Gaming Network Delays Investigation and Collection of Very Large-Scale Data Sets

Ben Ward, Youry Khmelevsky*
Computer Science, Okanagan College, Canada
Emails: ben.neil.ward@gmail.com
ykhmelevsky@okanagan.bc.ca
*Also Affiliated with UBC, Kelowna, Canada
†Also Affil. with LACL, Paris-Est Créteil, France

Gaétan Hains†
Huawei, France R&D Centre, Paris
Email: gaetan.hains@huawei.com

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

***Abstract*—The WTFast's Gamers Private Network (GPN®) is a client/server solution that makes online games faster. GPN® connects online video-game players with a common game service across a wide-area network. Online games are interactive competitions by individual players who compete in a virtual environment. Response time, latency and its predictability are keys to GPN® success and runs against the vast complexity of internet-wide systems.**

We have built an experimental network of virtualized GPN® components so as to carefully measure the statistics of latency for distributed Minecraft games and to do so in a controlled laboratory environment. This has led to a better understanding of the coupling between parameters such as: the number of players, the subset of players that are idle or active, the volume of packets exchanged, the size of packets, latency to and from the game servers, and time-series for most of those parameters.

In this paper we investigate specific game traffic connection types and show how we could collect very large-scale data sets of gaming metadata to control gaming network performance in real time. Such investigation and conducted experimentations will improve quality of service for GPN® systems and their reliability.

I. Introduction

The "GPNPerf" (2014-2015) project has built a laboratory version of a Games Private Network® (GPN) that is used for extensive and controlled-environment experiments to investigate the conditions of low and stable latency in online games. Experiments conducted since 2014 with the Minecraft network game have produced an ever-increasing quantity, quality and variety of measurements.

Our key objective was to understand the evolution of network traffic volume, latency of network + game server responses and game server CPU loads. Those target variables are measured against a mixture of:

- Time (as time series in minutes or seconds)
- Number of server VMs
- Number of physical cores/CPUs to run the servers
- Number of human game players
- Number of artificial (bot) game players
- Game-idleness or action of the players

Initial analysis published in 2015 led to the explanation of most measurement's average and standard-deviation values [1]. Precisely because our experiments are controlled, the measured time-series are constant or almost flat lines with a degree of noise that accounts for standard deviations of a few percent to 10% or 20%. The possible variation in game environment and server configuration are not mixed within those measurements but rather explored with multiple experiments, each one having a fixed environment and server configuration. This allows mapping the multi-dimensional space of game evolutions in a rational manner.

One experimental dimension that cannot be covered by our laboratory network is internet-scale measurements. To compensate for that lack of scale, the "GPNPerf2" project (2016-2019) is collecting a very large collection of meta-data of game network statistics.

To this end we introduce in this paper the data collector that will allow us to gather internet traffic performance data from hundreds and potentially thousands of networking hosts, served by hundreds of thousands of game servers in realistic online game environment. Once the data is completely analyzed it will lead to the exploration of specific scenarios and in particular those that lead to network overload or bottleneck (unlikely with games but important for general applications) as well as dynamic strategies for reinforcing GPN reliability and low-latency.

The next sections summarize existing and relevant works in the area of game networks, our study of existing platform and tools for the data collector's implementation, its design, implementation and measurements that allow us to conclude that it is a top-performance tools for collecting massive amounts of client-to-game-server network statistics.

II. Existing works

Predictable and sub-second response time has long been a key concern for interactive computer systems [2]. 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 [3]. 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 [4]. 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 [5], [6], 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. [7] 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 [8] 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 [9] 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 games' 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 [10] 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.

More recent studies [11], [12] of first-person shooter games have modelled 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 [13] 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 [14]. 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.

A study of different first-person games shows that the client traffic is characterized by an almost constant packet and data rate [15]. 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 [1].

Our research mostly concentrates on the servers' performance optimization, additionally to the network traffic analysis [16] and design and implementation of the custom bot for Minecraft [1]. As it was shown in [17] 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 or FPS [11], [12] and network traffic for such games [9]. They investigate network impact on the games and realistic traffic generators. In our infrastructure our aim was not just to 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 [14] online games become major contributors to Internet traffic. Latency is the another challenge for online games, as it's reported in [10], [7] 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 [16]. In [6] 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 [5] 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 patent [4] 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 [2] 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 divided 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 [8] the 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 [13] the authors investigated a multiplayer on-line game traffic including modelling traffic in mobile networks.

