

PARALLEL SIMULATION OF SCALE-FREE NETWORKS

A Dissertation
Presented to
The Academic Faculty

by

Vy Thuy Nguyen

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Computational Science & Engineering

Georgia Institute of Technology
August 2017

COPYRIGHT © 2017 BY VY THUY NGUYEN

PARALLEL SIMULATIONS OF SCALE-FREE NETWORKS

Approved by:

Dr. Richard M. Fujimoto, Advisor
School of Computational Science & Engineering
Georgia Institute of Technology

Dr. Richard Vuduc
School of Computational Science & Engineering
Georgia Institute of Technology

Dr. Brian Swenson
Information and Communication Laboratory
Georgia Institute of Technology

Date Approved: July 27, 2017

TABLE OF CONTENTS

LIST OF FIGURES	v
SUMMARY	vii
CHAPTER 1. Background & Motivation	1
1.1 Discrete Event Simulation (DES)	2
1.2 Parallel Discrete Event Simulation (PDES)	3
1.2.1 Conservative Synchronization Algorithms	5
1.2.2 Optimistic Synchronization Algorithms	7
1.3 Communication Network Simulation	9
1.4 Scale-free Networks	10
1.5 Related Work	12
1.5.1 Scale-free Graph Partitioning	12
1.5.2 Link Partitioning in PDES	13
1.5.3 Performance Analysis of Scale-free Graph Simulation	13
1.6 Contributions	14
1.7 Dissertation organization	15
CHAPTER 2. Node and Link Partitioning	16
2.1 Router Architecture	16
2.2 Node Partitioning	17
2.3 Link Partitioning	18
2.4 Lookahead	21
CHAPTER 3. Parallelism in the Node and Link Models	23
3.1 Experiment Setup and Terminology	23
3.1.1 Sequential Simulator & YAWNS Revisited	23
3.1.2 Network & Traffic Generation	24
3.2 Baseline Parallelism Using YAWNS	25
3.3 Impact of Lookahead	30
3.4 Performance Evaluation Using Critical Path Analysis	33
3.5 Discussion	36
CHAPTER 4. Parallel Performance Evaluation	38
4.1 Experiment Setup	38
4.1.1 Parallel Discrete Event Simulator	38
4.1.2 Network & Traffic Generation.	40
4.2 Baseline Parallelism Using YAWNS	40
4.3 Impact of Lookahead	46
4.4 Discussion	48
CHAPTER 5. Conclusion and Future Work	50
5.1 Contributions	50
5.2 Future Work	50

LIST OF FIGURES

Figure 1	– Router Architecture	16
Figure 2	– LP Components in a Link oriented model	18
Figure 3	– Physical Network vs Link-as-LP Model	20
Figure 4	– Parallelism obtained under different traffic rates	26
Figure 5	– Epoch-by-epoch measure of the max, min, and average number of events processed by each LP in the Node model	27
Figure 6	– Epoch-by-epoch measure of the max, min, and average number of events processed by each LP in the Link model	27
Figure 7	– Standard deviation of the numbers of events per LP per epoch	28
Figure 8	– Epoch-by-epoch measure of the max, min, and average number of events processed by each LP across top 10 busiest nodes	29
Figure 9	– Epoch-by-epoch measure of the max, min, and average number of events processed by each LP across top 10 busiest links	30
Figure 10	– Standard deviation of the numbers of events processed by each LP across top 10 busiest LPs	30
Figure 11	– Parallelism obtained under different link delay values and average interarrival time $\lambda = 5$	31
Figure 12	– Parallelism obtained under different link delay values and average interarrival time $\lambda = 3$	32
Figure 13	– Parallelism obtained under different link delay values and average interarrival time $\lambda = 2$	33
Figure 14	– Parallelism obtained under different link delay values and average interarrival time $\lambda = 1$	33
Figure 15	– Parallelism achieved by the Link and the Node models under different interarrival rates (traffic loads) and link delays	34
Figure 16	– Critical Path Length of the Link and the Node models under different interarrival rates and link delays	35
Figure 17	– Parallelism at average interarrival time $\lambda = 1$	36

Figure 18	– Parallelism at average interarrival time $\lambda = 5$	36
Figure 19	– High-level System Architecture	39
Figure 20	– Event Processing Loop	39
Figure 21	– Event rates across different traffic rates ($p = 1$)	41
Figure 22	– Event rates across different traffic rates ($p = 4$)	42
Figure 23	– Epoch-by-epoch measure of the max, min, and average number of events processed by each LP in the Node Model	43
Figure 24	– Epoch-by-epoch measure of the max, min, and average number of events processed by each LP in the Link Model	44
Figure 25	– Standard deviation of the number of events per LP in each epoch	44
Figure 26	– Number of events per LP per epoch among top 10 busiest nodes	45
Figure 27	– Number of events per LP per epoch among top 10 busiest links	46
Figure 28	– Standard deviation of the number of events per LP per epoch among the top 10 busiest LPs	46
Figure 29	– Impact of lookahead on event rate, $\lambda = 5 \text{ ms}$	47
Figure 30	– Impact of lookahead on event rate, $\lambda = 3 \text{ ms}$	48
Figure 31	– Impact of lookahead on event rate, $\lambda = 1 \text{ ms}$	48

SUMMARY

It has been observed that many networks arising in practice have skewed node degree distributions. Scale-free networks are one well-known class of such networks. Achieving efficient parallel simulation of scale-free networks is challenging because nodes with large degree can create bottlenecks that limit performance. While parallel discrete event simulation is commonly used to simulate many complex systems, skewed node degree topologies pose a challenging test case. To help address this problem we describe an approach called link partitioning where each network link is mapped to a logical process (LP) in contrast to the conventional approach of mapping each network node to an LP. Link partitioning is discussed in the context of packet-level simulations of telecommunication networks. The parallelism of link partitioning relative to node partitioning is examined in terms of an idealized execution using the well-known YAWNS synchronization algorithm. Further, a critical path analysis suggests that there is much more parallelism available in these simulations than can be exploited using the YAWNS algorithm. Finally, an implementation of the parallel simulation using YAWNS as the synchronization protocol is presented together with its performance evaluation.

