Game private networks performance: analytical models for very-large scale simulation

Gaétan Hains, Youry Khmelevsky, Rob Bartlett, Alex Needham

2016 IEEE International Conference on Cybercrime and Computer Forensic (ICCCF) (pp. 1-10). IEEE.

Game Private Networks Performance: Analytical Models for Very-Large Scale Simulation

Gaétan Hains*
Huawei, France R&D Centre, Paris
Email: gaetan.hains@huawei.com
*LACL, Univ. Paris-Est Créteil, France

Youry Khmelevsky†
Computer Science, Okanagan College, Canada
Email: ykhmelevsky@okanagan.bc.ca
†Computer Science, UBC Okanagan, Canada

Rob Bartlett and Alex Needham
WTFast, Kelowna, BC Canada
Emails: {rob, alex}@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 present a mathematical model of those system game network parameters and show how it leads to: (1) realistic simulation of each of those network or game parameters, without relying on the experimental setup; (2) very large-scale numerical simulation of the game setup so as to explore various internet-wide performance scenarios that: (a) are impossible to isolate from internet "noise" in their real environment and; (b) would require vast supercomputing resources if they were to be simulated exhaustively. We motivate all elements of our mathematical model and estimate the savings in computational costs they will bring for very large-scale simulation of the GPN®. Such simulations 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® 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 is 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. 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.

We are now building a better understanding of this space with a mathematical model of all the experiment variables and will continue to enrich it so as to cover the full variety of game situations for realistic network optimization. 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) will design, implement and use a mathematical simulation of very-large-scale game networks. To this end we introduce in this paper the model elements that will allow us to simulate hundreds of thousands of players, served by hundreds of thousands of game servers in realistic game simulations. Parallel computing, numerical modelling and symbolic computation will all be applied towards this end.

Once the model is completely implemented it will lead to the exploration of specific scenarios and in particular those that lead to server crashes or network overload (unlikely with games but important for general applications). The model and its simulations will thus become an experimental bench for characterizing some "critical" situations and allow internet applications to have a database of "situation signatures" to enable alert capabilities and better reaction times. This project objective is thus to have game networks serve the general benefit of internet reliability.

II. Model Elements

In this section we explain the mathematical model elements.

The time series are modelled as Markov chains, as in the work of Mallick-Hains-Deme [1] for predictive monitoring.

Each time-series vertical axis is divided into sequence of intervals from its lowest possible value to its largest

possible value. For example if the value is a percentage we could divide its vertical axis into 10 buckets $[0, 10\%), [10\%, 20\%), \ldots, [90, 100\%]$. The choice of 10 intervals arbitrary but creates a balance between precision and computational cost: n intervals yield an $n \times n$ Markov matrix as we now explain.

We consider the observed time-series as a trace of a stochastic change of vertical level i.e. as the evolution of a Markov chain. Following the classic textbook [2] section 10.6, we estimate the transition probabilities of this Markov chain by the observed frequencies of state changes. In our example the Markov chain's transition matrix M is 10×10 and $M(i, j)$, and the transition probability from state i to state j is estimated by the fraction of observed transitions $M(i, \_)$ that have j as destination. For example if interval 1 is $[0, 10\%)$ and interval 3 is $[20, 30\%)$ then $M(1, 3)$ is the fraction of immediate changes from a state in $[0, 10\%)$ that lead to a state in $[20, 30\%)$.

Once the Markov matrix of a time-series is estimated in this way, it can be used to produce random but realistically similar time-series by tracing one-dimensional random walks with the matrix:

1) compute the steady-state probabilities of being in each state, or just estimate that distribution of states by their frequency in the time-series.
2) draw an initial state randomly with probability distribution given by 1. Call this state $X(t) = i$.
3) draw the next state $X(t+1)$ with probability distribution given by $M(i, 1), M(i, 2), \ldots, M(i, n)$.

Many other things can be computed from the matrix but for the purpose of our initial model this will be sufficient. From an observed time-series compute its estimated Markov chain matrix M and from M an infinite variety of similar time-series can be cheaply and simply computed.

