Ultra Large-Scale Wireless Network Models using Massively Parallel Discrete-Event Simulation

Justin M. LaPre,* Christopher D. Carothers,* Kenneth D. Renard,† Dale R. Shires‡

* *Department of Computer Science*, *Rensselaer Polytechnic Institute, Troy, NY 12180*
{laprej,chrisc}@cs.rpi.edu
† *Secure Mission Solutions*, *Reston, VA 20190*
kdrenard2@gmail.com
‡ *Army Research Laboratory, Aberdeen Proving Ground*, *Aberdeen, MD 21005*
dale.r.shires.civ@mail.mil

***Abstract*—Mobile ad-hoc networks (MANETs) are becoming pervasive in our always-connected lives of iPhones, laptops, etc. While entertainment is important, next generation military applications can and will come to depend on these new technologies. Connectivity can no longer be optional; it has become a must-have requirement. Over the past decade, the DoD has spent billions of dollars investigating new approaches to radio including the Joint Tactical Radio System (JTRS) and Software-Defined Radios (SDRs). However, large-scale deployments (1 million+ nodes) have never been tested for a single homogeneous network of that magnitude. The goal of this paper is to demonstrate the efficacy of large-scale wireless network simulations on multiple HPC platforms. By exploiting the strengths of two different supercomputers, we evaluate our million node Optimized Link State Routing (OLSR) network on the CCNI Blue Gene/L as well as the Army Research Lab's SGI ICE 8200 (Harold) compute cluster. By exploiting infrequent synchronization we may push the simulation to its maximum potential for our model. We realize speedups between 315–450 in our normalized parallel simulation over sequential.**

***Index Terms*—massively parallel simulation, OLSR, 802.11, wireless networking**

I. Introduction

Mobile Ad-hoc Networks (MANETs) have clear and obvious advantages over classical infrastructure installations. They require no setup, no organization, and little to no user interaction in order to maintain connectivity and utility. However, all of these benefits do not come for free. Performing constant route maintenance uses valuable resources such as power and bandwidth, while it is possible that no useful traffic may be routed on the network. On the flip side, on-demand route construction often creates non-negligible latencies while route discovery and repair takes place.

Insight into desirable or required properties will allow one to make a choice as to which routing protocol will provide the most benefit. Attaching Quality of Service (QoS) support to various protocols will allow improvements to different protocol properties such as energy efficiency, route length, link-quality, etc.

This study will evaluate the Optimized Link State Routing (OLSR) [1] protocol in a large-scale scenario. OLSR is a proactive link-state routing protocol with some key optimizations making it more suitable for mobile scenarios than normal link-state protocols such as Open Shortest Path First (OSPF) [2] routing. Through the use of MultiPoint Relays (MPRs), power saving is achieved by not having to forward all overheard packets as well as maintaining shortest paths.

The military is naturally interested in such technologies. However, prototyping hardware and installing a large-scale deployment prior to validation would not be feasible. Consequently, modeling and simulation of these large-scale scenarios is the right approach for understanding their performance dynamics prior to acquisition and deployment. Our research here leverages the previous results generated using parallel ns-3 [3] to determine how an OLSR model and simulation engine optimized for massively parallel processing would perform on supercomputer systems.

For this task we use the Rensselaer Optimistic Simulation System (ROSS) which has demonstrated parallel model scalability on supercomputer systems containing more than 100,000 processors [4], [5]. ROSS supports Jefferson's [6] Virtual Time synchronization approach called *Time Warp* which is a paradigm where events are allowed to proceed until an anomaly is detected. At this point, the offending event is rolled back to a prior point where the anomaly no longer exists. The simulation is then allowed to resume forward progress from its valid state. The conservative approach to simulation was made possible by simultaneous realizations by Chandy/Misra and Bryant [7] [8] and is also supported in ROSS. Knowing the smallest time stamp event on an LP basis allows forward progress to be made by passing null messages between LPs. Ensuring the correctness of each LP will result in correctness of the entire simulation and prevent deadlock. In Chandy and Misra [9], deadlocks are allowed to occur. They are then detected via a distributed scheme and broken by allowing the LP with the smallest event time stamp to proceed. Our specific conservative implementation is based on Nicol's YAWNS protocol [10].

Our main contribution is to demonstrate the efficacy of ultra large-scale wireless networking simulations utilizing the OLSR protocol on HPC platforms. Section 2 will detail our implementation of OLSR as well as ROSS, our simulation engine. Section 3 will summarize our results. Section 4 will

cover new and related work followed by our conclusions in Section 5.

II. Implementation

A. OLSR Protocol

Optimized Link State Routing (OLSR) is a link-state routing protocol optimized for mobile ad-hoc networks. We will only give a high-level overview and not belabor the details of OLSR in this study; interested readers are referred to RFC 3626 [1] for an in-depth look at OLSR.

