Minecraft computer game performance analysis and network traffic emulation by a custom bot

Trevor Alstad, J. Riley Duncan, Simon Detlor, Brad French, Heath Caswell, Zane Ouimet, Youry Khmelevsky, Gaétan Hains, Rob Bartlett, Alex Needham

Science and Information Conference 2015
July 28-30, 2015 | London, UK

Minecraft Computer Game Performance Analysis and Network Traffic Emulation by a Custom Bot

Trevor Alstad, J. Riley Duncan, Simon Detlor, Brad French, Heath Caswell, Zane Ouimet and Youry Khmelevsky
COSC Dept., OC, Kelowna, BC V1Y 4X8, Canada
Emails: {prototypeTX37, john.dunkin13, wingmansd, bradfrench123, heath.caswell, khmelevsky}@gmail.com, zane.ouimet@outlook.com

Gaétan Hains*
Laboratoire d'Algorithmique, Complexité et Logique (LACL)
Univ. Paris-Est Créteil, Paris, France
Email: gaetan.hains@huawei.com
*Huawei France R&D Centre, Paris.

Rob Bartlett and Alex Needham
WTFast, Kelowna, BC
Emails: {rob, alex}@wtfast.com

***Abstract*—To simulate player traffic within the game we developed an automated bot for a popular online game Minecraft. The first emulation goal was for the bot to realistically replicate network traffic that a normal player would use while playing the game. The second emulation goal was to investigate the maximum possible workload on a virtual multicores and multi-CPUs CentOS server by running different number of active Minecraft games on many cores of the multi-CPU servers simultaneously.**

We created a scriptable bot capable of performing many common game actions, while generating comparable traffic to that of a player. This facilitates network and game server world optimization. It is allowed us to create a new testing and emulation environment to investigation network and server performance in our virtual Gaming Private Network (GPN) infrastructure.

I. Introduction

Minecraft [1], [23] is a multiplayer sandbox game in a procedurally generated world where players are able to build structures and creation out of textured cubes similar to Lego in the real world. Minecraft has several game modes including a survival mode where a player must acquire resources and build shelters to survive against NPCs and the elements, a creative mode where a player can simply build creations with unlimited resources, and an adventure mode where a player can play on custom maps built by the community. Minecraft which is heavily inspired by a similar game called Infiniminer [11] is built in the cross-platform language Java which allowed it to become one of the most popular games ever with a rich community that eventually brought it to the attention of Microsoft who purchased it in September 2014.

We chose Minecraft for our experimentation due not only it's vast customization ability, but because it is a well documented java application. We can see exactly what it is doing, in both packets and code, and can predict how the game will react. This is perfect for our controlled test network, so we can see when improvements are made, and when changes incur performance loss, as well as why. Most importantly we can scale to generate volumes of traffic and incredible workload on the game servers; Minecraft can allow a maximum of 2,147,483,647 players, all doing many various world changing actions, resulting in generating large amounts of data transfer and processing.

In this paper we start with a brief of information on a conducted research project in 2014 (see [3]). Following that we describe our bot, the solutions we used, and the outcome of the project, including performance experiments.

In the rest of this paper we discuss existing simulation tools which are used to investigate on-line game performance. Then we describe our experimental infrastructure, discuss our performance analysis results, including data collection, measurements, and their interpretation. In the Section V we describe the bot design and implementation for the Minecraft game traffic and server performance analysis. In conclusion (Section VII) we summarize our research results and outlined our future plans in the Section VIII.

II. Existing Works

"A study of different first-person games shows that the client traffic is characterized by an almost constant packet and data rate" [10]. The study found that "the average interpacket time for client to server traffic to be 51ms for the game being studied". Our new bot can send the action packets at 50ms intervals [4].

In the discussed research we are mostly concentrated on the servers performance optimization, additionally to the network traffic analysis [3] and design and implementation of the custom bot for Minecraft [4]. As it was shown in [2] the "bottleneck in the server is both game-related as well as network-related processing (about 50-50)." In our research we investigated the highest possible workload for the CentOS 6.5 virtual server by utilizing our custom based bot for Minecraft.

Some authors discuss interactive online games, especially ones related to the "first person shooter (FPS)" [5], [8] and network traffic for such games [25]. They investigate network impact on the games and realistic traffic generators. In our infrastructure our aim was not to just emulate 2 or 3 players, but 100 and even 1000 and more players. This is important for gaming companies, because as it is shown in [15] online games become major contributors to Internet traffic. Latency is the another challenge for online games, as it's reported in [6], [20] and [18] and it's an important factor of an online gaming experience. We built our infrastructure to emulate artificial latencies in the emulated traffic [3]. In [17] "massively multiplayer online games with a client-server architectures and

peer-to-peer game architectures" are investigated. The authors developed a hybrid game architecture to reduce game server bandwidth. In [16] authors even proposed to implement a zoned federation model for the multi-player online games trying to reduce workloads of the centralized authoritative game servers. A US 5956485 A patent [21] describes how to link multiple remote players of real-time games on a conferenced telephone line, which could reduce latency for the game players.

In the technical report from IBM [9] it was demonstrated, that "rapid system response time, ultimately reaching subsecond values and implemented with adequate system support, offers the promise of substantial improvements in user productivity" and it's even better to "implement subsecond system response for their own online systems". They mentioned, that not so many online computer systems are well balanced. They devided system response time for two large groups: computer response time and communication time, which are both critical for the game players user experience as well.

In [12] authors discuss online multiplayer gaming issues in wireless networks, which is an additional problem related to the game players experience on the Internet. These issues are not covered in the current paper. On the other hand, we experienced packet loss in our infrastructure too. In paper [24] authors investigated a multiplayer on-line game traffic including modelling traffic in mobile networks.

III. Infrastructure for the Minecraft Game Performance Analysis

The goal of the research project was to create a collection of Minecraft server instances and calculate the latency of real-time and/or simulation traffic using traffic control as well as flooding the network and servers to determine the maximum amount of processing the network can take before crashing. Also, another set goal is to collect data from in-game experiments and graph the experiments.

The infrastructure for our experiments is virtual machines (VMs) setup between three VMWare ESXi hosts (CISVMWare 3 - CISVMWare 5 in Fig. 1) and a Seanix SG166H desktop PC for the traffic monitor in a LAN consisting of 8 virtual CentOS 6.5 VMGuests (Fig. 2). The VMs are communicating through Virtual LANs (VLANs), isolating them from external networks to provide untainted results free of external influence; even live testers work from virtual machines as part of the test network [3]. Game server has been deployed using McMyAdmin [19].

