Proceedings of the 2016 Winter Simulation Conference
T. M. K. Roeder, P. I. Frazier, R. Szechtman, E. Zhou, T. Huschka, and S. E. Chick, eds.

EMULATION/SIMULATION OF PLC NETWORKS WITH THE S3F NETWORK SIMULATOR

Vignesh Babu
David M. Nicol

Department of Electrical and Computer Engineering
University of Illinois, Urbana Champaign
1308 W Main St, Urbana IL-61820, USA

ABSTRACT

Programmable Logic Controllers (PLCs) are devices frequently used in industrial control systems with tight real time constraints on operations. Using emulation and/or simulation to evaluate the behavior of a network of PLCs is difficult because of the lack of tools that accurately mimic the real-time behavior of such networks. This paper addresses this issue by showing how to tightly integrate instances of a PLC emulator Awlsim with the network simulator S3F, in such a way that emulations and simulation are advancing synchronously in virtual time. We demonstrate fidelity of the approach in capturing the operating behaviour of a PLC network under varied network conditions and stress levels.

1 INTRODUCTION

Programmable Logic Controllers (PLC) are an integral part of modern manufacturing plants and critical infrastructures like power stations, marine docking operations, and water treatment units. Their use helps to cut costs, minimize human errors and improve system capability. Networks of PLCs monitor data from different subsystems and send control signals to actuators that impact the behavior of a physical system.

It is difficult to test a network of PLCs without actual deployment. The program of an individual PLCs can be tested via emulation, but evaluation of a *network* of PLCs requires a communication infrastructure, and importantly, a means of ensuring that the temporal aspect of messaging behavior in the evaluation testbed is a good model of what will happen in the field. Effective testing therefore calls for a flexible, scalable and low-cost testbed which can produce realistic behavior. This testbed would facilitate research in cyber-security of systems using PLCs, where the difficulty associated with gathering generic data has been the main obstacle to development of efficient defensive measures (Vaughn Jr and Morris 2016). With the emergence of virtualization technology, parallel simulation/emulation is an economical solution.

Simulators offer control over experiment conditions, which is necessary to perform reproducible evaluations. In Jain, Lechevalier, Woo, and Shin 2015, the authors simulate a small virtual factory with four turning machines by writing simulation models for each subsystem in the manufacturing cell. However, writing simulation models of individual components in the system is often difficult and error prone and needs extensive validation.

With the advancement of virtualization technologies, this problem can be addressed by emulating components in the system. Emulations have the advantage over simulations of mimicking the exact functionality of their target components. Furthermore, emulations have been integrated with simulations, e.g., by the widely used ns-3 platform (The Ns-3 Consortium 2016). Typically these integrations are "best effort", in the sense that no explicit synchronization is done between the emulation and simulation. In the case of ns-3 the advance of the *virtual time* clock is tied to the *physical wall-clock*, to keep it from racing ahead in time past an emulator. Furthermore, within a typical virtual machine manager (e.g. Xen (Linux Foundation 2016) or Linux containers (LXC 2016)) the executions of the emulation instances they manage are not explicitly synchronized. In the case of LXC containers the synchronization is left entirely to the Linux kernel scheduler, as each container is a Unix process, and runs when the general scheduler decides it will, and for a length of CPU time that is chosen by the scheduler. To obtain tighter control over emulation execution we have developed TimeKeeper (Lamps, Nicol, and Caesar 2014), a small set of modifications and extensions to the Linux kernel that endows selected Linux processes with a notion of virtual time, and takes the responsibility of scheduling those processes so that they advance in through virtual time.

One way of thinking of TimeKeeper running with S3F is that execution bursts of an emulated entity can be thought of as being the execution of events, with an attendant advancement of virtual time, under the control of the S3F scheduler. A very useful ramification of this is that the approach enables one to evaluate large systems with modest amounts of computational resources, by accepting that the advancement of virtual time relative to the wall-clock is slower. We therefore address a scaling issue suffered by prior integrations of simulation and emulation.

This paper describes the integration of a PLC emulator AwlSim within the TimeKeeper+S3F system. S3F simulator (S3F Net 2016) is a parallel discrete event network simulator built on top of the Scalable Simulation Framework (SSF) API. SSF was redesigned in Nicol, Jin, and Zheng 2011 to improve performance of discrete event simulations by exploiting potential parallelism. S3F relies on standard C++ libraries which attest to its efficiency and simplicity. We chose S3F because of its capability to support creation of complex communication network models using devices like switches and routers operating on top of layered protocols like TCP/IP. In addition, S3F also supports emulation using Linux Containers and OpenVZ containers which can be conveniently used to run emulations of PLCs. Further, conventional PLC networks support serial (RS-232) capability and hence we augmented S3F with the notion of serial non-IP-speaking communication lines, and with a model of the ModBus communication protocol running on those serial lines. We demonstrate empirically that under TimeKeeper+S3F management, a network of PLC emulations produce expected behaviour, but when run under "best effort" control they do not.

This paper is organized as follows: Section 2 gives a brief introduction to TimeKeeper. Section 3 motivates the problem further while Section 4 briefly describes the architecture of the testbed. Section 5 presents our experimental results and Sections 6 and 7 discuss related work and conclusions respectively.

