Predictive Performance Modeling for Distributed Live, Virtual, Constructive Environments

Dr. Rebecca Cebulka
Naval Air Warfare Center Training Systems Division
Orlando, Florida
rebecca.s.cebulka@navy.mil

ABSTRACT

Modern distributed simulations require a substantial network infrastructure – be it satellite links, broadband internet, radio communications, or a mix of all of these. Modern, large-scale live, virtual, constructive (LVC) simulations may require high numbers both of human-in-the-loop participants and constructed physical entities, each of which must communicate in some way with the others. Thus, when planning and executing a large-scale distributed LVC simulation it is extremely beneficial to have an accurate prediction of whether your network is capable of facilitating the necessary data transfer and communications quickly enough to enable real-time human interactions. We aim to provide a government-developed modeling and simulation tool which will allow predictions of an arbitrary network's data transfer and communication capability, extensible to eventually include computational capability predictions. Currently, we are able to simulate an arbitrary number of applications and users interacting via simulation processes submitting jobs by broadcast and multicast methods through a specified network infrastructure to be analyzed on remote servers, with both memory (RAM) and storage (SSD) elements, jitter functionality, propagation delays, and network loading due to various real-world properties. Each link between network elements has a specified propagation delay (artificial latency), maximum transmission unit, and bandwidth. These elements are particularly important, as excessive latency can be greatly detrimental to the processing requirements of large-scale distributed simulation events. We believe that the behavior of the model as the network parameters are varied is reasonable, and that this model could be useful in predicting and planning capabilities of larger distributed networks. We support this conclusion with data taken from a small test network architecture modelled after a development enclave in our lab.

ABOUT THE AUTHOR

Becca Cebulka is a Computer Engineer at the Naval Air Warfare Center Training Systems Division (NAWCTSD). She holds a Ph.D. and Master's degree in Physics from the University of Central Florida, in addition to a Bachelor of Science degree in both Physics and Mathematics from Rutgers University. Becca has experience in computational modelling of physical systems and has previously been published for her work in nanoscience education research as well as in experimental condensed matter physics.

Predictive Performance Modeling for Distributed Live, Virtual, Constructive Environments

Dr. Rebecca Cebulka
Naval Air Warfare Center Training Systems Division
Orlando, Florida
rebecca.s.cebulka@navy.mil

INTRODUCTION

Live, virtual, constructive (LVC) training has quickly become prevalent in the defense community as a vital tool with which the warfighter can hone skills, prepare for real-world scenarios, and maintain fleet readiness (Office of Naval Research, 2011; I/ITSEC, 2017). It has been quickly adopted due to its advantages over conventional training – more realistic training opportunities due to equipment usage, less risk to participants, and greater control and flexibility when designing training scenarios, among others. The advent of distributed LVC training, which can link together participants across the globe, has enabled even greater strides towards a well-prepared and well-maintained fleet. Advances continue to improve the environments used to conduct large-scale distributed LVC training exercises, which require high numbers of human-in-the-loop participants and potentially many thousands of constructive entities to communicate. The nature of this sort of distributed training requires a substantial secure network infrastructure (Harper, 2015; Kauchak, 2019) as its bedrock – a network infrastructure that is still undergoing modernization and expansion (Department of Defense, 2019). In the midst of this effort, a tool that allows for prediction of future network capabilities and limitations could prove vital to minimizing costs and maximizing LVC environments as well as efficiency. The purpose of this paper is to introduce the development of such a tool, which would allow predictive performance modeling of an arbitrary network's data transfer and communication capability. One specific way this modeling would apply to human performance metrics and training after action reviews is to facilitate an accurate measure of a human-in-the-loop's *human* response time for completing a specific task as opposed to the total response time. That is, to enable separation of the human delay from the total system delay. An area in which human vs. system delay is vitally important is in pilot simulations, where the issue of communication latencies and pilot response delays has been studied across multiple agencies (Binias, Myszor, Palus, & Cyran, 2020; Zingale, McAnulty, & Kerns, 2003; Rantanen, McCarley, & Xu, 2004; Holden, et al., 2019; Vu, et al., 2014; Consiglio, Wilson, Sturdy, Murdoch, & Wing, 2010).

In general terms, a predictive performance model aims to provide, using a probabilistic approach, a good fit to a testing data set that was not used to determine the model's parameters. Data analytics aims to analyze large volumes of raw data to look for trends and provide answers. Both predictive performance modeling and data analytics have been used both commercially and within the defense community as predictive tools for years. This includes insight into optimal resource-usage algorithms (Kapadia, Fortes, & Brodley, 1999), acquisition decision-making (Anton, et al., 2019; Schwartz, 2016), effectiveness of security processes (Beres, Mont, Griffin, & Shiu, 2009), processor task completion and wait times (Witt, Bux, Gusew, & Leser, 2019), and network traffic analysis (Azzouni & Pujolle, 2018; Joshi & Hadi, 2015). These insights allow for decisions regarding planning and layout of architecture, as well as eventual equipment purchases, to be made with a higher degree of confidence. Our particular interest is to provide insight into potential network bandwidth bottlenecks, excessive latency, and jitter effects on the network infrastructure that supports distributed LVC training events. We have chosen to approach this by utilizing an open-source, academically developed Java toolkit called GridSim.

GridSim is an academically developed Java discrete event simulator for distributed architectures and communication protocols. Its original objective was to examine resource allocation techniques in conjunction with system scalability and varying algorithms – it was designed in part to study distributed computing in fields such as astrophysics, network design and high-energy physics (Buyya & Assuncao, 2020). As originally described by the creators: "The GridSim toolkit provides a comprehensive facility for simulation of different classes of heterogeneous resources, users, applications, resource brokers, and schedulers (Buyya & Murshed, 2002)." There have been several functionalities added on since initial conception, including advance reservation (Sulistio & Buyya, 2004), network service prioritization (Sulistio, Poduval, Buyya, & Tham, 2007), and failure detection (Caminero, Sulistio, Caminero, Carrión, & Buyya, Extending GridSim with an architecture for failure detection, 2008). In essence, the toolkit provides classes which can be put together so as to build virtually any desired network topology, and simulates packet behavior and job processing within that topology. Several examples of GridSim being used to model various systems can be found in (Caminero, Sulistio, Caminero, Carrión, & Buyya, 2008; Rawat, Yadav, & Barthwal, 2015; Buyya, Murshed, & Abramson, 2002; Murshed & Buyya, 2001).