Fig. 2 shows a UML representation of the actual VMs we tested with and the path they take through the network. A mirrored setup allows for redundancy as well as the ability to funnel traffic from multiple servers onto one wire for maximum traffic going through the node. There is actually two identical networks, our development and production networks running in parallel; the upper and lower rows in Fig. 2 show the two identical networks, running in different subnets. This allows us to run and re-run multiple tests on different components in parallel quickly [3]. We had 2 different versions of a network route, both containing a client replay, 2 routers and the game server. Both routes also connected with a node on vlan 11 which runs through proprietary software. A proxy was set on the top left router which reroutes the traffic to go through

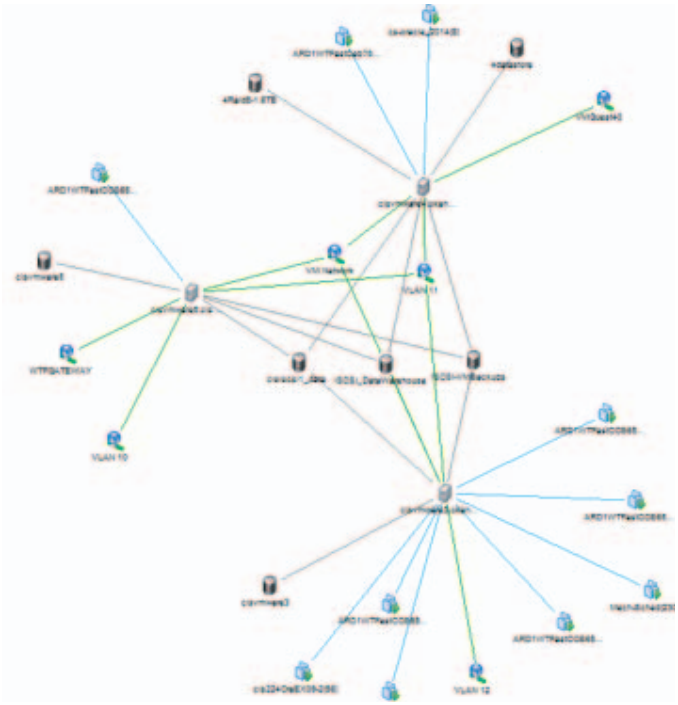

Fig. 3. Network topology diagram.

the node, back out to its normal route, comes back through the node again, and again returning to its normal route. Here we can collect data and latency can be tested when running through the dedicated software.

Fig. 3 shows the entire physical infrastructure, including all hosts, VMs, datastores, VLANs, and links between them. There are other clients shown that were not used in the experiments, but anything labeled ARDWTFast corresponds to one of the machines in the UML diagram. The "network is constructed to ensure that we can isolate and identify the causes of the latency we measure. The architecture structure fits in the process of the experiments in regard to which OS the environment runs and the state of the network (virtual or physical). The operating system that is installed on the network which runs the Minecraft server may affect network performance. Running a Linux CentOS (current OS) for example would have better network performance rather than a Windows or Mac operating system. The network performance may also be affected depending on if the network is virtual or is running on a physical configuration" [3].

IV. Experiments with a ChatBot, New Bot and a Human Player

The initial testing tools evolved when the initial ChatBot bot outgrew its use. The following experiments where preformed with a ChatBot we found online [13]. This tool was capable of replicating idle traffic, however it could not preform any kind of advanced functionality.

For the original network traffic testing the clients, routers and node were allocated with 2 CPUs running at 2.4 GHz and 2 Gb of RAM, while our servers used 2 CPUs with 1.6 GHz and 4 Gb of RAM. The later testing for the CPU usage on isolated cores used a single server with 8 cores running at 2.4

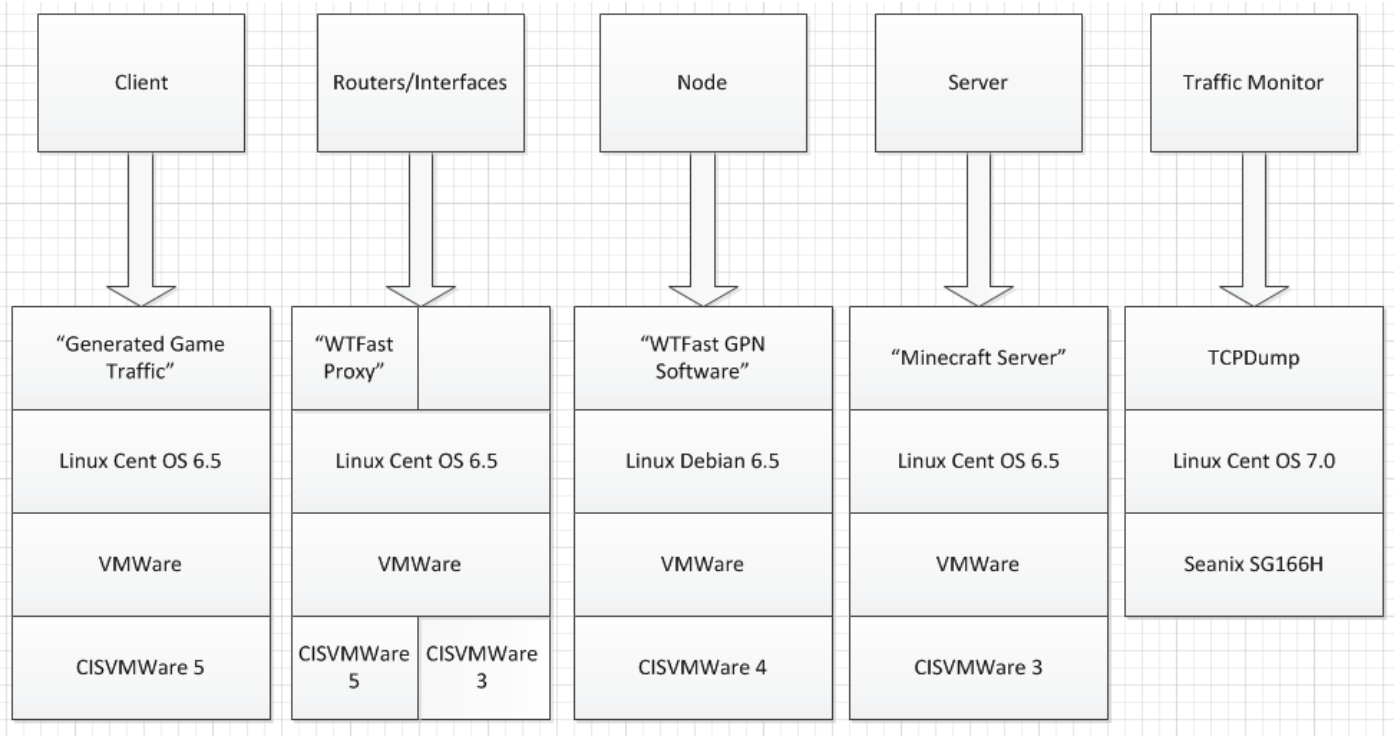

Fig. 1. Infrastructure architecture.