2 TIMEKEEPER

In a typical virtual machine manager (e.g., Xen (Linux Foundation 2016)) all virtual machines (VM) share the available CPU resources and when referencing the system clock (directly or indirectly) reference the clock of the manager. Therefore each process perceives the same wall-clock time (system clock time or real time) regardless of the number of processes in the system. An increase in the number of virtual machines reduces the available CPU time per VM per unit wall-clock time. When the behavior of the VMs depend in any way on measured time (e.g. real-time processes), this attribute of typical virtual machine managers can influence the behavior of the emulated machines. For example, insufficient allocation of CPU time to a VM could lead it to miss a deadline, and impact the emulated application's functionality.

Linux supports a lightweight style of virtual machines through a construct called a LXC *container*. Each container shares the underlying Linux resources such as cache, virtual memory, IO devices, and scheduling. A container is in fact just another Linux process, with a container API defined for ease of application development.

TimeKeeper is a small modification to the Linux kernel which addresses the shared wall-clock time problem among emulated applications. Each virtual machine (henceforth here, a container) is given its own virtual clock, advanced independently of other containers' virtual clocks. Clock advancement is based on measured execution time of the container when allocated the CPU, scaled by the *time dilation factor* (TDF). The TDF is the means by which we estimate the length of execution bursts *on the target architecture*. The time it takes a program to execute on the host Linux machine depends on the machine's architecture, as well as any supporting infrastructure. For example, the Awlsim emulator is actually a Python-based interpretation of Step-7 program instructions, which means the overhead of Python execution has to be factored into consideration. Furthermore, Awlsim can spawn additional processes in addition to the Python process, all of which are competing for a container's execution allocation. If we estimate that the advancement of an application is 5 times slower on the emulation platform than on the modeled platform we'd define the container's TDF to be 5. Formally, the TDF is the ratio of the rate of advance of real time to virtual time, so that the translation from measured execution time on the Linux platform to estimated execution time on the modeled platform is obtained by dividing measured time by the TDF.

Emulated applications can use system calls such as `gettimeofday`, `sleep`, `select` and `poll` to request the OS for time bound actions. TimeKeeper modifies the values returned by these system calls as a function of the TDF. TimeKeeper also offers finer control over the scheduling order and run time of processes under its control than is given when left to the Linux scheduler. TimeKeeper provides an API which allows a control application to

- dynamically change the TDF assigned to a process,
- pause/resume processes with high accuracy,
- run processes for a specified virtual time duration,
- schedule processes so that they advance together in virtual time.

Using this interface, a controlling application (such as a network simulation) can instruct emulation entities to run for a specified duration of virtual time, and be notified after a specified action is completed.

We previously used this functionality to integrate emulated Linux containers with popular network simulators ns3, CORE, EMANE and S3F (Lamps, Adam, Nicol, and Caesar 2015). With TimeKeeper simulation and emulation are tightly coupled because the network simulators use TimeKeeper's API to control advancement of emulation entities. We cannot stress the importance of this contribution enough, as existing hybrid emulation-simulation solutions are all best effort systems with simulators trying to process events in real time. Best effort emulations require the simulation to "keep up" with the emulation in real-time, and with increasing model size and complexity ultimately cannot. With finer coupling between simulation and emulation we were able to demonstrate significant scalability and fidelity benefits.

3 MOTIVATION

A programmable logic controller (PLC) is a computing device designed for use in industrial control systems. A PLC's role typically is to monitor inputs from sensors, and issue commands to actuators in a physical system as a result of the inputs it has read, and inputs other PLCs have read and communicated to it. Networks of PLCs are in the class of "hard real time systems", meaning that they are designed to respond to changes in inputs within a set period of time. Failure to do so, for any reason, can mean catastrophic consequences for the controlled system.

To support hard real-time operations, PLC applications run on top of custom operating systems. The applications themselves (and the languages used to describe those applications) are designed around this 'read-then-respond' model. A PLC program is described as organization of cycles, where in each cycle one or more inputs are read, a calculation is made, then one or more output messages to actuators or to other PLCs are issued. Determinism in execution time and predictability of program behavior are key in designing PLC networks with predicable execution timing. However, because a PLC's execution behavior can depend in part on communication with other PLCs, the timing of the network supporting inter-PLC communications is also critical to PLC functional behavior and meeting of real-time constraints.

Our penultimate goal is to study cyber-security issues in networks of PLCs. The work reported here is necessary to support that goal, in that our studies will initially involve *simulation* of cyber-attacks and simulation of the impacts that the detection and defensive measures have on the operations of the PLC networks. Therefore we need a means by which actual PLC programs can be responding to simulated inputs, can communicate with each other over a simulated network, and can send commands to simulated actuators. This integration of emulation and simulation must be as careful as possible to capture the temporal behavior of PLC networks as they operate in the field. As we have previously developed the integrated TimeKeeper/S3F combination to provide fine-grained temporal control over emulations in the context of network simulation, we identified a suitable PLC emulator to integrate with TimeKeeper/S3F. This task required some modifications to the PLC emulator, to TimeKeeper, and to S3F. In this paper we focus on how those modifications deliver much better predictability in temporal behavior than is achieved using "best effort" emulation. With this basis, our continuing work will focus on studying cyber-security issues in PLC networks.

4 TESTBED DESCRIPTION

Awlsim (Micheal Busch 2016) is an open source python based PLC emulator which supports a large subset of Step 7 programming intructions. It emulates Siemens S7-300 and S7-400 PLC CPUs. PLC programs are written in Step 7 STL language and the source file is simply presented as an input to the emulator. Awlsim also allows the user to simulate sensor inputs and read output values via a GUI interface. We choose Awlsim in part because it is open source, and in part because of its extensive support for real Step 7 programs. However, Awlsim does not itself support the notion of networked PLCs. We provided this by developing additional system blocks/instructions that would allow two Awlsim instances to communicate with each other using the MODBUS application layer protocol.

