Performance of Distributed *ns-3* Network Simulator

Sergei Nikolaev,* Peter D. Barnes, Jr., James M. Brase, Thomas W. Canales
David R. Jefferson, Steve Smith, Ron A. Soltz, Peter J. Scheibel
Lawrence Livermore National Laboratory
L-210, 7000 East Ave.,
Livermore, CA
nikolaev2@llnl.gov

ABSTRACT

This paper presents the results of parallel simulations of mixed networks using the *ns-3* simulator. The simulated networks consist of equal number of simple (leaf) nodes and router nodes, which form a small-world network. The active channels in the simulations link pairs of nodes with symmetric source and sink applications. All network traffic is restricted to a single-hop communications. The network partitioning among MPI ranks is accomplished using METIS library.

The performance and scalability of the *ns-3* network simulator is examined using a variety of metrics, *e.g.* memory footprint, packet transmission rate, and runtime statistics. Empirical relations are derived for the memory scaling. Further avenues are identified for improvement of the parallel *ns-3* simulator.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*network communications, network topology*; C.2.2 [**Computer-Communication Networks**]: Network Protocols—*applications, routing protocols*; D.2.8 [**Software Engineering**]: Metrics—*performance measures*

General Terms

Algorithms, Experimentation, Measurement, Performance

Keywords

High performance computing, MPI, Network simulation, ns-3, Parallel architecture, Performance

1. INTRODUCTION

ns-3 is a discrete-event network simulation framework designed to model realistic computer networks [6]. The simulator compares favorably with other existing network simulation tools [9]. Recently, it has incorporated a parallel scheduler [7], giving *ns-3* the capability to run distributed network simulations using MPI. Performance of the single-threaded *ns-3* network simulator was evaluated in [9] (and references therein). Here we examine the performance metrics for the parallel version of the simulator.

The parallel scheduler in *ns-3* is a straightforward global conservative scheduler, using MPI for communication between ns3 processes (ranks). The implementation allows the topology to be partitioned at simulated point-to-point (P2P) channels only; CSMA and wireless links may not cross rank boundaries. At the same time, each rank must have the full topology available.

The use of remote P2P links is handled by the `PointToPointHelper`, which detects that a P2P link crosses ranks, attaches a `PointToPointRemoteChannel` and an `MpiReceiver` object to the `PointToPointNetDevice` on each side of the link. To transmit a packet, the packet is serialized over the MPI link along with the receive time and destination node index and device index. (The destination node here is the other end of the P2P link, not the ultimate packet destination.) On the receiving side, the same data is deserialized and used to schedule the receive event for the destination node.

At initialization, each rank independently walks the topology to determine the smallest cross-rank P2P channel delay; this becomes the scheduler look-ahead, LA.

When a rank has no more events within the LA time, it enters a synchronization phase. It is guaranteed that all ranks will eventually exhaust their executable events and enter the synchronization phase. During this phase, the rank receives any outstanding messages, which will all be for events beyond the LA time. All ranks exchange messages containing the number of transmitted and received packets, and the time stamp of the next available event, using *MPI_ALLGATHER*. To check for transient (not yet delivered) MPI messages, each rank computes the total number of received and transmitted events. If these are unequal, indicating undelivered messages still exist, then the synchronization phase restarts. Once all messages have been received (events received equals events transmitted) each rank computes the global minimum next event time stamp (lower bound time stamp, LBTS), and adds the look-ahead, to obtain the maximum time stamp which is safe to process, called the *granted time*. The scheduler then proceeds

*Corresponding author.

Simutools 2013, March 05-07, Cannes, France

DOI 10.4108/icst.simutools.2013.251729

to process any events scheduled before the end of granted time. Because all ranks compute the same look ahead, the simulation effectively runs time-synchronous, with each rank processing events in the same look-ahead-sized virtual time window.

In addition to the restrictions noted above, the current implementation only transmits the receive time, the node index and device index, and any packet data. Packet and byte tags are not transmitted, nor is packet metadata, which is used to interpret the packet contents correctly.