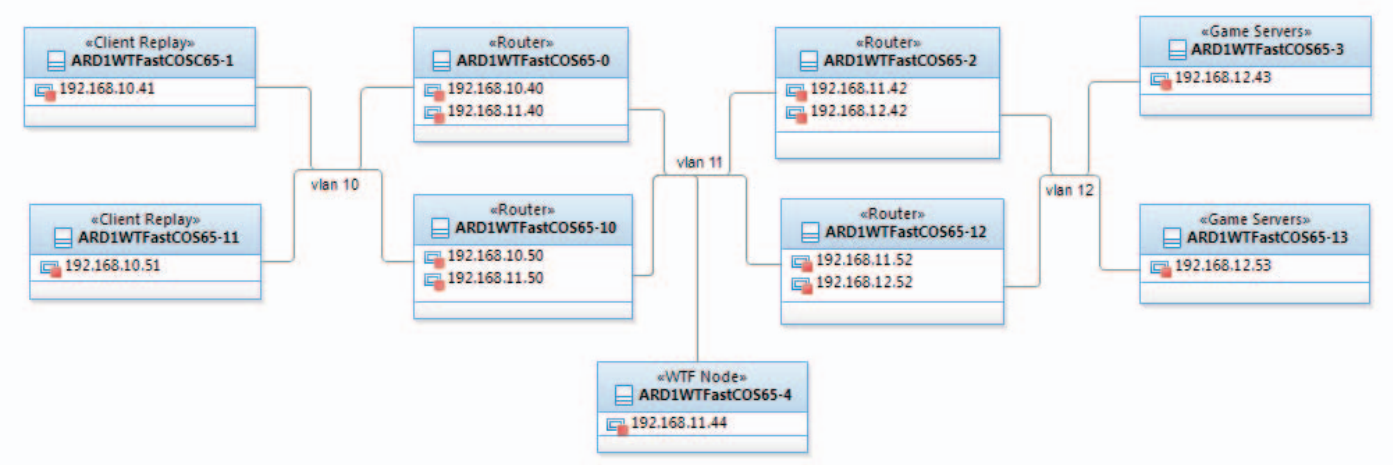

Fig. 2. VMGuests and the traffic monitor node.

GHz and 8 Gb of RAM. The CPU core isolation tests were all done using 20 bots per server.

"All experiments were run for sixty seconds. The packet count and traffic volume measurements are based on sums of the captured traffic, latency values are measured by calculating the time between the client sending a packet containing some form of player action data to the data and the receipt of a response packet from the server. We analyzed this latency data from different points in the network in an attempt to isolate the causes of latency" [3].

A. *Testing methods*

For our first experiments, a passive bot was used. This runs from a windows command line, with username, password, and server ip parameters, and simply connects to the server and sits idle. This method should no longer be used for testing as we have a new active bot now. The active bot is run as an executable jar with parameters for the server ip and the script to be loaded.

The CPU usage tests were run on a scaled down environment with only a client and server. CPU affinity was set using the Linux taskset command.

For each experiment, the script runs the top command and puts 20 samples taken once per second into a file. Ten such

experiments are run. The sample data is put into files, there is one file for each core for each experiment. For each of these files the sample values are added to a comma separated list and added to a file, the name of the output file is taken as a parameter and includes the number of cores and the number of games being tested. The data captured on the idle cores show the idle usage of the cpu, so that it must be altered by taking 100 - value during processing. The script also has an error when the idle usage is 100%, in which case it reports 0% instead, this is also managed during processing of the csv files. This script gathers the data from both the game servers and the idle cores and puts them in a file, game servers first, then idle cores. All data was stored in an Oracle SQL database for easy analysis. To access data from the tables, simple queries were crafted.

The following experiments were performed [3]:

- **One player (1P) initial testing:** We needed a base case, on how the game reacts to a player. This experiment involved having a human player join the Minecraft server, where we observed traffic as the player idled. After collecting a fair amount of idle traffic, we ventured out into the world, and noticed a rise in traffic. Then after further investigation, this large spike, was the server informing the player of new portions of the world.
- **One chatbot idling experiment:** We joined the chatbot into a server, and collected traffic. On observation, it performed similarly to the idling portion of the player results.
- **The 10 bot (10IB) experiment (Fig. 4):** We loaded ten bots into a single server, and simply observed. We noticed that number of bots, actually had an effect on the amount of traffic that each bot produced; that they where taking into account the other connections in the server.
- **The 1P2 experiment:** A human player did not move from his starting position, but interacted with blocks only. The latency was similar to the 1B case.
- **The 1 idle bot and 1 idle player (1IB1IP) connected to the same game server experiment:** The idle bot produced almost the same amount of traffic compared to the idle player.
- **The 1 idle bot and 1 active player (1IB1P) experiment:** The active player produced slightly less traffic to compare with the idle bot (see more information about this phenomena here in [3]).
- **The 1 active player and 1 idle player (1P1IP) experiment:** We found that idle players created more traffic to compare with the active players.

B. Performance Analysis and Analysis Results

Now that we had various results using the chatbot, we needed to figure out trends, and get solid results.

Idle players as traffic generators: When looking directly at the data for idle players compared to active players, we tend not to notice any jarring oddities, that the results are quite

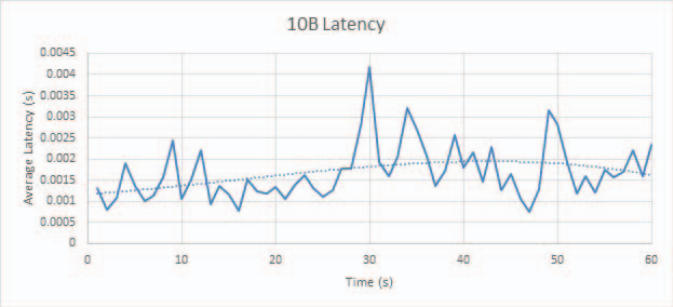

Fig. 4. 10B: Latency.

