PERFORMANCE OF DISTRIBUTED SIMULATION WITH MPI IN NS-3 (Final Project Report: GIPEDI Internship, Batch-2, 2014)

Sonika Yadav
IGDTUW, Delhi.
sonikayadav822@gmail.com

Faculty Mentor: Prof. Subrat Kar
subrat@ee.iitd.ac.in,
Block 2A Room 114, Bharti School of Telecommunication Technology and Management

Student mentor: Ms. Garima Misra
garima.misra84@gmail.com
Block 2A Room 112, Bharti School of Telecommunication Technology and Management

July 15, 2014

Contents

1 INTRODUCTION

1.1 Motivation

While small topology simulations are important for validation and educational purposes, large-scale network simulations are a fundamental part of active networking research. Therefore, it is important for a network simulator to provide scalable and efficient solutions to execute these types of scenarios. Using standard sequential simulation techniques, large-scale topologies with substantial network traffic will require lengthy simulation execution times and a considerable amount of computer memory. Often, these execution times are too long for a networking researcher to run multiple simulations and collect significant data.

Parallel and distributed simulation is one method that allows researchers to efficiently simulate these large topologies by distributing a single simulation program over multiple interconnected processors. To enable this scalable simulation methodology in ns-3, we formally present our distributed simulator, introduced in ns-3.19.

This simulator uses the Message Passing Interface (MPI) standard and a conservative lookahead mechanism. Using the distributed simulator, we conducted a performance study using a large-scale point-to-point wired cum wireless network scenario with a variable number of nodes distributed across several interconnected processors within a computer cluster. We show near-optimal improvements in simulation execution time are possible using the distributed simulator in ns-3.19.

Firstly, pure distributed wireless simulation is not currently possible. Simulator boundaries must be created along point-to-point links; therefore, a distributed simulation in ns-3 requires at least one point-to point link within the topology.Second, distributing the topology must be done manually by the user. Individual nodes must be assigned rank during the topology setup, and the user must determine which nodes should be placed on which LPs for efficient distributed simulation. Finally, all nodes in the topology are created on each LP, regardless of their rank. Distributing the simulation is instead handled at the application level. The user should only install applications on nodes when their rank matches the given LP.

1.2 Task Performed

The following four task, helped to built insight of ns-3 and working methodology

1.2.1 To implement Simple Client/Server in NS3

Expected learning outcome : NS-3 simulation basics. Basic client server paradigm. Reading pcap traces.

1. Create a simple topology of two nodes (Node1, Node2) separated by a point-to-point link.

2. Setup a UdpClient on one Node1 and a UdpServer on Node2. Let it be of a fixed data rate Rate1.

3. Start the client application, and measure end to end throughput whilst varying the latency of the link

4. Now add another client application to Node1 and a server instance to Node2. What do you need to configure to ensure that there is no conflict?

5. Repeat step 3 with the extra client and server application instances. Show screenshots of pcap traces which indicate that delivery is made to the appropriate server instance.

Observation :

Figure 1: Output on terminal

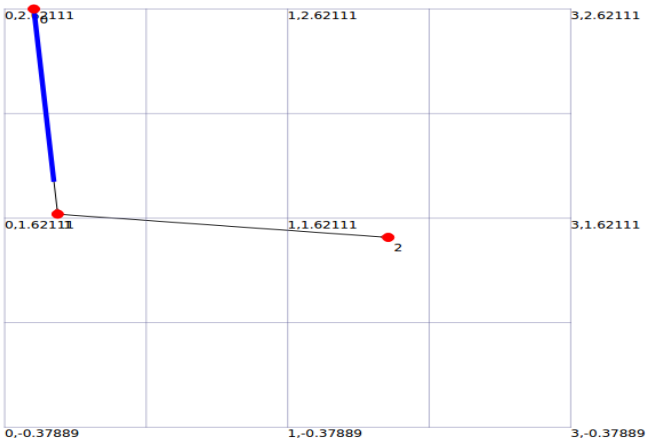

Figure 2: Topology for Task 1

1. Window size and throughput oscillations

2. Find average throughput, average queue size and also the loss rate of each tcp connection.

3. Construct a network with 2 TCP sources sharing one common link. Find the fraction of the bandwidth each source consume if,

- one source has delayed ACK option and another has simple ACK and both the sources use TCP NewReno
- one souce has TCP Tahoe and the other one uses TCP NewReno with simple ACK connections to both the receiving ends.