CHAPTER 1. BACKGROUND & MOTIVATION

How much parallelism one can achieve by adopting parallel execution is a topic of significant interest not only in the simulation community but also in the high performance computing community. The amount of parallelism obtained from parallel execution depends on many factors. One key limitation is the amount of computation that is inherently sequential. In a discrete event simulation, this depends on how the computation is partitioned among the logical processes making up the parallel simulation. Other aspects that impact parallel performance include the mapping of logical processes to processors and overheads for communication and synchronization.

In creating a parallel discrete event simulation program one must determine how to partition the system under investigation into elements that are then mapped to logical processes. For example, when modeling communication networks, a standard approach is to model each node in the network as a logical process. We call this approach node partitioning. However, communication networks have been found to possess scale-free and small-world properties [1]. This presents a challenging test case for parallel discrete event simulation. Their inherent properties can severely hinder the performance of the simulation.

While parallel discrete event simulation is a popular acceleration technique and it does provide mechanisms to speed up the simulation compared to a sequential execution, scale-free network topologies present many challenges. Existing parallelization techniques are not well suited for such networks.

This dissertation introduces a new partitioning scheme for topologies with skewed node degrees and performance evaluation of this technique comparing it to the traditional approach. This research can help gain a better understanding of the performance bottlenecks that arise in such topologies and ways to alleviate them

1.1 Discrete Event Simulation (DES)

Simulation is the process of imitating the behavior of a system, entity, or phenomenon over time. In a simulation a system, entity or phenomenon is represented as a model which is a simplified version of the real system represented in the form of mathematical or logical relationships. A simulation model specifies the evolution of a system's state over time.

Simulation is widely used in many fields. It facilitates understanding or prediction of a system's behavior in a controlled environment. Of particular interest here are communication network simulations that can provide insights into the behavior of network protocols under different operating conditions and workloads.

A discrete event simulation models a system's behavior as a discrete, chronologically non-decreasing sequence of event computations. Each event is assigned a timestamp – a discrete point in simulation time at which the event occurs. Each event computation may result in a change of system state and/or the scheduling of new events. There is no change of system state between two consecutive events. That is, the system's state only changes when an event computation occurs.

A discrete event simulation system can be divided into three main components: the simulation application, the simulation executive and the event processing loop. The simulation application specifies the model of the physical system. It contains the state variables, specification of the system behavior, and I/O or user interface software. The simulation executive is largely independent of the simulation application. It contains the future event list and the global clock. The future event list is simply a priority queue that sorts events based on their timestamp – the point in simulation time that they are scheduled to occur. The global clock indicates how far in simulation time the simulation has advanced. In each iteration of the event processing loop, the event with the smallest timestamp is removed from the future event list and processed by the event handler in the simulation application. The processing of an event, also called an event computation, advances the global clock to the timestamp of that event and typically modifies the state variables. An event computation may also result in the creation of new events that are scheduled into the future event list. This process continues until a termination condition is met. For example, the termination condition might be the global clock reaching some specified time or the event list is empty.

A simple discrete event simulator of activity at an airport may include variables such as the number of available gates. Events may represent the arrival and departure of aircraft. When an aircraft arrives, the number of available gates decreases. The arrival of an aircraft may also result in a new departure event being scheduled at some time in the future. Similarly, as an aircraft departs, the number of available gates increases. There should not be any changes in the number of available gates between two consecutive events.

1.2 Parallel Discrete Event Simulation (PDES)

Parallel computing enables acceleration of large-scale simulations by distributing the workload over multiple processors. Parallel discrete event simulation focuses on distributing event processing across different processors. This allows multiple events to be processed concurrently, potentially leading to a shorter execution time.

In a parallel discrete event simulation program the system being modeled is viewed as a set of physical processes that interact at various points in simulation time. The simulator is then constructed as a set of logical processes (LP), each of which represents a physical process and contains a portion of the state corresponding to the physical process. Each LP has a local clock denoting how far in simulation time the process has progressed. Each LP also has its own future event list that only contains events to be processed on that particular LP. These LPs communicate with one another by exchanging timestamped event messages. Each event message contains a single simulation event.

Since events in a parallel discrete event simulation program are processed concurrently, unless precautions are taken, an LP may process events out of timestamp order, leading to an incorrect computation. The processing of events out of timestamp order is referred to as a causality error. Causality errors can be avoided by ensuring that each LP processes events in non-decreasing timestamp order. This is referred to as the *local causality constraint*.

Consider the case where LP₁ sends a message with timestamp t_0 to LP₂ and the next event to be processed in LP₂ has timestamp $t_1 > t_0$. However, LP₂ has no knowledge of the message from LP₁ until after it has processed the event with timestamp t_1 . How does one

LP know which events can be processed without the possibility of a causality error later occurring? A synchronization algorithm is required to address this problem.

Algorithms to address the causality error problem generally fall into two categories: conservative and optimistic. Conservative approaches aim to strictly avoid causality errors. In contrast, optimistic approaches rely on rollback mechanism to recover the system state whenever a causality error is detected.

1.2.1 Conservative Synchronization Algorithms

Conservative synchronization algorithms ensure adherence to the local causality constraint by only allowing processors to process “safe” events, i.e., they strictly avoid processing events out of timestamp order. Since each process only has the information regarding the timestamp of the local events and not that of the entire system, communication between processes is required to determine which events are safe to be processed. Suppose it can be determined that only events with timestamp greater than T_1 are safe to process. Then processes without such events must block. This can potentially lead to deadlock in the system. Deadlock occurs when processes wait on one another and no process can advance.

The first generation of conservative algorithm used null messages to avoid deadlock. These algorithms executed asynchronously, i.e., no global synchronization operations were used. One well-known example is the Chandy-Misra-Bryant algorithm [2]. One disadvantage of this algorithm is that an excessive number of messages may need to be sent between LPs before the system can resume normal event processing. Another

conservative algorithm that used null message is that proposed by Peacock, Manning and Wong [3].

