Proceedings of the 2013 Winter Simulation Conference
R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds.

IMPACTS OF APPLICATION LOOKAHEAD ON DISTRIBUTED NETWORK EMULATION

Yuhao Zheng
Dong Jin
David M. Nicol

University of Illinois at Urbana-Champaign
Information Trust Institute
1308 West Main Street
Urbana, IL , 61801, USA

ABSTRACT

Large-scale and high-fidelity testbeds play critical roles in analyzing large-scale networks such as data centers, cellular networks, and smart grid control networks. Our prior work combines parallel simulation and virtual-time-integrated emulation, such that it offers both functional and temporal fidelity to the critical software execution in large scale network settings. To achieve better scalability, we have developed a distributed emulation system. However, as the number of computing servers grows, so does too the synchronization overhead. Application lookahead, the ability to predict future behaviors of software, may help reducing overhead for performance gain. In this paper, we study the impacts of application lookahead on our distributed emulation testbed. We find that application lookahead can greatly reduce synchronization overhead and improve speed by up to 3 times in our system, but incorrect lookahead may affect application fidelity to different degrees, depending on application sensitivity to timing.

1 INTRODUCTION

It is essential to evaluate applications and protocols for large-scale systems across development phases (e.g., design, implementation, testing, and verification) using testing systems, especially those with the capability to conduct large-scale network experiments. Simulation testbeds are widely used because of scalability and flexibility. However, the fidelity of simulation models is always in question because of model simplification and abstraction. Integration of emulation complements a simulation testbed by offering high fidelity, since real programs are executed on real operating systems instead of executing models to advance experiments. Therefore, we developed such a network testbed in our prior work by marrying a parallel simulator with a virtualization-based emulator (Jin, Zheng, Zhu, Nicol, and Winterrowd 2012). The testbed runs on a single shared-memory server, and is capable of emulating hundreds of virtual environments (VEs) concurrently in a single multi-core processor because of the lightweight kernel-level virtualization technologies.

To enable experiments with larger number of emulated nodes, we have extended our testbed to support distributed emulation. Good scalability implies not only the ability to emulate more nodes, but also emulate them efficiently. Since emulated nodes now reside on different physical machines, the synchronization overhead can dramatically increase comparing with the shared-memory system. Hence, we investigate techniques to extract application lookahead from traces of application-level behaviors to reduce the synchronization overhead, and thus achieve better scalability. The techniques we develop are based on trace history, and are not guaranteed to always yield a true lower bound on future times of process interactions. Lookahead is critical to the performance of conservative parallel discrete-event simulation, and our results indicate that it is also important to emulation, e.g. emulation with application lookahead is up to 3 times faster in some of our experiment setups.

The application lookahead we investigate in this work is defined as a lower bound of the time that an emulated nodes will next affect any simulated or emulated entities. The application lookahead is calculated by a neural-network-based model based on observed historical data. It is difficult for our predication model to produce lookahead predictions that are *always* lower bounds. In particular, the lookahead may be too large, which may negatively impact on fidelity, as it may allow an entity to advance too far in virtual time too quickly, and receive a message in its virtual past. The computational overhead of computing the lookahead is not negligible. Therefore, we have performed extensive studies on the impact of application lookahead, i.e. speed and fidelity, with various network scenarios. We find that application lookahead can greatly reduce synchronization frequency and overhead in some setups, but it may affect application fidelity to different degree depending on application categories. The results serve as guidelines for users of our testbed: in which conditions one ought to consider application lookahead to speed up their experiments, and in which conditions one should not.

Our contributions of this work are summarized as follows:

- We extend our network testbed with the capability to conduct distributed emulation experiments.
- We develop a neural-network-based model for predicting (approximate) application lookahead.
- We investigate the impact on speed and fidelity caused by the application lookahead, and provide a guideline of whether application lookahead may be enabled.

The remainder of the paper is organized as follows. Section 2 overviews the design of the distributed emulation version of our testbed with application lookahead support; Section 3 presents the implementation of the application lookahead model. Section 4 analyzes the impact of application lookahead; Section 5 reviews related work; and Section 6 concludes the paper and provides some future directions.

2 DISTRIBUTED DESIGN

2.1 SYSTEM ARCHITECTURE

Our testbed consists of three major components, an OpenVZ-based network emulator embedded in virtual time, a network simulator, and a parallel simulation engine responsible for coordinating operations of other two components. We have run 300+ emulated hosts on a single physical machine because of the lightweight OS-level virtualization; more emulated hosts are possible on a single machine simply by adding memory. To further increase the scale of the experiments the testbed can conduct by increasing the number of processors used, we have developed the distributed emulation capability in our testbed.

Figure 1 depicts the system architecture to support distributed emulation. A master server machine is connected to multiple slave machines via TCP/IP over gigabit Ethernet links. Each slave manages a group of local containers running on the same physical machine, and independently advances program states during its emulation window. The master has the knowledge on global network states. It is responsible to coordinate all slave machines through a global synchronization algorithm, to manage cross-slave events, and to perform the simulation experiments. Three types of information are communicated between the slaves and the master: control commands, network data packets, and application lookahead.