OLSR nodes establish, through the periodic transmission of HELLO messages, the notion of a *link*, i.e. another node we are able to "hear." This relation is asymmetric: placing a node in our link-set only indicates that we have heard the other node, not that we may be able to communicate with them. Additional information contained within the HELLO message is the sending node's entire 1-hop neighbor list. Symmetric neighbors are confirmed when node *a* sees its address in the neighbor list of node *b*'s HELLO message. A given node will collect all of its 1-hop links/neighbors into a repository and then broadcast that information through HELLOs. HELLO messages are never forwarded.

Data: HELLO message H
add H.sender to 1-hop neighbor set;
for $x \in H.neighbor\_set$ **do**
  add x to 2-hop neighbor set;
end
if *Either 1 or 2-hop neighbor set changes* **then**
  recalculate MPR set;
  recalculate routing tables;
end

Algorithm 1: Node behavior having received a HELLO message.

OLSR also maintains a 2-hop neighbor set. The 2-hop neighbor set can be determined by taking the union of the neighbor sets of all 1-hop neighbors. Note that a node cannot be its own 2-hop neighbor, nor may any 1-hop neighbors be in the 2-hop neighbor set. See Algorithm 1.

A MultiPoint Relay (MPR) is an OLSR node that is chosen to retransmit all broadcast messages from one of its 1-hop neighbors. By not requiring all nodes to retransmit broadcast data some power savings is achieved. A node's MPR set must be chosen such that all 2-hop neighbors are covered. A node also maintains the MPR selector set: a list of neighbors that have chosen it as a MPR. This is determined through additional information gathered from HELLO messages of 1-hop neighbors. See Algorithm 2.

Topology Control (TC) messages augment the local information gathered via periodic exchange of HELLO messages, i.e., 1 or 2-hop neighbors and MPRs. At a bare minimum, a node selected as a MPR must communicate to the network the various nodes which have selected it as a MPR. By broadcasting this information, network topology is established. TC messages are flooded to all nodes in the network and may be retransmitted by other MPRs. Each TC message includes an

Data: 1-hop neighbors $n1$, 2-hop neighbors $n2$
if $\exists$ *node* x *in* $n1$ *providing only path to* $n2$ **then**
  add x to MPR set;
end
Remove nodes in $n2$ now covered by MPR set;
while $\exists$ x *in* $n2$ *not covered by MPR set* **do**
  for $y \in n1$ **do**
    calculate reachability increase of adding y to MPR set;
  end
  add node $z \in n1$ with highest reachability;
  Remove nodes in $n2$ now covered by MPR set;
end

Algorithm 2: Recalculating MPR set.

Advertised Neighbor Sequence Number (ANSN), incremented when a change in its advertised neighbor set is detected, in order to determine which information is newer. Topology Tuples (T_dest_addr, T_last_addr, T_seq, T_time) are stored on each node. T_dest_addr is the main address of a node reachable through T_last_addr. T_seq is the sequence number associated with this entry and T_time is the time at which this information expires and should be removed. See Algorithm 3.

Data: Topology Control message T
if *T.sender* $\notin$ *node 1-hop neighbor list* **then**
  discard T;
end
if $\exists$ $x \in$ *topology set where T_last_addr == originator AND T_seq* $>$ *ANSN* **then**
  discard T;
end
for t *in topology set* **do**
  if *T_last_addr == originator AND T_seq* $<$ *ANSN* **then**
    remove t from topology set;
  end
end
for t *in T.top_set* **do**
  add or update t to topology set;
end

Algorithm 3: Node behavior having received a TC message.

Routing tables are built using the topology information. Should any piece of information be found stale among the link set, 1-hop or 2-hop neighbor set, or the topology set on a given node, the routing table will need to be updated. The shortest path algorithm is run to find routes between node *x* and all other nodes. Updating routing tables does not generate any traffic; it is a local construction only. Upon receiving a packet from a node in our MPR selector set, that packet must be forwarded. See Algorithm 4.

We also added a notion of *Situational Awareness* (SA) to our model. SA messages simply relay the coordinates of each node to the regional SA master node as defined

by our models. This may require routing to the SA master node as necessary. SA masters, in turn, relay the aggregated coordinates of its regional partners to a high-latency satellite link. These messages are, in fact, the only application-level (i.e. non-OLSR) messages carried by our network.

Data: Packet to forward P
if *P.sender* $\notin$ *node 1-hop neighbor list* **then**
  discard P;
end
if *P has already been forwarded* **then**
  discard P;
end
if *P.sender* $\in$ *MPR selector set* **then**
  forward P;
end

Algorithm 4: Default forwarding algorithm.