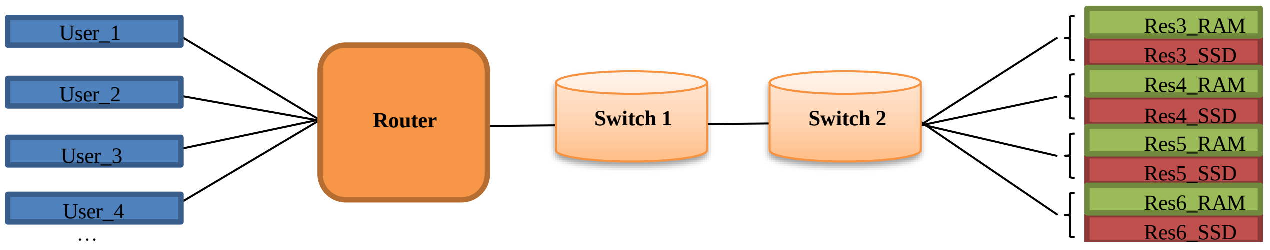
APPROACH

Our goal in this paper is to showcase our construction of a model using the GridSim toolkit in Java which is capable of simulating the following: an arbitrary number of applications and users interacting via simulation processes submitting jobs by broadcast and multicast methods through a specified network infrastructure to be analyzed on remote servers, with both memory (RAM) and storage (SSD) elements, jitter functionality, propagation delays, and network loading due to various real-world properties. We do this by following four steps: first, identifying a test network to simulate; second, building the model from the bottom up; third, estimating the parameters; and finally, running the model for a range of settings and examining the predicted behavior.

Identifying the Network

The initial test network architecture should simulate a basic development network: a small number of zero-clients (users) interacting with several processes or applications on one end, connecting through one Cisco ISR 4321 gateway router and two switches (another Cisco ISR 4321 configured as a layer 2-3 conversion switch and a Cisco 2960 X-TD used as a distribution core) to a collection of VMs hosted on a server cluster (resources). In the model, these resources are split into two partitions: memory (RAM) and storage (SSD), allowing us to simulate network traffic and communication packets designated as either or both types of processes. This is accomplished by setting two resource entities with different processing power to be linked together and referred to as ResX_RAM/ResX_SSD. The processing power itself is represented in the form of processing elements, PEs, which are analogous to a CPU core. The amount of processing power assigned to each partition is variable, so that more may be assigned to memory functions or storage capability as needed. Each resource also has an artificial propagation delay associated to it which represents real-world latency. As a key feature of this model, each resource propagation delay has the option to be set according to a random distribution to emulate some form of jitter.

The users create jobs, which can be based on a DIS protocol data unit for size and frequency of submission, that have a packet size in bytes for simulating packet transport and a measure in Million Instructions (MIs) for simulating processing. In the model there is the option to submit different amounts of jobs flagged as requiring storage (SSD) or memory (RAM) intensive access, redirecting each job to the appropriate partition of that resource. There is also the option to decide whether to send jobs from a single user to a single resource ("SUSR"), a single user to multiple resources ("SUMR"), or multiple users to multiple resources ("MUMR"). This makes it easy to implement unicast, broadcast, or multicast functionalities as desired; all three can be used in a single simulation if needed. The length, size, and number of jobs can be specified exactly, or randomized across some distribution. In essence, the jobs are created, transported through the router and switches, analyzed/processed on the servers, and returned to the users. This network topology is shown in **Figure 1**.



This simulation assumes a serial data transfer; that is, there are no other "invisible" systems on the network. In reality, of course, routers and switches work in a dynamic nature and typically are set up to utilize priority packets.

Figure 1. Initial Network Topology for Building the Model

Regarding the network behavior the model simulates, the router uses a First In First Out scheduler to direct jobs while the resources use a Time Shared scheduler to analyze them. There is also a limited background traffic generator, where each user can send junk packets at various intervals and of various sizes to random resources and other users; this is separate from the default background noise caused by IP fragmentation for larger packet sizes. This can be used to simulate the low-level network traffic inherent in using a zero-client of keyboard strokes, mouse movements, and graphically intensive applications. Each link between network elements has a specified propagation delay (artificial latency), maximum transmission unit, and, crucially, cable bandwidth. These elements are particularly important, as excessive latency can be greatly detrimental to the processing requirements of large-scale distributed simulation events, and bandwidth places significant limitations on the number of participants. It should be clarified here that we are currently concerned with direct, cable links between network elements; satellite links and their inherent challenges have not yet been explored.

Building the Model

The GridSim toolkit comes supplied with examples of how to use its various functionalities. Working through each example, each functionality was added to the overall model in turn and verified to behave appropriately. This was initially done for a set of default parameters (detailed in **Table 1**) that do not necessarily reflect any real-world networks. To assess the behavior of the model, the round trip time through the network of each job was examined. The equation used to calculate this round trip time is below

$$RTT = (time_{recieve} - time_{submit}) - time_{compute} \qquad (1)$$

where $time_{submit}$ is the simulation time at which the job is submitted and leaves the user, $time_{receive}$ is the simulation time at which the job returns to the user after having been completed, and $time_{compute}$ is the total simulation time spent by the job inside the resource, including waiting/scheduling time.