The scaling factors are modelled (naively for now) by polynomial interpolations and extrapolations. For example we found that some value is approximately quadratic in the number of bots b so we fitted a degree-two curve on this data with dependent variable b.

A. Future model components:

Time-series combined with scaling curves allow us to extrapolate the time series. In other words a linear- or quadratic fit will be found to a parametric series of time-series e.g. a similar time-series for $b = 1, 2, 3, \ldots$ may be extrapolated and simulated to a larger value of b. As always, the quality of this simulation will depend on the quantity of our statistical data.

Also, game-state variable vectors have yet to be analyzed and modelled. Initial analysis of data in this direction is described below in section III-E.

Finally Khmelevsky's model with signal + noise function networks will be applied in future versions of the model. This could lead to less naïve "glass-box" models of the network and perhaps interact intelligently with game-state simulation.

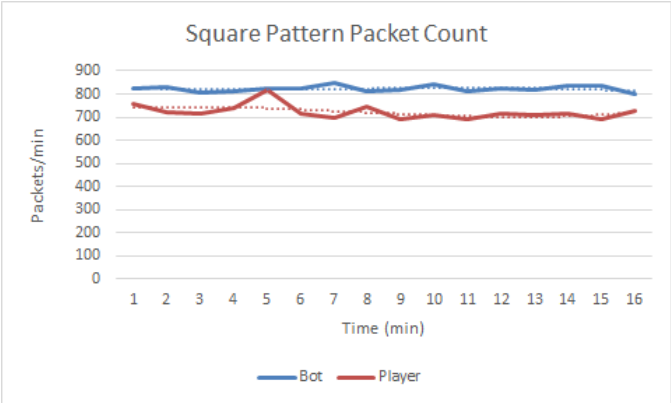

Fig. III.1. Square pattern experiment.

III. Measurement Data

We begin by organizing and then analyzing the measurement data. For each type of dataset we propose a mathematical model element to be used in the general model construction.

A. Packet count time-series for human and bot players

Here we compare square pattern group packet count for human and bot player. The square pattern is a specific game situation for which 16 minutes of network packet counts have been measured for both a bot (artificial player) and (human) player (Fig. III.1). The unit is packets/min.

The experiment (Table I) shows that similar communication volumes of 700-800 packets/min have been generated with a slight but consistent difference of $+14\%$ more packets for the bot. We analyzed the time series with a Markov chain to prepared for large scale simulation. Data for the bot sample is at Table I.

The 11 states of the corresponding Markov chain have been built by intervals 5-wide as follows at Table II.

Their relative and cumulative frequencies are used to generate a random initial state as follows (Table III). We generate a pseudo-random number between 0 and 1. If for example it falls between 81 and 87% then state 8 is chosen. The bot time-series Markov chain M_bot has been computed as explained in Section II.

A probabilistic simulation of the bot's time-series is obtained by:

1) Drawing a random initial state (explained above). This takes a number of instructions proportional to $O(\log S\_bot)$ where S_bot is the number of states. The reason is that the random number generation is in constant time and then searching through the sorted list of cumulative probabilities can be done by dichotomic search in logarithmic time.
2) From that initial state, for example state 4, draw the next state according to the probabilities in matrix line $M[4, \_]$. In our example this would mean 13 chance of remaining in state 4 and 23 chance of moving to state 5. This is also done in $O(\log S\_bot)$ operations by dichotomic search through the cumulative probabilities in that matrix line.