In [3], we outlined a standard benchmark model for evaluating the performance of the distributed *ns-3* simulator. In this paper, we present the results of distributed network simulations, developed under Livermore Computer Network Simulation Program. [4] We focus specifically on very large models (10^{4-6} nodes) with a large amount of available parallelism, and want to benchmark highly distributed implementations running on up to 10^3 computing cores. The scaling relations are derived in terms of both memory footprint and performance metrics. After reviewing the performance metrics, we demarcate the limits of *ns-3* parallel network simulation on distributed architectures and suggest potential avenues for its further improvement.

2. SIMULATED NETWORKS

The network topology and the choice of parameters for the benchmark models are motivated by the considerations in previous work. [3] In particular, to isolate the true performance metrics from routing and congestion effects, we consider network traffic between nearest neighbor nodes only (*i.e.* there is effectively no routing, as no packets are retransmitted from the receiving interface on to sending interface on the same router).

The simulated networks consist of equal number of simple (leaf) nodes and router nodes (Figure 1). The router nodes

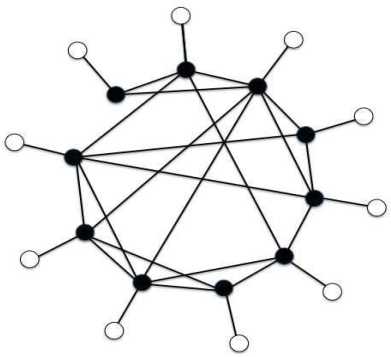

Figure 1: An example network topology used in the simulations, employing equal numbers of router nodes (filled circles) and simple nodes (open circles). The router nodes form a small-world network.

form a small-world network, generated by a Watts-Strogatz

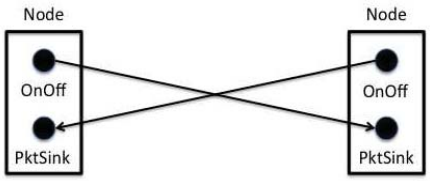

Figure 2: Connectivity diagram for application pairs residing on active channels.

algorithm with average node degree $k = 4$ and rewiring probability $\beta = 0.5$. The small-world network topology for router nodes was chosen to approximate a realistic internet backbone router network. In addition, each router in the simulation is connected to a single leaf node. The router-router connections are point-to-point (P2P) links, while router-leaf connections are CSMA. For the total number of nodes N in the network, the number of the P2P links is $Nk/4$, and the number of CSMA links is $N/2$. The total number of available channels in the network is thus $L_{\max} = N/2(1+k/2) = 3N/2$. The number of active (i.e. traffic-exchanging) channels is $L < L_{\max}$.

To simulate network traffic, we install cross-wired pairs of applications on randomly selected nodes in the network (see Figure 2). The applications can reside on either leaf or router nodes, with the only constraint that the nodes are directly connected. This 'single-hop' traffic constraint removes contaminating effects of queueing delays and limited model network bandwidth on the performance metrics. Each pair of applications includes the source (`OnOffApplication`) and sink (`PacketSink`), both of which are taken from the standard set of *ns-3* applications. The cross-wired connections result in channels that are symmetric and bi-directional.

Every `OnOffApplication` in the network generates packet traffic at the same rate of 5 Kbps (equal duration for On and Off intervals). The traffic is modeled by TCP/IP packets (`TcpSocketFacory`), with the packet size of 500 bytes.

All channels in the simulation, regardless of type (CSMA or P2P), have the same data rate of 100 Mbps. The link delay on each channel is a random parameter, uniformly sampled from the interval [2, 12] ms. Nix vector routing is used for the routing protocol.

All simulations run for 200 seconds, virtual time. To study potential delays due to route discovery, the application startup times are staggered, with traffic channels activating at random times. The channel activation times (`OnOffApplication` start times) are uniformly distributed within the initial 100 simulation seconds. All `PacketSink` applications start at simulation time zero.

For diagnostic purposes, and to monitor memory usage by the code (see §3.2), we used the *ns-3* trace system to output messages about the current RSS usage every 5 simulation seconds. The very limited output (a few kilobytes at most)

was limited to rank 0 process only and did not strain the I/O system.