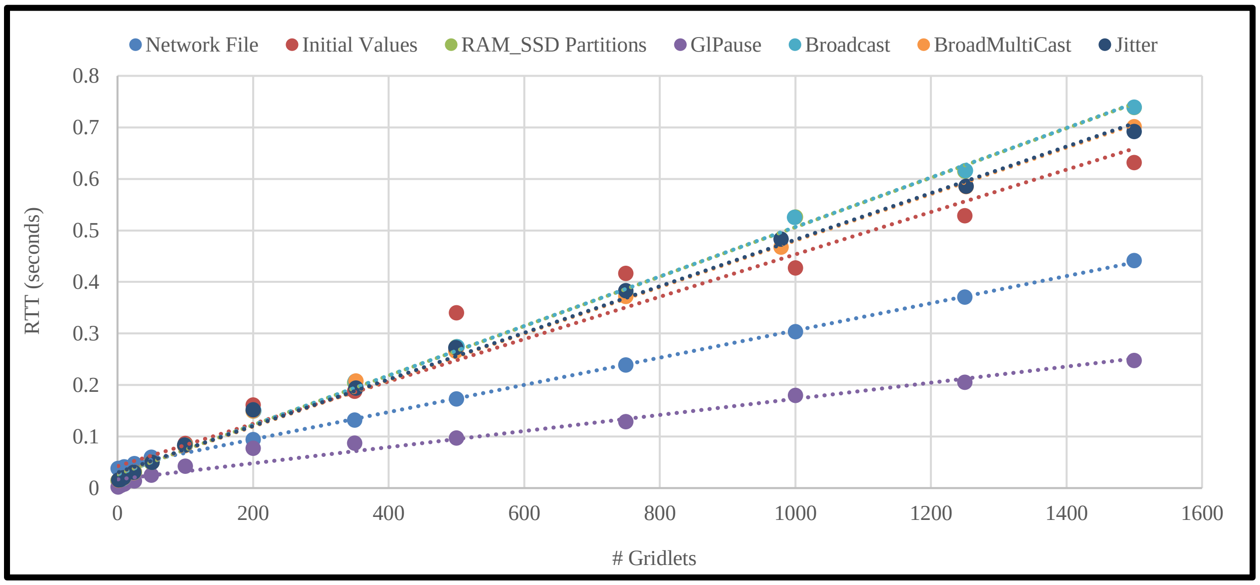
Table 1. Default Parameters for Building the Model

Variable	=	Value	Units	Notes
num_user	=	3	entities	One zero client connected to…
num_resource	=	3	entities	One server
totalGridlet	=	1	distinct jobs	Pretending to be a "ping" - a single packet of 32 bytes w/no computation that tracks RTT
length	=	500	MIs	Length of job to be completed on resource - affects computation time
file_size	=	200	bytes	Size of file sent to/from resource for each job
user_bandwidth	=	0.01	GB/s	Bandwidth of link to zero clients/users
res_bandwidth	=	0.1	GB/s	Bandwidth of link to servers/resources
RU_S1_bandwidth	=	0.025	Gbit/s	Bandwidth of link between router and first switch
S1_S2_bandwidth	=	0.025	Gbit/s	Bandwidth of link between router and second switch
user_propDelay	=	0.186673	ms	One-way delay due to the link between the zero client and network
res_propDelay	=	0.066673	ms	One-way delay due to the link between the server and network
RU_S1_propDelay	=	0.166668	ms	One-way delay due to the link between the router and first switch
S1_S2_propDelay	=	0.246668	ms	One-way delay due to the link between the first and second switch
bcktraffic	=	OFF		No junk packets sent as background traffic
freq_min - _max	=	9-10	pkts	Send ~ 10 junkpackets…
gap_min - _max	=	1-1.5	s	Every ~1 second…
pktsize_min- _max	=	1000-1001	bytes	With a size of ~1000 bytes
user_mtu	=	1200	byte	Assume the default for all network components is 1500
RU_S1_mtu	=	1500	byte	Assume the default for all network components is 1500
S1_S2_mtu	=	9198	byte	Assume the default for all network components is 1500
res_mtu	=	1200	byte	Assume the default for all network components is 1500
peRating	=	9600	MIPS	very rough estimate, only important for computationally intensive jobs
totalPE_RAM	=	14	PEs	# processing elements (like cores) for each RAM partition of the servers
totalPE_SSD	=	2	PEs	# processing elements (like cores) for each SSD partition of the servers

The steps to verify the model as it was built followed the examples provided in the toolkit. They are below in **Table 2**, along with a brief description of each. To briefly examine the effect of each functionality, the average RTT for each is plotted in **Figure 2** as a function of the total number of jobs sent.

Table 2. Steps to Verify Model

Functionality	Explanation
Network File	Loads the network topology from separate text file.
Initial Values	Uses approximate real-world values for parameters.
RAM_SSD Partitions	Duplicates each resource so that it has a RAM and SSD partition, with different processing power, and sends one out of every 5 of jobs to the SSD partition. This ratio is fully modifiable.
GlPause	Adds a pause functionality so jobs are not sent all at once. This literally stops the entity for some time, clearing the network and causing the next burst of jobs to be sent after that pause. *Use with skepticism, as pinging/pausing an entity effectively wipes the network and may throw off RTT calculations.*
Broadcast	Multiplies each job so that identical copies are sent to each resource.
BroadMulticast	Some subset of jobs are broadcast, the rest multiplied and sent out to a subset of resources.
Jitter	Randomizes the resource propagation delay to simulate jitter.



Note: This data is generated from the simulation.

Figure 2. Round Trip Time of Packets vs Number of Jobs as a Function of Model "Step"

Each run of the model shown above had exactly three users in communication with exactly three resources. Other than for Jitter, BroadCast, and BroadMultiCast, each user communicated with exactly one unique resource. For each step of verification, with the exception of Network File, the parameters were otherwise the same as listed in Table 1. The Network File step used the default parameters of the GridSim toolkit; these are listed in **Table 3**. The Initial Values step, as well as all subsequent steps, used more realistic value approximate to modern network and computing parameters; these are listed in **Table 4**.

The Network File bandwidths and propagation delays were significantly smaller than those of the other steps; this is reflected in the average RTT of the Network File step being much lower than all other steps – with the glaring exception of GlPause. GlPause introduced the ability to pause between sending Gridlets, so that jobs are not all sent concurrently. In the case shown above, the jobs were sent in five separate bursts with a delay of approximately 200 ms between them. The net effect of this seems to be clearing the network entirely and thus artificially deflates the RTT. See the Appendix for more discussion of this effect. The Parameters average RTTs is as expected higher than the Network File RTTs, and provides a baseline for the remaining steps, which use the same parameters and simply introduce new functionalities. Broadcast effectively cubed the amount of traffic, duplicating every job and sending it to each resource. This makes it the highest average RTTs of the steps. BroadMultiCast generated less traffic, with a portion of the jobs being only doubled and not tripled from each user. Thus it falls between the Broadcast and Parameters lines. Jitter was identical to BroadMultiCast, except for a randomization of the propagation delay to each resource. It follows the BroadMultiCast line very closely, with small deviations as expected.