1.2.2 Study of TCP Variants

Expected learning outcome: TCP internals and the difference between each of the variants. NS-3 tracing mechanism.

1. Create a simple dumbbell topology, two client Node1 and Node2 on the left side of the dumbbell and server nodes Node3 and Node4 on the right side of the dumbbell. Let Node5 and Node6 form the bridge of the dumbbell. Use point to point links.
2. Install a TCP socket instance on Node1 that will connect to Node3.
3. Install a UDP socket instance on Node2 that will connect to Node4.
4. Start the TCP application at time 1s.
5. Start the UDP application at time 20s at rate Rate1 such that it clogs half the dumbbell bridge's link capacity.
6. Increase the UDP application's rate at time 30s to rate Rate2 such that it clogs the whole of the dumbbell bridge's capacity.
7. Use the ns-3 tracing mechanism to record changes in congestion window size of the TCP instance over time. Use gnuplot/matplotlib to visualise plots of cwnd vs time.
8. Mark points of fast recovery and slow start in the graphs.
9. Perform the above experiment for TCP variants Tahoe, Reno and New Reno, all of which are available with ns-3.

Observations :

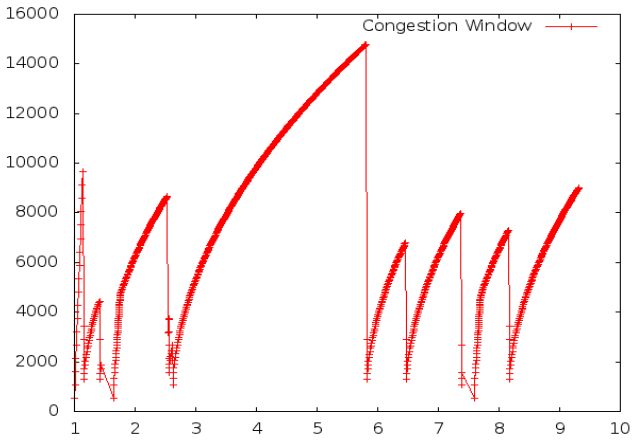

Figure 3: Window size variation of TCP Taheo

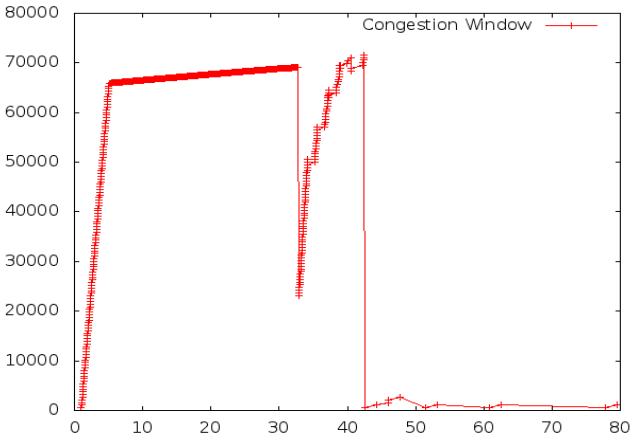

Figure 4: Window size variation of TCP New Reno

1.2.3 Error Model Analysis

- Realization simplex or duplex Error Model, which incoporates the up and down link flow of packets.
- Once you have your Error Model ready, in order to study the effect of loss probability of acknowledgements, modify your code and have link loss on the reverse link.
- Then analyse the throughput as a function of error (loss) probability for loss rates between 0 to 50.Check if TCP is more sensitive to forward losses or reverse losses.