Later conservative algorithms determine which events are safe to process by using global synchronization and reduction computations to compute a Lower Bound on Timestamp (LBTS) of future messages that each LP may receive [4]. One well-known algorithm in this class is the YAWNS protocol, which is also the main focus of this work. Other examples include simulation time windows technique described in [5].

As with other conservative algorithms, YAWNS aims to strictly prevent events from being processed out of order. It requires a lookahead value to be specified. Lookahead is defined as the minimum amount of time between the current simulation time of the LP scheduling the event and the timestamp of the new event. For example, if an LP is currently at simulation time T , and the system has lookahead L , then any future messages sent by this LP will have timestamp of at least $T + L$. The YAWNS algorithm processes events in iterations, also called epochs. In each epoch, each processor computes the local LBTS, which is the minimum timestamp of any future messages that the process may send if it does not receive any other new messages. The global LBTS is the minimum among all these local LBTS values. Any events with timestamp less than the global LBTS are safe to process. In the YAWNS algorithm, the lookahead value L defines the size of the epoch window. In general, larger epoch windows allow more events to be processed concurrently. Thus, the size of the lookahead value can have a significant impact on the amount of concurrency that can be exploited in simulations using YAWNS. This dissertation focuses primarily on YAWNS conservative algorithm.

One well-known drawback of all conservative algorithms is that they cannot fully exploit the parallelism available in a simulation because they are usually overly pessimistic [6]. For example, event E_1 might affect event E_2 in theory, but in practice, that may seldom be the case. Conservative approaches must always process event E_1 prior to event E_2 , while they could often be safely processed concurrently without violating the causality constraint.

1.2.2 Optimistic Synchronization Algorithms

In contrast to conservative algorithms which strictly avoid causality errors, optimistic algorithms detect and recover from such errors. One notable example of optimistic algorithms is the Time Warp mechanism [7] based on Virtual Time paradigm [8].

The Time Warp algorithm allows both optimistic event processing and optimistic message sending. In a Time Warp simulation, a causality error occurs whenever a process receives a message with timestamp less than the current local time – i.e., the timestamp of the last processed event on that process. The event that causes the causality error is called a straggler event. When a causality error occurs, recovery is done by canceling the effects of all events with timestamp greater than that of the straggler event but have been processed prior to the straggler event.

Canceling the effect of an event includes rolling back the LP to an earlier state and/or un-sending messages that might have been sent as the result of processing the rolled back events. LP rollback is accomplished by saving the LP's state before each event is processed. Whenever an event is rolled back, the state of the LP prior to the processing of that event is restored.

Un-sending messages is accomplished via the positive and negative message (also called anti-message) paradigm. Messages corresponding to a normal simulation event are called positive messages. Messages sent to annihilate some erroneously sent messages are called negative messages. When a logical process receives a negative message, it must cancel the effect of processing the positive message corresponding to that negative message if the positive message had already been processed. This means rolling back the LP state to some prior point in time and un-sending any message that might have been sent as the result of processing the positive message. This process is repeated recursively until all effects caused by the causality error have been canceled.

Additionally, in a Time Warp simulation, the smallest timestamp among all unprocessed positive and negative events is called the Global Virtual Time (GVT). No events with timestamp less than the GVT will ever be rolled back. Therefore, the LP's states corresponding to such events can be safely discarded.

Other optimistic algorithms include Breathing Time Bucket [9] and Breathing Time Warp [10]. The Breathing Time Bucket algorithm minimizes the risks in message sending [11] (Reynolds 1988) and thus does not employ the anti-message paradigm since all rollbacks are local. A Breathing Time Bucket simulator processes events in iterations. In each iteration, all events that do not precede one another are processed. Suppose the simulation is currently at time t and all newly generated events will have timestamp at least $t + \Delta t$. Then the minimum $t + \Delta t$ is called the Global Simulation Time (GST). All events with timestamp between t and $t + \Delta t$ do not precede one another, and thus can be safely processed in parallel. In order to compute the GST, each processor p first calculates its own local version of the GST, called the Local Event Horizon (LEH). The GST is then

the minimum among all LEHs. LEHs are generally greater than the GST, so some events are prematurely processed. Of course, those events can be rolled back. Breathing Time Warp is the hybrid version combining both Time Warp and Breathing Time Bucket techniques.

One advantage of the optimistic approach is that it allows for better exploitation of parallelism in a simulation compared to conservative approaches. On the other hand, the disadvantage of optimistic algorithms is the extensive amount of memory required during the simulation execution. Throughout the execution, the program must constantly checkpoint each LP before processing each event. That is, the entire state of the LP must be saved before each event can be processed, so that if out-of-order processing is detected, the simulation can be rolled back to a previous state. Later generation of optimistic algorithms offer mechanisms to reduce memory usage, such as using incremental state saving techniques as described in [12] and [13].

1.3 Communication Network Simulation

There are a variety of existing communication network simulators offering varying degrees of complexity. At the minimum, network simulators should enable users to specify a network topology, nodes and links as well as traffic between the nodes. More advanced simulators may allow users to define routing protocols or more complex traffic patterns.

NS-3 and OMNet++ [14] are two of the most popular open-sourced network simulators used in the research community. NS-3 is the latest version of NS (Network Simulator) which derives from REAL (Realistic and Large) [15]. OMNet++ (Objective Modular Network Testbed in C++) is a component-based simulation

framework. Both are discrete-event simulators with Graphical User Interface (GUI) support. The two simulators have generic and flexible architectures, which enable one to simulate networks ranging from communication networks to IT systems to queueing networks, or even business processes.

QualNet and OPNET are two other popular network simulators used extensively in industry. While NS-3 and OMNet++ are open sourced, QualNet and OPNET are proprietary products. They provide graphical user interfaces for modeling, simulation and analysis. One of the key features of these simulators is the use of a fast discrete-event simulation engine.

In the simulation of communication networks, the simulation workload can be characterized by the number of packet transmissions that must be simulated. The performance of a simulator can be quantified by the number of simulated packet transmissions that can be simulated per second (PTS) of wallclock time [16].