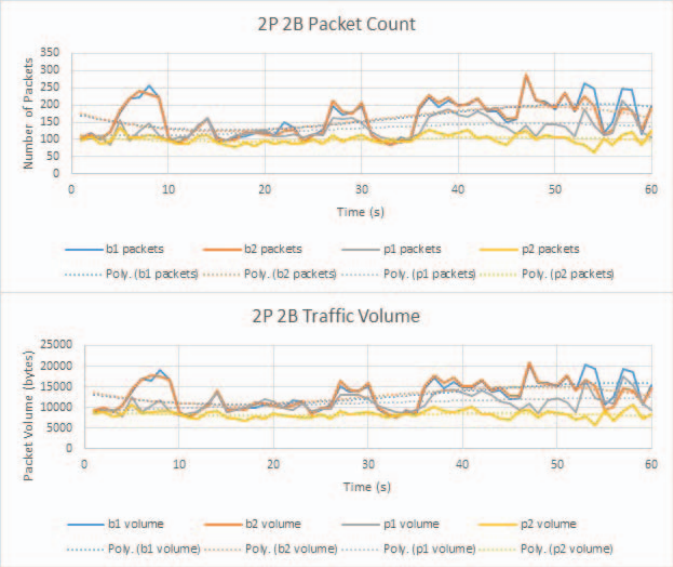

Fig. 5. 2P2B: Packet count and traffic volume.

similar. In fact, even the bot idling was quite close to the player traffic. This means, for the sake of tests, we can use idling connections to increase game traffic. On the other hand, this also means that an idle player will have negative repercussions on game play; if somebody joins, and does nothing, they are still causing the same influence on the server that an actual player would be creating, lowering the resources for other players by just that much. In the 2P2B experiments in Fig. 5 the bots produced approximately 140% of the traffic to compare with the players.

V. New Bot Implementation and Performance Measurement

We began our simulation bot development by searching for existing bots to emulate a Minecraft game player. We found a chat bot that could only connect to a Minecraft server and send chat commands, but it was too simple for the research project. We needed something more advanced, but not just a chat bot for Minecraft, something that would move in and interact with the Minecraft's game world. The student's capstone project group in COSC 470 course at Okanagan College in 2014 split up attempting to find an existing bot and was met with little success. We found the DarkBot [13] application in the Internet, but it was not really compatible with the current version of Minecraft. When it was clear we would not find a fully

functioning bot to meet our criteria, we decided we would have to create our own. We had one member of the group investigate a method using python scripts and raspberry pi hardware he had read about online, but it turned out unsuccessful. At the same time we examined the Minecraft source code to try and define the different packet types that were being sent between the client and server. We were able to define many of the packets created by the game code, using online resources detailing the Minecraft client/server protocol, but did not learn enough to be able to recreate said packets correctly. The solution we were able to use was called MCProtocolLib [14]. The MCProtocolLib is an open source library which allows simple communication between Minecraft clients and servers. Unlike Darkbot it was being actively developed and served a more general use of creating connections to a Minecraft server.

The following scripts were implemented:

- **Move Script:** The Move Script was developed so that the bot can move in a specified pattern on the game server. The script can successfully move in a rectangle in the game. When the script is executed, the bot connects to the game server, and then moves in a continuous rectangle in the same area which it spawns. The bot does not change its look direction, but only moves in its set pattern.
- **Place Block Script:** The Place Block Script was developed so that the bot can continuously place blocks in the game. The script can successfully place numerous blocks in the game. When the script is executed, the bot connects to the server, and then starts placing blocks in the game where the bot spawns.
- **Break Block Script:** The Break Block Script was developed so that the bot can break blocks on the terrain in the game. The script can successfully break blocks on the server. When the script is executed, the bot connects to the server, and then starts inputting commands to break the terrain which is in front of the bot. Once the bot breaks a block in front of it, it will step forward 1 square and then again start breaking the block in front of it. A real user has to place either dirt or sand blocks in front of it so that the bot can function properly.
- **Chat Script:** The chat script was developed to use chat functionality with the bots on the server. The chat script basically replaces the name of the bots when you input the following command: $self on the Minecraft server. The chat also allows the bots to produce chat messages in the server.

We started with a test program included with the library, which was modified to suit the needs of our project. By default, MCProtocolLib takes care of required server interactions, such as connecting and keep alive packets. Using the library, we generate player input packets similar to Minecraft itself, and then send packets to the desired server, and after verifying functionality, we setup an XML parser so that bots are created through XML scripts, and the appropriate packets are generated from the input. In addition the code allowed us to control the number of times packets were sent to the server per second. This would allow us to determine the minimum

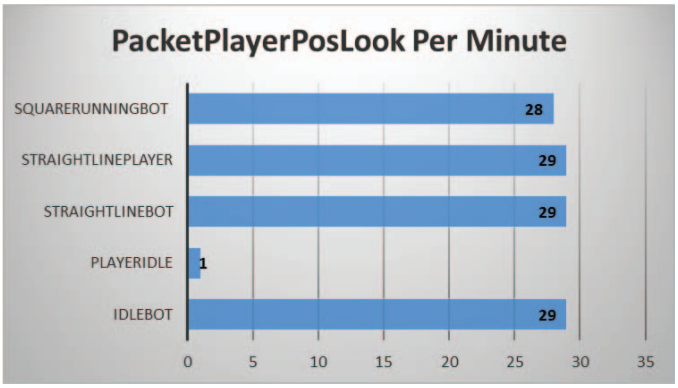

Fig. 6. PlayerPosLook packet.

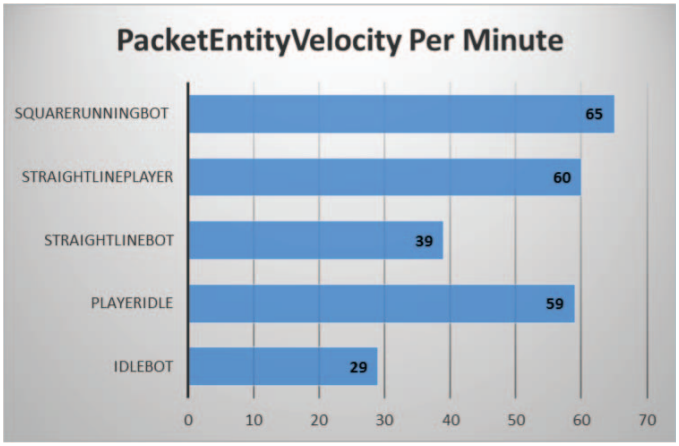

Fig. 7. EntityVelocity packet.

amount of packets that would be required for a player to appear fully connected and not 'lagged'. The bot was then built to perform several common user actions such as placing blocks, moving, and destroying blocks. Further development would add a greater degree of simple actions as well a more dynamic behaviour such as damage avoidance or resource gathering.

A. Comparison of types of packets generated

In verifying the success of our bot we have to ensure as well that not only is the amount of traffic correct, but that the types of packets are transmitted are similar to that of a real player.

PlayerPosLook packet is a Server-bound packet to update where the player is looking (see Fig. 6). As you can see there is a strong correlation between the running player and the running bot as well as the square running bot.

EntityVelocity packet is a Client-bound packet to update the velocity of a player (see Fig. 7). There is reduced amount of update packets in the case of the straight running bot and the idle bot. However, the square running bot generates comparable number of packets to a real player.