2.2 GLOBAL SYNCHRONIZATION ALGORITHM

Our systems consists of two sub-systems: emulation and simulation, and we designed a global synchronization algorithm to integrate the two systems based on virtual time (Jin, Zheng, Zhu, Nicol, and Winterrowd 2012). In this work, we revisited the synchronization algorithm to incorporate the application lookahead. Figure 2 describes how a network experiment advances in our testbed with application lookahead disabled and enabled. Emulation and simulation always execute their cycles alternatively. Without application lookahead, emulation always runs ahead of simulation, while with application lookahead, emulation and simulation appears like a racing game, one may run ahead of the other at any cycle. Application lookahead

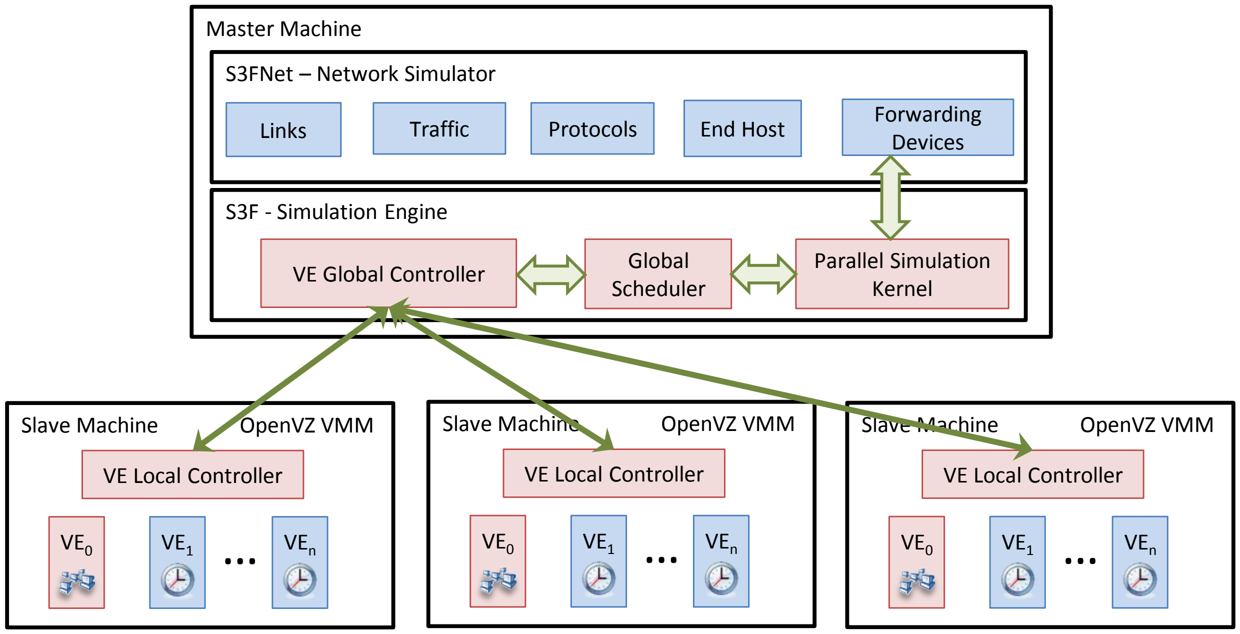

Figure 1: Network Testbed Architecture with Distributed Emulation Support.

is explored to improve scalability, especially for the distributed version. Three factors determine a high quality application lookahead:

- large average length of application lookahead, which implies small synchronization overhead,
- small overhead for computing the lookahead, and
- high accuracy of the predicted lookahead.

The three factors are often in tension with each other. We investigate the performance gain and the fidelity loss with various network scenarios, and results are presented in Section 4. The results serve as guidelines on the types of application that are beneficial for users to turn on the application lookahead functionality in our testbed.

The high-level global synchronization algorithm for the distributed testbed is described in Algorithm 1. Let us define t_{emu} be the current emulation time (OpenVZ virtual time), t_{sim} be the current simulation time, ESW be the emulation synchronization window (the length of the next emulation advancement), and SSW be the simulation synchronization window (the length of the next simulation advancement). t_{sim_target} is the time where the simulator hands the control over to the network emulator.

3 LOOKAHEAD IMPLEMENTATION

We next present implementation details about how application lookahead is predicted, as well as analysis of lookahead error. In this paper we mainly focus on the impacts of application lookahead, and we leave how to predict lookahead using various approaches as future work.