III. Specific Game Traffic Connection Types

A core concern of GPNPerf2 is the analysis of GPN traffic (see Table I and Table II) to characterize the statistical- and time variation of message latency. Those parameters is both the hearth of the business model and the key qualify feature of

TABLE I
ONLINE GAME SYSTEM ARCHITECTURES [19]

Peer-to-Peer	Peer-to-Peer (P2P) systems where game players interact directly with each other. They have not been very popular in recent systems.
Client-Server	Client-Server systems where a set of logically central servers store the game state and provide most operations except player interaction and complex view rendering (in a local client). This architecture has been very popular recently.
P2P Client-Server Hybrid	Peer-to-peer\Client-Server hybrid (lockstep) Hybrid types of implementations.
Cloud Gaming	Cloud Gaming where the client is a very thin piece of software are almost all functions are realized by the server. Like all cloud platforms, services or software, its advantages are in easy deployment, no installation for the players etc. But its disadvantages lie in heavier computation and communication. In [19] bandwidth requirements multiplied by 10 to 1000 for cloud games by comparison with traditional client-server games.

games networks. Indeed Saldana and Suznjevic [19] in *QoE and Latency Issues in Networked Games* have summarized existing literature that, among other things, highlights the insensitivity of most online games to bandwidth while confirming that latency is critical to most games except perhaps games of strategy.

In the rest of this section we summarize those concepts and results of [19] that define precise measurement and modelling objectives for GPNPerf2.

a) Game genres i.e. types of online games:

- *First Person shooters (FPS):* Games such as *Call of Duty* where the user sees himself as an armed warrior evolving alone or among a team of a few dozen members to eliminate virtual enemies. Average time between firing and death of the enemy is about 161 ms for the most popular FPS games. Studies have confirmed that such games require very low latencies.
- *Massively Multiplayer Online Role Playing Games* (MMORPG) are games where thousands of players, merge with artificial entities into a complex virtual world. They cooperate or fight each other so that, (simulated) firearm exchanges happen and put a constraint on latency as in FPS games but at a lesser frequency by the nature of the game. The element of tactics is also important in such games so virtual-world coherence is critical.
- *Real Time Strategy* (RTS) games where a dozen players share a virtual worlds where they build "civilizations" i.e. geometric and slowly-dynamic structures.
- *Multiplayer Online Battle Arena* (MOBA) games, a special-case of RTS where two teams try to conquer a battlefield.
- *Sports games* that simulate car racing or team sports. In vehicle racing it is possible that latency can be critical while team- or other realistic sports involve the simulation of balls, running humans and other objects that are very slow relative to the "firearms" and "bullets" of FPS games.

In [19] surveys that for FPS games, a one-way delay of 80 ms can be acceptable for most game users. Low latency convinces users to join the game, which confirms its importance for game-related business that is aimed at latency-reduction or latency-stabilization.

For MMORPG games, players started rating the game quality from "excellent" to "good" when one-way latency raised above 120 ms. When it rose further above 150 ms up to 200 ms, players started leaving their game sessions. The same phenomenon has been observed when latency in RTS games rose from 200 ms to 500 ms.

Studies have also shown that experienced players are more sensitive to those factors than ordinary players.

b) Connection Types: Network games are implemented with a variety of network connection types.

c) Geographical location: Geographical location of servers is correlated with latency for obvious reasons of transmission delays. Many games report the geographical location of their servers so players connect to the closest ones. Mapping IP addresses to geographical location is necessary for experimental and mathematical investigation of the above questions.

d) Latency reduction: Once empirical and mathematical tools are built to analyze and predict message latency and its variance, the results will be used to predict human- and business-effects in various games. For example player QoE in specific situations defined by combinations of the above factors (game genres, connection types, etc). In turn, the effect of this perceived or real QoE can be correlated to game session durations, popularity of the game service etc, and in the end of business objectives and economic factors for game players and GPN providers.