TABLE I
THE EXPERIMENT RESULTS

Time (min)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Bot	827	833	809	811	824	822	849	813	817	840	810	822	818	834	836	799
	7	8	3	4	6	6	11	4	5	10	4	6	5	8	9	1

TABLE II
11 STATES OF THE CORRESPONDING MARKOV CHAIN BY INTERVALS 5-WIDE

Bot state	Interval	Interval	Freq.	Cumul.
1	795	799	6%	0.06
2	800	804	0%	0.06
3	805	809	6%	0.13
4	810	814	19%	0.31
5	815	819	13%	0.44
6	820	824	19%	0.63
7	825	829	6%	0.69
8	830	834	13%	0.81
9	835	839	6%	0.88
10	840	844	6%	0.94
11	845	849	6%	1

TABLE III
THE BOT TIME-SERIES MARKOV CHAIN M_BOT

	1	2	3	4	5	6	7	8	9	10	11
1	9%	9%	9%	9%	9%	9%	9%	9%	9%	9%	9%
2	9%	9%	9%	9%	9%	9%	9%	9%	9%	9%	9%
3	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%
4	0%	0%	0%	33%	67%	0%	0%	0%	0%	0%	0%
5	0%	0%	0%	0%	0%	0%	0%	50%	0%	50%	0%
6	0%	0%	0%	0%	50%	50%	0%	0%	0%	0%	0%
7	0%	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%
8	0%	0%	50%	0%	0%	0%	0%	0%	50%	0%	0%
9	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
10	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%
11	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%

TABLE IV
THE TIME SERIES FOR THE HUMAN PLAYER

Player State	Interval	Interval	Freq.	Cumul.
1	690	694	19%	0,19
2	695	699	6%	0,25
3	700	704	0%	0,25
4	705	709	6%	0,31
5	710	714	19%	0,50
6	715	719	13%	0,63
7	720	724	6%	0,69
8	725	729	6%	0,75
9	730	734	0%	0,75
10	735	739	0%	0,75
11	740	744	13%	0,88
12	745	749	0%	0,88
13	750	754	0%	0,88
14	755	759	0%	0,88
15	760	764	6%	0,94
16	765	769	0%	0,94
17	770	774	0%	0,94
18	775	779	0%	0,94
19	780	784	0%	0,94
20	785	789	0%	0,94
21	790	794	0%	0,94
22	795	799	0%	0,94
23	800	804	0%	0,94
24	805	809	0%	0,94
25	810	814	0%	0,94
26	815	819	6%	1,00

Even if the number of states was about 1000, that would mean a simulation of T time steps could be completed in time $O(T \log 1000)$ $O(10T)$ or a few dozen Flops per time step. This means that for example one hour of this time series, simulated in steps of 1 minute would take time estimated to 60 x 10 $\sim$ 60 Flops or less than $100ns$ on modern architectures. At this rate, millions of such time-series can be simulated in a few seconds on a single processor or even a single core.

The time series for the human player required more states. It is listed here but the computational cost of using it for simulation is well below the estimate above for 1000 states (Table IV).

The player's time-series Markov chain M_player is 26 x 26 (Table V):

B. Multiple game servers on multiple cores, CPU load and balancing

We ran 1 or 2 or 3 Minecraft game servers per core and executed 10-50 bots against each game server and then collected information about each core CPU utilization. This was a stress test for each core. We started with 1 game server and finished 2 game servers per core out of 8 cores - totally 16 game servers. 3 game servers per core don't work — a core is used by 100% and games crashes.

For any given server-games configuration, experimental data yields a mostly flat time-series of CPU load. This can be modelled by a Markov chain with the method described above and very low computational cost. But game servers are a more critical resource in the games network than network bandwidth or simulation costs. As a result we must model and later simulate CPU load against the number of games and server computational cores. A key factor is load-balancing between the cores: only two games can be run on each core.

The first experiment ran 5 games on 8 cores: 3 games on core 7, two games on core 6 and cores 0 to 5 were left "idle" (with only OS activity but no game functions). If we ignore the time dimension its results measurements are the following (Table VI).