Rather than hard-coding the network topology and channel parameters in a C++ script, the conventional *ns-3* way, the network topology and channel parameters for the simulations are specified by an XML input file. One proposed specification for the *ns-3* XML format is described in [8]. In this work, we used our own XML spec, described in [2], to facilitate distributed simulation runs. The partitioning of the network nodes among the MPI cores is accomplished by METIS library. [5]

Partitioning random Watts-Strogatz graphs with METIS is straightforward. For large graphs ($N > 10^3$) partitioning with 1% of the nodes per rank results in rank degree $\sim N/10$, *i.e.* ranks communicate with 10% of all other ranks. This partitioning avoids all-to-all communication. On the other hand, the partitioning results in caterpillar graphs on each MPI rank, with half the links pointing to other ranks. Therefore half of all packets will be transmitted between ranks over MPI, providing a good test of the parallel implementation.

2.1 Distributed Simulations

The simulations are performed on a commodity cluster with 1296 total nodes (20736 cores) and 431.3 TFLOP/s theoretical system peak performance. The CPUs are 2.6 GHz Intel Xeon E5-2670 with InfiniBand QDR interconnect. Each 16-core node has 32GB shared RAM. The operating system on the machine is TOSS 2.0, based on CentOS 6/RHEL 6 x86_64. [1]

To compile the ns-3 simulator (ver 3.13), we used *openmpi-gnu-1.4.3* and *gcc-4.4.6*. The disk I/O used Lustre Parallel File System with 7GB/s bandwidth and 756TB capacity. The limited disk I/O by the simulation (see above) did not affect the bandwidth. All simulations were performed in batch mode (dedicated node allocation). The nominal runtime limit for batch jobs is 16 hours (wallclock time).

For a comprehensive study of the performance of the distributed *ns-3* network simulator, we conducted three sets of distributed simulations, using varying number of MPI cores and network sizes:

- Set A. Network size N ranging from 10K nodes to 100K nodes, with 10K increments. For each network size, the number of active channels is $L = \{0.5N, N, 1.5N\}$. Each combination of N and L is modeled using 2, 4, 8, 16, 32, 64, 128, 256 compute ranks. The ranks are allocated compactly, *i.e.* a 128 ranks run is allocated on $128/16 = 8$ cluster nodes. For most of these runs, the RSS (Resident Set Size) memory footprint is small enough to fit into the 2GB/core and 2 GB/rank.

- Set B, designed to study the performance for very large networks. In these runs, the network sizes are $N = \{100K, 250K, 500K, 750K\}$. For each network size, the number of active channels is $L = \{0.5N, N, 1.5N\}$. Due to large memory footprint for these networks, these runs are allocated using a single compute node per rank, running on a single core (to have entire 32GB of RAM available for each rank). The runs in this series use 2, 4, 8, 16, 32 ranks/nodes.

- Set C, designed to test the simulation performance and memory scaling for a fixed number of active channels while letting the network sizes vary. In these runs, the network sizes are $N = \{30K, 50K\}$, and the number of active channels in the modeled networks is $L = \{5K, 10K, 15K\}$. In this series of runs, we also wanted to examine the *ns-3* performance for a very large number of ranks, using 2, 4, 8, 16, 32, 64, 128, 192, 256, 384, 512, 768, 1024 ranks. As in run series A, the compact rank allocation is used for these jobs.

The allocation modes for each set of runs are chosen to accommodate the 32GB RAM limit for cluster nodes. The smaller networks (up to 100K nodes/channels) can be modeled with multiple MPI processes per node, as the RSS memory footprint is small enough to fit into RAM. However, for larger networks (100K nodes/channels and above), the total available RAM (32GB per node) becomes the limiting factor, and the simulations must allocate a limited number of ranks per node to increase the RAM available to each process. The 750K node network is the maximum possible for 32GB RAM per node. The memory scaling is described in detail in §3.2.

