Distributed ONE: Scalable Parallel Network Simulation

Vedavyas Duggirala
Dept of Computer Science
Virginia Tech
vduggira@vt.edu

Calvin J. Ribbens
Dept of Computer Science
Virginia Tech
ribbens@vt.edu

Srinidhi Varadarajan
Dell Inc
srinidhi_varadarajan@dell.com

ABSTRACT
In this paper we describe a distributed-memory parallel implementation of the Open Network Emulator (ONE), a network simulator that combines the controllability of simulation with the direct code execution advantages of emulation and experimental testbeds. ONE uses a scaled real-time model called Relativistic Time (RT). We describe a RT-based reactive global warp detection algorithm to exploit lookahead. The Distributed ONE system provides a platform for network simulation that combines model fidelity (existing network applications can be compiled and instantiated within ONE without modification), temporal fidelity, and good scalability. We present strong and weak scaling performance results for the Distributed ONE system on up to sixteen nodes of a distributed-memory parallel cluster, on simulations involving up to 16,000 virtual hosts.

Categories and Subject Descriptors
I.6.8 [**Computational Methodologies**]: Simulation

General Terms
Algorithms, Design, Performance

Keywords
Relativistic Time, Parallel Discrete Event Simulation, Time Dilation, Parallel Network Simulation.

1. INTRODUCTION
Simulation, emulation and experimental testbeds are three widely used network research methodologies. Each of these approaches has its advantages and limitations, reflecting tradeoffs in important properties such as programming model, accuracy and cost. Table 1 summarizes these tradeoffs between simulation on the one hand and emulation and testbeds on the other.

Ideally networking researchers use simulation to study prototypes of new applications and protocols, and then test and refine increasingly realistic implementations using emulation and testbeds before final deployment. However, it can be time consuming and expensive to move through each of these steps because of the differences in programming models and the overhead of experimental setup.

Simutools 2013, March 05-07, Cannes, France

DOI 10.4108/icst.simutools.2013.251733

To address the above limitations, we have developed the *Open Network Emulator* ([5]a network simulator that combines the controllability and scalability of simulation with the direct code execution advantages of emulation and experimental testbeds. ONE combines the best features of each approach (Table 1), thereby bridging the gap between simulation and emulation and testbeds, and allowing researchers to use the complementary strengths of one approach to verify and validate results from other approaches. In designing ONE we had three main goals:

1. **Model fidelity:** run real-world applications and implementations of network code, instead of simulation models.
2. **Temporal fidelity:** closely reflect the performance of applications in real-world settings, including reflecting the computational cost of processing packets.
3. **Scalability:** efficiently model thousands of high-fidelity virtual nodes.

Table 1. Comparison of Network Research Methodologies

Feature	Simulation	Emulation and Testbeds
Programming Model	Event based simulation API	Socket based general purpose languages
Time Model	Virtual time	Wallclock time
Scalability	Thousands of nodes	One to few tens of nodes
Controllability	High	Limited
Repeatability	High	Limited
Cost	Low	Very high
Modeling Power	Prototype future technologies	Limited by current state of art

In previous work [5] we described the design and implementation of a shared-memory parallel version of ONE, showing how it achieves the first two goals mentioned above. In this paper we describe our distributed-memory parallel implementation of ONE. The scalable distributed-memory platform provides the memory and processing capacity required to meet our third goal of simulating many thousands of virtual nodes.

The remainder of this paper is organized as follows. We first provide a brief summary of the design and key components of the shared-memory parallel implementation of ONE in Section 2. Section 3 describes our approach to distributed-memory parallelism for ONE, focusing on distributed warp detection, the key ingredient to achieving good parallel performance in this context. We report and discuss preliminary performance results in Section 4. Section 5 discusses related work and we conclude with some directions for future work in Section 6.

2. THE OPEN NETWORK EMULATOR (ONE)

ONE leverages three novel components to achieve model fidelity and temporal fidelity. The first component is the *Weaves*[10] compiler framework, which transforms existing network applications into a compositional model suitable for our simulator. Secondly, ONE users *Lunar*, a user-level network stack derived from the networking subsystem of the Linux kernel, to allow existing network applications to be compiled and instantiated without requiring source code modifications. Finally, ONE is based on *Relativistic Time*, a new time model that combines the controllability of virtual time with the fidelity of real time. We briefly introduce each of these three components below.