B. Simulation Engine — ROSS

The simulator we used was ROSS [11]. ROSS is a Time Warp simulator capable of reverse computation for rolling back erroneous[1] events. However, due to the complexities of this model as well as time constraints we were unable to complete the reverse event handler. Attempts were made to use `memcpy()` to perform state-saving on node-state akin to classic Time Warp. This caused our model to slow down immensely, mainly due to the rather large information repositories stored in each node. We do not store all OLSR information repositories, just a subset that we support including Neighbor Set, 2-Hop Neighbor Set, MPR Set, MPR Selector Set, Topology Information Base, and the Duplicate Set. See Figure 1 for an example. Therefore, we ran our simulation in conservative mode only. It is believed that once a correct reverse event handler is constructed, we can push our performance even further.

Data: Processing Element PE
while *true* **do**
  determine GVT;
  PriorityQ PQ = $PE.events$;
  if $GVT > simulation.end$ **then**
    break;
  end
  for x = *PQ.next* **do**
    process x;
  end
end

Algorithm 5: Conservative scheduling algorithm which runs on all Processing Elements (PEs).

Conservative simulations (see algorithm 5) must maintain the Local Causality Constraint, i.e. events on a given processing element (PE) are only processed in non-decreasing time stamp order. Additionally, conservative mode performance is tightly coupled with a notion of *lookahead*: the smallest time stamp delta that may be safely scheduled. While lookahead values may change throughout the course of a given simulation, our chosen lookahead never varies. Zero lookahead can be devastating to performance — no apparent progress can be made! Unfortunately for wireless simulations, lookahead is more or less determined by the speed of light, yielding a near-zero value. Instead, a small, non-zero constant was selected to counter this and a minimal (randomly-generated) delta was added to ensure no time stamp ties.

[1]Events that should never have been scheduled including all descendants.

```
for (i = 0; i < h->num_neighbors; i++) {
   if (s->local_address == h->neigh_addrs[i]) {
      // We are not our own 2-hop neighbor!
      continue;
   }
   // If h->neigh_addrs[i] is in our list already
   in = 0;
   for (j = 0; j < s->num_two_hop; j++) {
      if (s->twoHopSet[j].neighMainAddr ==
          m->originator &&
          s->twoHopSet[j].twoHopNeighAddr ==
          h->neighbor_addrs[i]) {
         in = 1;
      }
   }
   if (!in) {
      s->twoHopSet[s->num_two_hop].neighMainAddr =
        m->originator;
      s->twoHopSet[s->num_two_hop].twoHopNeighAddr =
        h->neigh_addrs[i];
      s->num_two_hop++;
   }
}
```

Fig. 1. This is the actual code from our model to recalculate the 2-hop neighbor set. This is the simplest (excluding 1-hop neighbors) information repository update. s is the node_state, m is the OLSR message, and h is the HELLO message embedded within m.

C. Model Optimizations and Assumptions

Our model was developed to mimic ns-3 as closely as possible. In fact, much of the ROSS model code was ported more or less directly from the ns-3 OLSR model including (but not limited to) many of the data structures, routing table computation, and the default forwarding algorithm. This was done to minimize performance differences due to implementation differences. Every attempt was made to translate the C++ code from ns-3 into corresponding C code for ROSS.

However, certain simplifying assumptions were made which differ from the ns-3 model. First we assume that all links are strictly symmetric. In other words, if node *a* can hear node *b* then node *b* can hear node *a*. This assumption actually removes the notion of a link and upgrades all asymmetric links with whom a HELLO message has been heard to symmetric links or *neighbors*. We expect minimal impact from this assumption as we are using a free space wireless communication model — any node that we hear will hear us given radio ranges and power are the same. We do not model any obstructions or terrain in this study. Lastly, we assume that each node contains only a single interface. The RFC states that, "A single OLSR interface node MUST use the address of its only OLSR

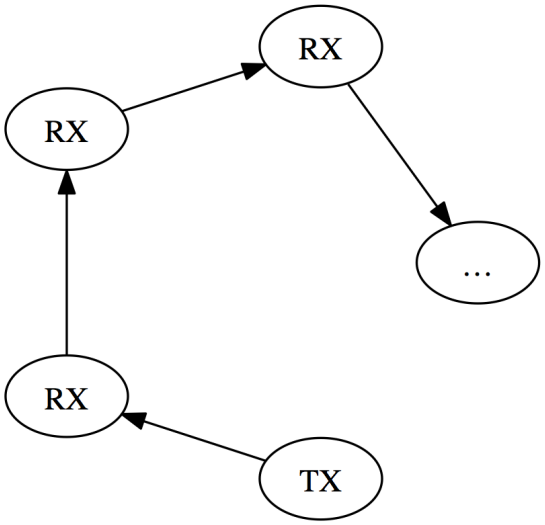

Fig. 2. A novel broadcast approach to minimize the event population. Every node is responsible for relaying a message to the next node.

interface as the main address." This frees us from having to implement Multiple Interface Declaration (MID) messages as well as Host and Network Association (HNA) messages.