In addition to the three main sets of distributed simulations, we looked at a set of smaller networks ($N = \{100, 1000\}$), at different degrees of parallelization. Our primary finding with regard to these simulations is the apparent sensitivity of the simulator to the METIS network partitioning: when the number of allocated MPI ranks is within a factor of a few of the number of model network nodes N, the "optimized" network partitioning algorithm may result in some MPI ranks having zero nodes and applications. This happened, for example, for the network simulation with 100 nodes running 100 application pairs on 32 MPI ranks, which has only 3 model nodes per rank, on average, or for the simulations with 1000 nodes running 1000 application pairs on 128 and 256 ranks. For these cases, the *ns-3* simulator exits with an error. This suggests a simple round-robin paritioning scheme is more appropriate when the number of MPI ranks is comparable to the number of nodes in the network.

3. RESULTS

This section summarizes our main findings, in terms of the simulator performance (packets per unit time) and memory footprint, and derives the scaling relations.

3.1 Network Performance Metrics

The primary performance metrics for the *ns-3* simulator are the total run time and the packet reception (Rx) rate as a function of both wall-clock and simulated time.

The total run-time statistic is dominated by the time for packet traffic. The other components of the run-time (parsing XML files, setting up the simulation, *etc*) do not contribute significantly to the total run-time. The scaling for these auxiliary operations is linear with the number of nodes and active channels in the network. The traffic time metric is shown in Figure 3 as a function of the number of computing cores. Note that some of the runs in Figure 3 are missing

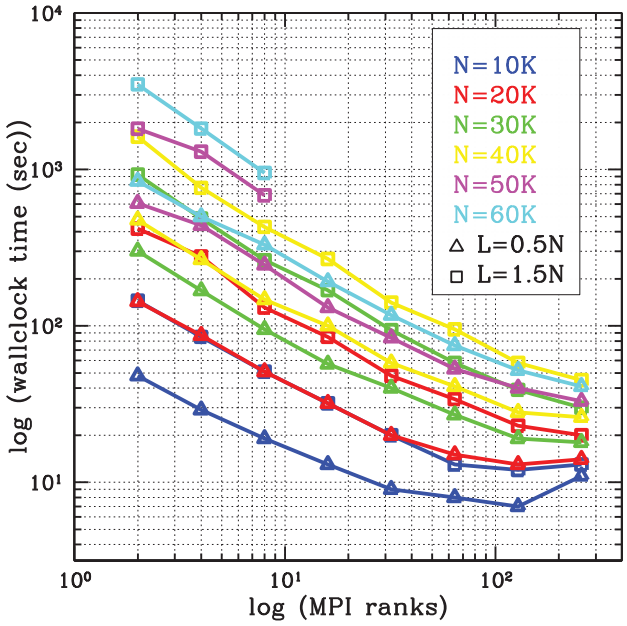

Figure 3: Total time modeling packet traffic for selected networks in set A. The color indicates the network size N, while the point type denotes the number of active channels in the network, $L = 0.5N$ (triangles) or $L = 1.5N$ (squares).

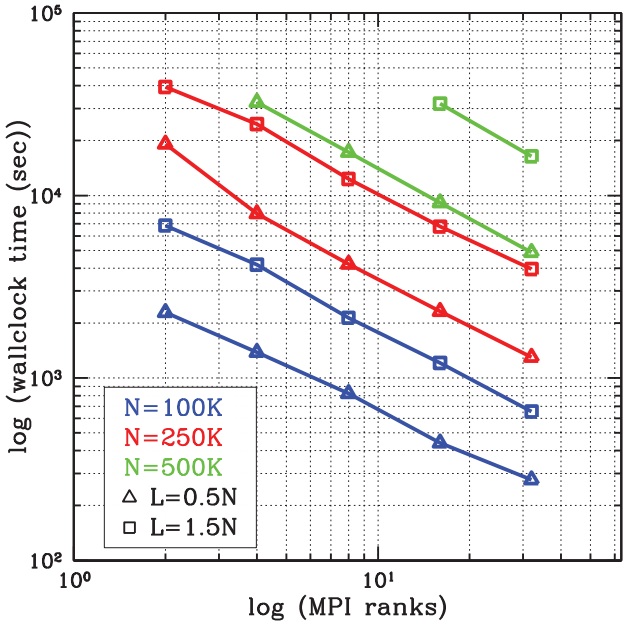

Figure 4: Total time modeling packet traffic for selected networks in set B. The point types have the same L/N ratios as in Figure 3.