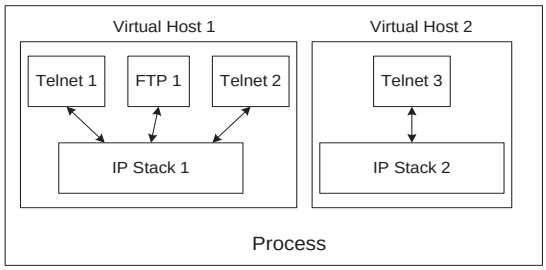

Figure 1. ONE's composition model: network in a process.

2.1 Weaves Compiler Framework

ONE supports the compositional model shown in Figure 1. Each virtual host has a unique IP stack, and each IP stack can be linked to multiple applications. Multiple such virtual hosts can be composed to model a network in a single process.

This compositional model introduces a challenging state-sharing problem. The global state of IP Stack 1 in Figure 2 must be shared among its applications (FTP 1, Telnet 1 and Telnet 2) but not with IP Stack 2. However, the global state of Telnet 1 and Telnet 2 should not be shared. Neither the standard process model nor the threads model supports this compositional model, which requires arbitrary sharing of global data

ONE uses the Weaves compiler and runtime to provide the necessary selective sharing of data between threads. A LLVM compiler[16] pass automatically transforms existing network applications/protocols written in any LLVM supported language like C and C++ into composable modules, which can then be combined to create arbitrarily complex network stacks.

The Weaves framework operates on the compiler intermediate representation to identify every global variable, replacing each global variable access with an instrumentation function call. The Weaves runtime has a configuration file that specifies the data sharing relationships. The instrumentation function uses the lookup tables generated by the configuration file to resolve each symbol to the correct module. The details of the compiler transformations and optimizations to reduce runtime overhead are specified in [5].

This compiler framework obviates the need for heavyweight virtualization. As we show in the results section, ONE's approach of virtualizing just the network is highly efficient and can emulate over 1000 virtual hosts on a single machine with 8GB of RAM. This is an order of magnitude more efficient than container based lightweight virtualization technologies like OpenVZ [19].

2.2 Lunar User Level TCP/IP Stack

To meet our goal of running unmodified socket API based network applications, ONE uses a full-fledged TCP/IP stack derived from the Linux kernel (version 2.6.18). We provide dummy abstractions for the rest of the Linux kernel (scheduler, processes, file system, etc.) to compile the networking subsystem as a user-level shared library. The original kernel source is otherwise unmodified.

The Lunar shared library exposes the POSIX Socket API to applications. The library has `send_msg, receive_msg`, and `timer` function calls to interface with a discrete event simulation. Using the timer function, the simulator can control the notion of time seen by the network stack and applications linked against it.

Lunar uses an asynchronous thread to provide a network hardware device abstraction. This thread is used to offload message transmission and reception from the virtual host. Interrupt enabling and disabling calls (`cli` and `sti`) in the kernel code are used to serialize any accesses to the network stack's internal data structures by the interrupt thread and application threads linked against the shared library.

Common network configuration utilities like *ifconfi*g and *route* are linked against the shared library to configure the network stack. The network stack can also dump packet traces in the standard libpcap format.

2.3 Relativistic Time

Real world applications run in wallclock time, while simulations run in virtual time. To achieve temporal fidelity with real applications running over a simulated network we need a new time model to reconcile these two notions of time.

Relativistic Time (RT) is derived by applying two operations, *dilation* and *warp*, on wallclock time. The *dilation* operation slows the progress of relativistic time w.r.t. wallclock time, while the *warp* operation speeds it up. RT is based on the observation that causality errors are due to a mismatch between the resources of the virtual and physical network. The capacity mismatch could be due to computation, communication or a combination of both. To measure this mismatch, we define a *scale factor* SF based on the ratios between capacities of the virtual and physical computational platform: $SF = Max\ (N_s/N_p,\ B_s/B_p\ )$, where

N_s – Number of virtual nodes
N_p – Number of physical nodes
B_s – Bandwidth of virtual network
B_p – Bandwidth of physical network

Slowing down the wallclock time by SF guarantees that the simulation has enough virtual resources to complete the simulation without causal inversions.