UpdateHealth packet is a Client-bound packet to update the health condition of a player (see Fig. 8). Once more it seems that our square running bot generates similar amounts of traffic to a real player. Conversely our straight line running

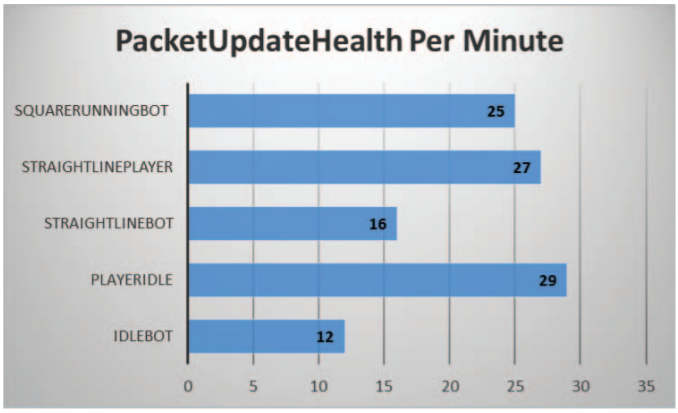

Fig. 8. UpdateHealth packet.

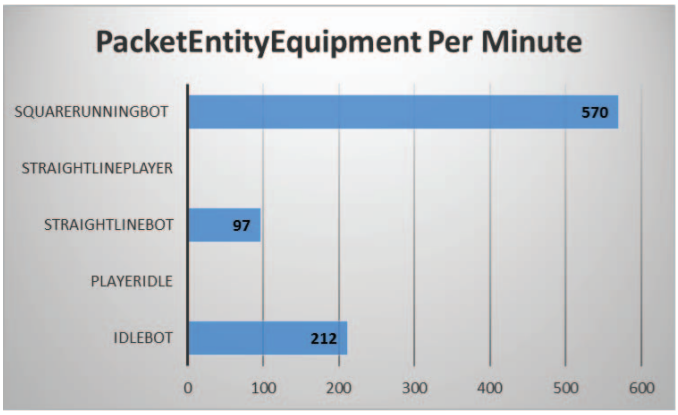

Fig. 9. EntityEquipment packet.

bot and idle bot do not generate similar amounts of health update packets. Perhaps this is due to the lack of commands and/or the complexity of the straight commands versus the square command set.

EntityEquipment packet see in Fig. 9. We are not sure why our bot generates so many of these packets whereas real players do not seem to generate any. During testing there was no involvement other than pushing a movement key in the straight line testing. It is interesting and may require further investigation.

The purpose of the new bot development was to generate and compare packet data produced from a test bot as we fed it commands from a script. The data was then analyzed and compared with a user bot as it completed the same task(s). In the short project duration we were able to implement a number of different methods for the test bot and as a team we are satisfied with the functionality our test bot affords. Using a script, the test bot (or test bots) are capable of following multiple commands in a row which allows for countless comparisons and tests in diverse scenarios of action.

The above graphs illustrate differences between the two bots, although they did not match up exactly, the majority of tests demonstrate a close resemblance in regards to the amount of data captured over a given time and command. In some comparisons we noticed data packets to be off more than anticipated which shows room for improvement of the test bot.

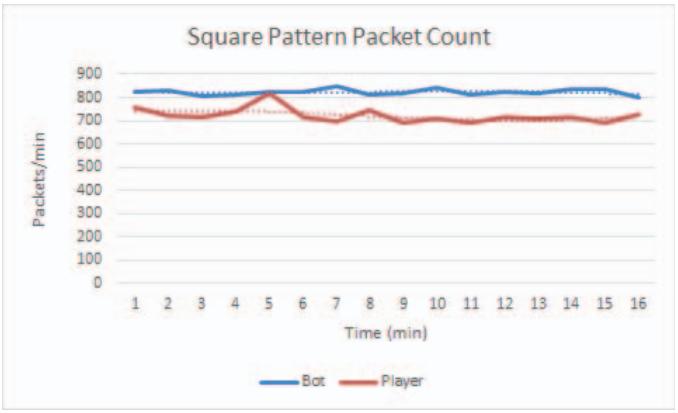

Fig. 10. Square pattern experiment.

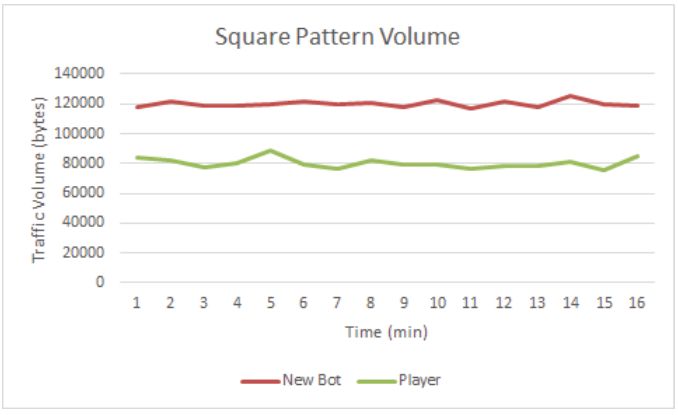

Fig. 11. Square pattern volume experiment.

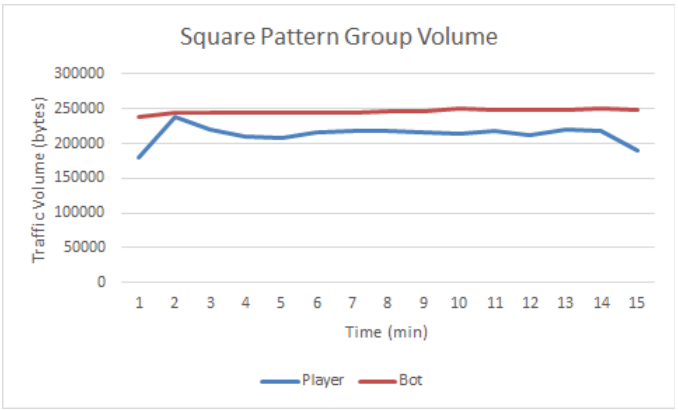

Fig. 12. Square pattern group volume experiment.

Our data analysis proves promising that the test bot is a viable testing tool for its intended use but could require refactoring in hopes of acquiring more accurate data by comparison.

On Fig. 10 is shown the comparison between the bot and the player traffic (about 14% difference). On Fig. 11 and 12 are shown experiment results for the Square Pattern Volume and Square Pattern Group Volume. See more chat bot test results discussion in [3] and new bot testing results discussion in [4].

In the followed experiments we built a scaled-down virtual-

ized GPN environment useable by game clients, then dumped and analyzed its traffic under a wide variety of load parameters. Those parameters include number of clients, network delay, network traffic and routing servers response time and effectiveness. Latency was both artificially generated and produced by the software processes themselves. This setup allowed us to produce stress-performance tests for clients and for the routing servers. Such measurements are too risky or impossible in the production environment and have not been published so far. The scaled-down experimental environment allowed us to produce numerical models that can predict extreme situations and thus allow capacity planning for the routing servers.