(*cf.* 50K and 60K runs for number of ranks greater than 8). This is due to the RSS memory footprint of these runs exceeding 32GB available RAM. In particular, all runs is Set A for network sizes of 60K nodes and greater can only run at 2, 4, and 8 core allocations (as their memory footprint is about 4GB per rank).

The total execution time (packet traffic modeling time) is a function of not only the respective share of the workload L/C, where C is the number of ranks, but also the total number of nodes. This can be seen in Figure 3, by comparing the traffic times for the same L and different N. For example, the time to run traffic in $N = 10K$, $L = 15K$ model (upper blue curve) is well below the corresponding times for $N = 30K$, $L = 15K$ model (lower green curve).

Figure 3 suggests that larger networks scale better to large number of compute cores: the wallclock time for a 40K network is monotonically decreasing all the way to 256 cores, while the 10K network at 256 cores is showing an increase in the execution time.

The traffic times for runs in sets B and C are shown in Figures 4 and 5, respectively. For large network sizes (*e.g.* 500K nodes in Figure 4), the effect of the wallclock run time limit (16 hours) manifests itself by removing points at low number of cores: the code does not finish in 16 hours.

In run set C, the turnover in the total run time is due to increased communication burden among compute cores. The shape of the curves in Figure 5 suggests there is the most effi-

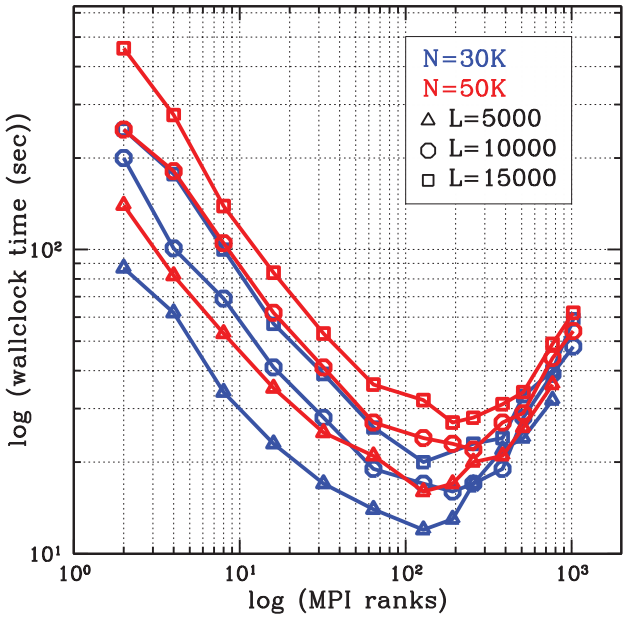

Figure 5: Total time modeling packet traffic for selected networks in set C. The type of points indicates the number of active channels in the networks: $L = 5K$ (triangles), $L = 10K$ (circles), $L = 15K$ (squares).

cient parallelization level for a fixed size network. This is not unexpected, as changing the number of ranks available for a given model size changes the communication/computation ratio of the model execution. The level is at ~ 200 cores for the networks of 30K-50K in size. The figure also hints that the parallelization level increases with the network size.

The rate of packet reception per wallclock second per rank is shown in Figure 6. The rate is measured in the steady state regime (for $t > 100$ simulation seconds), when all traffic channels have activated. The curves are very similar for a variety of N and L values, especially at low parallelization levels. The scatter in Rx rate values increases for larger number of CPUs.