3.1 LOOKAHEAD USING TIME SERIES FORECASTING

As discussed in Section 2, application lookahead is the ability to predict future behavior of applications. When it comes to network emulation, we are particularly interested in knowing the time of next packet sent from an application. Since application behaviors are fully determined by their executables and runtime

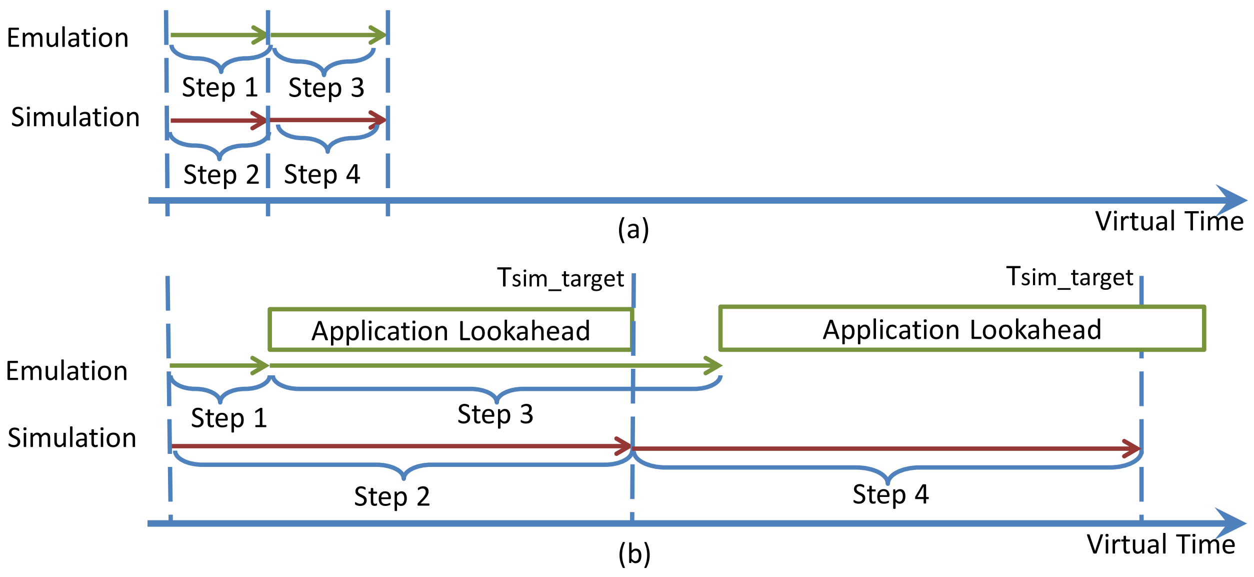

Figure 2: Experiments advancement (a) without application lookahead, (b) with application lookahead.

Algorithm 1 Global Scheduler

```
while true do
  if $t_{sim} = t_{sim\_target}$ then
    compute $ESW$
    run OpenVZ emulation for $ESW$
    inject packets to simulation
    $t_{sim\_target} \leftarrow t_{emu} + \min_{VE_e}\{lookahead_e\}$
  else
    compute $SSW$
    run simulation for $SSW$
    $t' \leftarrow$ the timestamp of the first executed deliver-to-VE event in this $SSW$
    $t_{sim\_target} \leftarrow \min\{t_{sim\_target}, \max\{t', t_{emu}\}\}$
  end if
end while
```

environments, one possible way to predict application lookahead is based on binary code. However, this approach is challenging not only due to the complexity of executable code, but also because runtime environment brings uncertainty, e.g. multi-thread scheduling controlled by operating system. While network emulation running native application code improves functional fidelity, it also makes lookahead impossible to predict exactly. *Any* implementable approach to computing lookahead will be affected by inescapable uncertainties.

Rather than estimate lookahead by analysis of the application code itself, we present an approach based on applications' packet inputs and outputs. Packets sent and received by an application can be viewed as a time series, and therefore the lookahead problem can be converted to a time series forecasting problem. To form a time series, all the packets sent or received by an application are sorted by timestamps, so that each packet naturally becomes an observation vector y:

$$y = (ts, a_1, a_2, \ldots, a_x),$$

where ts is the timestamp relative to the previous packet, and a_i is the i-th attribute of this packet. Packet attributes may contain packet size or other application specific fields. Packet attributes are to help indicate packet content and are crucial to forecasting accuracy. We found that attributes containing packet size (positive value for sent packets and negative value for received packets), TCP SYN flag, and TCP FIN flag perform well for the applications we have tested in this paper. For more complex applications, using more attributes may improve accuracy but at the cost of slower lookahead computation speed.

We use artificial neural networks (ANNs) to solve the time series forecasting problem (Zhang, Patuwo, and Hu 1998). Time series forecasting typically assumes there is an underlying relationship between future values and past observations. Formally, a future value is a function (either known or unknown) of past observations:

$$y_{t+1} = f(y_t, y_{t-1}, \ldots, y_{t-w+1}),$$

where y_t is the t-th observation and w is the forecasting window size. As ANNs are universal function approximators (Lippmann 1987) (Hertz, Krogh, and Palmer 1991), it can be trained to perform the f function mapping using historical data. Since we are only interested in the duration within which an application will not send a packet, the ANN is only trained to predict the timestamp field ts of observation vector y_{t+1}, ignoring all other attribute fields a_i. In case that the next observation in historical data y_{t+1} is a received packet, using timestamp ts of this packet as training target does not violate the semantic of application lookahead — a period within which an application will not send a packet. In fact, ts is the maximum lookahead amount given this training data, as the y_{t+1} packet receipt may suppress other packet sends.