As shown by the low standard-deviation, the CPU load time-series are quite flat. Their simulation as time series will therefore be extremely light in computation. Their average value is the most important statistic. As mentioned above that part of the test that loads core 7 with 3 games led to a crash with 100% CPU load. The test with two games on core 7

TABLE V
THE PLAYER'S TIME-SERIES MARKOV CHAIN M_PLAYER IS 26 X 26

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
1	0%	0%	0%	0%	67%	0%	0%	33%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
2	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%
3	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%
4	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
5	33%	0%	0%	33%	0%	0%	0%	0%	0%	0%	33%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
6	50%	50%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
7	0%	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
8	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%
9	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%
10	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%
11	50%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	50%
12	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%
13	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%
14	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%
15	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
16	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%
17	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%
18	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%
19	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%
20	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%
21	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%
22	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%
23	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%
24	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%
25	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%	4%
26	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%

TABLE VI
THE FIRST EXPERIMENT RAN 5 GAMES ON 8 CORES: 3 GAMES ON CORE 7, TWO GAMES ON CORE 6 AND CORES 0 TO 5 WERE LEFT "IDLE".

	Core 7 Game1 Average	Game2 Average	Game3 Average	Core 6 Game1 Average	Game2 Average	Core5	Core4	Core3	Core2	Core1	Core0
Average	34,0	33,4	32,6	42,2	44,5	1,0	1,2	5,0	7,8	4,8	8,7
StDev	0,6	0,6	0,4	1,0	0,9	0,3	0,3	0,4	0,5	0,7	0,7

loaded it with $\sim (42 + 44) = 86\%$ of its capacity, while the idle game instances only loaded those cores by a variable but low value of 1-9%. Immediate conclusions could be that:

- Running one game per core is never an optimal use of hardware resources
- Running three games per core is impossible or very risky
- Running two games per core puts the server hardware in a state close to CPU overload but stable in time (very low standard deviation).

A second experiment ran 10 games on 8 cores with measurements as follows: two games on core 7, two games on core 6, one game on core 5 and cores 0 to 4 in game-idleness (Table VI).

Here the load on Core 7 and Core 6 is very similar to that on Core 6 during the previous experiment: a total of 86% CPU load for Core 7 or Core 6. The game-idle run on Cores 0 to 4 led to a similar behaviour as before: between 1 and 7% load. But the run on Core 5 yielded a load of 44% despite executing a single game. The following conclusions are now formulated:

- Running one game per core is not an optimal use of hardware resources
- Running three games per core is impossible or very risky
- Running two games per core puts the server hardware at almost 90% CPU load, but stable in time (very low standard deviation).
- An idle game-server instance load its CPU core by 1 to 7%, stable in time.

A third experiment was run with 6 games on 8 cores: Five cores were left idle, two cores ran two games and one core ran three games (Table VIII).

The 3-game run led to a crash by CPU overload and the other runs confirmed our previous measurements. The conclusion on this set of experiments is rather simple:

- An idle game-server running alone on a CPU core loads it by less than 9%, stable in time.
- Two game servers running on the same CPU core load it by a little less than 90%, stable in time.
- Running three games per core is impossible or very risky.

One straightforward conclusion is that our laboratory setup can run experiments for $2 \times p$ game servers when installed on a p-core physical machine. As we saw earlier, millions of traffic volume time-series can be simulated easily, so a complete simulation would represent a 1000-fold or more scale

TABLE VII
A SECOND EXPERIMENT RAN 10 GAMES ON 8 CORES WITH MEASUREMENTS AS FOLLOWS: TWO GAMES ON CORE 7, TWO GAMES ON CORE 6, ONE GAME ON CORE 5 AND CORES 0 TO 4 IN GAME-IDLENESS.

	Core7 Game1 Average	Game2 Average	Core 6 Game1 Average	Game2 Average	Core5 Game1	Core4	Core3	Core2	Core1	Core0
Average	42,3	42,5	43,7	45,2	44,5	1,2	4,6	5,0	10,7	7,4
StDev	0,9	1,0	0,9	1,0	1,5	0,4	1,0	1,1	0,9	1,1

TABLE VIII
LOADS ON CORE 7 AND CORE 6.