MODBUS, developed by Modicon in 1979 is a widely used standard for communication over industrial networks still being widely used today (ModBus Protocol 2016). It is a simple Master-Slave based communication protocol which can run over many physical layers including RS-232, RS-485 and in the TCP/IP mode over ethernet. We implemented a python based MODBUS stack which is used by the Awlsim's ModBus instructions we developed to send and receive ModBus messages over simulated IP or serial networks.

4.1 General Architecture

Figure 1, shows the architecture of the TimeKeeper controlled hybird PLC emulation-simulation framework. Emulated PLC instances are run in separate Linux containers whose execution times and scheduling order are controlled by S3F through TimeKeeper. The user is allowed to assign an input generator script for each emulated PLC instance. The input generator script is invoked by the PLC instance at the start of every cycle and the script can be used to model the physical process being monitored by the PLC and alter the inputs to the PLC program. When the assumed connectivity is that of an IP network, packets that are injected by emulated PLCs are captured by the S3F network simulator and injected into the appropriate destination container after the specified link delay. Many legacy PLC systems use point-to-point serial communications. We needed to modify both TimeKeeper and S3F to support serial communication; our solution requires bypassing the Linux network stack completely.

4.2 Serial Connections

Serial connections were simulated by implementing a serial driver which manages reads and writes to device files. Each Linux container running an emulated PLC instance is assigned a separate device file to which it can read and write data to be sent over a simulated serial connection. The emulated PLC is allowed to maintain multiple serial connections simultaneously by specifying a connection ID along with the read or write requests. We augmented S3F to poll device files of every emulated entity for any available data among all active connections. It then simulates a half duplex RS-232 connection with handshakes (RTS,CTS) before sending the data over the simulated link. The received data is then written into the destination device file to be later read by the emulated PLC.

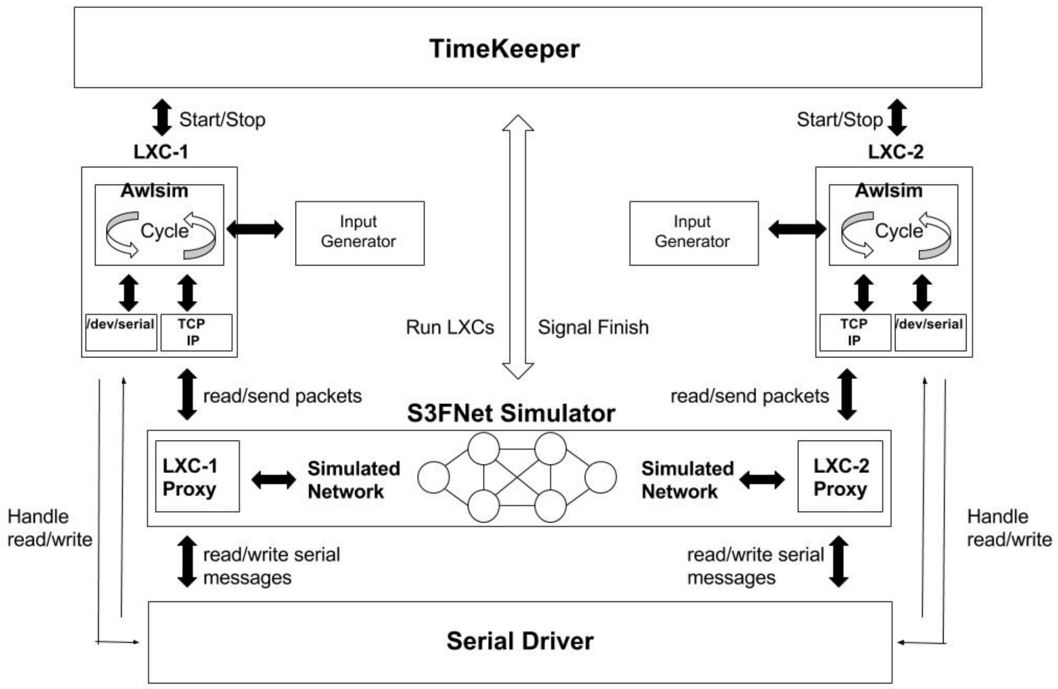


Figure 1: General Architecture of the PLC Network Testbed.

4.3 Experiment Configuration

Within a configuration file read at the beginning of the experiment, a user specifies the network type (IP or Serial) and the Time dilation factor for each emulated PLC. The general topology description and link delays are specified in a separate configuration file. In the IP mode, each PLC is connected to a simulated router by default and different routers are connected with each other in a user specified configuration. In serial mode, the connection is point to point, and thus the user specifies direct links and link delays between different PLCs.

4.4 HMI Devices and Compromised Routers

The testbed also supports emulation of Human Machine Interface (HMI) devices to interact with emulated PLCs. HMI devices are frequently used by human operators of Industrial Control systems to send and receive messages to monitor and update the operator's view of the current system state. HMI devices function as ModBus master devices and can send commands to PLCs functioning as ModBus slaves. The user can leverage the implemented ModBus stack API to write models of HMI devices and send arbitrary commands to slave PLCs.

The testbed also allows the user to experiment with different attack scenarios on the system and study its resilience. The user can designate certain routers in the topology as compromised. Routers which are classified as compromised invoke a specified attack script upon reception of each simulated packet. The user defined script can passively examine the packet or even perform man in the middle attacks by modifying it. We are using this feature of the testbed to study the effectiveness of intrusion detection algorithms under different active and passive adversarial attack models.