ANN models for forecasting are application dependent, as different applications behave differently. To train an ANN for a given application, we run the application under its typical setup and collect the sent and received packet trace. For better accuracy and generalization, the trace had better be comprehensive to cover most the functionalities of the application. Consequently, when collecting trace for training, we run the application under different setup, e.g. with different parameters, and under different network conditions.

3.2 LOOKAHEAD ERROR HANDLING AND ANALYSIS

As defined in Section 3.1, application lookahead is defined as a prediction of a time in future before which an application will not send a packet. Ideally, application lookahead should be the exact time of the next sent packet given that the application does not receive a packet before that. Unfortunately, due to application complexity and runtime uncertainty, lookahead may not be exact. On one hand, an actual packet send may occur later than lookahead lower bound. According to our scheduling algorithm presented in Section 2, this underestimated lookahead may result in an unnecessary synchronization, i.e. a synchronization that does nothing. However, this only has impact on speed but not fidelity, as packets will be delivered to destination at the same time as that without lookahead.

On the other hand, an application *may* send a packet before the predicted lookahead. In this case, it may result in some emulation host advancing too far ahead without noticing the presence of a potential received packet. Since our virtual time system used in emulation system is not exact and may introduce error (Zheng and Nicol 2011), it naturally allows packet arrival with a timestamp in the past and will try delivering late packets to the destination containers at the earliest possible time. However, an event with a timestamp in the past is strictly prohibited in conservative parallel discrete-event simulation. To fix this, when a packet sending event is presented to the simulation system, if this event has a timestamp smaller than the current simulation time, its timestamp is set to the current simulation time.

A lookahead error may increase the one-way delay of a packet sending from one emulation host to another. Our previous work shows that temporal error up to a timeslice (100 μs) does not introduce additional error to certain classes of application behaviors (Zheng, Jin, and Nicol 2012). However, incorrect lookahead may introduce additional temporal error beyond that, and this may impact fidelity. Section 4.2 present experimental results demonstrating the impact of incorrectly lookahead to application fidelity, but we analyze such impacts to different application categories as follows.

Different applications have different sensitivity to the increased one-way packet delays caused by incorrect lookahead. We classify applications into the following three categories according to how much impact incorrect lookahead may introduce.

- **Local impact**: some applications are relatively insensitive to lookahead error. An increased one-way delay of a single packet brings minimal impact to the application functionality. Such error does not cumulate and thus will not affect later functionality. Typical instances of this category are those applications with long-term one-way traffic, e.g. transferring large files using FTP protocol. As long as the lookahead error is not large enough to trigger a functionality change (e.g. TCP retransmission due to timeout), when an increased delay occurs on a single packet, later functionality is performed as if the lookahead error never occurs. In this case, lookahead error creates a *jitter* in application behavior.
- **Global impact**: some other applications are sensitive to lookahead error. An increased delay on a single packet may cumulate and postpone all later functionality. Many communicating protocols behave in this way: when an application receives a packet, it performs some action and send a reply packet, and the application on the other end will not advance until it receives this reply. In fact, FTP protocol may belong to this category if the file size is very small and most time is spent in control messages. However, as long as the increased delay is not long enough to trigger a timeout, application still perform the same subsequent actions, only at a later time. In this case, lookahead error creates a *skew* in application behavior.
- **Drastic impact**: finally, some applications are extremely sensitive to network delays. Any small error in delays may lead to either different application behavior or incorrect application output. Typical examples are those applications mainly based on network delay, e.g. ICMP ping message that measures round trip time. In Section 4.2, we provide an example by implementing an end-to-end available bandwidth measurement approach proposed by (Jain and Dovrolis 2002). This approach mainly relies on measuring one-way delay at different sending rate, and therefore could be extremely sensitive to any lookahead error.

We provide the above framework to estimate applications' sensitivity to incorrect lookahead. Note that this classification is vague, i.e. some applications may locate around the boundary between two categories and have the characteristics of both. Detailed experiment results are presented in Section 4.2.

4 EVALUATION

We evaluate the impacts of application lookahead in a distributed setup. The overall architecture is shown in Figure 1. The master machine is a server with 32 logical processors and 64GB memory. The master machine is connected with 4 slave machines via a gigabit switch. Each slave machine is a commodity laptop with 2 processors and 2GB memory. This setup demonstrates our distributed design can use relatively low-profile machines to achieve scalable distributed emulation.

Next we present some preliminary results showing the impacts of application lookahead to speed and fidelity respectively. For experiment with application lookahead, as stated in Section 3.1, ANNs models are application dependent. In our experiments, we pre-trained ANNs for different applications, and the training time is not counted into the run time. Training an ANN is usually time-consuming, but a well-trained ANN is reusable until there is a functionality change in the application. In addition, we let the forecasting window size w defined in Section 3.1 be 10.