However, the implementation of the simulation engine alone may not be sufficient to improve the performance a simulation, especially for irregular networks. Therefore, this dissertation introduces a new approach to modeling communication networks with a scale-free network topology, which allows for greater exploitation of parallelism

1.4 Scale-free Networks

Scale-free networks are a topology that present challenges in parallel simulation, as will be described momentarily. Previously, in the absence of complex network models, network topologies were often modeled based on random network theory. A well known

model is that developed by Erdős and Renyi – the ER-network model. In the ER-network model, the node degree distribution follows a Poisson distribution. Each pair of nodes are connected with a given probability p , and the majority of nodes have a degree approximately the same as the average node degree in the network.

In 1999, results from Albert et al.'s study of the Internet have shown that linkages between web pages were, in fact, not random as expected. Only a few web pages have a large number of links, while most of the pages have very few links. The ER-network model is inadequate to describe networks of this type where the node degree distribution is skewed. A new network model called the scale-free network model was introduced.

In scale-free networks, the node degree distribution follows a power law distribution, at least asymptotically. That is, the probability distribution function $P(k)$ of node degree k of scale-free networks is given by

$$P(k) \approx k^{-\gamma}$$

where $k, \gamma > 0$, and γ is called the scale-free exponent.

Since their node degrees follow a power law distribution, scale-free networks exhibit great imbalance in degrees among the nodes in the network. Additionally, these networks possess the small world property, which means the average minimum path length between any two given nodes in the network tends to be small even when the total number of nodes in the network is large. This is because the number of nodes reachable from any given node increases exponentially with the number of hops.

It has been observed that many natural and engineered systems possess scale-free properties. Examples include the Internet topology at the autonomous system (AS) level [17] (Faloutsos), protein interaction networks [18] (Hase, Takeshi), financial networks [19] (Soramaki), social networks, the world wide web [20], airline transportation networks, or human interaction networks used to model the spread of diseases. Other networks such as the physical topology of the Internet at the router level also exhibits a skewed node degree distribution that leads to hub nodes with much larger degrees than that of most of the other nodes [21].

Given the size and prevalence of scale-free networks in a wide range of areas, techniques to efficiently simulate such networks are essential. However, the inherent properties of scale-free networks pose a challenge for parallel discrete event simulation. For example, the imbalance in node degrees in scale-free networks means that if each node is mapped to a logical process in a parallel simulation, a small number of logical processes may be responsible for a disproportionately large amount of work, while many other logical processes will have comparatively little computation to perform. Consequently, this can severely limit the amount of parallelism that the system can achieve. Besides, the small world property of such networks makes determination of the set of events that can affect another event challenging since the number increases exponentially with the number of links. This is another challenge for conservative synchronization algorithms.

1.5 Related Work

1.5.1 Scale-free Graph Partitioning

Partitioning of scale-free graphs has been studied extensively in the data mining and database community. Many efficient techniques have been presented, including the vertex-cut (also known as link partitioning) approach. In link partitioning approach, a vertex can be split and assigned to different partitions while edges are assigned without cut. For example, Gonzalez et al. utilize the method in their PowerGraph framework [22]. Pearce, Gokhale and Amato also propose an edge list partitioning technique to address the problem caused by high-degree nodes in scale-free graphs [23].

1.5.2 Link Partitioning in PDES

Although link partitioning of scale-free graph partitioning is not a new technique, to our knowledge, it is still new for parallel discrete event simulations and has not been studied for communication network simulations. Further, issues such as lookahead introduce new considerations that have not been previously studied. In their study of large-scale Social Contact Networks (SCN) simulation Wu et al. also propose a similar technique [24]. However, due to the dynamic contact relationship between the agents in a SCN, it is difficult to completely partition hub nodes. Therefore, in their approach, after a hub node is partitioned, its partitions still have to maintain knowledge of and communication with one another during the simulation. In a sense, the approach still employs node-oriented processing. In other words, the functionality of a hub node is not entirely partitioned.

1.5.3 Performance Analysis of Scale-free Graph Simulation

Various prior works have examined the performance of parallel discrete event simulation for scale-free networks. Kunz, et al. introduced a scalable performance prediction methodology that computes the best possible performance of parallel discrete

event simulation programs based on linear programming [25]. An empirical study by Liu and Chien showed that severe load imbalances can be observed in parallel discrete event simulation of communication networks [26]. The load distribution issue in parallel discrete simulation of scale-free networks is also studied in D'Angelo and Ferretti [27]. Pienta and Fujimoto have also shown through analytical models and simulations, that very limited parallelism can be obtained in very large network simulations [28]. Other works examining the parallelism in parallel modeling and simulation of scale-free graphs include work by Hruz et al [29]. All of these studies, however, primarily focus on the node-oriented partitioning approach (i.e., each node is assigned to one LP.)

1.6 Contributions

The objective of this work is to study the limits to parallelism in scale-free network simulation as well as to design, develop and evaluate the performance of a new approach to modeling scale-free communication networks in parallel discrete event simulation. We also present a simple implementation of a parallel discrete event simulation of scale-free communication networks using YAWNS.

The primary contributions of this work are as follows:

- We propose a new approach to modeling networks called link partitioning that addresses the bottleneck problem in parallel discrete event simulation of scale-free communication networks. We show how this approach can be applied to a wired communication network simulation application.
- We showed that the new link model yields better parallelism than the traditional node model due to better distribution of events across LPs. To quantitatively

understand the limits to parallelism in parallel simulation of networks exhibiting scale-free properties, we developed a sequential simulator to emulate the parallel discrete event simulation being studied. Utilizing critical path analysis, we also examine the amount of parallelism obtained from the node and link approach independent of the synchronization protocol. The results suggest that the link approach may result in even greater amount of parallelism that could potentially be exploited by other synchronization algorithms

- We show the performance advantage of the link-partitioning approach compared to node partitioning by developed a parallel discrete event simulation implementation using YAWNS as the synchronization algorithm. The performance results are consistent with the parallelism results described above.

1.7 Dissertation organization

The remaining of this dissertation is organized as follows. Chapter 2 introduces the link partition scheme, a new approach to modeling scale-free topologies in PDES. In Chapter 3 we demonstrate how we used a sequential simulator to emulate a PDES of a communication network and to evaluate its performance. We also compare the performance of the Link vs. the Node partitioning approach. In Chapter 4 we provide the performance evaluation of the actual implementation of the parallel simulator. Finally, Chapter 5 concludes this dissertation and discusses possible directions for future work.