5 EVALUATION

We now demonstrate how the testbed manages to deliver predictable low variance behavior as exhibited in a real system, as compared to behavior of the same emulation under best effort coordination. All of the experiments described here were performed on a single dual core machine with 16 GB of RAM. We consider a hypothetical job routing scenario in a bottling plant. Jobs arrive at an incoming root node which performs some operations on the incoming job before routing the job to one of its child nodes. The topology is organized as a full binary tree of PLCs, with leaves being dispatch units where the finished product leaves the manufacturing plant. For simplicity, we assume that all nodes at a particular level in the binary tree perform the same action (i.e., run the same application) on its incoming jobs. This scenario is similar to automated baggage transfer in modern airports and bottle filling operations in bottling plants.

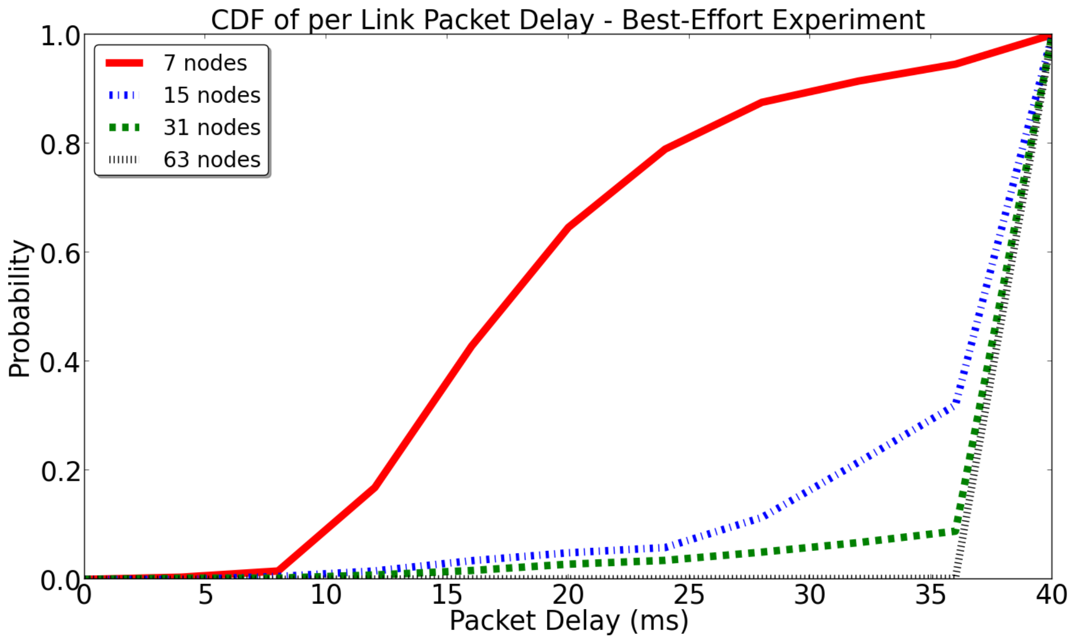
In this scenario, each PLC controls two conveyor belts connected to two other PLCs (child nodes in the binary tree,) and is also linked to both nodes over a high speed Ethernet connection. An incoming job is routed by the PLC to the child node with the least congestion in its sub-tree. To maintain a real time view of congestion in each child sub-tree, nodes repeatedly query their children for congestion levels in their sub-trees. The performance of such a dispatch system relies heavily on the frequency of updates and the speed of processing of these messages. Incoming jobs are simulated as sensor inputs at each node. Sensor inputs are scheduled at the chosen child node after the routing decision is made at the current node. Dispatch jobs are also modeled as simulated sensor inputs to the leaves of the tree. All inter-node communication links were assigned the same delay of 4ms. We impose stress on the test topology by forcing nodes to send updates at the fastest possible rate. Each node queries both its children, waits for their reply, and then initiates the next query immediately. We define the following metrics to analyse the accuracy of the testbed in simulating the expected behaviour of the target system.

- **Per packet delay:** Isolated industrial control networks typically have static nodes with fairly regular traffic patterns. Hence, the variance in the delay experienced by each packet is expected to be small. A testbed simulating these networks must also be able to guarantee stable packet delays under fixed traffic generation patterns.
- **Throughput of updates:** The number of update messages sent and received over the course of the run time of the system relies heavily on the network delay experienced by each packet and on the speed at which each update is processed. PLCs have stable per cycle execution times which can guarantee bounded information processing delays. The emulation testbed must therefore be able to exhibit low variance in cycle exhibition times and steady throughput values.