Multiple communication options were implemented: a unit circle model as well as the Friis equation [12]. Identical frequencies ($\sim$5 GHz) were used in both the ns-3 implementation and the ROSS implementation. It is worth noting that regardless which communication method was chosen, we were never out of range or had too little power for successful transmissions within our 100 m $\times$ 100 m grid. Therefore no packets were ever lost.

One optimization we made that is worthy of mention is our approach to wireless communication. As opposed to one "sender" node communicating simultaneously with up to n "receiver" nodes, we instead choose to send a single message with a tiny epsilon timestamp increment to the "first" member of the region. That node will process the message and in turn send *an identical copy of the message* to the next member of the region until all regional nodes have received it. In other words, from the perspective of all receiving nodes, they have effectively just received a broadcast message This is similar to a token ring network broadcast (see Figure 2). While slightly awkward, this optimization allows us to avoid the explosition in event population and associated demands placed on the memory subsystem that may occur due to wireless broadcasting.

Finally, we introduced the notion of *regions*, a small well-defined area over which the nodes could communicate and move. A region is intended to encapsulate a small military troop unit or detachment in the field. (Note that node movement is not required for all of our experiments).

Event Type	Event Count
HELLO_RX	972,368,126
HELLO_TX	60,817,997
TC_RX	613,491,662
TC_TX	24,641,680
SA_RX	10,809,707
SA_TX	11,534,336
SA_MASTER_TX	65,536
SA_MASTER_RX	1,274,695
RWALK_CHANGE	11,059,830

TABLE I
EVENT COUNTS FOR THE VARIOUS TYPES OF EVENTS IN OUR SIMULATION OVER 120 SECONDS.

III. RESULTS

A. Experimental Setup

Our experiments primarily deal with effects from the protocol itself, i.e. very little data was actually routed over the simulated networks via OLSR.

Harold, one of the ARL's supercomputers, is an SGI Altix ICE 8200. The compute nodes consist of dual Intel Xeon Nehalem-EP quad-core processors running at 2.8 GHz with 24 GBytes of DDR3 memory paired with a 4x DDR InfiniBand network. The model was compiled using the Intel compiler and the MPI implementation was MPICH2.

The Computational Center for Nanotechnology Innovations (CCNI) Blue Gene/L is a 700 MHz PowerPC 440 with two cores per node. It is a 32-bit machine with 1024 MB of DDR memory split per node. While slower than Harold, it has a better-paired network at 175 MBps in either direction, as well as a fast (1.5 us latency) global collective and global barrier network. The model was compiled using the IBM XLC compiler and the MPI implementation was MPICH2.

Last, the machine we developed our OLSR model on was a 24-way SMP machine which we will refer to as the *baseline* configuration. It has 12 2.667 GHz hyper-threaded CPUs. As opposed to networks in the other two configurations, our baseline machine uses shared memory. The model was compiled using gcc 4.5 and the MPI implementation was again MPICH2.

B. ROSS Results

We ran all simulations for 120 seconds of simulated time. In this model we used a *Logical Process* (LP) to simulate a single node. The number of nodes in a given region was held constant at 16 although we did vary the number of regions a single *Processing Element* (PE) could contain. The number of LPs per PE was always a power of two. Additionally, the nodes had mobility within their region but inter-region movement was prohibited. Depending on the model configuration, nodes may follow a random walk mobility pattern where their course was changed every 1–20 seconds along a uniform random distribution. Node starting locations were chosen randomly in a 100 m $\times$ 100 m grid. See Figure 3 for an example.

Regardless of the core count, both supercomputers simulated 1,048,576 distinct nodes. Dividing this by the number of

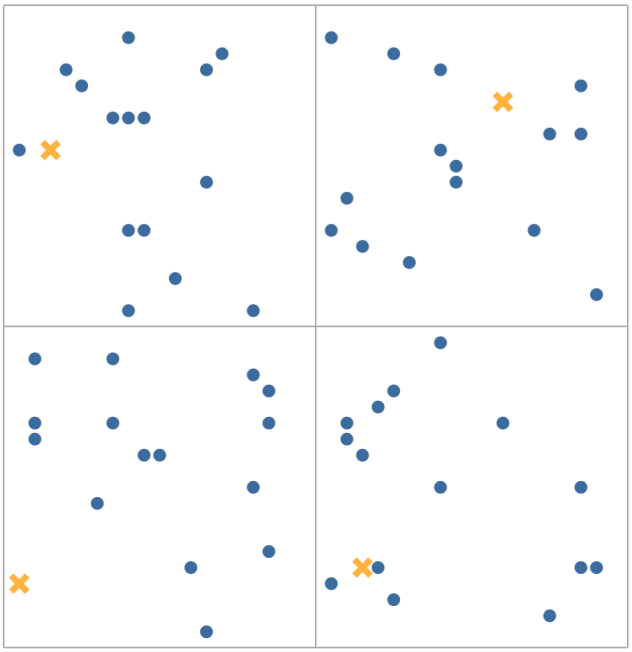

Fig. 3. Random node placement. Regular nodes are dots while regional SA masters are labeled with an X. Only four regions are displayed.