	Core7 Game1 Average	Game2 Average	Game3 Average	Core 6 Game1 Average	Core5 Game1	Core4 Idle	Core3 Idle	Core2 Idle	Core1 Idle	Core0 Idle
Average	33,0	33,5	33,6	43,9	45,1	1,1	8,1	4,3	8,6	9,0
StDev	0,3	0,6	0,7	1,5	1,0	0,3	0,6	0,5	0,9	0,9

TABLE IX
TWO SERVERS-CORES LOADS

isol2	%CPU Average	Sdev
s1	20,4	0,42
s2	20,0	0,57
StDev	0,3	

TABLE X
THREE CORES

isol3	%CPU Average	Sdev
s1	19,2	0,58
s2	19,6	0,73
s3	17,9	0,55
StDev	0,9	

TABLE XI
FOUR CORES

isol4	%CPU Average	Sdev
s1	18,0	0,54
s2	18,8	0,98
s3	18,6	0,22
s4	18,4	0,26
StDev	0,3	

TABLE XII
FIVE CORES

isol5	%CPU Average	Sdev
s1	19,5	0,76
s2	18,9	0,37
s3	20,6	0,24
s4	20,5	0,30
s5	19,8	1,50
StDev	0,7	

TABLE XIII
SIX CORES

isol6	%CPU Average	Sdev
s1	18,8	0,25
s2	18,8	0,33
s3	19,6	0,43
s4	20,7	0,54
s5	19,8	0,37
s6	20,2	0,23
StDev	0,8	

improvement over explicit laboratory experiments with our current hardware. It remains to measure how running one or two game-server per core affects game latency. Once that effect is understood, a mathematical model will be designed to simulate it over hypothetical- and very-large server configurations.

C. Multiple game servers on isolated cores and maximal CPU load

In those experiments we measured two to 6 game servers running on isolated CPU cores. Each one was running a game instance and serving 50 bots (artificial players). The value of 50 bots was found to be a maximum: beyond this capacity the game servers crashed. In each configuration the experiment was repeated ten times. The standard deviations over the ten runs are very low, on order of a few percent, which confirms the average values computed over the set of runs. With two servers-cores the loads are as follows (Table IX):

The right column means that on server 1 the value of 20,4% load only varied by +/- 0,42% over the 10 runs. Similarly for server 2 at 20% +/- 0,57%. This measures time variability of loads and is very low. The last line means that the variation between both servers' loads (20,4 or 20,0%) is $+/-0.3$: it measures load imbalance and is also very low. With three to six servers-cores the results are as follows (Table X, Table XI, Table XII, Table XIII).

In every case time-variations and load-imbalance are very low and we can neglect those effects unless further experiments indicate otherwise. There is no clear correlation between the number of server-core pairs and their loads. This is intuitively natural since their computational loads are independent, they are running different game instances and communicating

TABLE XIV
TRAFFIC VOLUME AND LATENCY AGAINST AN INCREASING NUMBER OF BOTS

n	Packets/s	KB/s	
1	70	5,8	measured
3	97	7,3	measured
5	251	18	measured
10	2101	160	measured

with different sets of 50 bots. The provisional conclusion of this experiments is:

- Up to six game servers running independent game instances, sharing the network but each serving 50 independent bots (artificial players) are each loaded at approximately $20\% CPU +/- 0,4\%$ stable over experimental runs and independent of the number of server-core pairs.
- Beyond 50 bots per server the same configuration led to a server crash.

It remains to explain how a $\sim 20\%$ CPU load can become an overload and server crash when running the experiment with more than 50 bots per game. In other words: what was the cause of this crash if the CPU load did not rise to 100% before it happened?

A possible explanation for this phenomenon is given in the next subsection.

D. Traffic volume for increasing number of bots. Maximal server capacity

Here we measured traffic volume and latency against an increasing number of bots.

In experiments conducted in 2014 and analyzed in 2015 we measured the (time-series of) network traffic in volume and number of packets for a variety of player configurations. From zero to two (human) players played with 1 to 10 bots (artificial players). Ignoring the time-variations (that can be modeled efficiently by Markov chains) we observed the rate of growth of traffic with an increasing number of bots (n) playing the game (Table XIV):

To extrapolate on this we applied a quadratic regression and obtained the following curves:

$$Packets/s = 214 - 159n + 35n^2$$

$$KB/s = 17 - 13n + 2.7n^2$$

Extrapolation of the traffic volume leads to the following values (Table XV):

So if 51 bots shared the network and shared also a game server they would have generated $8.9\ MB/s = 71\ Mb/s$ which is unlikely to have overloaded the switches. A possible explanation for the server crash could be an out-of-memory error in its queues or local data structures.