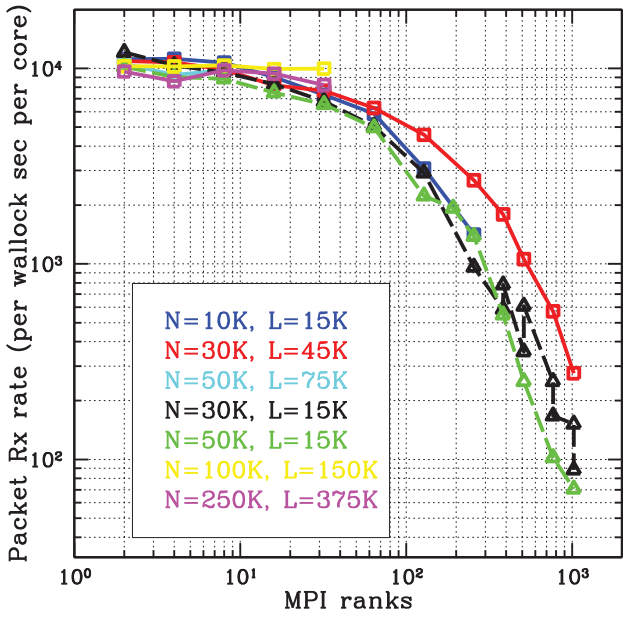

Figure 6: Packet Rx rate per process per wallclock second for selected runs.

Figure 7 shows the total packet Rx rate per wallclock second (all ranks combined). For the majority of the runs, the rate increases up to parallelization levels of $\sim 100 - 200$ ranks, where it flattens out and starts decreasing, due to inefficiencies of inter-rank synchronization and communication for large number of MPI ranks. The turning point for the larger networks (run set A) occurs at higher core count than for smaller networks (run set C). The largest networks (run set B) appear to have the best Rx rates for the number of ranks greater than 16. The turning point for these networks will occur at even larger parallelization levels.

Figure 8 shows the rate of advancement of simulated time during the run as a function of wallclock time. The performance is visibly worse (fewer simulated seconds processed per same wallclock time interval) during the first 100 seconds of the simulation, when channels activate. This is probably due to the extra time needed for route discovery for nix-vector routing algorithm.

3.2 Memory Scaling

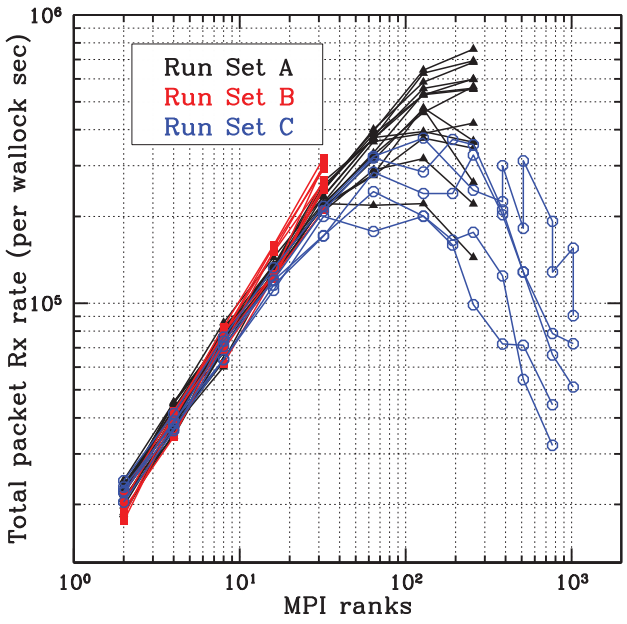

Figure 7: Total packet Rx rate per wallclock second for all runs.