nodes in a region (16) yields 65,536 individual regions for us to distribute amongst our cores. On our baseline machine, we simulated 131,072 distinct nodes leaving us 8,192 regions. One additional LP was added per region to improve performance of SA aggregation, bringing our total number of LPs to 1,114,112 and 139,264, respectively. Our initial approach consisted of sending the aggregated SA information from the SA masters to a single node to act as a satellite uplink. Naturally, this did not scale well as that node then became a bottleneck of the entire network. Adding an SA aggregator to each region eliminated this problem. See Table I for a typical event breakdown during a run with mobility.

Figure 4 shows how many seconds it takes for our model to simulate 131,072 LPs for 120 seconds on 1, 2, 4, and 8 cores. A single core running our model requires 3,000 seconds for no mobility and approximately 7,500 seconds including mobility. The event rates, shown in Figure 7, show the baseline machine clocking in at 70,000 events per second for no mobility and 30,000 for the scenario including mobility one processor. The baseline machine was mainly used for development and debugging of our model.

Figures 5 and 6 shows the wall-clock time for our simulation on ARL's Harold and the CCNI Blue Gene/L, respectively. Both stationary nodes and random walk models were tested. Here the Blue Gene/L begins to show its age — using just 64 cores, nearly 8,000 seconds of compute time were required to simulate 120 seconds of OLSR and situational awareness chatter in the random walk scenario. Again, no other data beyond this was routed on this network. The same scenario on Harold took less than 1,000 seconds. The most cores we ran either simulation on was 2,048. The Blue Gene/L ran for just under 500 seconds in the random walk scenario while

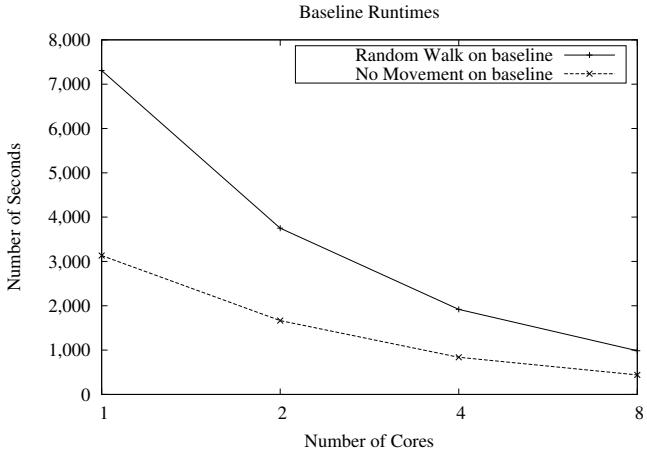

Fig. 4. Wall-clock time for the two configurations on baseline.

Harold took approximately 120 seconds.

Figures 8 and 9 show the event-rates (of all events) obtained on the BG/L and Harold, respectively as a function of the configuration and core count. Notice again that Harold outperforms the Blue Gene/L by a wide margin. However, Figure 8 shows an interesting situation nearly imperceivable on Figure 5: a *decrease* in the event rate as we increase the number of cores. This is the point at which we exhaust the available work to be done in between GVT computations.

This data clearly indicates that we could push more messages into the network and sustain our event rates. However, Situational Awareness messages are actual routed data. As shown in [13], OLSR control traffic remains largely constant and independent of traffic patterns.

These numbers make sense when we consider that very little inter-processor communication occurs in this model. All HELLO, topology control, routing, and a majority of situational awareness communications are restricted to a single region. Recall that inter-region node movement was prohibited as well.

Additionally, there appear to be some "knees" in the plot without mobility in Figure 8. These are likely due to caching effects of our model on Harold. The L3 cache size on each processor in Harold is 8 MBytes and there are two processors per node. Isolated, 8 MBytes of cache will not have a large impact. Combined with potentially hundreds or thousands of cores, we may find ourselves running nearly exclusively within cache which enables events to be processed at a much higher rate.