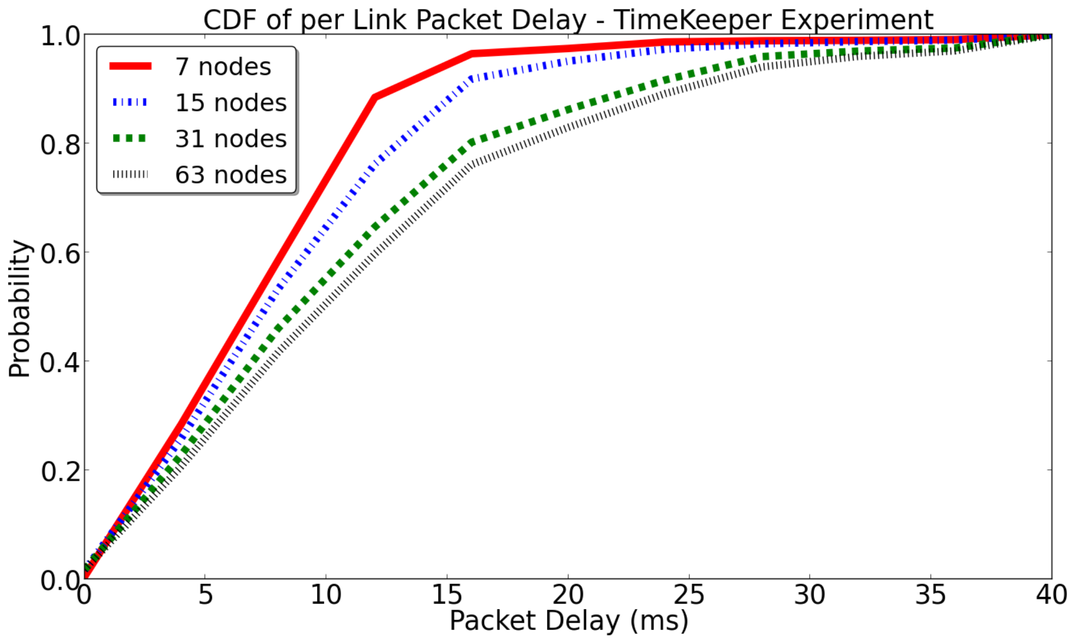
We vary the topology size in our experiments, and study the impact of topology size on the packet delays, cycle execution times and throughput of update messages. In an actual system we expect that packet delay and the cycle execution times will be insensitive to topology size, and we expect the aggregate update throughput to scale linearly with topology size. We compare our observations in *controlled* (under TimeKeeper) and *best-effort* experiments, and by doing so emphasize the benefits of tight coupling between emulation and simulation.

5.1 Experiments

For running best-effort experiments, we designed a simple setup where Linux containers running emulated PLCs are directly interconnected with each other. As TimeKeeper and S3F are not involved (and are required for support of serial communication), the emulated PLCs use the IP protocol to carry their Modbus communications. At the point a source constructs a packet, we note the “send time” and have the source delay transmitting the packet for a length of time equal to the modeled transmission delay. The time a packet is received by the target PLC is likewise noted, and the observed packet transmission delay was calculated by logging the receive times and send times of each packet.



(a) CDF of packet delay for best-effort experiment for different topologies.



(b) CDF of packet delay for TimeKeeper controlled experiment for different topologies.

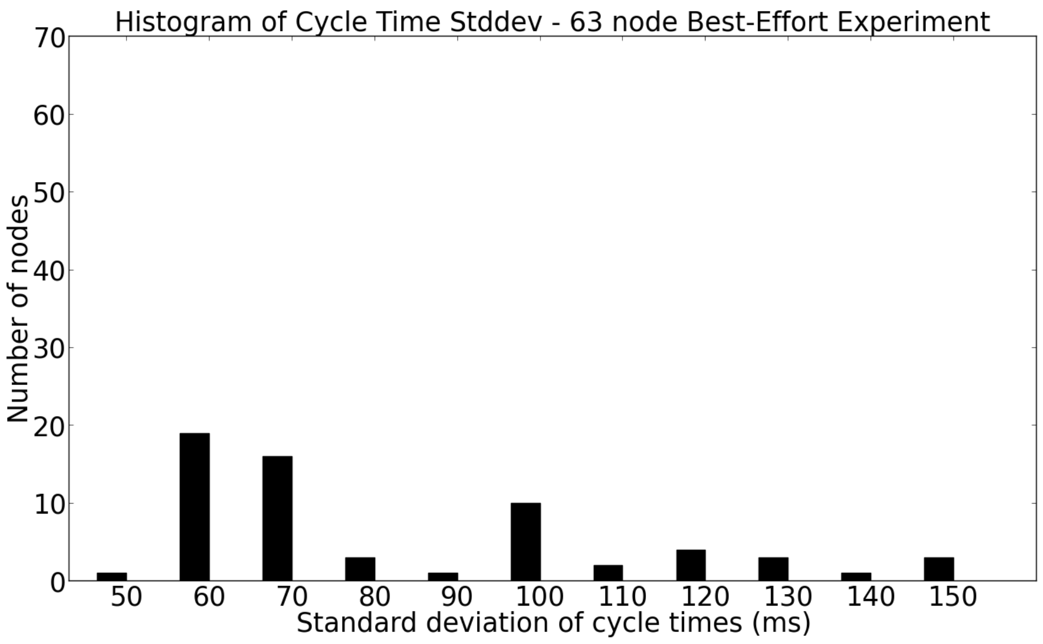
Figure 2: Comparison of CDF of standard deviation of cycle times and Maximum throughput of update messages for best effort and controlled experiments.