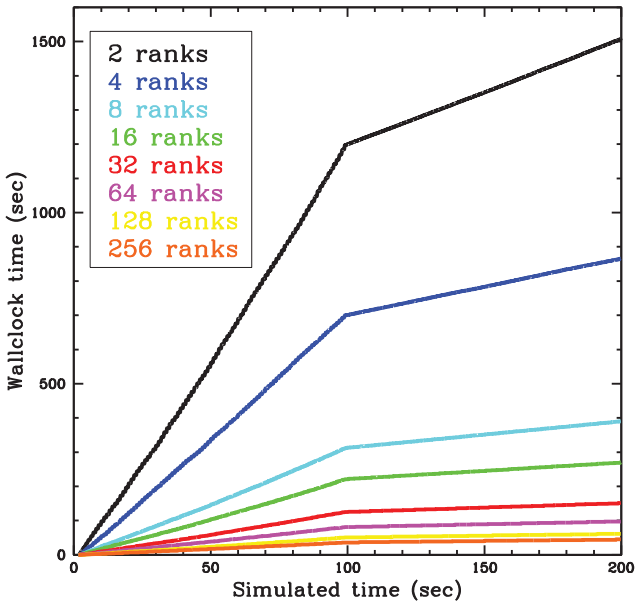

Figure 8: Simulated time vs. wallclock time for a series of N=50K, L=50K runs. Top to bottom, the curves correspond to runs from 2 to 1024 MPI cores, increasing by power of twos. The slope change around 100 simulated seconds is due to all channels becoming active.

Another performance metric we examine is the resident set size (RSS) of the model, *i.e.* including both the code and the data. The parameter was measured for each of the runs

in sets A, B, and C.

For each of the runs, the RSS was measured at 5 second intervals of simulated time, to monitor its behavior as the run progressed. Due to staggered nature of the channel activation in our simulations, the RSS is expected to grow with simulated time, at least for the first 100 seconds of simulation time, when the applications activate. (We did observe a small increase in the RSS even beyond 100 seconds simulated time, which might be indicative of a small memory leak). For consistency, as the reported metric in fits below, we recorded the RSS number at the final simulation time (200 seconds).

We've already seen that memory bounds play a significant role in determining if a particular model can be run on a given number of MPI ranks or compute nodes. In general, while the memory usage of *ns-3* is in general quite good, [9] the memory scaling of the distributed implementation is less than ideal, since each rank has to instantiate the global topology. An accurate estimate of the memory requirements would aid significantly in planning the parallel execution of a given model.

To quantify the scaling of the RSS with respect to different parameters, we fit the RSS data to a variety of models. The dependent variables we consider are the number of model nodes N, the number of active channels L, the number of ranks C, and the average application load per rank L/C. The node load for each rank is still N, as the entire network topology is reproduced at each rank. Ideally, the memory usage would scale with $N/C + L/C$, instead of $N + L/C$. The fits below also include the constant term, to represent the size of the *ns-3* code in memory.

The fit of RSS footprint using all the independent variables is as follows:

$$\begin{aligned} RSS(Kb) = \; & (71142 \pm 18420) + \\ & (24.1 \pm 0.2)N + \\ & (10.3 \pm 0.2)L - \\ & (47.0 \pm 69.0)C + \\ & (10.1 \pm 0.5)L/C, \end{aligned} \tag{1}$$
$$R^2 = 0.9995$$

The fit quality is not very good, and some of the parameter errors seem very large, suggesting that those variables have small effect, so we tried to fit with smaller parameter sets, if only to give approximate rules of thumb. (A full ANOVA analysis to reject weak dependent parameters seemed overkill.) The resulting fits are shown below:

$$\begin{aligned} RSS(Kb) = \; & (71225 \pm 46080) + \\ & (24.0 \pm 0.5)N + \\ & (12.4 \pm 0.5)L - \\ & (113 \pm 175)C, \end{aligned} \tag{2}$$
$$R^2 = 0.9965$$

$$\begin{aligned} RSS(Kb) = \; & (92995 \pm 86410) + \\ & (30.5 \pm 0.6)N + \\ & (21.0 \pm 2.5)L/C, \end{aligned} \tag{3}$$
$$R^2 = 0.9842$$

$$\begin{aligned} RSS(Kb) = \; & (65242 \pm 16260) + \\ & (24.1 \pm 0.1)N + \\ & (10.3 \pm 0.2)L + \\ & (10.1 \pm 0.5)L/C, \end{aligned} \tag{4}$$
$$R^2 = 0.9994$$

Model-to-model, the fits display large variations in the value of the free term, which also carries large uncertainty. Strictly speaking, none of the fits is statistically significant, and should be treated more as a guidance for the expected scaling rather than firm statistical relationship. This is also the reason we give several expressions for different combinations of dependent variables.