The environment for these experiments was very active, bot players would all spawn within a small radius that was surrounded by active game elements. The players were sometimes in the same area and sometimes elsewhere. Tests were done where the player would travel through the game world quickly and force large amounts of information transfer. The published data specifically applies to the case where the bots are together in the active environment and both human players are alone in other areas. Both game state factors like this and technical details like the method of sending bot packets are heavy factors in the results.

VI. Performance Optimization

Optimization in our project refers to the selection of the best method of game deployment and configuration in order to maximize CPU resource use and minimize in-game latency. This section outlines the methods we researched in order to achieve those means.

The goal of this optimization is to evaluate the interaction between a Minecraft client and server on our isolated network, and work to optimize network latency and CPU resource usage via various methods. A large part of this work depended on having a realistic way of simulating game traffic, so that we could determine the major causes of latency and have a better idea of how it could be combated. Various tests were performed to test the new bot as it was developed. In addition to the data published in the [3] other key comparisons were made. In Fig. 13 the composition of packets is compared between the bot and a real human player, this composition is important for creating realistic network traffic. The second comparison in Fig. 14, deals with game state issues and helped us decide the environment for our experiments in the paper. Reducing the complexity of the environment resulted in a large decrease in the traffic generated by the bot we had used previously, showing how strongly the game state affects the network traffic. The new bot was developed so that it would remain consistent with a real player without the additional load from a busy in game environment. The experiments were done in a purely simple environment to try to isolate the effects of player traffic in regards to server performance.

The following charts describe the cpu performance information gathered on an 8-core game server running varying numbers of game servers on varying numbers of cores. Each game instance had 20 bots connected and running around. The infrastructure available at the time did not allow us to collect network information at the same time as doing cpu testing, so this work may be replicated in the future from a network perspective in order to see the effects of cpu overuse on game traffic. Note that there would have been severe latency in the game instances whenever the cpu usage for the core it is on is above 90%. In many cases game instances would fail due to cpu saturation, several tries were required to maintain a stable state for experimentation.

Fig.15 shows the combined CPU usage of all cores, labeled with the number of game servers on each core. This chart shows the values for zero to 4 games using approximate average values for the idle cores as we did not gather the data for idle cores during these experiments.

Fig.16 shows the combined CPU usage of all cores, labeled with the number of game servers on each core (3 1 means 3 games on one core, 1 game on another). This chart shows the values for 5 to 16 games using actual values for the idle cores. (A few values are the average as in CPU Stack 1, but most are real data)

Fig.17 shows each core and the workload on it. For cores running a game server (or servers) the CPU usage of the game server process is shown, for cores without a game server running, the system usage of hat core is shown by subtracting the idle percentage from 100. This chart shows the values for zero to 4 games using approximate average values for the idle cores as we did not gather the data for idle cores during these experiments.

Fig.18 shows each core and the workload on it. For cores running a game server (or servers) the CPU usage of the game server process is shown, for cores without a game server running, the system usage of hat core is shown by subtracting the idle percentage from 100. This chart shows the values for 5 to 16 games using actual values for the idle cores.

VII. Conclusion

This paper discussed results of the network and server performance investigation by using two different bots, one publicly available in Internet and another one developed specially for this project for the online game Minecraft. The goals for the bots were to replicate network traffic that as a normal player would produce while playing the game and to investigate the maximum possible workload on a virtual multicores and multi-CPUs CentOS server by running a different number of active Minecraft games on many cores of the multi-CPU servers simultaneously.

We designed and developed a scriptable bot capable of performing many common game actions, while generating comparable traffic to that of a player. This facilitates network and game server world optimization. It is allowed us to create a new testing and emulation environment to investigation network and server performance in our virtual gaming infrastructure.

CPU workload evaluation shows that with 8 cores, we can deploy 2 x 8 = 16 Minecraft game servers per a physical host. In the future we can move the Minecraft servers from an overloaded core to another core on the same CPU or to another not highly loaded CPU or even to another server automatically.

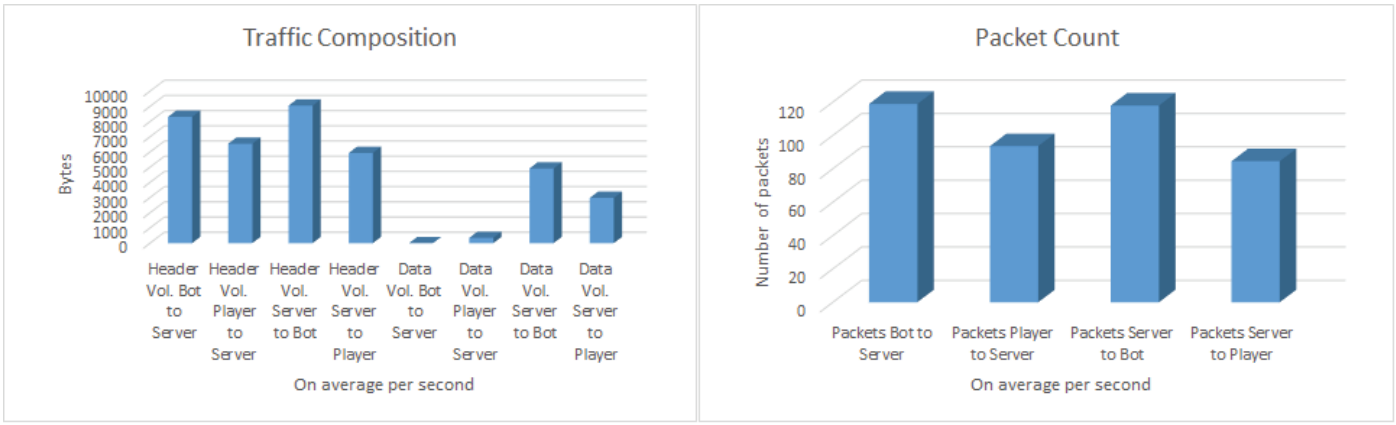

Fig. 13. Traffic composition.