IV. Related Work

Clausen [13] compared OLSR to both Ad hoc On-Demand Distance Vector (AODV) routing as well as Dynamic Source Routing (DSR) using ns-2 [14]. Being reactive protocols, both AODV and DSR outperformed OLSR in high-mobility scenarios although OLSR fared better while supporting many simultaneous streams. Adjih et al. [15] compared OLSR to OSPF [2], a non-ad-hoc link state routing protocol, and found it to be wanting in many respects such as overhead and

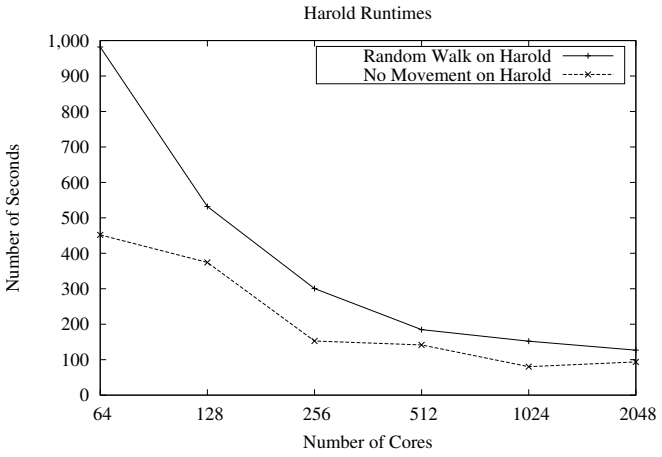

Fig. 5. Wall-clock time for the two configurations on Harold.

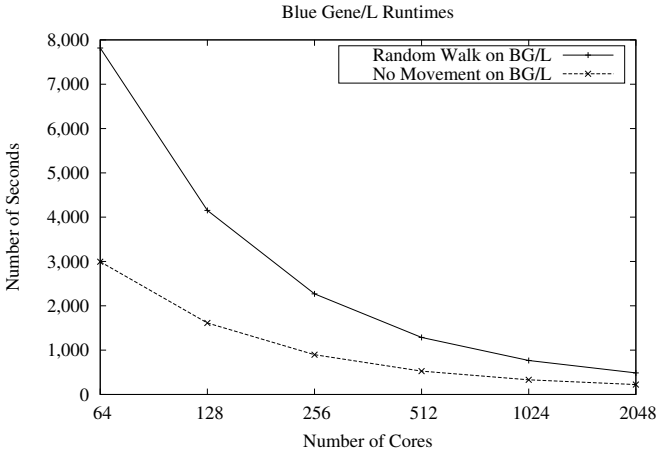

Fig. 6. Wall-clock time for the two configurations on BG/L.

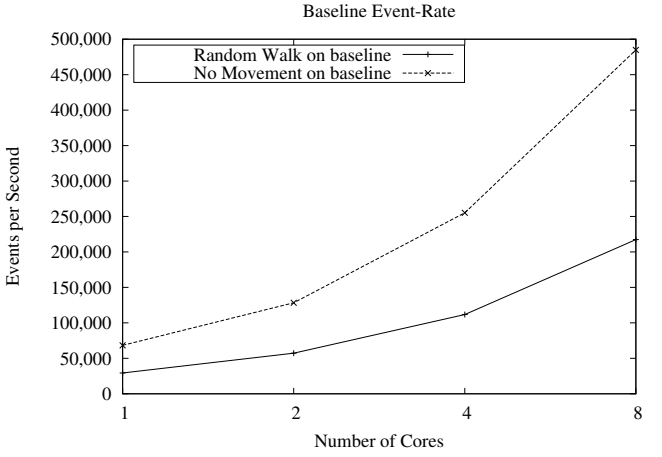

Fig. 7. Event rate for the two configurations on baseline.

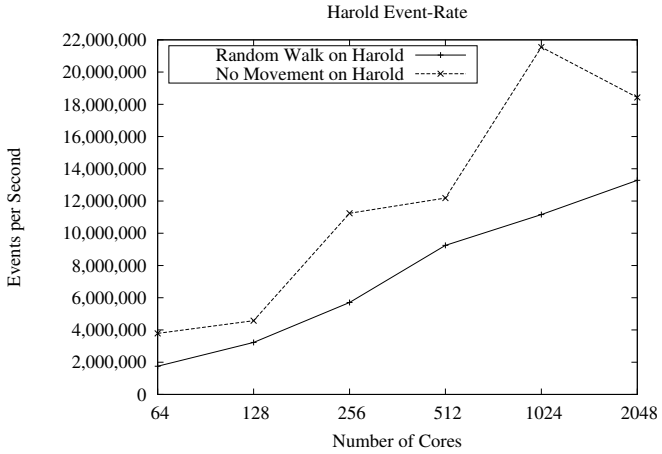

Fig. 8. Event rate for the two configurations on Harold.