The memory scaling work is still under active development.

4. CONCLUSIONS

We presented and analyzed performance metrics for the distributed *ns-3* network simulator, including the packet Rx rate and the resident set memory footprint size. We derived several scaling relations for the memory footprint which can be used for planning larger distributed network simulations.

The performance analysis suggests optimal parallelization level at $\sim 100 - 200$ ranks, depending on the size of the network (larger networks can use a larger number of ranks efficiently). Beyond that limit, the performance suffers due to increased burden of inter-process synchronization and communication.

One potential improvement for the distributed *ns-3* network simulator is to implement a more scalable parallelization approach, by removing the requirement that every rank instantiates the global topology. A much more scalable solution would only instantiate the rank-local nodes, and perhaps ghost nodes for the endpoints of any cross-rank links emanating from the current rank. At the same time, there will have to be a method to identify the remote receiving node without knowing the remote rank topology, in order to create and schedule the receive event correctly. In the current implementation non-blocking receives are scheduled for all other ranks. When receiving messages, the source rank of the message is not used to determine which P2P link is involved. Rather, the destination node (global) index is transmitted as part of the MPI message. Instead of identifying the receiving node by global index, the MPI interface should track which rank sent the message, and with that one rank build a common picture of the shared P2P links and end nodes. An alternative would be to add a globally unique identifier, independent of the current globally sequential node Id, to each simulation node with a cross-

rank channel. Each cross-rank packet transmission would then identify the receiving node global id.

This approach will also require implementation of a distributed routing algorithm. All current ns3 automatic routing algorithms (global and Nix-vector routing) ultimately require access to the global topology. The need to carry the global topology and global routing tables at each rank limits the memory scalability. Removing this bottleneck will require implementation of a distributed shortest path algorithm, for example Δ-Stepping. [7]

5. ACKNOWLEDGMENTS

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. The Information Management release number for the document is LLNL-CONF-607992.

6. REFERENCES

[1] Chaos 5.0/toss 2.0 release notes. *Livermore Computing Technical Bulletin*, (474), 2012.

[2] P. D. Barnes, Jr. Xml format for ns-3 network topology. In *Workshop on ns-3, 2013*, WNS3, pages 1–3, ICST, Brussels, Belgium, Belgium, 2013. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[3] P. D. Barnes, Jr., J. M. Brase, T. W. Canales, M. M. Damante, M. A. Horsley, D. R. Jefferson, and R. A. Soltz. A benchmark model for parallel ns3. In *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, SIMUTOOLS '12, pages 375–377, ICST, Brussels, Belgium, Belgium, 2012. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[4] P. D. Barnes, Jr., J. M. Brase, T. W. Canales, M. M. Damante, M. A. Horsley, D. R. Jefferson, and R. A. Soltz. Livermore computer network simulation program. In *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, SIMUTOOLS '12, pages 223–225, ICST, Brussels, Belgium, Belgium, 2012. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[5] G. Karypis and V. Kumar. Metis - unstructured graph partitioning and sparse matrix ordering system, version 2.0. Technical report, 1995.

[6] ns3 Collaboration. The ns-3 network simulator. 2011.

[7] J. Pelkey and G. Riley. Distributed simulation with mpi in ns-3. In J. Liu, F. Quaglia, S. Eidenbenz, and S. Gilmore, editors, *SimuTools*, pages 410–414. ICST/ACM, 2011.

[8] G. Riley and J. Pelkey. An xml experiment description language for ns-3. In J. Liu, F. Quaglia, S. Eidenbenz, and S. Gilmore, editors, *SimuTools*, pages 447–453. ICST/ACM, 2011.

[9] E. Weingärtner, H. vom Lehn, and K. Wehrle. A performance comparison of recent network simulators. In *Proceedings of the IEEE International Conference on Communications 2009 (ICC 2009)*, Dresden, Germany, 2009. IEEE.