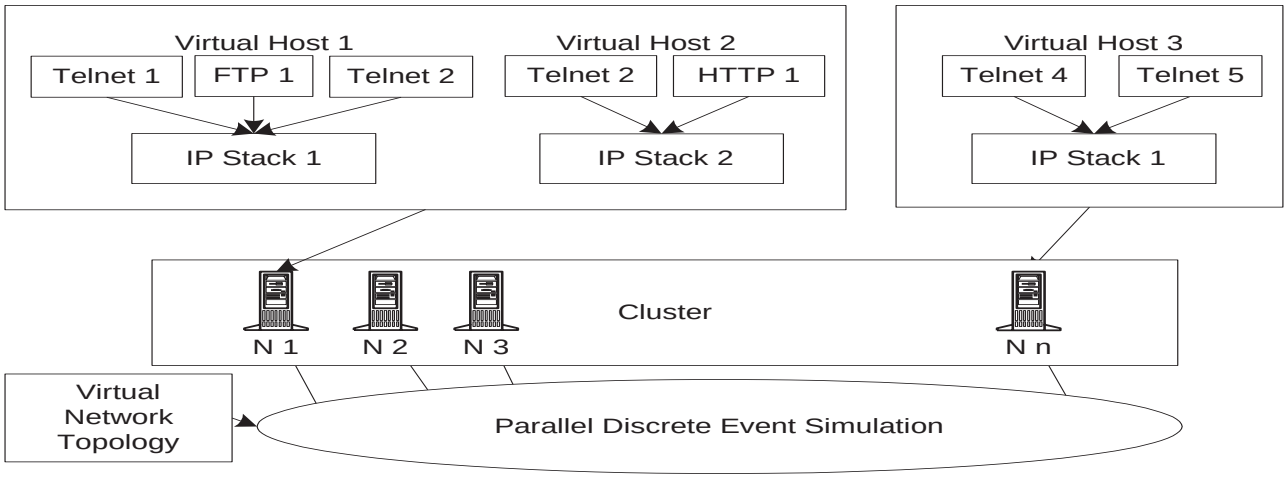

Figure 2. Architecture of Distributed Open Network Emulator

Each virtual node in the simulation has access to two clocks, its local clock and the Global Relativistic Time (GRT). GRT is derived by slowing down (*dilation*) wallclock time by SF. GRT progresses with wallclock time. On the other hand, the local clock is updated only during the execution of that particular virtual host. Any virtual node can process events below GRT. The simulation doesn't let any node get ahead of GRT. In this way, GRT is similar in concept to GVT (Global Virtual Time) used by conservative time synchronization algorithms [3, 4, 11]. However, unlike GVT, GRT advances by itself and doesn't need the exchange of *null* messages.

If all the virtual nodes in the simulation are blocked, the simulation performs a *warp* operation and GRT is set to the next event in the pending event list. The scale factor SF represents the worst case lookahead of the simulation. Due to various reasons (e.g., application processing overhead, protocol state and traffic patterns) the lookahead in the simulation is potentially larger than the pessimistic estimate represented by SF. When that happens all the virtual nodes would be blocked and ONE performs a warp, thereby speeding up the simulation.

Due to its reactive nature, RT can take advantage of sources of lookahead without explicitly calculating it. The theory behind RT is described in detail in [5, 18].

3. DISTRIBUTED ONE

Figure 2 shows the high-level architecture of the distributed-memory parallel Open Network Emulator. The changes needed to extend the shared-memory implementation of ONE to take advantage of scalable distributed-memory platforms are primarily in two areas: event queue interaction and warp detection.

3.1 Event Queue Interaction

Each physical node that a distributed ONE simulation is running on corresponds to a *Logical Process* (LP). We maintain an event queue for each LP. As applications running in a given LP generate events (e.g., send_msg, receive_msg) the system determines whether these events are local to that LP or whether they correspond to a virtual node that is part of a physically remote LP. This determination relies on the fact that each LP holds a full copy of the topology structure of the network being simulated, i.e., the mapping between virtual nodes (IP addresses) and physical nodes (LPs). ONE currently uses static routing between the nodes. Remote enqueuing operations generate MPI messages, which include application data as well as simulation-specific data. Distributed ONE is based on MVAPICH2 [9]. While any thread can send MPI messages, a dedicated thread in each LP handles all receive operations.