CHAPTER 2. NODE AND LINK PARTITIONING

This chapter describes a simulation model of a packet-switched telecommunication network using the node and link partitioning approaches. In a packet-level simulation the operation of the network focuses on the transmission of packets among network nodes (e.g., hosts, routers) over communication links.

2.1 Router Architecture

Figure 1 derived from [21] shows a high-level view of a network router. While this figure represents a particular switch architecture, it reflects the main functions performed by network routers in general.

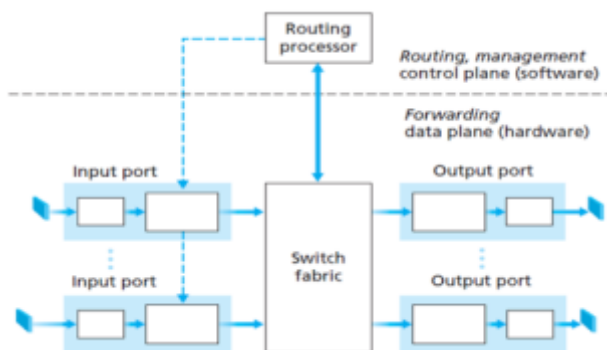


Figure 1 – Router Architecture

Each router typically consists of input ports, the switch fabric and output ports. Packets come into the router through the input ports. They then pass through the switch fabric where they are forwarded to the appropriate output port determined by the routing processor. Finally, the packages leave the router through the output port to which they are forwarded.

Packet-level simulations usually focus attention on the movement of packets in the data plane. Specifically, incoming packets may be received concurrently on each input port where they are buffered until the entire packet has arrived and can be transmitted to an output port via the switch fabric. Each output port contains a queue that holds packets waiting to be transmitted over the outgoing link. The switch fabric is a hardware device such as a crossbar switch that transmits packets within the router from an input port to the appropriate output port. Here, we assume the switch fabric can transmit packets within the switch concurrently so long as they utilize a different output port. The control plane includes a routing processor that manages aspects such as the routing of packets from input to output ports in the switch based on the packet's intended destination.

We assume that all packets are the same size and each queue has unlimited capacity. The link model discussed momentarily can easily be extended to include finite capacity queues and packet dropping should queue overflow occur.

2.2 Node Partitioning

With node partitioning, each router shown in Figure 1 is simply modeled by a single LP. This is the common approach used by most PDES simulation models of communication networks. A typical discrete event simulation will have two types of events: arrival and departure events. An arrival event denotes a packet arriving at an input port. It is routed through the switch fabric to an output port, and a departure event is scheduled with a timestamp indicating when the router begins transmission of the packet on the outgoing link, taking into account queuing delays at the output port. The departure

event schedules a new arrival event at the LP modeling the next router indicating when it arrives at that router.

We note that with this approach, there is one LP-to-LP communication in the parallel simulation for each hop traversed by a packet as it is forwarded through the router – the arrival event. LP-to-LP communications refer to events scheduled between different LPs and are important because they entail certain overheads, especially if the sending and receiving LPs are mapped to different processors.

As noted earlier, node partitioning is problematic for routers containing a large number of links. A simple solution is to implement each input port and each output port as separate LPs. However, this approach doubles the number of LP-to-LP communications required to model a single hop through the network, a significant drawback relative to the original node partitioned model. For this reason, we do not consider this approach further.

2.3 Link Partitioning

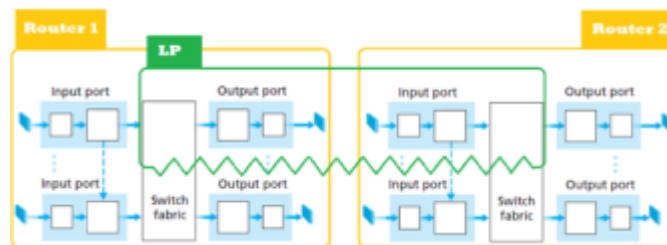


Figure 2 – LP Components in a Link oriented model

In the link partitioning approach each link is mapped to a single LP, as shown in Figure 2. The figure highlights the components modeled by an LP for traffic traveling in one direction on the link. Each LP includes two output ports (one for each direction of

traffic over the link), the communication link, two input ports (similarly, one for each direction of traffic), and the switch fabric. The green box in Figure 2 shows half of the elements modeled in one LP; the other half of the LP models components for traffic moving in the opposite direction of the bi-directional link. Note that these queues model the queues of the physical system, and should not be confused with the event queues managed for each LP.

The following figures shows an example of a network's physical state compared to the simulated model. In the left-hand-side figure, green rectangles and blue ovals represent the nodes in the network. Each node can be a host or a router. Yellow arrows represent the links between these nodes. The figure to the right shows the corresponding link model of the network. Yellow rectangles represent logical processes (LPs) in the model. There is a blue arrow between two LPs if they can communicate. Two LPs can communicate if the links they represent share the same router or host in the physical network. For example, consider link R0-H1 (connecting Host 1 and Router 0) and link R0-H0 (connecting Host 0 and Router 0). In the simulated model, there's an arrow between the two LPs representing these two links because they both touch router 0.