Observation :

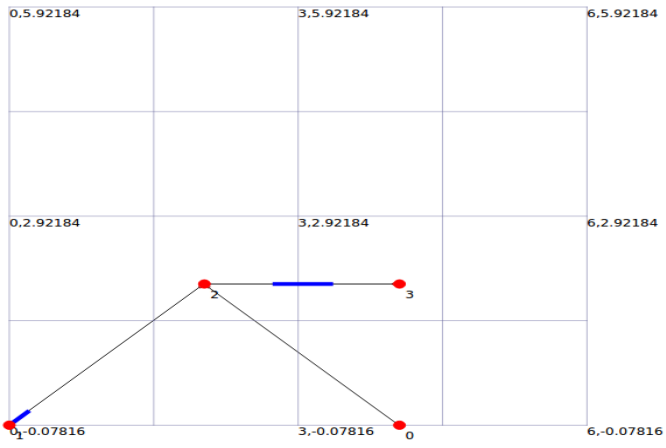

Figure 5: Animation for Duplex Error Model Analysis

1.3 History

1.3.1 Literature Survey

Distributed Simulation with MPI by J Pelkey,2011 paper describes the simple-distributed topology with fixed nodes and two LP's. Parallel and Distributed discrete event network simulation are implemented as a set of sequential simulations. An individual sequential simulation is referred to as a logicalprocess (LP), and the unique number for each LP is referred to as the rank.

Potentially, each LP can be placed on a different processor for execution. During the course of a distributed simulation, LPs may need to be notified of events occurring in different LPs. For networking simulation this occurs often, as packets in one LP are destined for a different LP. Each time a packet needs to change LPs, or cross simulator boundaries, a time-stamped message must be sent between LPs.

For distributed simulation in ns-3, we use MPI and a remote point-to-point channel to handle simulator boundaries and message passing. During the setup of a network topology, individual nodes are assigned numbers to indicate their LP. As point-to-point links are created, if a link between two nodes with differing LPs is encountered, a remote point-to-point channel is created between them. The transmit method for a remote point-to-point channel calls the MPI send operation, using our MPI interface in ns-3. Thus, any packet destined for a different LP will cross a remote point-to-point channel via a time-stamped MPI message, and the receiving LP can unwrap this message to obtain the packet and proceed with its simulation.

Along with message passing, ensuring events are processed in time-stamped order is a fundamental part of distributedsimulation. Without strict sequential event ordering at all LPs, distributed simulation will fail to produce identical results to sequential simulation.To ensure sequential ordering,several synchronization algorithms exist for distributed simulation.

It has chosen a conservative synchronization algorithm in ns-3. Conservative synchronization guarantees that events will be executed at an LP only if it is certain that no new events with a smaller timestamp will be received at that LP. To accomplish this, conservative synchronization is dependent upon predetermining a value known as lookahead. In the simulation model, the lookahead value is the minimum amount of time that must elapse before any ac- tion at LP-A can possibly affect anything at LP-B. In the case of network simulations, a packet transmission at router A cannot possibly be known to router B until at least the speed-of-light delay on the wire connecting routers A and B.

The lookahead value is used by each LP as the longest time-window allowed for local event processing before synchronizing with the remaining LPs. Events outside of this lookahead window are blocked until synchronization occurs.Subsequently, the lookahead window advances, and LPs are free to execute events within this new window. By carefully choosing a lookahead value, conservative synchronization can guarantee strict sequential event ordering. In ns-3,after topology

creation, each remote point-to-point link is visited and the time delay for that link is noted. The smallest time delay value for all remote point-to-point links is used as our lookahead value.It is important to note a few limitations of the current implementation of distributed simulation in ns-3.

1.3.2 Past Work

An important part of network simulation research includes the simulation of large-scale networks coupled with substantial network traffic. With standard sequential simulation techniques, these simulations can require lengthy execution times or may exceed available computer memory and fail to run completely. In either case, simulation is rendered ineffective. To run these large-scale simulations efficiently, parallel and distributed simulation techniques can be used. Earlier discrete event network simulators, such as Parallel/Distributed ns and the Georgia Tech Network Simulator (GTNetS) [2, 7], have implemented distributedsimulation.