Table 3. Network File Parameters

Variable	=	Value	Units	Notes
user_bandwidth	=	0.01	GB/s	Bandwidth of link to zero clients/users
res_bandwidth	=	0.01	GB/s	Bandwidth of link to servers/resources
RU_S1_bandwidth	=	0.01	Gbit/s	Bandwidth of link between router and first switch
S1_S2_bandwidth	=	0.01	Gbit/s	Bandwidth of link between router and second switch
user_propDelay	=	0.00000000001	ms	One-way delay due to the link between the zero client and network
res_propDelay	=	0.00000000001	ms	One-way delay due to the link between the server and network
RU_S1_propDelay	=	0.00000000001	ms	One-way delay due to the link between the router and first switch
S1_S2_propDelay	=	0.00000000001	ms	One-way delay due to the link between the first and second switch
user_mtu	=	1500	byte	Assume the default for all network components is 1500
res_mtu	=	1500	byte	Assume the default for all network components is 1500
S1_S2_mtu	=	1500	byte	Assume the default for all network components is 1500
totalPE_RAM	=	4	PEs	# processing elements (like cores) for each RAM partition of the servers
totalPE_SSD	=	4	PEs	# processing elements (like cores) for each SSD partition of the servers

Table 4. Initial Values Parameters

Variable	=	Value	Units	Notes
user_bandwidth	=	0.01	GB/s	Bandwidth of link to zero clients/users
res_bandwidth	=	0.1	GB/s	Bandwidth of link to servers/resources
RU_S1_bandwidth	=	0.025	Gbit/s	Bandwidth of link between router and first switch
S1_S2_bandwidth	=	0.025	Gbit/s	Bandwidth of link between router and second switch
user_propDelay	=	0.50005	ms	One-way delay due to the link between the zero client and network
res_propDelay	=	0.25005	ms	One-way delay due to the link between the server and network
RU_S1_propDelay	=	0.015	ms	One-way delay due to the link between the router and first switch
S1_S2_propDelay	=	0.0042	ms	One-way delay due to the link between the first and second switch
user_mtu	=	1200	byte	Maximum transmission unit for zero client links
res_mtu	=	1200	byte	Maximum transmission unit for server links
S1_S2_mtu	=	9198	byte	Maximum transmission unit for switch links
totalPE_RAM	=	14	PEs	# processing elements (like cores) for each RAM partition of the servers
totalPE_SSD	=	2	PEs	# processing elements (like cores) for each SSD partition of the servers

The final version of the model consolidated all the parameters and variables into the main file, added explicit variables to denote the ratio of both broadcast to multicast jobs and memory-based to storage-based jobs, included options to disable the jitter and background traffic functionalities, as well as enable a “single-user-single-resource” mode where each user talks only to one unique resource, and put in some safeguards against mismatched numbers of users and resources (see **Table 5**). After doing so, a comparison was made to the relevant steps before and after these improvements to verify the model was fundamentally unchanged.

Table 5. Explicit Variables Added to Verified Model

Variable	=	Value	Units	Notes
bcktraffic	=	FALSE		False means no junk packets sent as background traffic
susr	=	TRUE		True means each zero client talks to only one unique server
jitter	=	FALSE		False means no jitter implementation
multi	=	2	resources	# resources to multicast to
glPause	=	0		# of bursts Gridlets are sent in - default 0 or 1 means all sent at once
percBroad	=	20	%	% jobs broadcast to every resource
modSSD	=	0		Modulus to determine number of SSD jobs vs RAM (ie, every 5th is SSD)

Estimating Parameters

In order to establish a working model on a small scale, it needed to have parameters that approximate the test system's *network* and *computing* capabilities. It should be explicitly stated that the test system's "user computer" is in fact a zero client, and will therefore inherently have low-level background noise as it communicates back and forth over the network to the server. In order to minimize the effect of this, no other programs were running on the user zero client during testing – only keyboard and mouse inputs were occurring.

Computing Capability

The GridSim toolkit assesses processing power in terms of MIPS (million instructions per second). As no modern processor gives a MIPS rating, this must be approximated based on other measures: the servers in the lab environment network can be assumed to be at least on the level of an Intel Xeon Processor E5-2630 v3 (Intel, 2020; Microway, 2020). From specs on this processor* there should be 8 cores, with each core able to compute 16 floating point operations (FLOPs) per cycle, and a processor base frequency of 2.40 GHz. Total floating point operations per second (FLOPS) is a similar measure of computing power to MIPS; MIPS is more appropriate for database queries and running multiple virtual operating systems, while FLOPS is more appropriate for computational research and simulation-based computing. FLOPS is also a more accurate measure of computing performance overall. Given the FLOPs per cycle, we can calculate the total FLOPS:

$$FLOPS = \#\, cores \times \frac{cycles}{second} \times \frac{FLOPs}{cycle} \quad (2)$$

The total should thus be approximately 307 Giga FLOPS, or 38.4 Giga FLOPS per core. We can use this as a rough order-of-magnitude rating for the processing power of each core, or processing element. If a detailed study of computing performance is needed additional estimates will need to be run for each specific system, as a MIPS rating is inherently biased due to the type of instructions used. For the initial testing the number of cores in the memory (RAM) partition was set to be 7, leaving only one core for the storage (SSD) partition. This can easily be changed, and was only decided so as to simulate a system and interaction that is not storage intensive. Accordingly, all jobs were designated as memory-focused.