Once this analysis is put in place it will be natural to apply latency-reduction techniques such as *zoning* and *mirroring*. Zoning partitions the virtual world into geographical areas called zones, handled independently by separate machines. Mirroring, targets parallelization of game sessions with a large density of players located and interacting within each other's geographical vicinity.

e) Methods to enhance QoE: Scalability:

- *Logical Sharding:* A virtual world spans over multiple servers.
- *Sharding:* Multiple instances of separate but identical virtual servers.
- *Zoning:* Separable locations in a virtual world are handled by different servers.

TABLE II
NETWORK GAME SCALING

Peer-to-Peer	Connection Type: TCP or UDP Quality of Experience (QoE) considerations: No server authority (cheating/hacking). No hardware considerations for game developer. Game Example(s): StarCraft 1 (1998) Game Genre(s): RTS
Client-Server	Connection Type: TCP or UDP Description: Clients synchronize to a single source which propagates the synchronization outwards. QoE considerations: Central server authority for anti-cheating, pay-gate, etc. Ability to deploy dedicated servers if included with game (LAN parties w/ low latency). Individual players game-client may also act as server (Game interruption if player leaves). Game Example(s): Counter Strike, Warcr aft 3, Call of duty, Minecraft Game Genre(s): FPS, MMORPG, Sandbox
Cloud Gaming	Description: Game logic and virtual world rendering occurs server-side and a video stream is sent to the client. The client only sends commands to the server. Game Example(s): Playstation Now Game Genre(s): Game Streaming service

- *Mirroring:* Distributes a zone load by replicating the zone on multiple servers. The state of a subset of active entities is calculated by each participating server and communicated between servers.
- *Instancing:* Sharding on a smaller(zone) level.

IV. THE DATA COLLECTOR

To investigate gaming network performance issues our project requires collecting from upwards of 400,000 client's gaming sessions network latency data in order to statistically analyze and improve their network performance. Using the open source programming language *golang*, a data collection web application (named GPerf2 Collector, or just the collector), was constructed that is able to accept a vast number of rapid incoming connections. The collector then transmits some amount of data to a data store before disconnecting. This operational information from clients, would first be authenticated by the collector and then at some point bulk transmitted to some form of data store as was possible. Experiments were conducted as to the best way to create and tear down connections to achieve the optimal cross between the greatest number of connections and the greatest volume of data processed in the future. The goal is for the collector to be able to handle at least one million connections per second.

The collector server generates a number of workers which listen for incoming TLS connections. When a worker receives a request it uses a multiplexed handler function to authenticate the data packet, extracts the information, generates a job object using the extracted information, and adds the new job object to the job channel. The job channel is a queue from which objects are processed and inserted into a bulk Elasticsearch processor which finally sends a bulk packet of data to a data store. The server also regularly prints updates on the screen with the number of connections processed, the incoming connection's host and IP address as well as the target Elasticsearch index.

The server has a configuration file that contains general server operations, profiling options, and Elasticsearch options. The general server operations involve opening a specific port, maximum packet size and connection keep alive options. The profiling options determine the type of profiling the application will record which include memory, cpu and blocking. Elasticsearch options regard functionality of the Elasticsearch processor and the IP address of the Elasticsearch server location. Other configuration variables determine the number of workers and jobs the server will be able to handle.

The server has three changeable variables which vary the server's capacity that can be found at the top of the main function. MaxQueueSize sets the number of jobs that the job channel can hold. MaxWorkers dictates the number of workers that are generated to listen for connections. The more workers there are the more connections that can be handled in a shorter period of time; however, adding more workers will quickly consume more resources.

V. FRAMEWORK BENCHMARKING

The initial performance comparison of web frameworks for the use of developing the Collector was obtained from TechEmpower [20], an online organization that performs standard benchmarking tests on web frameworks. Their round 12 tests, which concluded February 25th of 2016, were hosted on a static environment which was created according to best practices and community input. Each instance of crowd submitted frameworks are implemented and go through a variety of standard tests for comparative analysis. The frameworks evaluations are comprised of JSON serialization, single query, multiple queries, fortunes, data updates and plaintext. TechEmpower's benchmark framework used 'wrk', a load simulator, to send a request packet containing 20 updates at a rate limited only by the infrastructure and framework. The test results for each web framework are contrasted to observe performance results.