3.2 Distributed Warp Detection

Warp detection on a shared-memory machine is straightforward. On a distributed-memory machine, however, it is complicated by the lack of global state and the possibility of messages in transit. Our parallel warp detection algorithm is described below. Table 2 lists the important parameters that are used in this algorithm.

Table 2. Variables used in distributed warp detection algorithm

N_{TOTAL}	Total number of virtual nodes
N_{LP}	Number of Logical Processes
N_{LP_NODES}	Number of virtual nodes per LP
$msg\_inflight_i$	Incremented for every remote message sent and decremented for every remote message received
warp_benefit	(next_event_time - current_time)/Scale factor
WARP_COST	A tunable parameter. It depends on the cost of MPI collective. It determines how aggressive we would like to be in warp detection.
WARP_TIMEOUT	A tunable parameter of how long to wait for warp.

Each LP increments *msg_inflight* for every remote message sent and decrements it for every message received. Whenever an LP detects a local block, it sends a warp request to the coordinator thread (a special thread running on the LP corresponding to MPI rank 0) with its next event time and *msg_inflight.*

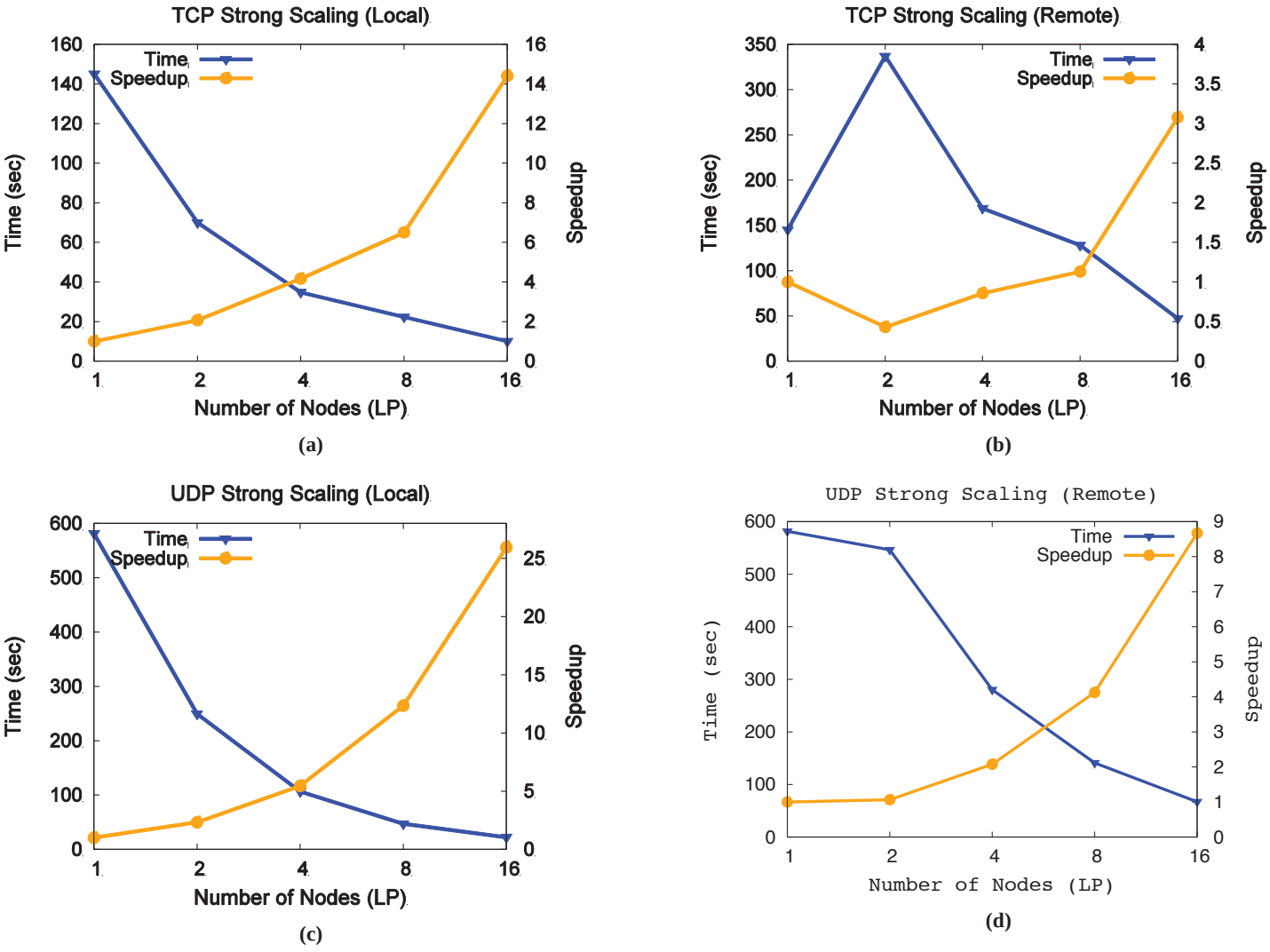

Figure 3. Strong scaling results with 2000 virtual nodes for TCP (top), UDP (bottom) and for Local (left) and Remote (right) mapping.

The coordinator starts a timeout timer as soon as it receives a warp request. If it receives N_{LP} requests and the sum of all the *msg_inflight* values is zero, it selects the lowest next event as the new GRT and sends a warp success response to all LPs. If the timeout expires, or if N_{LP} warp requests are received but there are still messages in transit, a warp fail message is sent.

Aggressive warp detection is highly beneficial, since warp speeds up the simulation by jumping over idle time. However, due to the The warp detection algorithm has two important tunable parameters. The first, *warp_cost* reflects the fact that warp detection itself takes some time. Since GRT progresses independently of wallclock time, a blocked LP can process the next event with the passage of enough time. If the next event is less than warp_cost microseconds away, an LP will not initiate a warp detection, even if it is blocked.

If the second parameter, *warp_timeout*, is set too low, the algorithm can potentially go into an infinite loop. The coordinator would too quickly abort the warp, while the LPs would continually restart the warp attempt. The downside of having a large warp_timeout value is that it increases the warp cost.