4.1 LOOKAHEAD IMPACTS ON SPEED

We create a network scenario to which application lookahead could be potentially beneficial. The network has high bandwidth (10 Mbps) and low latency (10 μs), which makes simulation-emulation synchronization windows small when lookahead is not presented. On the other hand, the applications are sending traffic very infrequently — we use iperf (Iperf 2012) sending UDP traffic at 100 Kbps — which means there are lots of unnecessary synchronizations and this can be reduce by application lookahead. We vary the number of slave machines as well as the number of virtual environments (VEs) per slave. In each setup, the VEs are paired: each VE is connected and only connected to another VE via a 10Mbps / 10 μs link. Different link pairs are independent and are just to increase simulation/emulation load. As all the packets have to pass through the simulator within the master machine, it does not matter whether two VEs of a link pair are on the same slave or not. Baseline results without lookahead are shown in Table 1(a), and the ones with lookahead is given in Table 1(b).

We observed small synchronization windows (100 μs) when lookahead is not presented, resulting in a high synchronization overhead (more than 50% when VE density is low, e.g. 20 VEs/slave). As VE density increases, the synchronization overhead decreases due to improved synchronization efficiency per slave machines. But the overhead still increases when the number of slaves grows, and this is negative to scalability. On the other hand, when application lookahead is given, simulation-emulation synchronization windows are greatly enlarged, resulting in lower overhead and higher execution speed (2-3 times faster). As we can see, application lookahead brings performance gain.

We also notice the synchronization window decreases when the total number of VEs increases. This is due to the synchronization mechanism we use (Section 2). Every time emulation is about to run, the global scheduler computes an ESW for all VEs to advance. This approach is a barrier-based synchronous approach, whose performance is sensitive is to minimal latency and minimal lookahead (Nicol and Liu 2002). As seen from Table 1(b), the average lookahead is independent to the number of VEs, since all the applications are performing the same functionality. However, the average minimum lookahead decreases very quickly as the size grows, because different link pairs are independent and are running out of sync. We conclude that the reduced window size is the nature of barrier-based synchronization. Changing to asynchronous approach may improve performance for this scenario as the node degree is small, but this remains future effort.

4.2 LOOKAHEAD IMPACTS ON FIDELITY

To investigate the impact of application lookahead to fidelity, we use one slave machine with 2 VEs, maximizing the impact of incorrect lookahead. As explained in Section 4.1, increased number of VEs will result in smaller synchronization window, which reduces the likelihood of a VEs advancing too far ahead due to incorrect lookahead and causing an increased one-way packet delay. Detailed analysis is provided in Section 3.2. The experiment setup we use is this subsection is 2 VEs connected by a link with 10 Mbps bandwidth and 10 μs latency. Next we present the impacts of lookahead to applications fidelity, by discussing the three categories in Section 3.2 respectively.

To model the application category which only suffers local impact from incorrect lookahead, we use a traffic generator sending controllable one-way UDP traffic. The traffic source is sending 10-Kbyte constant size packets, and the inter-packet arrival time is exponential distribution with mean of 10 ms. We use the same seed for random number generator to ensure repeatability, but we use different seeds when training the ANNs to ensure generalization. The results are shown in Figure 3. We only plot the first 100 packets for conciseness.

Because the application is sending one-way traffic and the sink application does not have any feedback to the source, the source application is performing exactly the same behavior regardless of incorrect lookahead, as shown in Figure 3(a). On the other hand, when an incorrect lookahead occurs, i.e. the source application sends a packet before its predicted lookahead, and because the network delay is small, the sink application may advance too far ahead without noticing the packet. In this case, the packet arrival time will be late due to the error lookahead, as shown in Figure 3(b), near packet #10 and #65. However, such error is not propagating due to the one-way characteristic, and the subsequent packets may arrive at the right time. Application level statistics show lookahead does not affect overall throughput (0.976 Mbps without lookahead, and 0.975 Mbps with lookahead), but it does increase jitter (102 μs vs. 466 μs).

Table 1: Impacts of Lookahead to Simulation/Emulation Running Speed.

Abbreviations:
Diff — speed difference between without/with lookahead
Sync Win — size of emulation synchronization windows (ESW)
Overhead — percentage time spent in synchronization
Avg LA — average lookahead amount over all VEs
Avg Min LA — average minimal lookahead amount
vtime — simulation time / virtual time

(a) No Lookahead.

Setup	Speed (vtime/sec)	Diff (%)	Sync Win (vtime μs)	Overhead (%)	Avg LA (vtime μs)	Avg Min LA (vtime μs)
20 VEs/slave, 1 slave	0.445	n/a	100	51%	n/a	n/a
20 VEs/slave, 2 slaves	0.398	n/a	100	53%	n/a	n/a
20 VEs/slave, 4 slaves	0.350	n/a	100	56%	n/a	n/a
50 VEs/slave, 1 slave	0.234	n/a	100	30%	n/a	n/a
50 VEs/slave, 2 slaves	0.207	n/a	100	31%	n/a	n/a
50 VEs/slave, 4 slaves	0.193	n/a	100	32%	n/a	n/a
100 VEs/slave, 1 slave	0.138	n/a	100	17%	n/a	n/a
100 VEs/slave, 2 slaves	0.111	n/a	100	19%	n/a	n/a
100 VEs/slave, 4 slaves	0.102	n/a	100	20%	n/a	n/a

(b) With Lookahead.