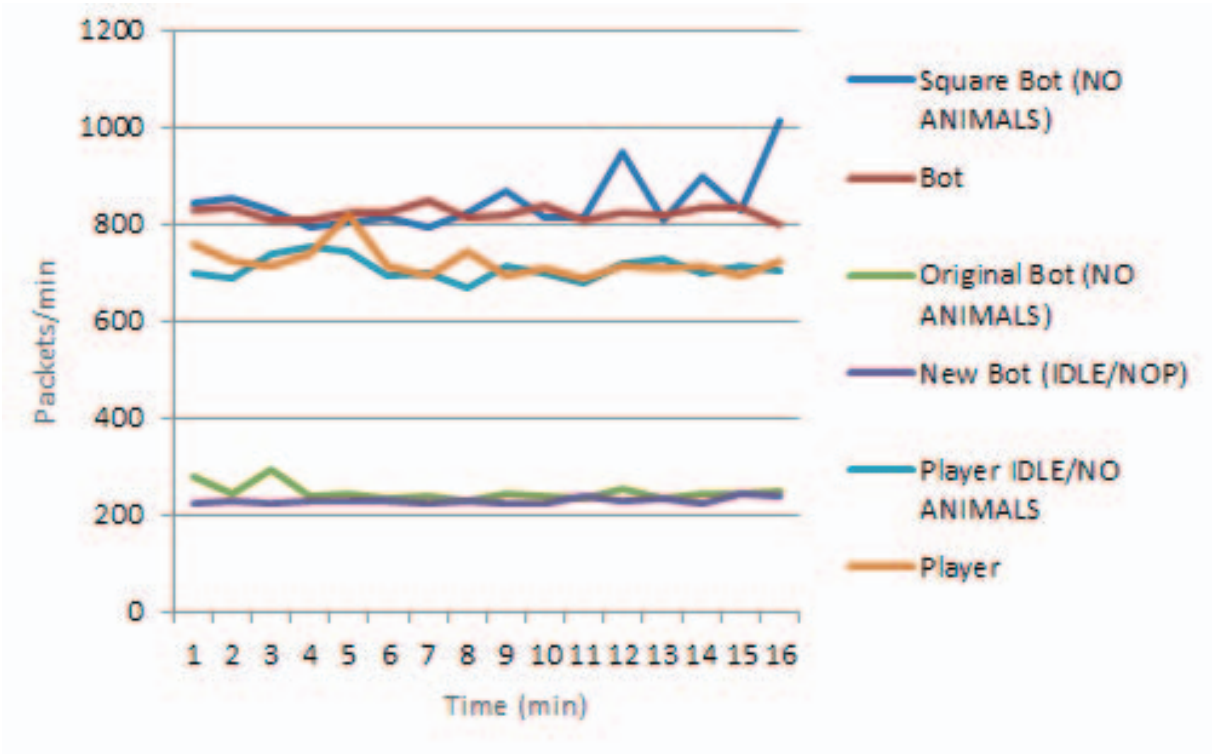

Fig. 14. Different game states.

VIII. Future Work

Our future work will be related to automatic game servers workload and network traffic optimization, and automatic online game deployment. By using collected statistical data related to network traffic and game servers performance we are going to develop predictive models for new servers provisioning. The game servers can be automatically deployed into new servers depending on the workload of the multicore multi-CPU game servers. "Later, the 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" [4].

Acknowledgment

The research project results, described in this paper were achieved under support of the Computer Science department at Okanagan College and by the NSERC of Canada in 2014 (ARD1 465659 - 14): GPN-Perf: Investigating performance of game private networks. The custom Bot was developed by the students in COSC 470 SW Engineering capstone project course [7].

References

[1] Mojang AB. Minecraft home page: `https://minecraft.net/`, 2015.

[2] Ahmed Abdelkhalek, Angelos Bilas, and Andreas Moshovos. Behavior and performance of interactive multi-player game servers. *Cluster Computing*, 6(4):355–366, 2003.

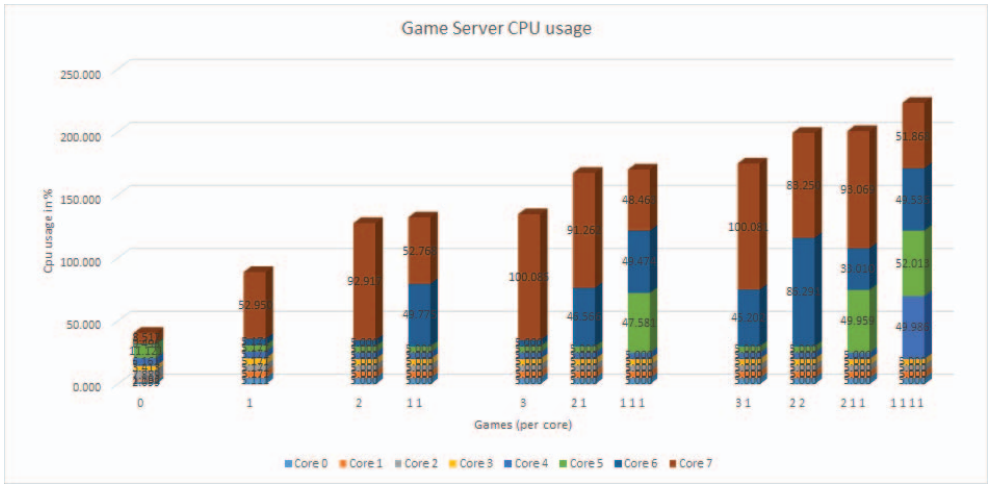

Fig. 15. Total CPU from 0 to 4 game servers deployed.

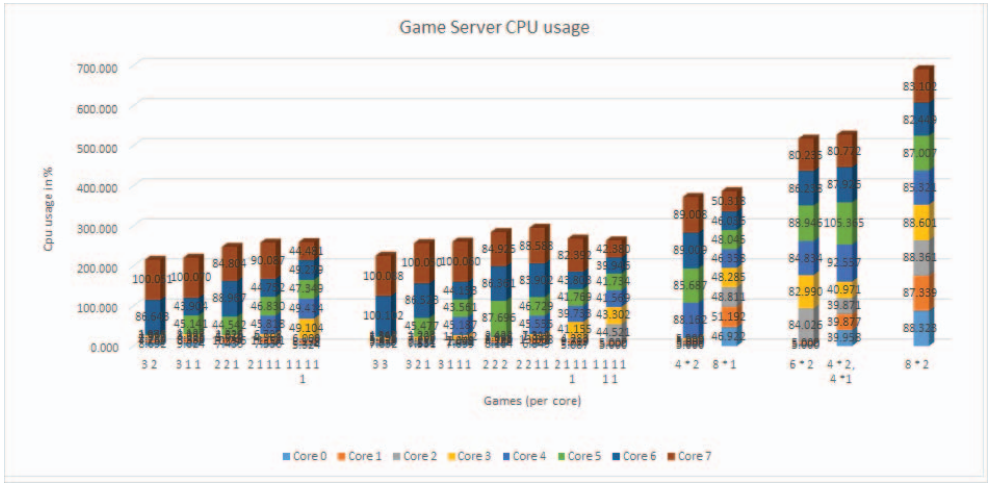

Fig. 16. Total CPU usage with more game servers deployed.