asynchronous nature of distributed systems, the warp detection algorithm can occasionally fail to detect a warp even when there is the possibility of one. For example, if there are messages in flight or all warp requests haven't arrived before the timeout, the coordinator sends a warp fail message; any LPs can then restart the warm detection algorithm if they are still blocked. Note that a missed warp opportunity can result in some performance degradation, but does not affect correctness.

4. EXPERIMENTAL EVALUATION

To evaluate the potential parallel performance of Distributed ONE we carried out experiments using thousands of virtual nodes connected by point-to-point 1 Gb bandwidth, 4 us latency links. In this section we present results for both strong scaling (with 2000 virtual nodes) and weak scaling (with 1000 virtual nodes per physical node). Each simulation involves half of the virtual nodes running a source application and half running a sink application.

Each source application sends 10,000 512 byte messages to a partner sink application at constant bit rate. We ran the applications using both UDP and TCP. (We also generated traffic data using a Pareto distribution. The parallel performance results were very similar to the CBR cases reported here.) We looked at two mappings of virtual nodes to physical nodes, one ("local mapping") where corresponding source and sink applications are on the same physical node, and the other ("remote mapping") where the sink application is always on a different physical node from the source application. These two mappings represent a best and worst case in terms of communication requirements for the parallel simulation, since the first case minimizes the number of remote event enqueues (to zero) and the second case maximizes the number of remote enqueues (i.e. all packet transmissions cross a physical node boundary). Since our distributed warp detection algorithm is agnostic to any particular mapping, these experiments also allow us to isolate the overhead of warp detection from the overhead of remote enqueues.

ONE was tested on the SystemG cluster. Each node in the cluster has two 2.8 GHz Intel Xeon E5462 Quad core processors (total 8 cores) with 8 GB of RAM. The nodes are connected by a QDR (40Gbps) Infiniband network.

Figure 3 presents the strong scaling results. All cases involve 2000 virtual nodes. Note that 1 physical node case corresponds to original shared-memory version of ONE.