Setup	Speed (vtime/sec)	Diff (%)	Sync Win (vtime μs)	Overhead (%)	Avg LA (vtime μs)	Avg Min LA (vtime μs)
20 VEs/slave, 1 slave	1.302	+193%	3577	5%	58500	5282
20 VEs/slave, 2 slaves	1.226	+208%	2075	7%	58500	3424
20 VEs/slave, 4 slaves	1.088	+211%	1192	14%	58500	2388
50 VEs/slave, 1 slave	0.534	+128%	1711	4%	58500	3068
50 VEs/slave, 2 slaves	0.483	+133%	982	7%	58500	1920
50 VEs/slave, 4 slaves	0.403	+109%	525	12%	58500	1289
100 VEs/slave, 1 slave	0.272	+97%	983	4%	58500	1892
100 VEs/slave, 2 slaves	0.214	+93%	497	6%	58500	1252
100 VEs/slave, 4 slaves	0.169	+66%	297	11%	58500	744

(a) Source Sent Packet Trace.

(b) Sink Received Packet Trace.

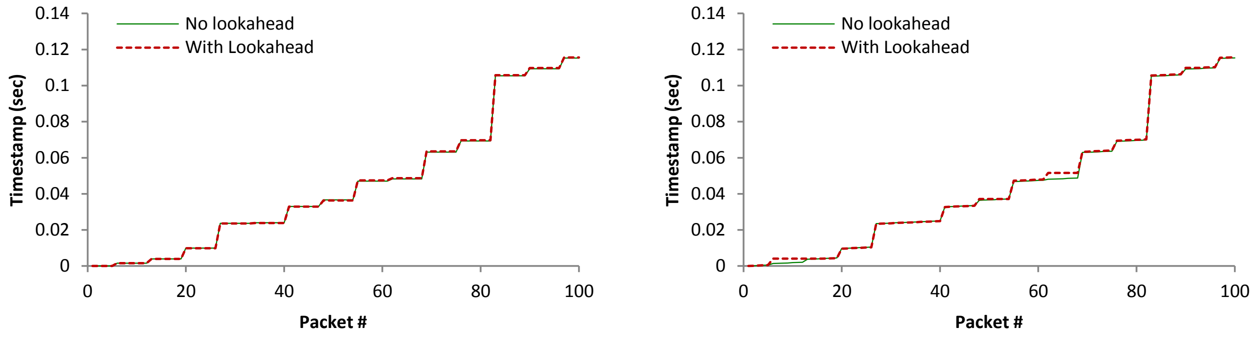

Figure 3: Impacts of Lookahead to Fidelity — Local Impact.

(a) Source Sent Packet Trace.

(b) Sink Received Packet Trace.

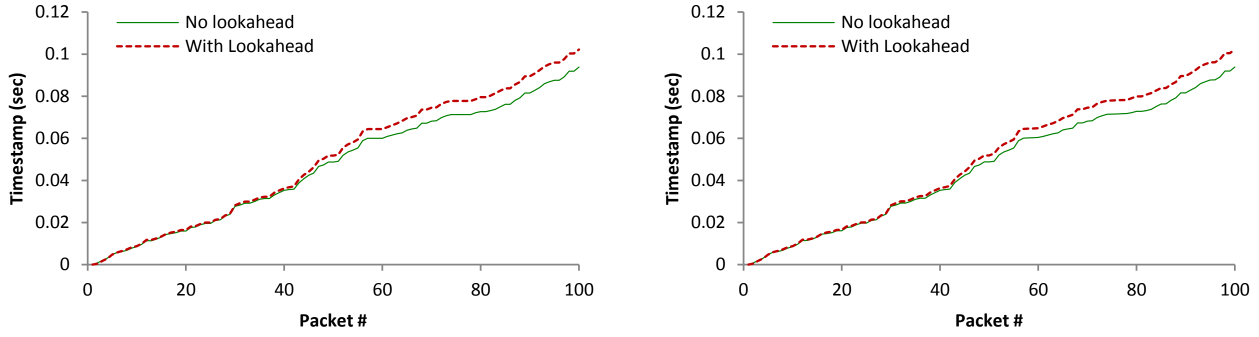

Figure 4: Impacts of Lookahead to Fidelity — Global Impact.

To model the application category which may suffer global impact from incorrect lookahead, we change the traffic generator from open-loop to closed-loop, i.e. the source application must wait for a reply to proceed for each sent packet. We also change the packet size to exponential distribution with mean of 1 Kbyte, and set the packet arrival time to exponential distribution with 1-ms mean. The result are shown in Figure 4.