Figure 2a shows the plot of the cumulative density function (CDF) of the observed packet delays for different sizes of full binary trees. At each step, the topology size was nearly doubled but throughout the experiment was run on the same Linux server. Each tree node represents a PLC emulation, so as we increase the size of the topology, the computational resources available to a container decreases; Figure 2a shows that this has a clear impact on processing time, increasing topology significantly increases the packet delay. The reason for this is that as these containers are scheduled purely by the Linux kernel, when a container is swapped out in the midst of a transmission (i.e., the send time has been recorded and the wait time before delivery has not yet completed) the time it is swapped out will increase with the number of other processes clamoring for CPU attention. The larger the topology, the longer a container will be swapped out. For the topology with 7 nodes, we measured the mean and 90 percentile delays as 25.26*ms* and 32*ms* (approximately) respectively. On the other hand, for the topology with 63 nodes, the mean delay equalled 191.52*ms* whereas the 90 percentile delay exceeded 200*ms*. Whatever the true packet delay might be, we know that it should not change with increasing topology size, and in this system it does. We repeated the same experiment under a Time Dilation Factor of 15 with S3F and TimeKeeper. Recall that Awlsim is a Python program that interprets Step-7 instructions, and furthermore spawns a number of other processes in support of that interpretation. On a PLC the instructions run almost on bare metal. While a TDF of 15 is admittedly arbitrary, it does reflect that there is considerable overhead in interpreting Step-7 instructions. Fine-tuning a TDF requires instrumenting the modeled PLC to acquire a solid understanding of its native performance, and a study of the cost of interpreting its instructions within Awlsim.

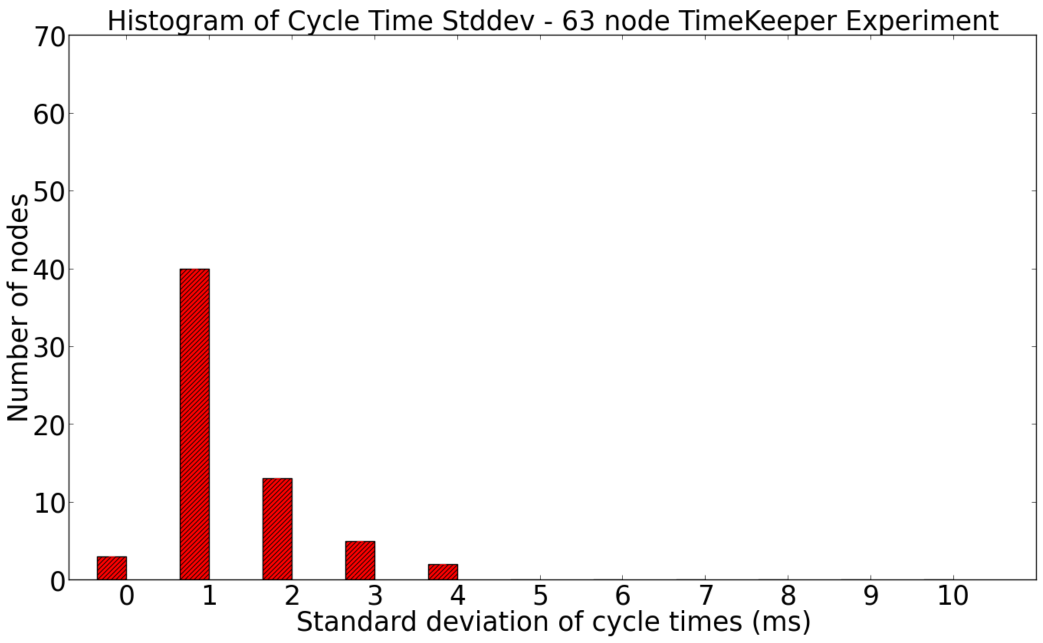
Figure 2b, shows the CDF of the observed packet delays in the dilated experiment. Packet delay is considerably smaller; the cumulative distribution functions is largely insensitive to topology size. For the 7 node topology, the mean and 90 percentile delays were approximately 11.42*ms* and 16*ms* respectively whereas the corresponding values for the 63 node topology equalled 15.57*ms* and 26*ms* respectively. We attribute the slight increase in the packet delay values to TimeKeeper's timing errors in starting and stopping processes and detecting and adding newly spawned processes to the experiment. It is also important to note that while the specified link delay was 4*ms*, the observed mean delays were all greater than 4*ms*. This is because each process inside the container is assigned a default fixed virtual time-slice of 1*ms* and scheduled in a round robin fashion by TimeKeeper. This can introduce a maximum delay proportional the number of processes running in the container. In our experiments, each container ran a total of 10 processes including the PLC emulator, which can in theory introduce an additional packet reception delay of up to 10*ms*. We understand the source of this inaccuracy; to deal with it more directly will require somewhat extensive modifications to the Linux kernel scheduler.

For both sets of experiments, we also compared the standard deviation of each emulated PLC's cycle execution time. Figure 3 shows the histogram (taken over all network nodes) of the standard deviation of cycle execution times in the controlled and best-effort executions of the 63 node topology. Each marked point on the x axis represents the a standard deviation interval starting from the specified point until the next one. The y axis indicates the number of nodes with standard deviation of cycle execution time within the specified interval. From the figures, it is easy to see that the standard deviation of the per cycle execution time is very low in the controlled experiment with the 90 percentile standard deviation below 3*ms* whereas it is of the order of hundreds of milliseconds for the best-effort experiment. The best-effort performance can again be attributed to the impact of a container being suspended in the midst of an execution burst, with increasing length of suspension as the number of other containers grows. The fact that the cycle execution times of PLCs in a controlled experiment remain fairly constant at each node further underlines the benefits of the tight coupling between emulation and simulation.

We also measured the maximum number of processed update messages for both classes of experiments. Figure 4 plots these measurements for various topology sizes. As expected in the real systems, we observed a near linear increase in the throughput values in the controlled experiment whereas the throughput fell dramatically in the best-effort experiment. The observed variance in the maximum throughput of the controlled experiment was also fairly small, as should be observed in the real system.



(a) Histogram of standard deviation of cycle times for undilated 63 node topology.



(b) Histogram of standard deviation of cycle times for dilated 63 node topology.