E. Network traffic vs game states and player types

Packet counts over time or over a series of tests. Here we measure traffic volume in packet counts. The measurements

TABLE XV
EXTRAPOLATION OF THE TRAFFIC VOLUME

n	Packets/s	KB/s	
1	70	5,8	measured
3	97	7,3	measured
5	251	18	measured
10	2101	160	measured
20	11034	837	extrapolated
30	26944	2057	extrapolated
40	49854	3817	extrapolated
50	79764	6117	extrapolated
51	83140	6377	extrapolated
60	116674	8957	extrapolated

TABLE XVI
EXPERIMENT THAT HAD A NON-NEGLIGIBLE TIME-VARIATION

Test (packets/min)	Average
Bot	2342
Player	2276

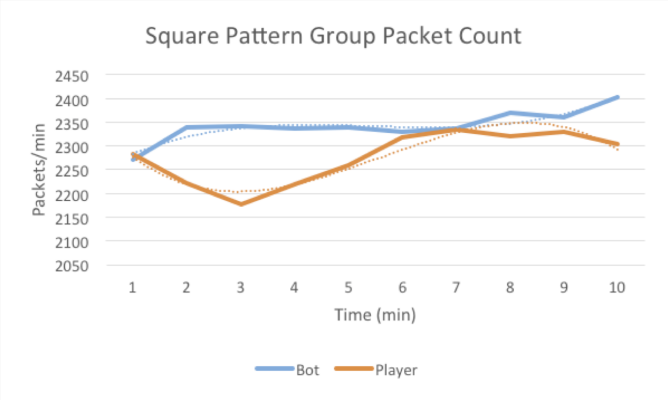

Fig. III.2. Square Pattern Group Packet Count.

are either a series of 16 statistical tests OR a time-series over 16 minutes (not to be confused). A novel and interesting dimension is the new varieties of player-bot. As was often the case, the controlled-environment experiments lead to rather flat time series or flat series of tests. In this initial analysis we ignore their time dimension, knowing that they can be modelled and efficiently simulated by Markov chains.

This series of experiments is an initial step towards of mathematical understanding and simulation of the game state. It measured the traffic generated by different types of players and player-situations. Values are in packets/min. In all cases the bidirectional traffic is very heavily dependent on the game state and the updates sent by the server.

The only experiment that had a non-negligible time-variation is a pure comparison between bot and human player (Table XVI).

The human produces almost as much volume as the bot but he does so with time variation of 5 to 6% (Fig. III.2).

Future experiments will analyze packet types to investigate this minor but unexplained difference between bot and player. It will be ignored in the rest of this section because we

TABLE XVII
BOT-VS-PLAYER IN A QUALITATIVE DIMENSION

Test (packets/min)	Average
Original Bot	647
Square Bot	2128
Square Player	2153

TABLE XVIII
A SECOND EXPERIMENT COMPARED MORE PLAYER-GAME VARIABLES

Test (packets/min)	Average
Square Bot (NO ANIMALS)	847
Original Bot (NO ANIMALS)	245
Player IDLE/NO ANIMALS	709
Player	722

concentrate on a less well understood aspect of our model elements: the multi-dimensional space of game states and player types.

A first experiment compared bot-vs-player in a qualitative dimension. The "original" bot is a passive software to emit packets that we recovered from an open-software source without adaptation to Minecraft. All other bots have been developed by our team specifically for the Minecraft experimental setup. "Square" refers to repeated behaviour and cyclic game-world geometry (Table XVII).

The results of this first experiment show that:

1) Minecraft-specific bots and human players generate 3.5 times more traffic than the original bots when in "square" behaviour
2) Square bots and square players are almost identical in traffic volume.

A second experiment compared more player-game variables (Table XVIII).