By dividing a single simulation topology into distinct pieces and distributing these pieces across multiple interconnected processors, the simulator can execute different parts of the network in parallel during runtime. Along with a speedup in simulation execution time through parallel simulation, distributing a simulation across multiple nodes provides an increase in computer resources, specifically main memory, and enables larger topologies to be simulated. It is mandatory that simulations using distributed techniques produce the same results as identical sequential simulations.

To ensure proper distributed simulation execution in ns-3, it has been implemented an ns-3 interface with the MPI standard for message passing and time synchronization using a conservative lookahead algorithm. MPI interface in ns-3 allows time-stamped messages to be sent and received asynchronously between processes.

The implementation of a lookahead algorithm ensures that simulation events are processed in time-stamped order. Failure to re- quire sequential event ordering may lead to causality errors and, ultimately, incorrect simulation results. Demonstration of distributed simulation in ns-3, a performance study using a large point-to-point campus network topology has been conducted.

1.3.3 Planned Work

I will present simulations of wired cum wireless networks using the ns-3 simulator. The simulated networks consist of wired(csma) nodes and wireless(wifi) 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 using openmpi++.I will present and analyse performance metrics for the distributed ns-3 network simulator, including the packet Rx rate and for varing number of nodes.

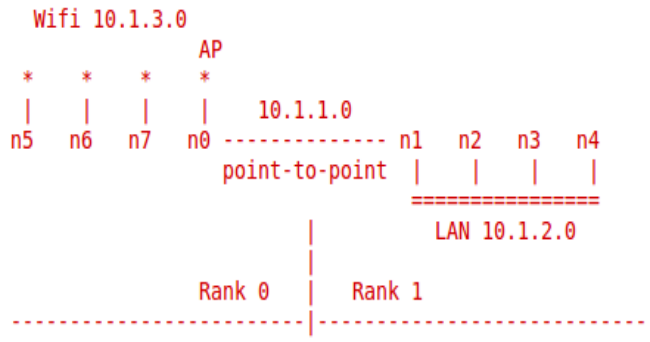

Figure 6: Wired cum Wireless Topology

1.4 Organisation of the thesis

Distributed discrete event simulation allows the execution of a single simulation program on multiple processors. By splitting up the simulation into logical processes, LPs, each LP can be executed by a different processor.

This simulation methodology enables very large-scale simulations by leveraging increased processing power and memory availability. In order to ensure proper execution of a distributed simulation, message passing between LPs is required. To support distributed simulation in ns-3, the standard Message Passing Interface (MPI) is used, along with a new distributed simulator class. Currently, dividing a simulation for distributed purposes in ns-3 can only occur across point-to-point links.

2 METHODOLOGY, APPROACH AND TOOLS

2.1 Approach

The network topology and the choice of parameters for the benchmark models are motivated by the considerations in previous work. [1] During the course of a distributed simulation, many packets must cross simulator boundaries. In other words, a packet that originated on one LP is destined for a different LP, and in order to make this transition, a message containing the packet contents must be sent to the remote LP. Upon receiving this message, the remote LP can rebuild the packet and proceed as normal. The process of sending an receiving messages between LPs is handled easily by the new MPI interface in ns-3.

Along with simple message passing between LPs, a distributed simulator is used on each LP to determine which events to process. It is important to process events in time-stamped order to ensure proper simulation execution. If a LP receives a message containing an event from the past, clearly this is an issue, since this event could change other events which have already been executed. To address this problem, two conservative synchronization algorithm with lookahead are used in ns-3. For more information on different synchronization approaches and parallel and distributed simulation in general, please refer to “Parallel and

Distributed Simulation Systems" by Richard Fujimoto[2].

The default parallel synchronization strategy implemented in the DistributedSimulatorImpl class is based on a globally synchronized algorithm using an MPI collective operation to synchronize simulation time across all LPs. A second synchronization strategy based on local communication and null messages is implemented in the NullMessageSimulatorImpl class, For the null message strategy the global all to all gather is not required; LPs only need to communication with LPs that have shared point-to-point links. The algorithm to use is controlled by which the ns-3 global value Sim- ulatorImplementationType.

