirement and state consistency in three multiplayer game architectures. In *Proceedings of the 2Nd Workshop on Network and System Support for Games*, NetGames '03, pages 52–59, New York, NY, USA, 2003. ACM.

[12] S. G. Perlman. Network architecture to support multiple site real-time video games. United States Patent number 5,586,257, Dec. 17, 1996.

[13] D. H. Sitrick. Video game network. United States Patent number 4,572,509, Feb. 25, 1986.

[14] Yi Wu, Hui Huang, and Dongmei Zhang. Traffic modeling for massive multiplayer on-line role playing game (mmorpg) in gprs access network. In *Communications, Circuits and Systems Proceedings, 2006 International Conference on*, volume 3, pages 1811–1815, June 2006.

[15] Qili Zhou, C.J. Miller, and Victor Bassilious. First person shooter multiplayer game traffic analysis. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pages 195–200, May 2008.