The speedup in local mapping is good for both TCP(Fig. 3.a) and UDP (Fig3.c). This shows that distributed warp detection is low and it scales well. The superlinear speedup in the case of UDP is due to reduced contention in the calendar queue. For TCP, the flow control and packet coalescing reduces contention on event queue.

Remote mapping (Fig 3.b, Fig 3.d) doesn't do as well from 1 to 2 physical nodes, especially for TCP where we actually slow down,

due to the overhead of remote enqueues. ONE events carry full application payloads and are at least the size of MTU (1500 bytes) of virtual link being emulated. But as the number of physical nodes increase beyond two we realize relatively steady improvements, since the communication overhead gets amortized.

Figure 4 presents similar results using weak scaling, with the number of virtual nodes per physical node held constant at 1000.

Again we see local mapping has near constant overhead. For Remote we again see a significant increase in time from 1 to 2 physical nodes, but then the time per physical node is relatively constant.

5. RELATED WORK

Algorithms to determine the set of events that can safely be

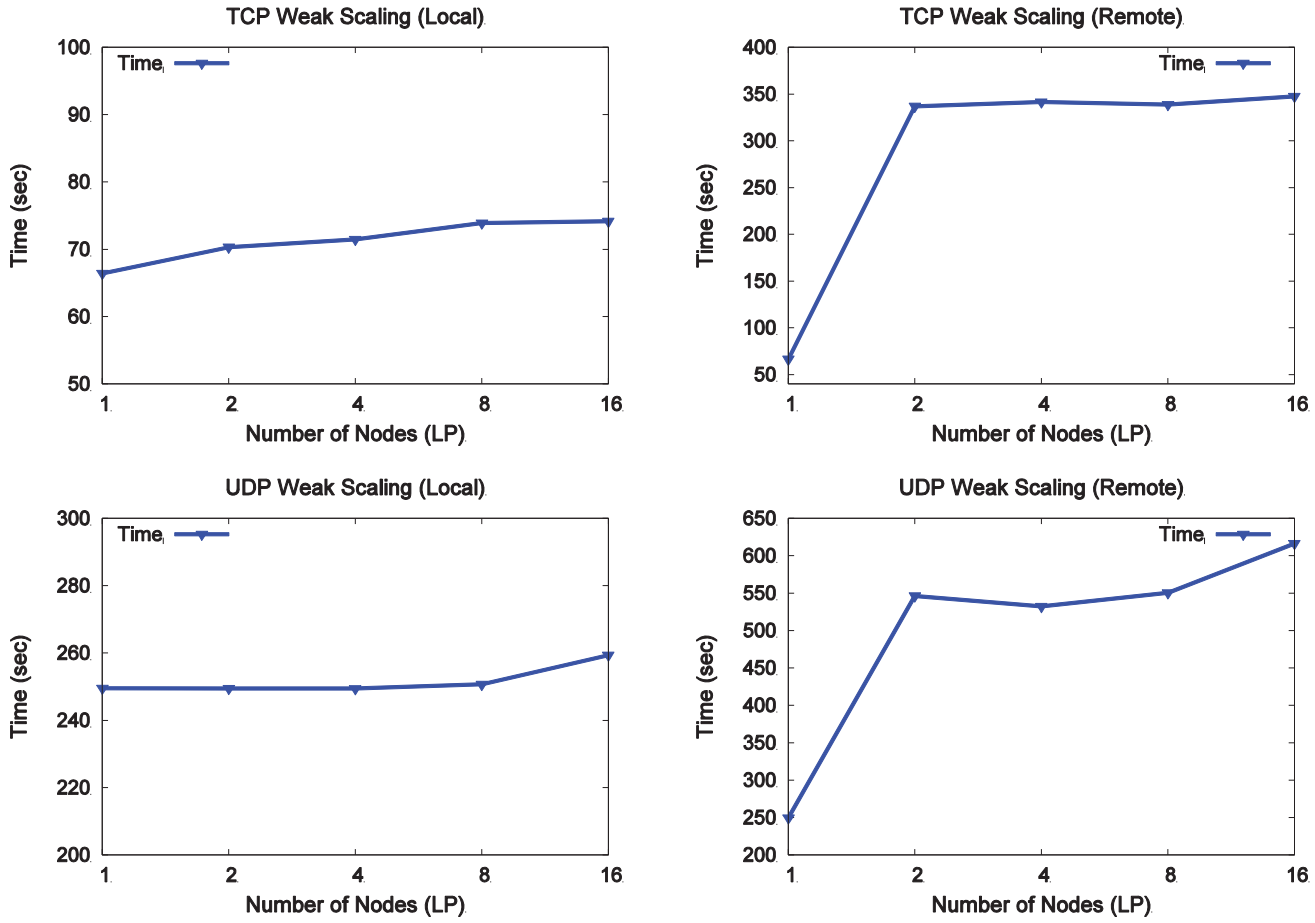

Figure 4. Weak scaling results using 1000 virtual nodes for one physical node for TCP (top), UDP (bottom) and Local (left), Remote (right) mapping.

executed in parallel in a discrete-event simulation fall into two broad categories: conservative and optimistic.

Optimistic algorithms need a mechanism to checkpoint state and rollback, if there are message inversions. Due to the high cost of check-pointing the state in network simulators, these algorithms haven't been used in this domain.

Conservative time synchronization algorithms have been the mainstay of parallel network simulation. CMB [2, 4] is the first algorithm that was proposed to solve the time synchronization problem. This algorithm was prone to deadlocks; so null messages were added as a deadlock avoidance measure. The number of null messages sent can be very high on a highly connected network. The cost of null messages is discussed in [11].

Another approach is discussed in [3]. This algorithm uses a compute and a synchronization phase. All the LPs perform a barrier synchronization and use the local next event time and lookahead to determine the safe set of events to be executed in the compute phase. LPs synchronize after each compute phase.

Another approach to parallel network simulation is the use of federation [21]. Federation was initially developed to promote model reuse and interoperability between heterogeneous simulators. Compass [14] is an example of this approach that combines heterogeneous simulators like ns-2 [17] and GloMoSim [20] to take advantage of their differing strengths. It combines the transport models of ns-2 with wireless models of GloMoSim.