The best algorithm to use is dependent on the communication and event scheduling pattern for the application. In general, null message synchronization algorithms will scale better due to local communication scaling better than a global all-to-all gather that is required by DistributedSimulatorImpl. There are two known cases where the global synchronization performs better. The first is when most LPs have point-to-point link with most other LPs, in other words the LPs are nearly fully connected. In this case the null message algorithm will generate more message passing traffic than the all-to-all gather. A second case where the global all-to-all gather is more efficient is when there are long periods of simulation time when no events are occurring. The all-to-all gather algorithm is able to quickly determine then next event time globally. The nearest neighbor behavior of the null message algorithm will require more communications to propagate that knowledge; each LP is only aware of neighbor next event times.

2.1.1 Remote point-to-point links

As described in the introduction, dividing a simulation for distributed purposes in ns-3 currently can only occur across point-to-point links; therefore, the idea of remote point-to-point links is very important for distributed simulation in ns-3. When a point-to-point link is installed, connecting two nodes, the point-to-point helper checks the system id, or rank, of both nodes.

The rank should be assigned during node creation for distributed simulation and is intended to signify on which LP a node belongs. If the two nodes are on the same rank, a regular point-to-point link is created. If, however, the two nodes are on different ranks, then these nodes are intended for different LPs, and a remote point-to- point link is used. If a packet is to be sent across a remote point-to-point link, MPI is used to send the message to the remote LP.

2.1.2 Distributing the topology

Currently,I have dealt with 2 logical processes, the full topology is created on each rank, regardless of the individual node system ids. Only the applications are specific to a rank. For example, consider node 1 on LP 1 and node 2 on LP 2, with a traffic generator on node 1. Both node 1 and node 2 will be created on both LP1 and LP2; however, the traffic generator will only be installed on LP1. While this is not optimal for memory efficiency, it does simplify routing, since all current routing implementations in ns-3 will work with distributed simulation.

2.2 Tools Used

ns-3.19 has been used to simulate the described topology.ns-3 (from network simulator) is a name for series of latest discrete event network simulators. ns-3 is free software, publicly available under the GNU GPLv2 license for research, development, and use. Since the process of creation of a network simulator that contains a sufficient number of high-quality validated, tested and actively maintained models requires a lot of work, ns-3 project spreads this workload over a large community of users and developers.

2.2.1 Design

ns-3 is built using C++ and Python with scripting capability. The ns-3 library is wrapped to python thanks to the pybindgen library which delegates the parsing of the ns-3 C++ headers to gccxml and pygccxml to generate automatically the corresponding C++ binding glue. These automatically-generated C++ files are finally compiled into the ns-3 python module to allow users to interact with the C++ ns-3 models and core through python scripts. The ns-3 simulator features an integrated attribute-based system to manage default and per-instance values for simulation parameters. All of the configurable default values for parameters are managed by this system, integrated with command-line argument processing, Doxygen documentation, and an XML-based and optional GTK-based configuration subsystem.

The large majority of its users focuses on wireless simulations which involve models for Wi-Fi, WiMAX, or LTE for layers 1 and 2 and routing protocols such as OLSR and AODV.

2.2.2 Components

ns-3 is split over couple dozen modules containing one or more models for real-world network devices and protocols.

ns-3 has more recently integrated with related projects: the Direct Code Execution extensions allowing the use of C or C++-based applications and Linux kernel code in the simulations, and the NetAnim offline animator based on the Qt toolkit.

2.2.3 Simulation workflow

The general process of creating a simulation can be divided into several steps:

1. Topology definition: to ease the creation of basic facilities and define their interrelationships, ns-3 has a system of containers and helpers that facilitates this process.
2. Model development: models are added to simulation (for example, UDP, IPv4, point-to-point devices and links, applications); most of the time this is done using helpers.
3. Node and link configuration: models set their default values (for example, the size of packets sent by an application or MTU of a point-to-point link); most of the time this is done using the attribute system.