The GPERF2 project decided on Aliaksandr Valialkin's fasthttp web framework implemented in golang based off of the results from TechEmpower's round 12 test results. Fasthttp data update was able to handle 3,959 requests (20 update queries) over 15 seconds. The update request's average latency was 62.9 ms with a standard deviation of 62.3 ms and a maximum of 893.7 ms. These results as well as the plaintext test results were the closest to resembling what we wanted for performance. Further information regarding test frameworks utilized are shown below in Table III.

TABLE III
TECHEMPOWER TEST ENVIRONEMENT

Hardware	**Dell R720xd dual Xeon E5-2660 v2** (40 HT cores) with 32 GB memory; database servers equipped with SSDs in RAID; switched 10-gigabit Ethernet **i7: Sandy Bridge** Core i7-2600K workstations with 8 GB memory (early 2011 vintage); database server equipped with Samsung 840 Pro SSD; switched gigabit Ethernet **EC2: Amazon EC2 c3** large instances (2 vCPU each); switched gigabit Ethernet (m1.large was used through Round 9)
Operating System	Ubuntu Linux 12.04 64-bit Windows Server 2012 64-bit
Databases	MySQL MongoDB PostrgeSQL
LoadSimulator	Wrk
Tests	JSON serialization Single query Multiple queries Fortunes Data updates Plaintext

We implemented our own tests to validate and benchmark our fasthttp application. The results of our tests are shown in Table IV and Table V (see the first line with the GPerf2 results), where we have almost 30 times more requests # in 15 seconds and much less average latency to compare with the other frameworks. The Plaintext Test in Table V is shown just for the comparison only.

Fig. V.1 shows the number of requests the frameworks are able to handle for plaintext HTTP GET requests. The results of our test can be seen in the tables. Our collector web application is shown to be considerably slower than fasthttp's implementation of 6 million requests per 15 seconds by a factor of 10. This can be explained by using the *wrk* benchmarking tool on a lower powered computer over a 100 Mbps switch. Fig. V.2 further shows that the latency is quite low much like fasthttp. The system is actually faster however the tests were done on the same network that the collector which explains how much faster the system is. Fig. V.3 shows the number of database queries that the frameworks can handle. Due to utilizing a buffered method to accept database queries we are able to accept queries at a higher rate as compared to the frameworks shown here. This however is a trade off as we run into buffer bloat (latency created due to network hardware buffering too much data) which can slow down the system if the collector receives requests at a faster rate than it can execute them. This is shown in Fig. V.4 where the overall latency is lower but the standard deviation is 50%.