Parallel and Distributed NS (PDNS) [12] uses multiple instances of a serial simulator (ns-2) to simulate disjoint parts of a network. It uses a federation based interface to combine results from these disjoint simulations. PDNS was handicapped by a number of limitations in its base simulator ns-2.

GTNetS[15] is a serial simulator developed to address the various shortcomings of ns-2. A parallel version was also developed using the federation approach [14]. ns-3's [8] design was greatly influenced by GTNetS; ns-3 also supports distributed simulation using [13]. Unlike ONE all of the above approaches use models written in their own specific API.

We now discuss the network simulators based on virtual machines. Time jails [6] and DieCast [7] and [19] all use virtual machines (VM) to run real-world network applications and overcome the state sharing problem. [7] uses Xen, [6] uses BSD jails while [19] uses OpenVZ. Virtualization technologies like Xen are heavyweight and consume significant resources. The context switching cost is also very high for VMs compared to ONE where all virtual hosts are just threads in a single process.

OpenVZ [19] uses container based OS level virtualization. Containers are lighter on resource consumption and it is possible to run a couple of hundreds of containers on the hardware that can run only ten or so VMs. A downside is that all containers use the same underlying OS network stack; hence it is not possible to configure the network stack of individual hosts differently. This would preclude certain kinds of experimental setups where different end hosts use different congestion control or active queue management algorithms. As we show in the results section ONE's virtual network stack is even lighter.

All VM based schemes, e.g. [7], [19], need modifications to the underlying OS kernel or hypervisor to implement time dilation, while ONE doesn't need any.

DieCast [7] uses uniform time dilation on end hosts to simulate faster machines than physically available. ONE uses time dilation, to emulate a higher number of same capacity nodes. It also warps past idle states in the simulation without affecting temporal fidelity.

Unlike Distributed ONE, all VM based parallel simulators are confined to a single physical node.

A previous version of distributed ONE is discussed in [1]. However, due to a number of design choices in both the event management and communication layer, it didn't prove to be very scalable.