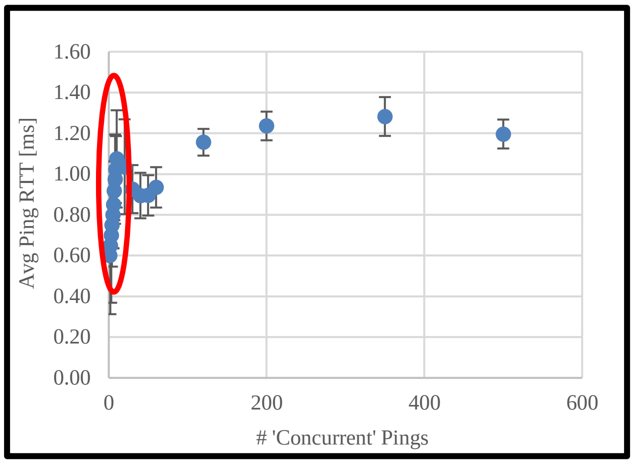
Network Performance

The basic setup for the network was from a zero client through the router and two switches to a host server, each dedicated explicitly to this scenario at the time of testing. Therefore, the number of users and resources was 1 each. The simplest way to test the real-world system was by a ping, so the number of jobs was set to one, with a size of 32 bytes and a length of 1 MI – this would simulate a small ping with little to no computational requirements. The bandwidth for the LVCDOC network should be capable of supporting an average of 5 Gigabytes per second (GB/s) transfer speeds without degradation (each zero client can reach speed up to 1 GB/s, while the internal network components reach speeds up to 10 GB/s). As the testing was the only project being performed on the network, no graphically intensive applications were being run on the zero client, and there was only one path available for the traffic to take, both the jitter and background traffic were turned off. The single-user-single-resource (SUSR) mode was turned on, as there was neither broadcast nor multicast traffic. No pausing was introduced between successive jobs being sent, and there was no split between jobs designated as memory- and storage-intensive. The maximum transmission unit for the network was 1500 bytes. Thus, the only parameter left to estimate was the propagation delays for the user, resource, router, and switch links. The highest delay should be the router, followed by the delay due to virtualization (the user), then switch forwarding latency, and finally the server I/O latency (resource delay), which should be almost negligible for a ping.

Ping Tests

In order to get some estimate for the propagation delay times (that is, the latency for each network element), a script was written to open multiple command windows and have each 'simultaneously' ping the zero client. It was necessary to perform this sort of backwards ping test because of the nature of zero clients, that the script was actually being executed on the host server itself. Thus the round trip time of the ping should be twice the time to travel from the host server to the zero client in front of me. The term simultaneous is used very optimistically – we were hopeful that the pings would be sent at least within a millisecond or two of each other, but that did not seem to be the case. In fact, for any number of windows greater than 2 there was typically more than a 10 ms difference between some of the pings. However, it is our belief that at least some useful information could be collected as a lower bound for delays.

Each command windows would send five pings, separated by 1 second, and write the average round trip time to a file. The script was repeated for a ping size of 8192 bytes and between 1 and 500 windows. The size of the ping, 8192 bytes, was chosen to be the maximum buffer size of the zero client – no larger pings can be received by it. Smaller ping sizes were attempted, but the very small latency made the data indistinguishable – there was virtually no difference in RTT as the number of windows increased. As may be expected, the more windows were opened, the larger the temporal spread of 'simultaneous' pings. Each number set of windows was averaged over 10 distinct runs of the batch file. While completing these tests, no graphically intense applications were running on the zero client, and neither should there have been any traffic other than minimal mouse and keyboard strokes. Thus the only PCoIP traffic should have been negligible. The results are plotted in **Figure 3** as the average round trip time (RTT) vs the number of simultaneous windows. Looking at this graph, there is a clear linear increase in RTT for less than 10 simultaneous windows opened (the red circle). The arrow shows the location of exactly 10 'concurrent' pings sent at a time. Above this value the RTT drops off, and even at much higher numbers of windows there is not much change. Thus the red circle denotes the area of interest where the ping RTT would be significantly more accurate.



Note: This data is generated by real-world testing.

Figure 3. Average Round-Trip-Time vs. # Concurrent Pings

Depending on the individual run, there is a different temporal concentration of pings, especially at higher numbers of windows. Fairly often, there are groupings of pings sent within 10ms of each other, corresponding to higher times. It is clear that the pings are not sent exactly concurrently, but that they can be somewhat grouped together to achieve a similar (but underestimated) effect – at least to a point. Looking at the graph, it appears that for more than about 10 concurrent pings (beyond the red circle), the temporal spread is just too great and there is not much significant effect in increasing the number of pings. In fact, it seems the pings may begin to spread out enough to cause a decrease in the RTT.

After experimenting with different propagation delays, it was possible to approximately match the results of the ping testing with the GridSim BlkDev model:

res_propDelay = 0.0055ms   user_propDelay = 0.0491ms

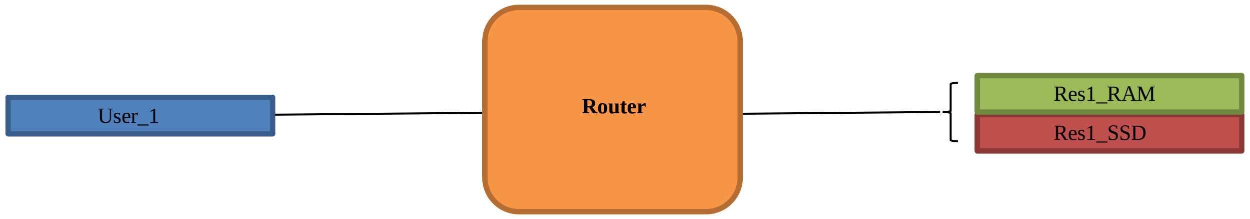
For the maximum size of 8192 bytes, these values give a RTT of approximately 0.60 ms for a single job and 4.04 ms for 10 jobs. While the graph above suggests the latter should be closer to 1.2 ms, it must be recalled that the pings had a significant break between them on the order of tens of milliseconds, while the jobs sent in the model may be considered to be exactly concurrent. In fact, during the ping tests for 10 windows, 60% of the runs contained RTTs of greater than 1 ms for closer groupings of pings. This lends credence to the assumption that 4 ms is in the right ballpark for this case.