4. Execution: simulation facilities generate events, data requested by the user is logged.

5. Performance analysis: after the simulation is finished and data is available as a time-stamped event trace. This data can then be statistically analysed with tools like R to draw conclusions.

6. Graphical Visualization: raw or processed data collected in a simulation can be graphed using tools like Gnuplot.

3 RESULTS AND CONCLUSIONS

3.1 Summary

The entire project is divided into four chapters, The first chapter provides an insight about the motivation behind the project. It describes what work has already been done and how do we plan to go ahead with the project. Second chapter describes how people have done it and how we are doing it. The components used are described and the reason for using these components are also justified. This chapter provides the result that we have obtained and the suitable graphs are also drawn. An interpretation of these results is also provided. What work we have done, what work can be done in future is also listed.

3.2 Distributed vs. Non-distributed

Distributed and non-distributed performances of the network simulations on the descibed topology(figures 6). To compare dis- tributed simulations against sequential simulations, a lookahead value of 200 ms(intuitive) was chosen. Distributed simulations run on the topolgy were significantly faster than identical sequential simulations.

The maximum number of nodes for a 250 nodes each of wired and wireless network simulation was with execution times of 156 and 1304 seconds for distributed (200 ms lookahead) and non-distributed runs, respectively.The 2, 4, 6, 8, and 10 network simulations ran on average 1.8, 3.3, 5.8, 6.9, and 8.3 times faster than the non-distributed simulations, respectively.

Using the distributed simulations, were faster than identical sequential simulations. For a 2 campus network simulation with 1028 nodes, execution times were 50 and 94 seconds for distributed (200 ms lookahead) and non-distributed runs, respectively.

The maximum number of nodes for a 250 node topology network simulation was 5140 with execution times of 322 and 712 seconds for distributed (200 ms lookahead) and non-distributed runs, respectively. Linear speedup was only observed for the described topology for network simulation runs. The average speedup values for 2, 4, 6, 8, and 10 campus network simulations were 1.9, 1.6, 2.0, 2.3, and 2.4 times faster than the non-distributed simulations, respectively.

3.3 Simulated Network

The network topology and the choice of parameters for the benchmark models are motivated by the considerations in previous work. [1] .The simulated networks consist of wired(nCsma) nodes and wireless(WiFi) nodes (see Figure 6). The router nodes of each are connected by point-to-point application.To simulate network traffic, we install cross-wired pairs of applications on randomly selected nodes in the network.

The applications can reside on either leaf or router nodes, with the only constraint that the nodes are di- rectly 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 250 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.

Global 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. We used the ns-3 trace system to output messages about the current RSS usage every 5 simulation seconds.

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 for Animation. In this work, we used our own XML spec, described in [2], to facilitate distributed simulation runs.

3.4 Distributed Simulations : Network Performance Metrics

The simulations are performed on a commodity cluster with 250(maximum, depending on command line input) each nodes. The CPU is Intel® Core™ i3-2348M CPU @ 2.30GHz has 3.8GB shared RAM. The operating system on the machine is UBUNTU 12.04 LTS, based on x64.To compile the ns-3 simulator (ver 3.19), I used openmpi-gnu-1.4.3 and gcc-4.4.6. All simulations were performed in batch mode (dedicated node allocation). The nominal runtime limit for batch jobs is 3.7 secs (wallclock time).

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 traffic time metric is shown in Figure – as a function of the number of computing cores. The rate of packet reception per wallclock second per rank is shown in Figure 0–0. 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.