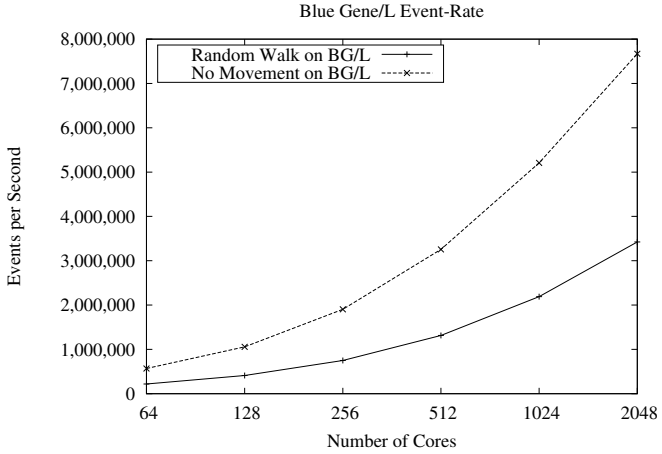

Fig. 9. Event rate for the two configurations on BG/L.

adaptability when compared to OLSR in a MANET environment. Plesse [16] et al. found that Quality of Service (QoS) would increase performance of OLSR, especially in military contexts while Ge et al. [17] determined that QoS could aid in finding optimal paths while maintaining bandwidth constraints. Nguyen and Minet [18] chose to pursue QoS by augmenting MPRs with QoS MPRs in order to build routes with QoS properties. Benzaid et al. [19] developed Fast-OLSR for fast-moving nodes wishing to use OLSR using reasonable bandwidth and control traffic tuned for mobile scenarios. Haerri et al. [20] investigated Vehicular MANETs or VANETs. While comparing AODV and OLSR, OLSR outperforms AODV in all but packet delivery ratio though the performance loss was limited to 10%.

Perumalla [21] covers both traditional and modern techniques in his survey on parallel and distributed approaches to simulation. Ji et al. [22] discuss multiple avenues for optimizing performance such as extracting greater lookaheads from the model and minimizing LP communications. Zeng et al. [23] introduce the GloMoSim library, a modular plug-and-play approach to building wireless network models from various pre-built components. Bauer [5] demonstrated ROSS

running the PHOLD benchmark on both Blue Gene/L and P in optimistic mode on up to 131,072 cores. At the other end of the spectrum, Carothers et al. [24] simulated a PCS network using distributed Time Warp on a cluster of workstations. This setup was a predecessor to ROSS running on supercomputers and many similarities remain. Our model was fairly CPU-intensive and little communication took place. Like the results from Nicol's YAWNS protocol [10], we as well have very local state changes which allow for a high degree of parallelism

While we chose to use the Friis equation [12] due to its simplicity and to mimic ns-3 [25] as much as possible, we could have chosen a more realistic radio propagation technique. For example, Bauer and Carothers [5] show that Transmission Line Matrix (TLM) can be implemented in a Time Warp simulator while maintaining a high degree of accuracy and resolution. TLM models light as a wave while capturing first-order effects such as diffraction, reflection, and scattering. This work was built on Nutaro [26] which describes a Finite Difference Time Domain (FDTD) interpretation of wave propagation. Improving upon TLM, Nutaro et al. [27] developed a less computationally-intense approach to determining path-loss while Seal and Perumalla [28] applied Nutaro's newer work to a Time Warp context and, in particular, formulated reverse computation rollbacks and demonstrates the highest model performance to date.

V. Conclusions

Modeling and simulation are necessary to determine the viability and scale of networking protocols before their widespread adoption for a particular purpose. While ns-3 excels at the micro-level, ROSS on the other hand excels at the macro-level. Through our simulation on multiple HPC platforms, we have demonstrated the ability to take advantage of various hardware offerings.

Development and debugging of our model was conducted on the baseline system and no porting (other than recompiling) was necessary to run ROSS on the HPC equipment. Although the Blue Gene/L has the potential to keep pace amongst various models, the compute-intensive nature of the OLSR simulation favored Harold with its much faster clock speeds.

The normalized speedup achieved was between 250 for the scenario without mobility and $\sim$375 for the random walk scenario.

In the future, we feel an obvious course of action would be to develop an optimistic model for OLSR. Optimistic models have the ability to exploit non-obvious parallel lines of execution embedded within the model.

In the field, power often becomes a major concern, yet this simulation did not address this issue at all. While there is an abundance of literature concerning power issues in sensor networks, relatively little has been published in non-sensor network studies.

Additionally, while we ourselves have mentioned alternate methods for wireless propagation, in the end we employed the simple Friis equation. While serving the community well for many years, many investigations require finer fidelity.

References

[1] Thomas H Clausen and Philippe Jacquet, "Optimized link state routing protocol (olsr)," Tech. Rep., 2003.

[2] J Moy, "Rfc 2328: Ospf version 2," Tech. Rep., 1998.

[3] K. Renard, C. Peri, and J. Clarke, "A performance and scalability evaluation of the ns-3 distributed scheduler," in *2012 NS-3 Workshop (WNS3)*, March 2012.