Figure 3: Comparison of standard deviation of cycle times for dilated and undilated 63 node topologies.

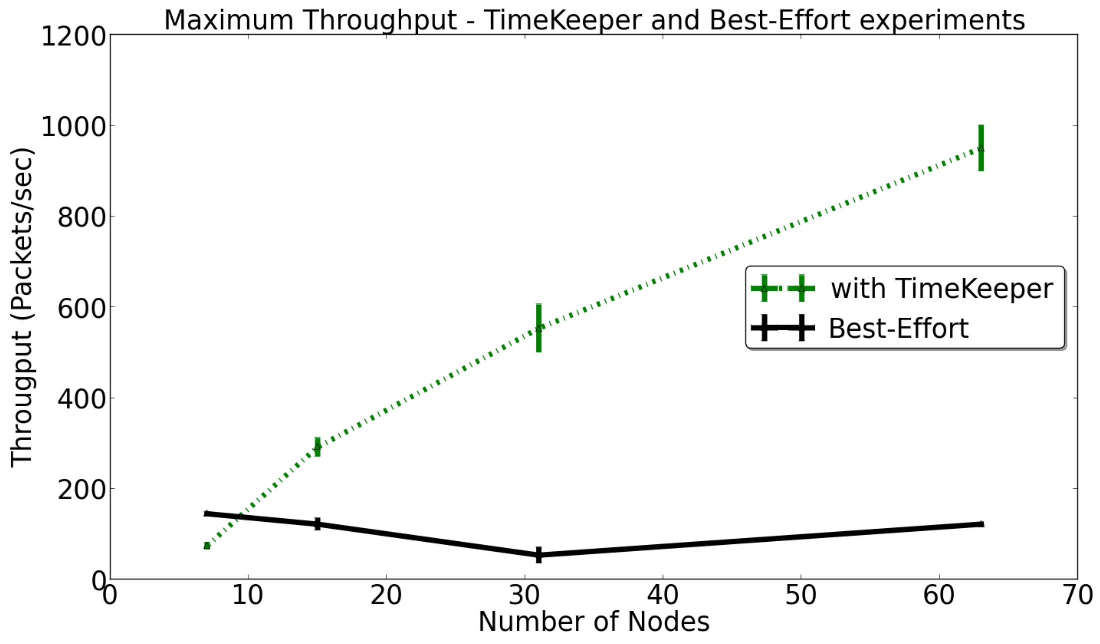


Figure 4: Comparison of Maximum throughput of update messages for best effort and controlled experiments.

6 RELATED WORK

The related work in this area is broadly divided into two categories: Testbeds for Industrial Control Systems (ICS) and Virtual Time based Emulation.

6.1 ICS Testbeds

Testbeds for Industrial control systems are either implementation based or emulation based or a combination of both. Giani, Karsai, Roosta, Shah, Sinopoli, and Wiley 2008 is an example of an implementation based testbed consisting of real PLCs and HMI devices. Data collected from implementation based testbeds is more accurate and realistic but the costs of expanding the testbed and maintaining it often outweighs the benefits. In Reaves and Morris 2012, the authors propose an open virtual testbed for ICS which interconnects emulated models of PLCs and Master Terminal Units (MTU) and Remote Terminal Units (RTU) with each other and with a central process simulator which simulates inputs to these virtual devices. However, the testbed does not simulate network links between its components, making the experiments non-repeatable. Best effort emulation and implementation testbeds such as Morris, Srivastava, Reaves, Gao, Pavurapu, and Reddi 2011 also exist where physical devices are interconnected with virtual emulated devices. However they are still expensive to maintain and less flexible.

6.2 Virtual Time Based Emulation

The notion of virtual time in emulation is not new. In Gupta, Yocum, McNett, Snoeren, Vahdat, and Voelker 2005, the authors proposed the notion of time dilation as a potential solution to scalable hybrid emulation-simulation systems. They defined time dilation factor (TDF) as the ratio of rate of progress of real time to virtual time and simply scaled the real time by TDF. SVEET! (Erazo, Li, and Liu 2009) is a TCP protocol evaluation testbed built using the time dilation technique discussed above. It stresses the importance of virtual time systems in performance-scalability studies of emerging technologies and demonstrates the cost benefits of time dilation by accurately predicting TCP performance on slower hardware. In Gupta's solution, virtual time is completely tied to the advancement of real time and there is no control over the scheduling order of dilated entities in the experiment which can lead to synchronization errors. In Zheng and Nicol 2011, the authors adopt a different approach to advancing virtual time which is less tied to the advancement of real time. They propose a virtual time system with a simulation control phase which decides how far each container should advance in virtual time. Containers which have advanced too far in virtual time are blocked to allow others to catch up. This keeps all containers closely synchronized in virtual time. TimeKeeper builds on this work by bringing the notion of TDF to the forefront and allows more finer control over process execution times.

7 CONCLUSION

In this work, we report on the development of a cost effective testbed for PLC networks where actual Step-7 programs are emulated, and their intercommunication is simulated. We demonstrated that the testbed delivers expected PLC network behavior in a hypothetical job routing scenario by studying the per packet delay and throughput under varied topology sizes. We see that our testbed yields low variance packet delays and low variance program cycle times (as expected in a real system) whereas best-effort emulation shows marked increases in both as the size of the network topology increases. Our observations highlight the advantages of tight temporal integration between emulation and simulation.