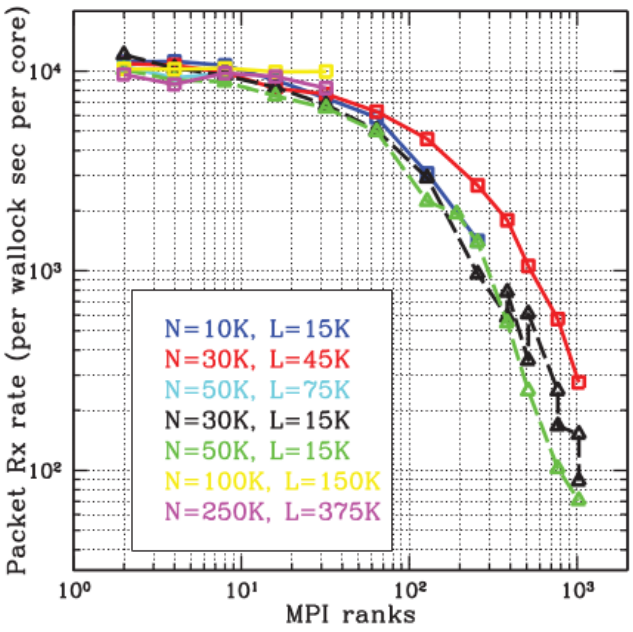

Figure 7: Packet Rx rate per process per wallclock time for some runs

3.5 Results

Presentation and analysis of 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 100 to 250 nodes, depending on the size of the network (larger networks can use a larger number of ranks efficiently).I have formally presented our distributed simulator, released in ns-3.19, and described its implementation details.

Specifically, the MPI interface, remote point-to-point channel, and the conservative synchronization algorithm were discussed. Using this distributed simulator,

performance study using a large-scale point-to-point wired cum wireless network topology was presented.

The study showed nearly optimal linear speedup is achievable using the distributed simulator in ns-3. Furthermore, the study confirmed that speedup was greatly affected by the lookahead value used,as expected for any distributed simulation using the conservative synchronization technique.

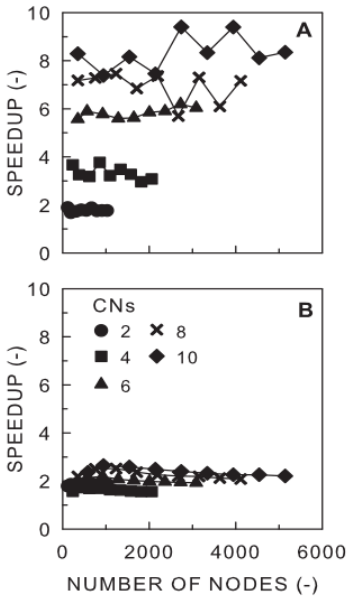

Figure 8: Speedup Graph of Non-Distributed and Distributed

4 FUTURE WORK

For future work, several additions and improvements will be examined. First, an MPI helper class could be designed to facilitate creating distributed topologies. Rather than a user assigning rank to each node individually, nodes could be assigned rank automatically by examining the topology and determining the optimal simulator boundary positions.With this addition, existing simulation scripts could be distributed with relative ease.

Second, distributing the topology could occur at the node level, rather than the application level. Distributing at the node level would greatly reduce memory consumption and lead to a speedup in topology generation. Finally, pure distributed wireless is a commonly requested feature and will be added to the distributed simulator in ns-3.

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 delta-Stepping. [1]

Acknowledgement

I take this opportunity to express my profound gratitude and deep regards to our Faculty Mentor Prof. Subrat Kar,our Student Mentor Ms. Garima Misra, Mrs. Sagarika Dutta Bishwas, Lab Incharge Mr. Deepak Priyadarshi, Ms. Monika Rajan, for their constant guidance and supervision and also for their support in completing the project. I would also like to thank FITT(Foundation for Innovation and Technology Transfer) for giving me such a knowledgeable exposure. I would like to express our gratitude towards my parents and members of IIT Delhi for their kind co-operation and encouragement which helped me in completion of this project.

My thanks and appreciation also goes to all the people who have willingly helped us out with their abilities.

Sonika Yadav

Place: New Delhi.

References

[1] J Perkley, 2012. "Distributed simulation with MPI in ns-3."

[2] Richard M. Fujimoto, 2001."Parallel and Distributed Simulation Systems."

[3] E. Weingartner, H. vom Lehn, and K. Wehrle, 2010."A Performance Comparison of recent Network Simulators."