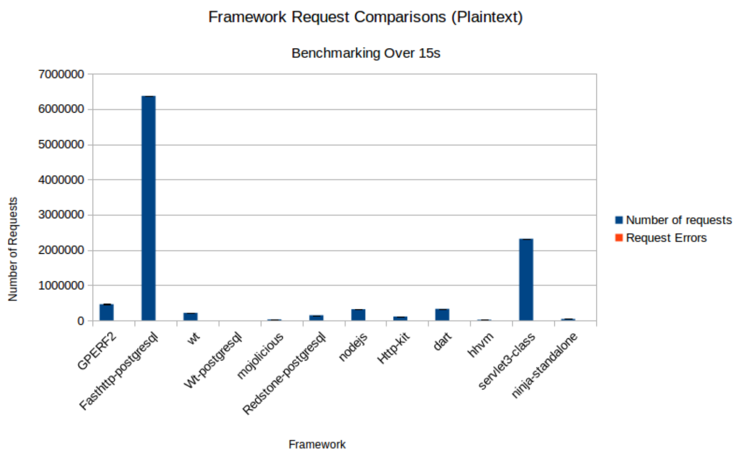

Fig. V.1. Plaintext Requests Comparisons

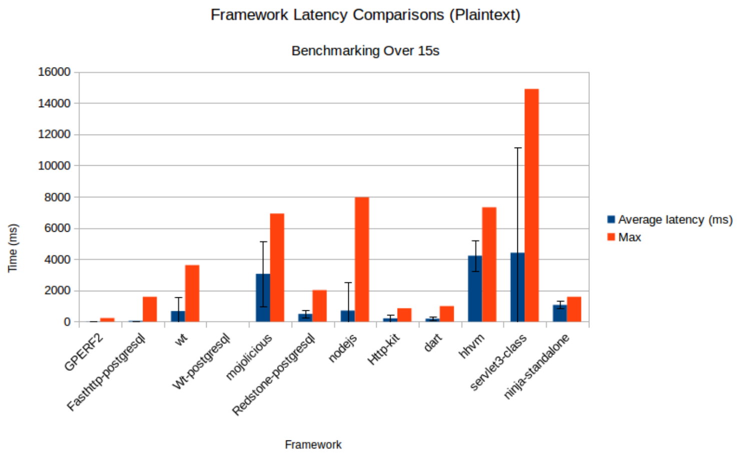

Fig. V.2. Plaintext Latency Comparisons

Future tests of the system will utilize remote benchmarking machines to further emulate the test environment executed by TechEmpower as well as performing longer tests to show the long term functionality and determine what affects bufferbloat will have on the networked machines.

VI. CONCLUSION

In this work we have built the elements and general structure of a web application that accepts requests to insert data to a database from clients. The framework utilized was determined

TABLE IV
UPDATE TEST

Framework	Language	# of requests (in 15 sec)	Average latency (ms)	σ (SD) (ms)	Max (ms)	Request Errors
GPerf2	Go	98816	26.86	36.5	279.23	256.9
Fasthttp-postgresql	Go	3959	62.90	62.30	893.70	0
wt	C++	3583	67.60	28.00	731.70	0
Wt-postgresql	C++	3344	74.30	27.00	146.50	0
mojolicious	perl	3157	78.00	37.70	285.50	101
Redstone-postgresql	dart	3148	76.80	25.80	179.20	0
nodejs	JavaScript	2862	84.00	11.70	152.60	0
Http-kit	Clojure	2727	88.00	7.70	124.10	0
dart	Dart	2539	97.70	37.00	329.40	0
hhvm	Php	2472	114.80	66.50	714.60	0
servlet3-class	Java	2298	106.60	18.00	214.20	0
ninja-standalone	Java	2165	108.10	140.60	700.20	3006

TABLE V
PLAINTEXT TEST

Framework	Language	# of requests (in 15 sec)	Average latency (ms)	σ (SD)	Max	Request Errors
GPerf2	Go	456756	9.09	10.71	229.35	0
Fasthttp-postgresql	Go	6371358	47.5	8.2	1590	0
wt	C++	210989	679	915.3	3620	0
Wt-postgresql	C++	N/A				
mojolicious	perl	25960	3060	2060	6920	0
Redstone-postgresql	dart	142724	493.5	226.4	2020	0
nodejs	JavaScript	314418	710.4	1830	7960	0
Http-kit	Clojure	107446	213	216.1	856.3	0
dart	Dart	324392	188.9	107.9	997.2	0
hhvm	Php	15074	4220	1010	7320	0
servlet3-class	Java	2314918	4410	6770	14890	0
ninja-standalone	Java	38005	1070	235.4	1590	0