MODEL PREDICTIONS

Final Parameters and Test-Case Setup

The generic network test scenario for the model consists of one user linked through a router to a resource with a large memory (RAM) partition and small storage (SSD) partition (shown in **Figure 4**). The smallest packet size commonly used today is 32 bytes, while the maximum buffer size of our user zero-client was as stated above 8192 bytes. Thus, aside from testing purposes, by default a single job of either 32 or 8192 bytes is sent from the user to the resource with minimal computation (a length of 1 million instructions, MIs – the smallest the program allows). The goal was to simulate the ping testing previously discussed between the zero client and its host server. The parameters listed in

Table 6 were used as static values, while one or more were varied to assess the capability of the model to predict round trip times (RTT) in the network.



*As in **Figure 1**, this simulation assumes a serial data transfer; that is, there are no other "invisible" systems on the network.*

Figure 4. Network Topology for Testing the Model

Table 6. Default Parameters for Building the Model

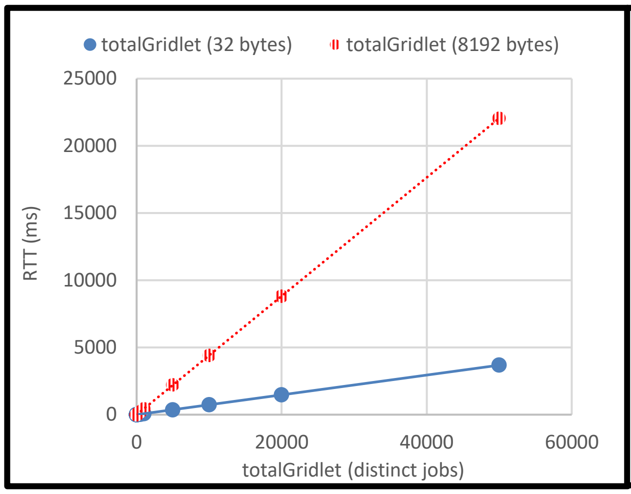
Variable	=	Value	Units	Notes
num_user	=	1	entities	One zero client connected to…
num_resource	=	1	entities	One server
totalGridlet	=	1	distinct jobs	Pretending to be a "ping" - a single packet of 32 bytes w/no computation that tracks RTT
length	=	1	MIs	Length of job to be completed on resource - affects computation time
file_size	=	32	bytes	Size of file sent to/from resource for each job
user_bandwidth	=	5	GB/s	Bandwidth of link to zero clients/users
res_bandwidth	=	5	GB/s	Bandwidth of link to servers/resources
user_propDelay	=	0.0491	ms	One-way delay due to the link between the zero client and network
res_propDelay	=	0.0055	ms	One-way delay due to the link between the server and network
jitter	=	FALSE		False means no jitter implementation
susr	=	TRUE		True means each zero client talks to only one unique server
bcktraffic	=	FALSE		False means no junk packets sent as background traffic
freq_min - _max	=	1-2	pkts	Send ~ 1 junkpacket…
gap_min - _max	=	1-1.5	s	Every ~1 second…
pktsize_min- _max	=	32-33	bytes	With a size of ~32 bytes
multi	=	2	resources	# resources to multicast to
glPause	=	0		# of bursts Gridlets are sent in - default 1 means all at once, no pause
percBroad	=	20	%	% jobs broadcast to every resource; set to zero if susr = TRUE
modSSD	=	0		Modulus to determine number of SSD jobs vs RAM (ie, every 5th is SSD)
user_mtu	=	1500	byte	Assume the default for all network components is 1500
res_mtu	=	1500	byte	Assume the default for all network components is 1500
peRating	=	38400	MIPS	very rough estimate, only important for computationally intensive jobs
totalPE_RAM	=	7	PEs	# processing elements (like cores) for each RAM partition of the servers
totalPE_SSD	=	1	PEs	# processing elements (like cores) for each SSD partition of the servers

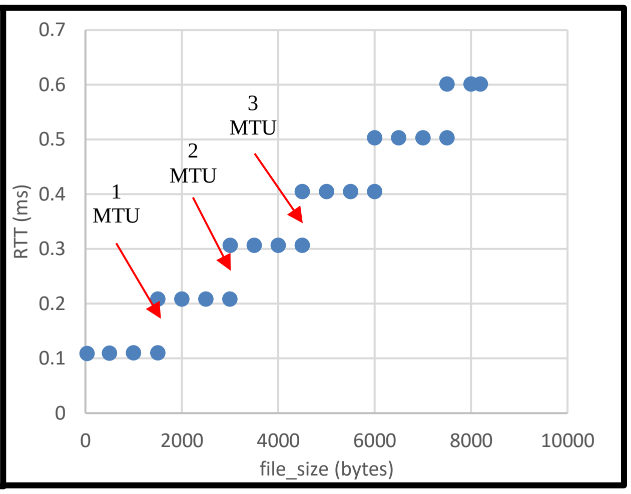
Results

As the number of jobs increases, the average round trip time increases linearly as well. This is shown in **Figure 5**. As expected, the slope is much higher for the 8192 byte jobs than the 32 byte jobs. Somewhat unexpectedly, even at extremely high number of simultaneous jobs the curve does not appear to bend upwards, an indication that the model does not take into account the eventual oscillation due to processing high volumes of jobs. If it were possible to send twenty 32 byte pings simultaneously, this model predicts it would take them an average of less than 1.4 ms to return – something barely detectable by Windows' default ping functionality. For the larger size of 8192 bytes, twenty simultaneous pings are predicted to have a RTT of about 8.4 ms.

Figure 6 shows data for a single job being sent (analogous to a single ping). The size ranges from below the maximum transmission unit of the network to several times that value. Thus, it would be expected to see some effect due to breaking up the job into separate packets near those multiples of the MTU – packet fragmentation. In order to look for this, simulations were run near both the upper and lower bounds for the size of the MTU (1500 bytes) around several multiples of it. For example, for a job size of 1499 bytes, and then 1501 bytes. There is a clear increase in RTT at each of these locations, shown for greater clarity by the red arrows.