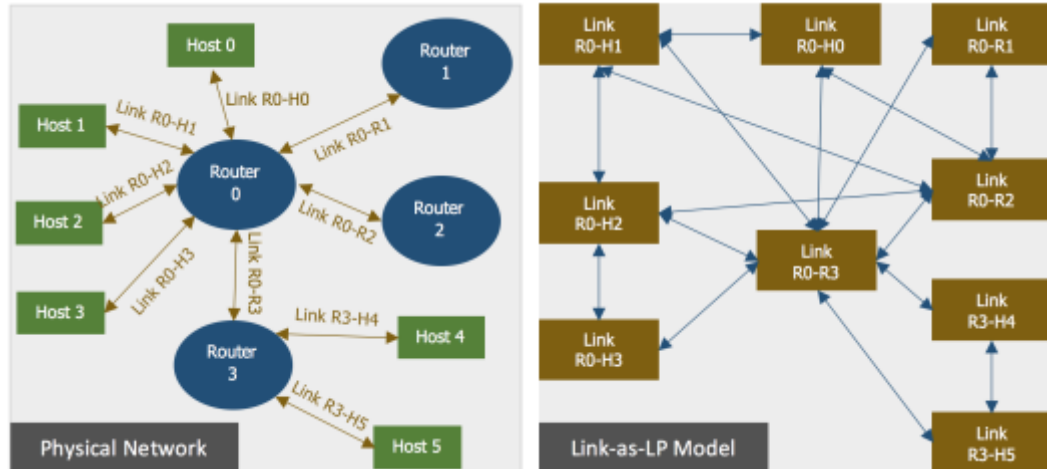


Figure 3 – Physical Network vs Link-as-LP Model

This model includes two types of events. The first is a packet “switching” event that denotes a packet being passed from the switch fabric to the output port. Multiple switching events may occur at an LP at approximately the same time. This would occur if multiple packets arrive simultaneously on different input ports that are routed to the same output port. In this case, the LP will model the queueing of these packets at the output port. The second event, called a “transmitted” event is scheduled by a link LP for itself to denote that the packet has been transmitted over the communication link, and arrived at the input port of the receiving router. When the transmitted event is processed the LP will determine the direction to route the packet, and schedule a switching event at the LP responsible for modeling the selected outgoing port. Like the node-partitioning model, this model requires one LP-to-LP communication to model a hop through the network, namely the switching event.

It is important to note that in this model the switch fabric is shared among the LPs representing the input ports of the router. The main function performed by the switch fabric

is to determine the output port to which the packet should be routed. Logically, the switch fabric model is partitioned among the input ports with any shared state replicated, with identical copies implemented in LPs modeling input ports as needed. If static routing is used, this part of the model does not change during the simulation. If routes do change events may need to be scheduled between LPs modeling the input ports of the router depending on the detail to which such changes are included in the model.

The experiments described later assume the link-partition model described above. We note that in some cases, e.g., if static routing and first-come-first-serve (FCFS) queues are used, one could optimize the implementation to eliminate the transmitting event, and allow the switching event to immediately schedule a switching event at the LP modeling the outgoing link used at the next hop (router). This approach also has important implications with regards to lookahead, as will be discussed momentarily.

2.4 Lookahead

There are four types of delays in packet-switched networks: packet processing delay (typically on the order of microseconds or less), queueing delays (microseconds to milliseconds), transmission delays (microseconds to milliseconds) and propagation delay (depends on the physical distances between the nodes in a network and the speed of the physical medium used for the link, which is usually in the range of 2×10^8 to 3×10^8 m/s, which implies delays ranging from microseconds to milliseconds depending on the length of the link).

For the node model the lookahead is the minimum amount of time for a packet to be transmitted over a link. This includes the speed-of-light propagation delay as well as the

time to transmit the packet over the link, which in turn depends on the link bandwidth and the size of the packet. We refer to the sum of these quantities as the link delay. The processing delay and queueing delays reside within the node and as such, do not affect lookahead in the node model.

For the link model, the lookahead is the minimum amount of time to pass the packet from the input port to the output port of the router. We refer to this as the node delay. The node delay includes the processing delay and queueing delay. Propagation delay and transmission delays do not affect lookahead in the link model.

If static routing and FCFS queues are used, the lookahead can be enhanced in the link model. In particular, the lookahead can also include the propagation and transmission delays. This is significant for links traversing a long distance. The experiments described next do not consider this optimization. In this sense, we regard the experimental results to be conservative with respect to link-partitioned model performance.

CHAPTER 3. PARALLELISM IN THE NODE AND LINK MODELS

The link-partitioning approach offers the possibility of increased parallelism in networks with high degree nodes. However, link partitioning may result in less lookahead than the node partitioning approach. This chapter presents a series of experiments conducted to evaluate this tradeoff and assess the amount of parallelism each approach can obtain for a scale-free network topology. These experiments were conducted using a sequential simulator that simulates the behavior of the parallel simulator. The experiments conducted using an actual parallel simulator are presented in Chapter 4.

The first part of this section explores the parallelism of the link and the node partitioning approaches using YAWNS assuming both approaches have the same lookahead. This establishes a baseline for further experimentation. We then examine the impact of lookahead on parallelism in both models. The second part evaluates the amount of available parallelism independent of the synchronization protocol being used by the simulation. This is done using a critical path analysis.

3.1 Experiment Setup and Terminology

3.1.1 *Sequential Simulator & YAWNS Revisited*

These experiments were conducted using a sequential simulator that emulates a parallel discrete event simulation using the YAWNS synchronization protocol. It is assumed that each event computation requires one unit of time to complete. Further, overheads for communication and synchronization are assumed to be zero. While these are

not realistic assumptions, they do enable us to explore the parallelism that can be obtained by the two partitioning methods separately from other implementation aspects. These experiments assume each LP executes on its own processor.

Recall that simulation using YAWNS processes events in iterations. Each iteration is called an epoch. Since we assume that each processor only has one LP, the longest amount of time taken by any LP in an epoch constitutes the length of that epoch. Besides, since we assume that each event takes one unit time to process, the length of an epoch is the same as the maximum number of events that each LP processes. For example, in epoch 2, if LP_0 processes 100 events and this is the largest number of events processed by any LP in this epoch, then epoch 2 requires 100 units of time to complete.

Further, parallelism in this chapter is defined as the total number of events processed by the simulator divided by the execution time. The execution time is the sum of the maximum numbers of events in each epoch over the entire simulation run. For example, in a simulation with 3 epochs, the maximum number of events process in epochs 1, 2, and 3 are 10, 15, and 5, respectively. Then the execution time is 30 units of time. Suppose the simulation processes 100 events in total, then the parallelism is $p = 100/30$ events/unit time.

3.1.2 Network & Traffic Generation

The network that is modeled is a telecommunication network with a scale-free topology. The network topology was generated using a general-purpose network analysis and graph mining tool call SNAP written in Python [30]. The topology generator creates a

network with a minimum degree M for any node. For these experiments M was set at 20. The node delay in the network corresponds to the lookahead value of the link model.