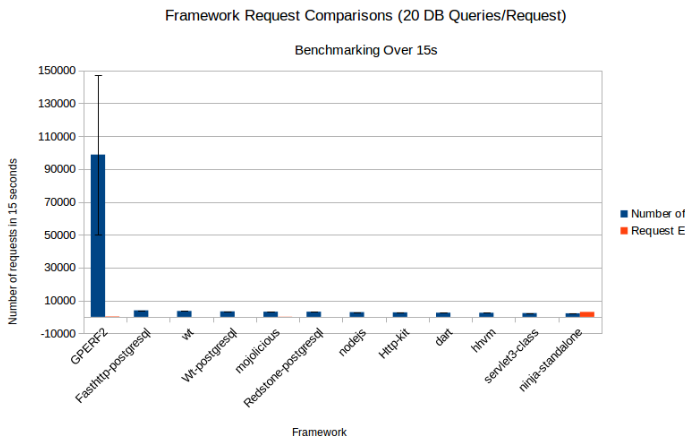

Fig. V.3. Database Insert Request Comparisons

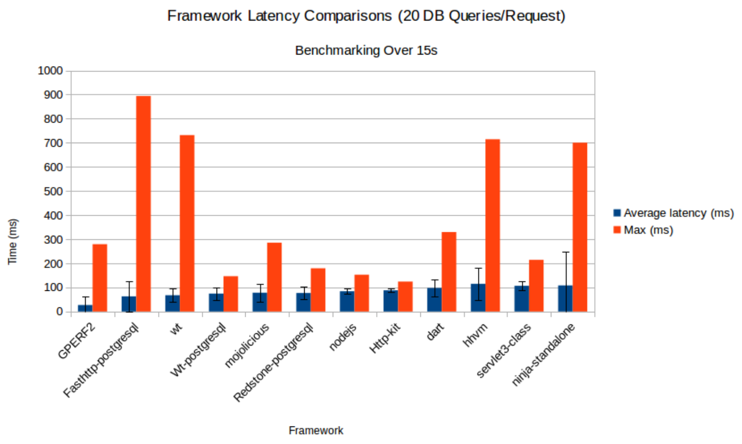

Fig. V.4. Database Insert Latency Comparisons

through tests employed by TechEmpower which were then emulated to confirm that the framework performed in the way described. It can be seen from the data that the tests were successful. The current performance measurements indicate that our collector outperforms other solutions for collecting similar client-server statistics. It should lead to the collection of massive datasets whose analysis will improve the performance of GPNs and their reliability against fluctuating internet- and online game conditions.

ACKNOWLEDGMENT

The research project results described in this paper were achieved with support from Computer Science department at Okanagan College and by the NSERC of Canada CCI ARD Level 2 grant in 2016: "GPN-Perf2: Game private networks and game servers performance optimization". This research is conducted in partnership with WTFast Inc.

References

[1] T. Alstad, J. R. Dunkin, S. Detlor, B. French, H. Caswell, Z. Ouimet, and Y. Khmelevsky., “Game network traffic emulation by a custom bot.” in *2015 IEEE International Systems Conference (SysCon 2015) Proceedings*, ser. 2015 IEEE International Systems Conference. IEEE Systems Council., April 13-16 2015.

[2] W. Doherty and A. Thadhani. (1982) The economic value of rapid response time (IBM Technical Report GE20-0752-0). [Online]. Available: http://www.vm.ibm.com/devpages/jelliott/evrrt.html

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

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

[5] T. Iimura, H. Hazeyama, and Y. 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*, ser. NetGames ’04. New York, NY, USA: ACM, 2004, pp. 116–120. [Online]. Available: http://doi.acm.org/10.1145/1016540.1016549

[6] J. Jardine and D. Zappala, “A hybrid architecture for massively multiplayer online games,” in *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*, ser. NetGames ’08. New York, NY, USA: ACM, 2008, pp. 60–65. [Online]. Available: http://doi.acm.org/10.1145/1517494.1517507

[7] J. D. Pellegrino and C. Dovrolis, “Bandwidth requirement and state consistency in three multiplayer game architectures,” in *Proceedings of the 2Nd Workshop on Network and System Support for Games*, ser. NetGames ’03. New York, NY, USA: ACM, 2003, pp. 52–59. [Online]. Available: http://doi.acm.org/10.1145/963900.963905

[8] P. Ghosh, K. Basu, and S. K. Das, “Improving end-to-end quality-of-service in online multi-player wireless gaming networks,” *Computer Communications*, vol. 31, no. 11, pp. 2685–2698, 2008.

[9] Q. Zhou, C. Miller, and V. Bassilious, “First person shooter multiplayer game traffic analysis,” in *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, May 2008, pp. 195–200.

[10] M. Claypool and K. Claypool, “Latency and player actions in online games,” *Commun. ACM*, vol. 49, no. 11, pp. 40–45, Nov. 2006. [Online]. Available: http://doi.acm.org/10.1145/1167838.1167860

[11] P. A. Branch, A. L. Cricenti, and G. J. Armitage, “An ARMA (1, 1) prediction model of first person shooter game traffic,” in *Multimedia Signal Processing, 2008 IEEE 10th Workshop on*. IEEE, 2008, pp. 736–741.