In Figure 4, we observe a clear skew in the packet trace. This is due to the above-mentioned closed-loop characteristic, in which the error is cumulative and any late packet arrival will defer all subsequent actions. We also observe a higher prediction error, as the source application is sending smaller packets which are usually not fragmented. The ANN is unable to exactly predict the random distribution inter-packet arrival time. The application level statistics show lookahead shows lookahead results in a lower throughput (0.792 Mbps vs. 0.720 Mbps) as well as a larger jitter (87 μs vs. 185 μs).

Finally, to model the application category which is extremely sensitive to lookahead errors, we implement the end-to-end available bandwidth measure approach proposed in (Jain and Dovrolis 2002), and we slightly tune the algorithm to make it more sensitive one-way delay variations. For the setup with or without lookahead, we run the application for 10 times, and the results are shown in Table 2. When lookahead is not presented, the application correctly measures the available bandwidth (10 Mbps), and the results are quite stable given the small standard deviation. However, when lookahead is introduced, the application produces wrong answer and with a very large variation. That is because lookahead affects one-way delay, and the application performs different functionality as its behavior highly depends on such delay.

Table 2: Results of End-to-End Available Bandwidth Measurement.

	Average Output (Mbps)	Std Dev (Mbps)
No Lookahead	9.93	0.14
With Lookahead	3.84	1.48

In summary, we find the degree to which application lookahead may affect fidelity depends on applications' sensitivity of network delay. For those applications that can tolerate slight changed delay, lookahead may bring performance gain while not affecting fidelity too much. For those applications that are extremely sensitive to network delay, lookahead is not suitable regardless the potential faster execution speed.

5 RELATED WORK

Lookahead has been extensively studied in conservative parallel discrete-event simulation, because good lookahead significantly improves the simulation performance by increasing the length of time by which logical processes can independently advance. Lookahead is defined as the ability to predict what will occur and what will not occur in simulated future (Fujimoto 1990). A lookahead is loosely defined as the minimum time that a logical process will not affect event lists of other logical processes. Dimensions of lookahead based on the knowledges extracted from model characteristics and simulated applications are classified in (Nicol 1996). A good lookahead reduces synchronization overhead and hence results in performance gain. Extensive researches have been performed to show the importance of simulation lookahead (Reed, Malony, and McCredie 1988) (Fujimoto 1988) (Fujimoto 1987). Sample applications of exploiting good simulation lookahead include simulations of stochastic queuing networks (Nicol 1988), continuous-time Markov chains (Nicol and Heidelberger 1995), and wireless ad-hoc networks (Liu and Nicol 2002). The difference with our history-based emulation lookahead is that the emulation lookahead is not absolutely correct (an event may occur before the lookahead). The quality of the emulation lookahead is dependent on how accurate our lookahead model can predict application-level behaviors.

Inspired by biological systems, artificial neural networks (ANNs) are mathematical models that emulate biological neural networks (Lippmann 1987) (Hertz, Krogh, and Palmer 1991). An ANN consists of a group of nodes (called neurons, or perceptrons), and these nodes are interconnected and perform functions in parallel. As the number of nodes increases, the capability of the network increases dramatically. For example, multi-layer perceptrons (MLPs) are universal function approximators, and it can approximate any continuous function at any desired accuracy (Hornik, Stinchcombe, and White 1989). ANNs are data-drive, and they can learn and generalize from past experience, in the way that they can correctly infer some unseen part from the training data. Due to these natures, ANNs have been widely used in many domains (Widrow, Rumelhart, and Lehr 1994), including forecasting (Zhang, Patuwo, and Hu 1998). Forecasting models usually assume there is an underlying relationship between the future and the past, and such relationship is usually unknown and complicated. ANNs are powerful tools for learning such underlying relationship, making them very capable of forecasting. Our work also uses ANNs for forecasting, but we have demanding accuracy requirements as network delays are in the order of milliseconds or even microseconds. The ANN-based forecasting only takes previously packet trace as input, and is occasionally inadequate to capture those uncertainties created by application internal states.

6 CONCLUSION

We extend our network simulation/emulation testbed to support distributed emulation. We find synchronization overhead increases as the system size grows, and application lookahead may help reduce such overhead and achieve better scalability. We proposed an approach of application lookahead based on time series forecasting using artificial neural network. Through experiments, we find application lookahead can enlarge simulation-emulation synchronization windows and improve speed up to 3 times. However, the downside is that it may affect fidelity as lookahead could be wrong, and the degree to which lookahead affect fidelity highly depends on application behaviors. We conclude that lookahead is suitable for those applications that can tolerate small changes in network delay, in which case lookahead can improve speed without affecting fidelity too much.

In this paper we mainly focus on studying the impacts of application lookahead, while investigation about how to provide better lookahead using other approaches (e.g. application code analysis) is left as future work. Another possible future effort is to implement asynchronous synchronization between simulation and emulation, as it may overcome the reduced synchronization window size caused by scale increases.