Packet traffic is generated as follows. Each of the minimum degree nodes is viewed as a traffic source/sink. These nodes might represent routers that connect to a subnetwork of hosts not represented in the network. It is assumed that each source generates traffic following a Poisson distribution, with mean interarrival time λ . This parameter is held constant across all traffic sources. Packet destinations are selected at random among the source/sink nodes in the network. All traffic sources behave in an identical fashion. Packets are routed using Dijkstra's shortest path algorithm assuming all links have equal weight. A single simulation run models the transmission of N packets through the network. The simulations using the node- and link-partitioned models produce identical results and both execute the exact same number of events.

The interarrival time λ is the amount of time between each packet arrival into the system. The inverse of the interarrival time is the arrival rate or the traffic rate. For example, there are 10 packages entering the network every second. The interarrival time is $\lambda = 1/10$ and the arrival rate is $1/\lambda = 10$.

3.2 Baseline Parallelism Using YAWNS

A set of experiments were conducted to compare the parallelism obtained by the two partitioning approaches using the YAWNS synchronization protocol. The parallelism is computed as the total number of events processed by the parallel simulator divided by the execution time, again assuming communications and synchronization require no time. We first compare the parallelism obtained using the node- and link-partitioning models

when both models have the same lookahead values. In other words, the link delay and the node delays are assumed to be same and equal 1 unit of simulation time. This establishes a baseline result on which we can later compare the parallelism of the two partitioning approaches as the lookahead is varied.

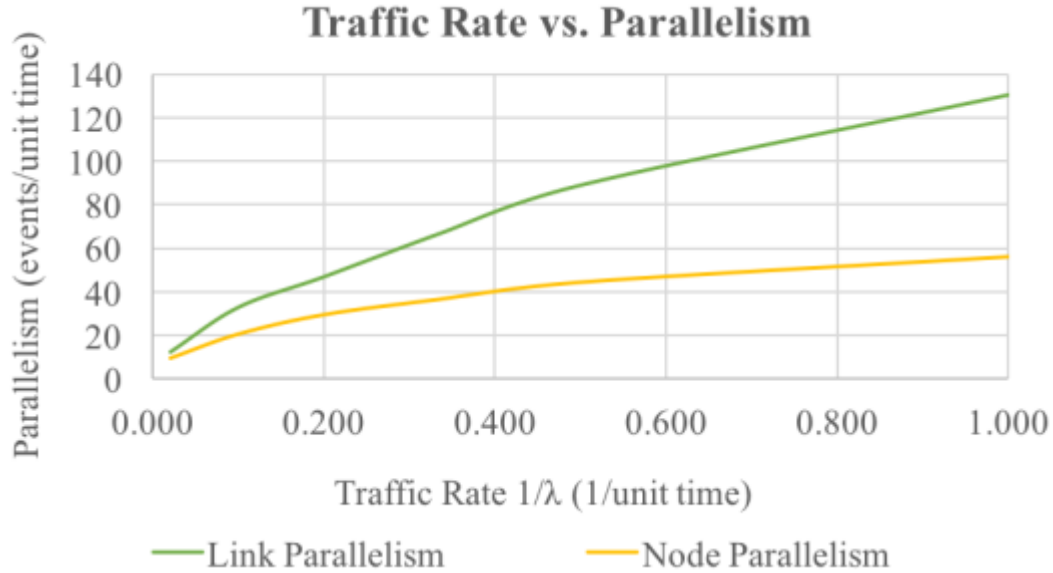


Figure 4 – Parallelism obtained under different traffic rates

The network modeled here has 1,000 nodes and 19,792 links. The simulation routes 50,000 packets through the network. The graph shown in Figure 3 compares the amount of parallelism obtained using the node and link-partitioned models, with the same lookahead value of 1 time unit, for different rates of traffic flow $1/\lambda$. It can be seen that the link-partitioning approach yields greater parallelism. This difference increases as the traffic rate increases.

The link-partitioning model yields better parallelism because of a more balanced distribution of events among the LPs. This can be explained by examining the behavior of

YAWNS during each epoch. Figure 5 and Figure 6 show the minimum, maximum, and average number of events processed in each LP, epoch by epoch, with each partitioning approach when the traffic rate $1/\lambda$ is set to 1.

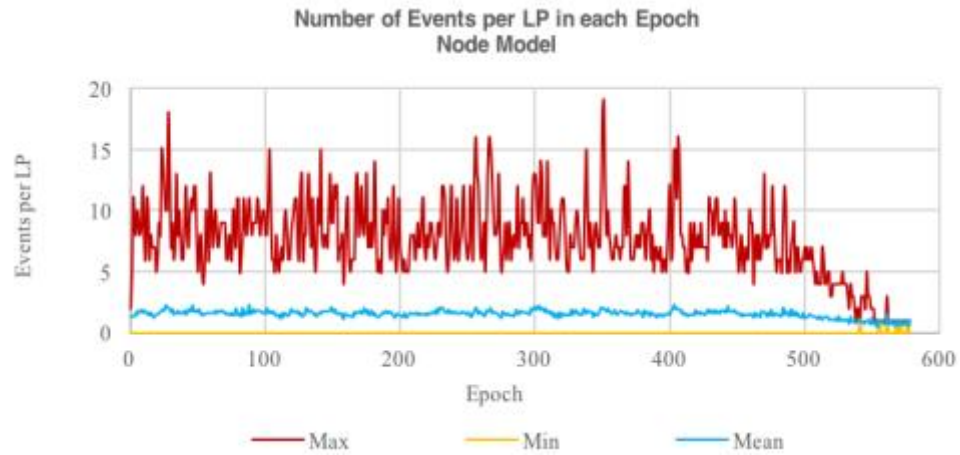


Figure 5 – Epoch-by-epoch measure of the max, min, and average number of events processed by each LP in the Node model