Here "NO ANIMALS" means that the game-state contains no animals and otherwise it does. Only humans moved in this experiment (neither bots nor animals did). Parameter "IDLE" means "just connected without activity" i.e. only connection for the voice communication. Initial observations can be made here.

- The "original" bot, non-specific to Minecraft, generates very little activity because it ignores the game world.
- The human player generates from 709 to 722 (+2%) traffic from an idle state without animals to an active state in the presence of animals. It has been observed that in a normal game world (hills, trees and animals) the server sends a lot of information about the environment.
- The square bot generates about 20% more traffic without animals than the idle player without animals. This is coherent with, but more significant, than the excess in volume generated by the bot vs the player in their pure comparison experiment above.

A last set of experiments with even more varied setups gave the following measurements (Table XIX).

Despite the early and incomplete nature of those experiments, some clear trends are emerging, thanks to the controlled-environment of our laboratory network. The following observations can be made and many others are possibly emerging from this cube of binary variables.

TABLE XIX
A LAST SET OF EXPERIMENTS WITH EVEN MORE VARIED SETUPS

Test (packets/min)	Average
Square Bot	2084
Square Player	2161
Square Bot (NO ANIMALS)	847
Original Bot (NO ANIMALS)	245
New Bot (IDLE/NOP)	229
Player IDLE/NO ANIMALS	709
Player Square Running (NO ANIMALS)	722

- The excess volume generated by a bot over a human player is not systematic: in the first two lines we see an active "square" bot generate 4% less traffic than the "square" active human player. In all cases bot and player generate similar traffic volumes for an equivalent situation.
- The square bot (NO ANIMALS) generates more than 3 times the volume of a passive "original" bot. That had already been measured in the second experiment above.
- Square behaviour takes a human player (without animals in the environment) from 709 to 722 (+2%) packets/min.
- The idle new bot's traffic is similar to that of the "original" game-oblivious bot at 229 or 245 packets/min.

For the purpose of simulation, this data is still insufficient but its use is clear: for a given game configuration we can estimate from the above tables the traffic volume generated by each type of player in its context. Then we can apply the quadratic curve-fit to yield traffic vs n (n = number of player type) as was done in earlier sub-sections for the bots. More experiments are needed for the exact curve fits for each type of player. Once that is done, we can extrapolate the volume generated for each type of player, and add them to obtain a total expected traffic volume. This can be done by simple arithmetic in time proportional to the number of types of players.

IV. LARGE-SCALE GPN SIMULATION

The last section has analyzed our experimental measurements and given us many model elements that will be integrated into a global game model. This model will allow the very-large scale simulation of game performance.

The datasets and their analysis should be completed by similar model elements for network latency. If latency can be related to our, now predictable, parameters of traffic volume and server loads, its simulation should follow.

For now we can already outline the general version of this black-box model and its computational cost.

1) Millions of time series can be simulated on a single computational core by their Markov chain model. That holds for all of our parameter's time-series because they are mathematically similar: relatively flat with noise but vertical level highly dependent on context.

2) Traffic volume can be estimated by a quadratic extrapolation in a few floating-point operations from the number of players, for a given type of player (confirmed for bots, to be confirmed for other types of bots). This quadratic form of curve is likely to be universal because we explain it through a basic property of game communications: state-changes from every player need to be communicated to every other player, hence their quadratic count.
3) Server CPU load is predictable and even constant for a given game state: zero, one or two servers per (simulated) physical core lead to possibilities of sharing the network with many game instances. The upper limit in number of players per game appears to be in the few dozens (e.g. 50 bots).
4) Varied types of players and game situations lead to network behaviors that are stable in time ("noise" variations of a few %) and whose quadratic growth curve for traffic is specific to the type of player-situation. Once the parameters are known for all important cases, they can lead to an estimate of the traffic in a very few arithmetic operations per type of player-situation.

The resulting model will operate as follows once calibrated: an automaton whose states are the known configurations of number of players-situations x number of servers and cores. It will generate a probabilistic Markov-chain sequence of traffic volumes, CPU loads, network latencies, until it changes state etc. This kind of simulation can be done over tens of thousands of players with a single computational core if a few dozen player-situations appear in each state. With a parallel program and a multicore or multi-node computer it can be lifted to dozens of millions of players simulated.