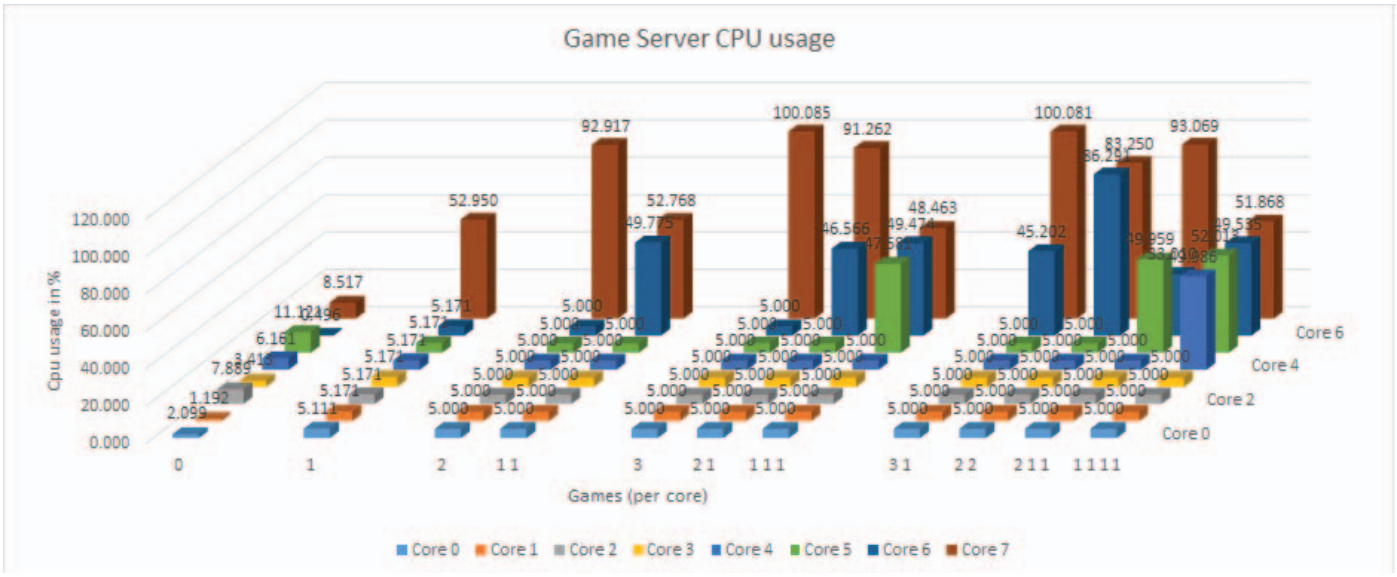

Fig. 17. CPU usage per core.

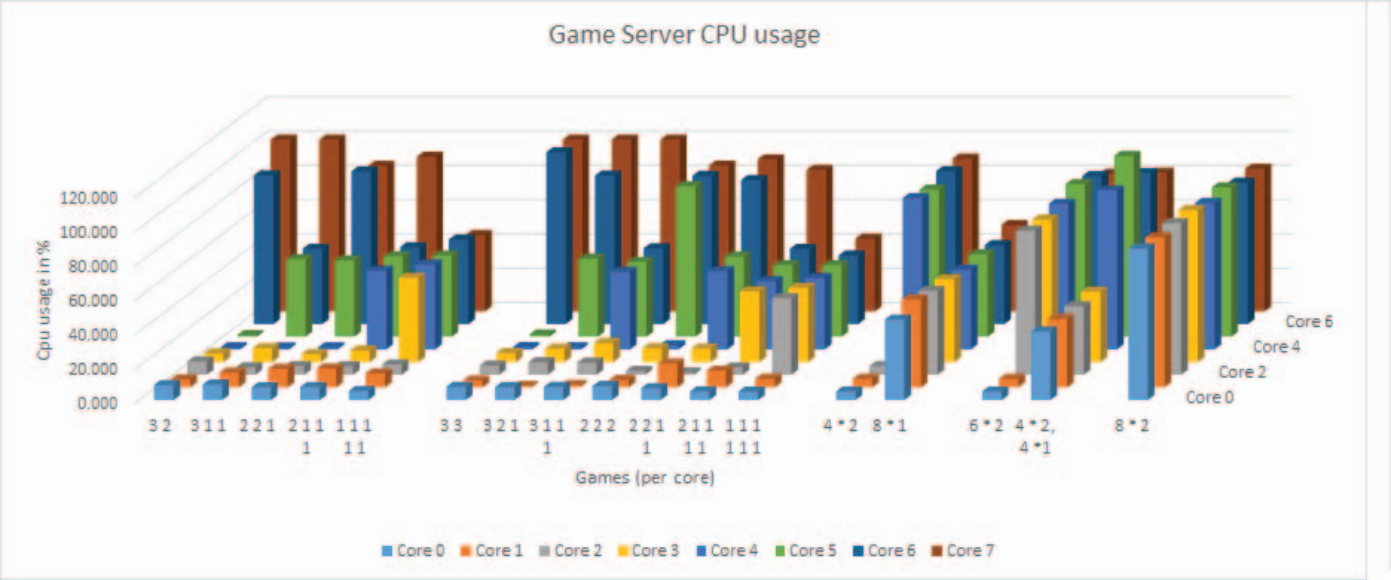

Fig. 18. CPU usage per core.

[3] 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.

[4] Trevor Alstad, J. Riley Dunkin, Simon Detlor, Brad French, Heath Caswell, Zane Ouimet, and Youry Khmelevsky. Game network traffic emulation by a custom bot. In *2015 IEEE International Systems Conference (SysCon 2015) Proceedings*, 2015 IEEE International Systems Conference. IEEE Systems Council., April 13-16 2015.

[5] 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.

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

[7] 2014. COSC 470 SW Engineering Capstone Project Course Team, Okanagan College. A short video clip with 50 bots running in a square: `https://www.youtube.com/watch?v=KYrIO7yWekw`, 2014.

[8] 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.

[9] 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/evrrt.html`, 1982.

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

[11] Gamepedia. Infiniminer: `http://tinyurl.com/o5plsbk`, 2015.

[12] 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.

[13] Inc. DarkStorm652/DarkBot GitHub. Minecraft thin client and automation framework: `https://github.com/Steveice10/MCProtocolLib`, 2015.

[14] Steveice10/MCProtocolLib. GitHub Inc. A library for communications with a minecraft client/server http://spacehq.org: `https://github.com/Steveice10/MCProtocolLib`, 2015.

[15] 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.

[16] 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.

[17] 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.

[18] Tom Jehaes, Danny De Vleeschauwer, Toon Coppens, Bart Van Doorselaer, Eva Deckers, W Naudts, K Spruyt, and R Smets. Access network delay in networked games. In *Proceedings of the 2nd workshop on Network and system support for games*, pages 63–71. ACM, 2003.

[19] CubeCoders Limited. Mcmyadmin 2 the minecraft control panel: `https://www.mcmyadmin.com`, 2015.

[20] 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.

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

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

[23] Wikipedia. Minecraft: `http://tinyurl.com/294abv9`, 2015.

[24] 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.

[25] 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.