ACKNOWLEDGMENTS

This material is based upon work supported by the Department of Energy under Award Number DE-OE0000097, and work supported by the Boeing Corporation. (Disclaimer: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.)

REFERENCES

Fujimoto, R. M. 1987. “Performance Measurements of Distributed Simulation Strategies.”. Technical report, DTIC Document.

Fujimoto, R. M. 1988. “Lookahead in parallel discrete event simulation”. Technical report, DTIC Document.

Fujimoto, R. M. 1990. “Parallel discrete event simulation”. *Communications of the ACM* 33 (10): 30–53.

Hertz, J. A., A. S. Krogh, and R. G. Palmer. 1991. *Introduction to the theory of neural computation*, Volume 1. Westview press.

Hornik, K., M. Stinchcombe, and H. White. 1989. “Multilayer feedforward networks are universal approximators”. *Neural networks* 2 (5): 359–366.

Iperf 2012. http://iperf.sourceforge.net/.

Jain, M., and C. Dovrolis. 2002. “End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput”. In *ACM SIGCOMM Computer Communication Review*, Volume 32, 295–308. ACM.

Jin, D., Y. Zheng, H. Zhu, D. Nicol, and L. Winterrowd. 2012, July. “Virtual Time Integration of Emulation and Parallel Simulation”. In *Proceedings of the 2012 Workshop on Principles of Advanced and Distributed Simulation (PADS)*, 120–130. Zhangjiajie, China.

Lippmann, R. 1987. “An introduction to computing with neural nets”. *ASSP Magazine, IEEE* 4 (2): 4–22.

Liu, J., and D. M. Nicol. 2002. “Lookahead revisited in wireless network simulations”. In *Proceedings of the sixteenth workshop on Parallel and distributed simulation*, 79–88. IEEE Computer Society.

Nicol, D. M. 1988. *Parallel discrete-event simulation of FCFS stochastic queueing networks*, Volume 23. ACM.

Nicol, D. M. 1996. “Principles of conservative parallel simulation”. In *Proceedings of the 28th Winter Simulation Conference*, 128–135. IEEE Computer Society.

Nicol, D. M., and P. Heidelberger. 1995. "A comparative study of parallel algorithms for simulating continuous time Markov chains". *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 5 (4): 326–354.

Nicol, D. M., and J. Liu. 2002. "Composite synchronization in parallel discrete-event simulation". *Parallel and Distributed Systems, IEEE Transactions on* 13 (5): 433–446.

Reed, D. A., A. D. Malony, and B. D. McCredie. 1988. "Parallel discrete event simulation using shared memory". *IEEE Transactions on Software Engineering* 14 (4): 541–553.

Widrow, B., D. E. Rumelhart, and M. A. Lehr. 1994. "Neural networks: Applications in industry, business and science". *Communications of the ACM* 37 (3): 93–105.

Zhang, G., B. E. Patuwo, and M. Y. Hu. 1998. "Forecasting with artificial neural networks: The state of the art". *International journal of forecasting* 14 (1): 35–62.

Zheng, Y., D. Jin, and D. M. Nicol. 2012. "Validation of application behavior on a virtual time integrated network emulation testbed". In *Proceedings of the Winter Simulation Conference*, 246. Winter Simulation Conference.

Zheng, Y., and D. Nicol. 2011. "A Virtual Time System for OpenVZ-Based Network Emulations". In *Proceedings of the 2011 Workshop on Principles of Advanced and Distributed Simulation (PADS)*.

AUTHOR BIOGRAPHIES

YUHAO ZHENG holds a Ph.D. degree in computer science from University of Illinois at Urbana-Champaign (2013). He will be a software engineer in Google. He also holds a B.S. in computer science and technology (ACM horned class) from Shanghai Jiao Tong University, China (2007). His research interests lie in the areas of wireless network, computer security, large-scale computer and communication system modeling and simulation. His email address is zheng7@illinois.edu.

DONG (KEVIN) JIN holds a Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign. He will be an assistant professor of Computer Science at Illinois Institute of Technology. He also holds a B.Eng. with first class honors in computer engineering from Nanyang Technological University, Singapore (2005), and a M.S. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign (2010). His research interests lie in the areas of cyber-security, networking, modeling and simulation of large-scale systems and networks. His email address is dongjin2@illinois.edu.

DAVID M. NICOL is Professor of Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign, and is Director of the Information Trust Institute. He holds a B.A. in mathematics from Carleton College (1979), and M.S. and Ph.D. degrees in computer science from the University of Virginia (1983, 1985). Prior to joining UIUC, he taught at the College of William & Mary, and Dartmouth College. He has served in many roles in the simulation community (e.g. Editor-in-Chief of ACM TOMACS, General Chair of the Winter Simulation Conference Executive Board of the WSC), was elected Fellow of the IEEE and Fellow of the ACM for his work in discrete-event simulation, and was the inaugural recipient of the ACM SIGSIM Distinguished Contributions award. His current research interests include application of simulation methodologies to the study of security in computer and communication systems. His email address is dmnicol@illinois.edu.