As an additional check the real-world ping testing was redone from the host server to the zero client for a ping size of 1499 and 1501 bytes (10 concurrent pings). It was found that over an average of ten runs, the RTT for 1499 bytes was 0.09 ms while for 1501 bytes the RTT was 0.18 ms. This supports the assumption of an MTU of 1500 for the model.





Note: This data is generated from the simulation.

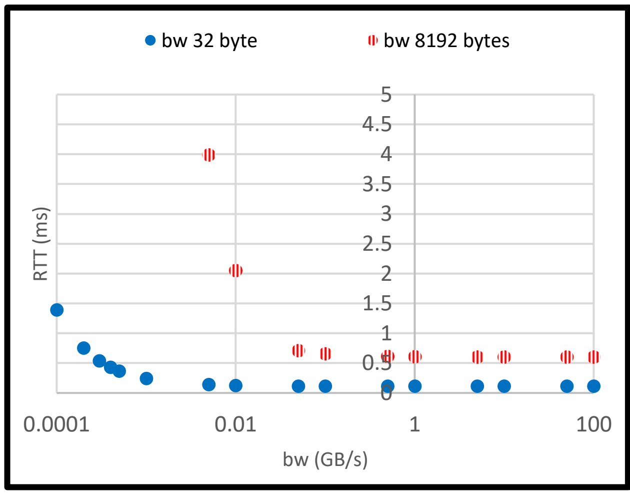
Figure 5. RTT vs Total Number of Jobs

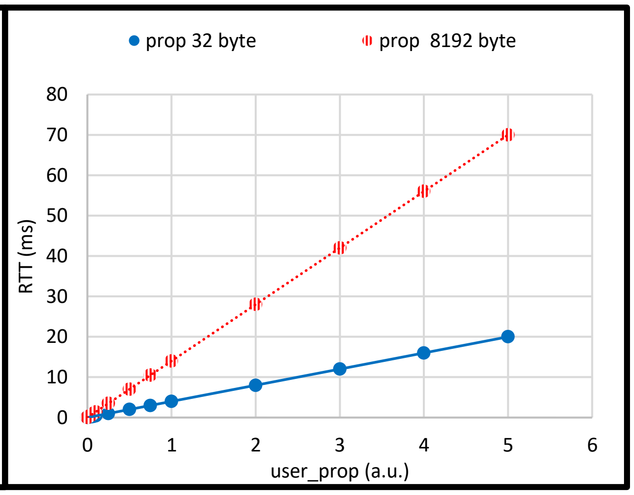
Figure 6. RTT vs Size of Each Job

In **Figure 7**, the bandwidth of every network component was kept equal and varied between 100 kB/sec and 100 GB/sec. As expected, the average RTT for the large file size is significantly larger than that for the smaller. The RTT also increase much faster, starting at about 0.1 GB/sec as opposed to 0.01 GB/sec. Most interesting to me was the fact that for 100 kb/sec in the large file size data set, the RTT was close to 190 ms. However, doing some back of the envelope calculations as below showed an expected RTT in the realm of 164 ms, which appears to be the correct magnitude.

$$RTT \approx 2 * \frac{amount\ of\ data}{speed\ of\ data\ transfer} = 2 * \frac{8192\ bytes}{100\ kbytes/sec} = \ 0.164\ sec \qquad (3)$$

As a final check, for 100GB/sec in the small file size data set the RTT was just the total one-way propagation delay for one packet multiplied by two - 0.1092 ms.





Note: This data is generated from the simulation.

Figure 7. RTT vs Bandwidth

Figure 8. RTT vs Propagation Delay

Finally, in **Figure 8** we examine the effect of propagation delay on RTT in the model following the same method as the bandwidth above, with the delay of every network component being kept equal and varied between 1 ns and 5 s. The larger file size sets also have a significantly higher RTT. The relation between increasing delay and increasing RTT is a linear one, exactly linear to be precise. This too is expected, as the propagation delay is effectively increasing the time for each packet to travel through the network by a simple addition.

CONCLUSIONS

In conclusion it is our belief that the behavior of the RTT as the network parameters are varied in the model is reasonable for an isolated laboratory system, and the model could be useful in predicting capabilities of larger networks. However, as it stands the model assumes a perfect world - there is no oscillation in RTT due to increasingly high numbers of jobs being processed on the server, bandwidth is exact and constant, and propagation delay is treated as an exact linear addition. These shortcomings could be addressed in several ways: an artificial oscillation in RTT could be introduced by adding periodic delays in computing time, and both bandwidth and propagation delays can be randomly varied within a set margin to closer approximate the physical network.

Of greater concern is the fact that for smaller networks with low latencies, it is quite difficult to get an accurate assessment of normal operating conditions as all round trip times will be less than one millisecond. The next step would then be to look at a test bed larger network, with the ability to artificially insert delays and bandwidth throttles in a physical real-world setting. This would mitigate the risk of training the model to work solely for a specific network, and provide verification of its predictive accuracy. In fact, we are currently looking into options for network virtualization and emulation (modeling the inner workings of the network, as opposed to its behaviors alone as with simulation) as a method of more accurately assessing parameters and model behavior.

Ultimately, it is our belief that we have successfully adapted the GridSim toolkit into an idealized model for simulating the behavior of a laboratory, at-base network environment. In order to best apply this model to a distributed training LVC network environment there is still more work to be done, including adjusting some model parameters and investigating network emulation rather than pure simulation, while verification and validation must be performed to ensure the simulation is accurately modelling a real-world system. In the long term, our goal is to identify, model, and predict the timeframes for human-to-human interaction across a system which includes both human performance factors and hardware factors.

REFERENCES

Anton, P. S., McKernan, M., Munson, K., Kallimani, J. G., Levedahl, A., Blickstein, I., . . . Newberry, S. (2019). Assessing the Use of Data Analytics in Department of Defense Acquisition. Santa Monica, CA, USA: RAND Corporation.