[4] David Bauer Jr, Christopher Carothers, and Akintayo Holder, "Scalable time warp on blue gene supercomputers," 2009, pp. 35–44.

[5] D.W. Bauer and C.D. Carothers, "Scalable rf propagation modeling on the ibm blue gene/l and cray xt5 supercomputers," in *Simulation Conference (WSC), Proceedings of the 2009 Winter*, dec. 2009, pp. 779–787.

[6] David R Jefferson, "Virtual time," *ACM Trans. Program. Lang. Syst.*, vol. 7, no. 3, pp. 404–425, 1985.

[7] KM Chandy and J Misra, "Distributed simulation: A case study in design and verification of distributed programs," *Software Engineering, IEEE Transactions on*, vol. SE-5, no. 5, pp. 440 – 452, Sep 1979.

[8] R E Bryant, "Simulation of packet communication architecture computer systems," Tech. Rep., 1977.

[9] K M Chandy and J Misra, "Asynchronous distributed simulation via a sequence of parallel computations," *Commun. ACM*, vol. 24, no. 4, pp. 198–206, 1981.

[10] David Nicol, "The cost of conservative synchronization in parallel discrete event simulations," *J. ACM*, vol. 40, no. 2, pp. 304–333, 1993.

[11] Christopher Carothers, David Bauer Jr, and Shawn Pearce, "Ross: A high-performance, low-memory, modular time warp system," *Journal of Parallel and Distributed Computing*, vol. 62, no. 11, pp. 1648 – 1669, 2002.

[12] HT Friis, "A note on a simple transmission formula," *Proceedings of the IRE*, vol. 34, no. 5, pp. 254 – 256, May 1946.

[13] T Clausen, "Comparative study of routing protocols for mobile ad-hoc networks," *hal.inria.fr*, Jan 2004.

[14] L Breslau, D Estrin, K Fall, S Floyd, J Heidemann, A Helmy, P Huang, S McCanne, K Varadhan, Ya Xu, and Haobo Yu, "Advances in network simulation," *Computer*, vol. 33, no. 5, pp. 59 –67, May 2000.

[15] C Adjih, E Baccelli, and P Jacquet, "Link state routing in wireless ad-hoc networks," Oct 2003, vol. 2, pp. 1274 – 1279 Vol.2.

[16] T Plesse, C Adjih, P Minet, A Laouiti, A Plakoo, M Badel, P Muhlethaler, P Jacquet, and J Lecomte, "Olsr performance measurement in a military mobile ad hoc network," *Ad Hoc Networks*, vol. 3, no. 5, pp. 575–588, 2005.

[17] Y Ge, T Kunz, and L Lamont, "Quality of service routing in ad-hoc networks using olsr," *System Sciences*, Jan 2003.

[18] D Nguyen and P Minet, "Analysis of mpr selection in the olsr protocol," *Advanced Information Networking and ...*, Jan 2007.

[19] M Benzaid, P Minet, and K Al Agha, "Integrating fast mobility in the olsr routing protocol," *Mobile and Wireless ...*, Jan 2002.

[20] J Haerri, F Filali, and C Bonnet, "Performance comparison of aodv and olsr in vanets urban environments under realistic mobility patterns," *Proceedings of the 5th IFIP Mediterranean ...*, Jan 2006.

[21] Kalyan S Perumalla, "Parallel and distributed simulation: traditional techniques and recent advances," 2006, pp. 84–95.

[22] Zhengrong Ji, Junlan Zhou, Mineo Takai, Jay Martin, and Rajive Bagrodia, "Optimizing parallel execution of detailed wireless network simulation," 2004, pp. 162–169.

[23] X Zeng, Rajive Bagrodia, and M Gerla, "Glomosim: a library for parallel simulation of large-scale wireless networks," May 1998, pp. 154 –161.

[24] Christopher Carothers, RM Fujimoto, Y-B Lin, and P England, "Distributed simulation of large-scale pcs networks," Jan 1994, pp. 2 –6.

[25] Thomas R Henderson, Sumit Roy, Sally Floyd, and George F Riley, "ns-3 project goals," 2006.

[26] James Nutaro, "A discrete event method for wave simulation," *Transactions on Modeling and Computer Simulation (TOMACS*, vol. 16, no. 2, Apr 2006.

[27] J Nutaro, PT Kuruganti, R Jammalamadaka, T Tinoco, and V Protopopescu, "An event driven, simplified tlm method for predicting path-loss in cluttered environments," *Antennas and Propagation, IEEE Transactions on*, vol. 56, no. 1, pp. 189 –198, Jan 2008.

[28] Sudip K Seal and Kalyan S Perumalla, "Reversible parallel discrete event formulation of a tlm-based radio signal propagation model," *ACM Trans. Model. Comput. Simul.*, vol. 22, no. 1, pp. 4:1–4:23, 2011.