[12] A. L. Cricenti and P. A. Branch, “A generalised prediction model of first person shooter game traffic,” in *Local Computer Networks, 2009. LCN 2009. IEEE 34th Conference on*. IEEE, 2009, pp. 213–216.

[13] Y. Wu, H. Huang, and D. 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*, vol. 3, June 2006, pp. 1811–1815.

[14] B. Hariri, S. Shirmohammadi, and M. R. Pakravan, “A hierarchical HMM model for online gaming traffic patterns,” in *Instrumentation and Measurement Technology Conference Proceedings, 2008. IMTC 2008. IEEE*. IEEE, 2008, pp. 2195–2200.

[15] J. Färber, “Traffic modelling for fast action network games,” *Multimedia Tools and Applications*, vol. 23, no. 1, pp. 31–46, 2004.

[16] T. Alstad, J. R. Dunkin, R. Bartlett, A. Needham, G. Hains, and Y. 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.

[17] A. Abdelkhalek, A. Bilas, and A. Moshovos, “Behavior and performance of interactive multi-player game servers,” *Cluster Computing*, vol. 6, no. 4, pp. 355–366. [Online]. Available: http://dx.doi.org/10.1023/A:1025718026938

[18] T. Jehaes, D. De Vleeschauwer, T. Coppens, B. Van Doorselaer, E. 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*. ACM, 2003, pp. 63–71.

[19] J. Saldana and M. Suznjevic, *QoE and Latency Issues in Networked Games*. Singapore: Springer Singapore, 2015, pp. 1–36.

[20] “Web framework benchmarks,” November 2016. [Online]. Available: https://www.techempower.com/benchmarks/

[21] H. Al-Bahadili, *Simulation in Computer Network Design and Modeling: Use and Analysis: Use and Analysis*. IGI Global, 2012.

[22] B. Marchenko and S. Leonid, “Liniear stochastic processes and thier applications (in russian).” Naukove Dumka, 1975.

[23] K.-T. Chen, P. Huang, C.-Y. Huang, and C.-L. Lei, “Game traffic analysis: An mmorpg perspective,” in *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video*, ser. NOSSDAV ’05. New York, NY, USA: ACM, 2005, pp. 19–24. [Online]. Available: http://doi.acm.org/10.1145/1065983.1065988

[24] Y. Khmelevsky, “Parameterization and Statistical Analysis of Stochastic Signals in Biological Research,” Abstract to the Ph.D. thesis, The Institute of Simulation Problems in Power Industry, Kyiv Scientific Typography at Scientific Book Publisher, Kyiv, Ukraine, September 1992.

[25] E. Westrub and F. Pettersson, “Using the go programming language in practice,” Master’s thesis, Lund University, 2014.

[26] S. M. Ross, *Introduction to probability models*. Academic press, 2014.

[27] ——, *Introduction to probability and statistics for engineers and scientists*. Academic press, 2004.

[28] S. Mallick, G. Hains, and C. S. Deme, “An alert prediction model for cloud infrastructure monitoring,” 2013.

[29] M. AB. Minecraft home page. [Online]. Available: https://minecraft.net/

[30] COSC 470 SW Engineering Capstone Project Course Team, “A short video clip with 50 bots running in a square.” Computer Science Department, Okanagan College. [Online]. Available: https://www.youtube.com/watch?v=KYrIO7yWekw

[31] Gamepedia. Infiniminer. [Online]. Available: http://tinyurl.com/o5plsbk

[32] GitHub, Inc. DarkStorm652/DarkBot. Minecraft thin client and automation framework. [Online]. Available: https://github.com/DarkStorm652/DarkBot

[33] GitHub Inc., Steveice10/MCProtocolLib. A library for communications with a minecraft client/server. [Online]. Available: https://github.com/Steveice10/MCProtocolLib

[34] C. Limited. Mcmyadmin 2 the minecraft control panel. [Online]. Available: https://www.mcmyadmin.com.

[35] Wikipedia. Minecraft. [Online]. Available: http://en.wikipedia.org/wiki/Minecraft