Azzouni, A., & Pujolle, G. (2018). NeuTM: A neural network-based framework for traffic matrix prediction in SDN. *Proceedings of the 2018 IEEE/IFIP Network Operations and Management Symposium*, (pp. 1-5). doi:10.1109/NOMS.2018.8406199

Beres, Y., Mont, M., Griffin, J., & Shiu, S. (2009). Using security metrics coupled with predictive modeling and simulation to assess security processes. *Proceedings of the 2009 Third Int Symp Empirical Software Eng Measure*, (pp. 564-573). doi:10.1109/ESEM.2009.5314213

Binias, B., Myszor, D., Palus, H., & Cyran, K. A. (2020). Prediction of Pilot's Reaction Time Based on EEG Signals. *Frontiers in Neuroinformatics, 14*, 6. doi:10.3389/fninf.2020.00006

Buyya, R., & Assuncao, M. D. (2020). *The Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne*. Retrieved from GridSim: A Grid Simulation Toolkit For Resource Modelling And Application Scheduling For Parallel And Distributed Computing: http://www.cloudbus.org/gridsim/

Buyya, R., & Murshed, M. (2002). GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *Concurrency and Computation Practice and Experience, 14*, 13-15. doi:10.1002/cpe.710

Buyya, R., Murshed, M., & Abramson, D. (2002). A Deadline and Budget Constrained Cost-Time Optimisation Algorithm for Scheduling Task Farming Applications on Global Grids. *Proceedings of the 2002 International Conference on Parallel and Distributed Processing Techniques and Applications* (p. 2183). CSREA Press.

Caminero, A., Sulistio, A., Caminero, B., Carrión, M., & Buyya, R. (2008). Extending GridSim with an architecture for failure detection. *Proceedings of the 2008 International Conference on Parallel and Distributed Systems, 2*. doi:10.1109/ICPADS.2007.4447756

Caminero, A., Sulistio, A., Caminero, B., Carrión, M., & Buyya, R. (2008). Simulation of Buffer Management Policies in Networks for Grids. *Proceedings of the 2008 41st Annual Simulation Symposium*, (pp. 51-60). doi:10.1109/ANSS-41.2008.6

Consiglio, M. C., Wilson, S. R., Sturdy, J., Murdoch, J. L., & Wing, D. J. (2010). Human in the loop simulation measures of pilot response delay in a self-separation concept of operations. *Proceedings of the 2010 27th Congress of the International Council of the Aeronautical Sciences (ICAS).* Nice, France.

Department of Defense. (2019). DoD Digital Modernization Strategy . *DoD Information Resource Management Strategic Plan FY19-23*.

Harper, J. (2015). Live, Virtual, Constructive Training Poised for Growth. *National Defense*. Retrieved from https://www.nationaldefensemagazine.org/articles/2015/11/30/2015december-live-virtual-constructive-training-poised-for-growth

Holden, J., Francisco, E., Lensch, R., Tommerdahl, A., Bryan Kirsch, L. Z., Dennis, R., & Tommerdahl, M. (2019). Accuracy of different modalities of reaction time testing: Implications for online cognitive assessment tools. doi:https://doi.org/10.1101/726364

I/ITSEC. (2017). Navy Works Toward LVC Future. *Official Daily News Digest*, pp. 1, 6.

Intel. (2020). *Intel® Xeon® Processor E5-2630 v3 Technical Specifications*. Retrieved from https://www.intel.com/content/www/us/en/products/processors/xeon/e5-processors/e5-2630-v3.html

Joshi, M. R., & Hadi, T. H. (2015). A Review of Network Traffic Analysis and Prediction Techniques. *arXiv e-prints*, arXiv:1507.05722.

Kapadia, N., Fortes, J., & Brodley, C. (1999). Predictive Application performance modeling in a computational Grid environment. *Proceedings of the 1999 Eighth International Symposium on High Performance Distributed Computing*, (pp. 47-54). doi:10.1109/HPDC.1999.805281

Kauchak, M. (2019). Closing in on the LVC Holy Grail. *Military Simulation & Training Magazine*(2).

Microway. (2020). *Detailed Specifications of the Intel Xeon E5-2600v3 "Haswell-EP" Processors*. Retrieved from microway.com/knowledge-center-articles/detailed-specifications-intel-xeon-e5-2600v3-haswell-ep-processors/

Murshed, M., & Buyya, R. (2001). Using GridSim Toolkit for Supercharging Grid Computing Education.

Office of Naval Research. (2011). Live, Virtual and Constructive (LVC) Training Fidelity. *ONR BAA Announcement # 11-005*.

Rantanen, E., McCarley, J. S., & Xu, X. (2004). Time Delays in Air Traffic Control Communication Loop: Effect on Controller Performance and Workload. *International Journal of Aviation Psychology, 14*(4), 369-394. doi:10.1207/s15327108ijap1404_3

Rawat, P. S., Yadav, A. K., & Barthwal, V. (2015). Grid resource computing environment simulation using GridSim toolkit. *Proceedings of the 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, (pp. 1069-1073). New Dehli.

Schwartz, M. (2016). *Using Data to Improve Defense Acquisitions: Background, Analysis, and Questions for Congress.* Congressional Research Service.

Sulistio, A., & Buyya, R. (2004). A grid simulation infrastructure supporting advance reservation. *Proceedings of the 2004 16th International Conference on Parallel and Distributed Computing and Systems*, (pp. 1-7).

Sulistio, A., Poduval, G., Buyya, R., & Tham, C. K. (2007). On incorporating differentiated levels of network service into GridSim. *Future Generation Computer Systems, 23*(4), 606-615. doi:10.1016/j.future.2006.10.006

Vu, K.-P., Chiappe, D., Morales, G., Strybel, T., Battiste, V., Shively, J., & Buker, T. (2014). Impact of UAS Pilot Communication and Execution Latencies on Air Traffic Controllers' Acceptance of UAS Operations. *Air Traffic Control Quarterly*. doi:10.2514/atcq.22.1.49

Witt, C., Bux, M., Gusew, W., & Leser, U. (2019). Predictive performance modeling for distributed batch processing using black box monitoring and machine learning. *Information Systems, 82*, 33-52. doi:10.1016/j.is.2019.01.006

Zingale, C. M., McAnulty, D. M., & Kerns, K. (2003). *The Effect of Voice Communications Latency in High Density, Communications-Intensive Airspace Phase II: Flight Deck Perspective and Comparison of Analog and Digital Systems.* Federal Aviation Administration.

DISCLAIMER