6. CONCLUSIONS AND FUTURE WORK

We have described Distributed ONE, a new parallel implementation of the ONE direct code execution network simulator. Performance results indicate very scalable performance up to 16 nodes of a distributed-memory cluster, in both the strong scaling (2000 virtual nodes) and weak scaling (1000 virtual nodes per physical node) sense, even in the case where all communication generated by the applications must cross the network. The results also show that our distributed warp detection algorithm scales extremely well. The overhead of remote event enqueues is relatively high when compared against the single LP case, but this overhead parallelizes well, so that scalability is still achieved.

We are currently investigating the performance of warp detection with a variety of distributions and network topologies.

7. REFERENCES

[1] Bergstrom, C. 2006. *The Distributed Open Network Emulator: Applying Realtivistic Time*. Virginia Tech.

[2] Bryant, R.E. 1977. *Simulation of Packet communications architecture computer systems*. Technical Report #188.

[3] Chandy, K.M. and Misra, J. 1981. Asynchronous distributed simulation via a sequence of parallel computations. *Commun. ACM*. 24, 4 (Apr. 1981), 198–206.

[4] Chandy, K.M. and Misra, J. 1979. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Trans. Softw. Eng.* 5, 5 (Sep. 1979), 440–452.

[5] Duggirala, V. and Varadarajan, S. 2012. Open Network Emulator: A Parallel Direct Code Execution Network Simulator. *Proceedings of the 2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation* (Washington, DC, USA, 2012), 101–110.

[6] Grau, A. et al. 2008. Time Jails: A Hybrid Approach to Scalable Network Emulation. *Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation* (Washington, DC, USA, 2008), 7–14.

[7] Gupta, D. et al. 2005. To infinity and beyond: time warped network emulation. *Proceedings of the twentieth ACM symposium on Operating systems principles* (2005), 1–2.

[8] Henderson, T.R. et al. 2006. ns-3 project goals. *Proceeding from the 2006 workshop on ns-2: the IP network simulator* (2006), 13.

[9] Huang, W. et al. 2006. Design of High Performance MVAPICH2: MPI2 over InfiniBand. *Sixth IEEE International Symposium on Cluster Computing and the Grid, 2006. CCGRID 06* (May. 2006), 43 –48.

[10] Mukherjee, J. and Varadarajan, S. 2005. Develop once deploy anywhere achieving adaptivity with a runtime linker/loader framework. *Proceedings of the 4th workshop*

on Reflective and adaptive middleware systems (New York, NY, USA, 2005).

[11] Nicol, D.M. 1993. The cost of conservative synchronization in parallel discrete event simulations. *Journal of the ACM (JACM)*. 40, 2 (1993), 304–333.

[12] PDNS - Parallel & Distributed NS: *www.cc.gatech.edu/computing/compass/pdns/*.

[13] Pelkey, J. and Riley, G. 2011. Distributed Simulation with MPI in ns-3. (2011).

[14] Riley, G.F. et al. 2004. A federated approach to distributed network simulation. *ACM Trans. Model. Comput. Simul.* 14, 2 (2004), 116–148.

[15] Riley, G.F. 2003. The Georgia Tech Network Simulator. *Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research* (New York, NY, USA, 2003), 5–12.

[16] The LLVM Compiler Infrastructure: *http://llvm.org*.

[17] The ns-2 Network Simulator: *http://nsnam.isi.edu/nsnam/index.php*.

[18] Varadarajan, S. and Nance, R. 2004. Relativistic Time: The Evolution of a Temporal Model for Multi-Domain Simulation. *Proceedings of the Operations Research Society* (Birmingham, England, Mar. 2004).

[19] Yuhao Zheng and Nicol, D.M. 2011. A Virtual Time System for OpenVZ-Based Network Emulations. *2011 IEEE Workshop on Principles of Advanced and Distributed Simulation (PADS)* (Jun. 2011), 1–10.

[20] Zeng, X. et al. 1998. GloMoSim: a library for parallel simulation of large-scale wireless networks. *Proceedings of the twelfth workshop on Parallel and distributed simulation* (Washington, DC, USA, 1998), 154–161.

[21] 2010. IEEE Standard for Modeling and Simulation (M amp;S) High Level Architecture (HLA)– Framework and Rules. *IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000)*. (2010), 1 –38.