A second phase in the project will lead to a stochastic exploration of the game-state evolutions (which we only understand on their own for now: how many player-situations of each type) and this will lead to another dimension of Markov models. The result will a realistic very-large scale simulation of the game with its game-specific evolutions. In all cases the model can be scaled to the simulation of internet-size situations because it highly compact.

Once the model is completely implemented it will lead to the exploration of specific scenarios and in particular those that lead to server crashes or network overload (unlikely with games but important for general applications). The model and its simulations will thus become an experimental bench for characterizing some "critical" situations and allow internet applications to have a database of "situation signatures" to enable alert capabilities and better reaction times. This project objective is thus to have game networks serve the general benefit of internet reliability.

V. Existing works

Predictable and sub-second response time has long been a key concern for interactive computer systems [3]. 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 [4]. 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" [5]. 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 [6], [7], 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. [8] 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 [9] 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 [10] 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 [11] 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 [12], [13] 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 [15]. 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" [16]. 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 [17].

Our research mostly concentrates on the servers' performance optimization, additionally to the network traffic analysis [18] and design and implementation of the custom bot for Minecraft [17]. As it was shown in [19] 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)" [12], [13] and network traffic for such games [10]. 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 [15] online games become major contributors to Internet traffic. Latency is the another challenge for online games, as it's reported in [11], [8] and [20] and it's an important factor of an online gaming experience. We built our infrastructure to emulate artificial latencies in the emulated traffic [18]. In [7] "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 [6] 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 [5] 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 [3] 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 [9] 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 [14] the authors investigated a multiplayer on-line game traffic including modelling traffic in mobile networks.

VI. Conclusion

In this work we have built the elements and general structure of a mathematical performance model of GPN latency time-series in the presence of a variety of hardware-player configurations. Human vs artificial players are covered, scalability with the number of players, virtualization of game servers is also modelled with its (limited) scalability and effect on latency. Elementary aspects of the game-state have begun to be explored for their effect on latency: the presence of animals, idleness of players etc. Finally, the computational cost of numerically computing the model has been estimated. This provides an initial estimate of its application to very-large scale simulations.

The GPNPerf project aims at a deep understanding of game network latency. It appears natural not to consider bandwidth saturation effects for online games are not as heavy in traffic consumption as video streaming, data streaming for example. In all our experiments the amount of traffic was very low compared with the overall bandwidth of the network equipment. The only overload that could be measured is the computational load (CPU monitoring levels) on the game servers. That is being measured precisely and modeled mathematically. But currently game traffic never overloads the experimental network bandwidth and in practice never overloads internet bandwidth. Any configuration where bandwidth becomes a bottleneck is likely to correspond to a very degraded state of services. This is either beyond the scope of our research, or will be precisely estimated once our large-scale simulations are operational.

Our ultimate goal of a lab simulator for internet-scale GPN is now a little closer. Remaining work on the way contains at least the following elements:

- Extensive statistical testing of the time-series and of all aspects of the model.
- More realistic bots with game-state models and their lightweight simulation.
- Design of various game scenarios and system configurations to be simulated in the very-large internet scale.
- Choice of pseudo-random number generators with very long periods: to avoid simulation bias that is invisible at small scale but important for very-large simulations.
- Choice of the resulting trace, visualization or statistics to be produced out of each simulation for interpreting its result.

A general long-term question to investigate: is large-scale GPN simulation similar to weather prediction or more like the stock exchange? The former is chaotic but follows general laws while the latter is unpredictable beyond the near future or global tendencies.

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 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 [21].

REFERENCES

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

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

[3] 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

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

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

[6] 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

[7] 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

[8] 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

[9] 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.

[10] 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.

[11] 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

[12] 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.

[13] 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.

[14] Y. Wu, H. Huang, and D. Zhang, "Traffic modeling for massive multi-player 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.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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

[20] 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.

[21] 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