Our future work includes two thrusts. It is reasonable to expect that the execution time of a straight-line sequence of instructions in a Step-7 program might be estimated. TimeKeeper/S3F presently rely purely on measured execution time on the Linux platform for an estimate of computational effort expended. But here the actual time spent related to Step-7 instructions is muddied considerably with the overhead of Python interpretation and helper processes introduced by Awlsim. It seems to us plausible to annotate a Step-7 program with estimates of execution times on the modeled architecture, and somehow communicate those to TimeKeeper and/or S3F to allow a more direct (and repeatable) estimate of execution time on actual PLCs. In a second thrust we will use our testbed to design, implement and test intrusion detection algorithms for Industrial Control Networks.

ACKNOWLEDGEMENTS

This work was supported in part by the Boeing Corporation, in part by Department of Energy contract DE-OE0000780and in part by DHS contract 2015-ST-061-CIRC01The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Department of Homeland Security.

REFERENCES

Erazo, M. A., Y. Li, and J. Liu. 2009. “SVEET! a Scalable Virtualized Evaluation Environment for TCP”. In *Testbeds and Research Infrastructures for the Development of Networks & Communities and Workshops, 2009. TridentCom 2009. 5th International Conference on*, 1–10. IEEE.

Giani, A., G. Karsai, T. Roosta, A. Shah, B. Sinopoli, and J. Wiley. 2008. “A Testbed for Secure and Robust SCADA Systems”. *ACM SIGBED Review* 5 (2): 4.

Gupta, D., K. Yocum, M. McNett, A. C. Snoeren, A. Vahdat, and G. M. Voelker. 2005. “To Infinity and Beyond: Time Warped Network Emulation”. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, 1–2. ACM.

Jain, S., D. Lechevalier, J. Woo, and S.-J. Shin. 2015. “Towards a Virtual Factory Prototype”. In *Proceedings of the 2015 Winter Simulation Conference*, edited by L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, 2207–2218. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Lamps, J., V. Adam, D. M. Nicol, and M. Caesar. 2015. “Conjoining Emulation and Network Simulators on Linux Multiprocessors”. In *Proceedings of the 3rd ACM Conference on SIGSIM-Principles of Advanced Discrete Simulation*, 113–124. ACM.

Lamps, J., D. M. Nicol, and M. Caesar. 2014. “Timekeeper: A Lightweight Virtual Time System for Linux”. In *Proceedings of the 2nd ACM SIGSIM/PADS Conference on Principles of Advanced Discrete Simulation*, 179–186. ACM.

Linux Foundation 2016. “The Xen Project”. Accessed Jan. 14, 2016. http://www.xenproject.org/.

LXC 2016. “Linux Containers”. Accessed Jan. 14, 2016. https://linuxcontainers.org/.

Micheal Busch 2016. “Awlsim”. Accessed Jan. 14, 2016. https://bues.ch/cms/hacking/awlsim.html.

ModBus Protocol 2016. “ModBus Protocol”. Accessed Jan. 14, 2016. http://www.modbus.org/.

Morris, T., A. Srivastava, B. Reaves, W. Gao, K. Pavurapu, and R. Reddi. 2011. “A Control System Testbed to Validate Critical Infrastructure Protection Concepts”. *International Journal of Critical Infrastructure Protection* 4 (2): 88–103.

Nicol, D. M., D. Jin, and Y. Zheng. 2011. “S3F: The scalable Simulation Framework Revisited”. In *Proceedings of the 2011 Winter Simulation Conference*, edited by S. Jain, R. Creasey, J. Himmelspach, K. White, and M. Fu, 3288–3299. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Reaves, B., and T. Morris. 2012. “An Open Virtual Testbed for Industrial Control System Security Research”. *International Journal of Information Security* 11 (4): 215–229.

S3F Net 2016. “S3F Net”. Accessed Jan. 14, 2016. https://https://s3f.iti.illinois.edu/.

The Ns-3 Consortium 2016. “Ns-3”. Accessed Jan. 14, 2016. https://www.nsnam.org/.

Vaughn Jr, R. B., and T. Morris. 2016. “Addressing Critical Industrial Control System Cyber Security Concerns via High Fidelity Simulation”. In *Proceedings of the 11th Annual Cyber and Information Security Research Conference*, 12–24. ACM.

Zheng, Y., and D. M. Nicol. 2011. “A Virtual Time System for Openvz-based Network Emulations”. In *Proceedings of the 2011 IEEE Workshop on Principles of Advanced and Distributed Simulation*, 1–10. IEEE Computer Society.

AUTHOR BIOGRAPHIES

VIGNESH BABU is currently a Graduate student and Research Assistant in Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign. He is advised by Prof. David M. Nicol and his research is primarily focussed on cyber security issues in the Smart Grid and discrete event simulation frameworks. He received his Bachelor’s degree from the Indian Institute of Technology Guwahati.His email address is babu3@illinois.edu.

DAVID M. NICOL is the Franklin W. Woeltge Professor of Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign, and Director of the Information Trust Institute. He is the PI for two recently awarded national centers for infrastructure resilience: the DHS-funded Critical Infrastructure Reliance Institute, and the DoE funded Cyber Resilient Energy Delivery Consortium. His research interests include trust analysis of networks and software, analytic modeling, and parallelized discrete-event simulation,research which has lead to the founding of startup company Network Perception, and election as Fellow of the IEEE and Fellow of the ACM. He is the inaugural recipient of the ACM SIGSIM Outstanding Contributions award. He received the M.S. (1983) and Ph.D. (1985) degrees in computer science from the University of Virginia, and the B.A. degree in mathematics (1979) from Carleton College. His email address is dmnicol@illinois.edu.