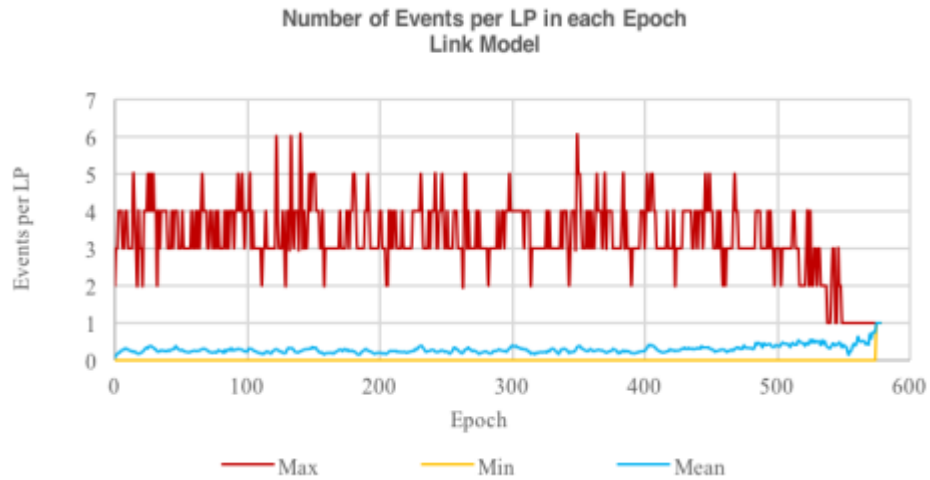


Figure 6 – Epoch-by-epoch measure of the max, min, and average number of events processed by each LP in the Link model

As can be seen, both the node and link models execute the same number of epochs. This is expected because the size of each epoch in simulation time is equal to the lookahead,

which is set to be the same for both models in these initial experiments. Because both models process the same number of total events, this means the total number of “safe” events processed in each epoch is about the same for both models.

The metric of interest in each epoch is the maximum number of safe events in any LP in that epoch. This determines the amount of time required to complete the execution of the epoch. The link-partitioned model exhibits a smaller maximum number of events per LP indicating a more uniform distribution of events among the LPs in each epoch, and less time to complete each epoch. Figure 7 confirms this observation by showing the standard deviation of the number of events per LP in each epoch for the node and link partitioning models. The higher variance in the node model illustrates that the node-partitioned model is more susceptible to bottlenecks where some nodes, i.e., hubs, have many more events to process in the epoch than leaf nodes. This imbalance leads to an inefficient execution because many processors executing leaf node LPs will become idle during the epoch, waiting for hub node LPs to complete processing their events.

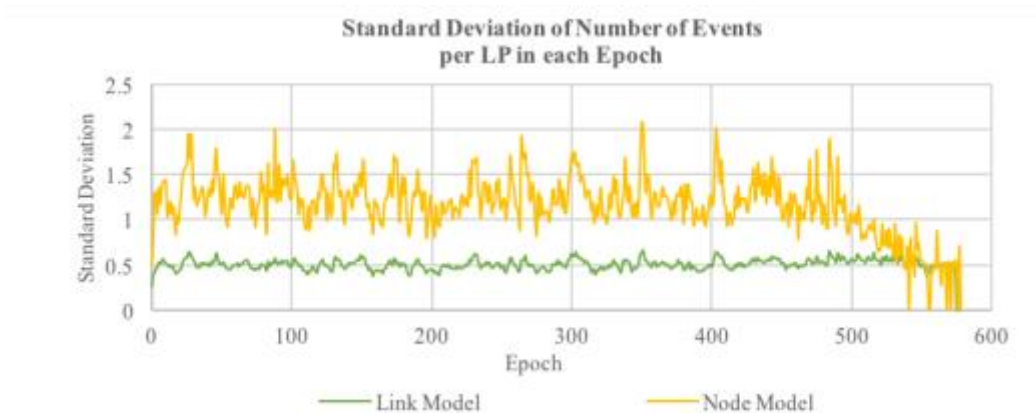


Figure 7 – Standard deviation of the numbers of events per LP per epoch

As the graphs in Figure 5 and Figure 6 show, the node model’s maximum number of events per LP per epoch is approximately twice that of the link-partitioned model. This translates to twice as much parallelism in the link-partitioned model compared to the node-partitioned model. This stems from the irregular nature of scale-free networks’ where high-degree hub nodes tend to have more events to process than leaf nodes. This bottleneck is alleviated by the link model as shown in the graphs.

Additionally, it is known that there can be bottleneck links in scale-free networks – links that tend to have more events to process than others. We analyzed the number of events processed by the top ten busiest nodes and links in each epoch. The busiest nodes and links in each epoch are defined as those that processed the most number of events in that epoch. As shown in figures 8, 9 and 10, even among the busiest links, or bottleneck links, the link model exhibits a smaller maximum number of events per LP as well as a better event distribution comparing to the node model.

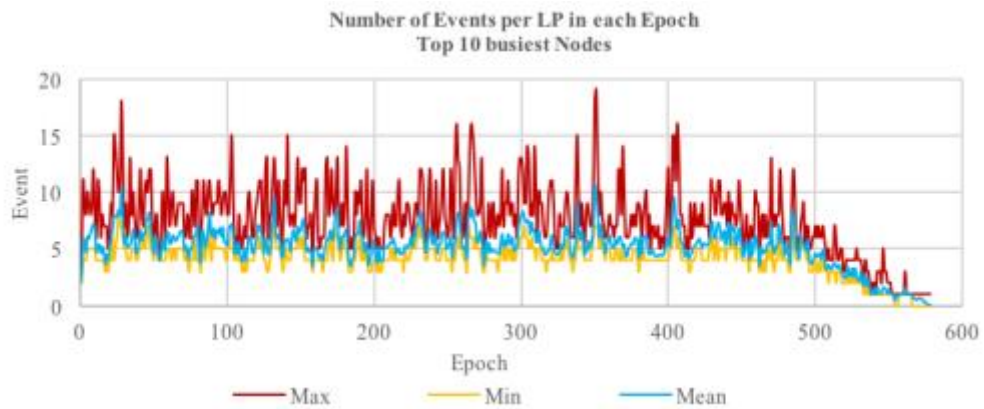


Figure 8 – Epoch-by-epoch measure of the max, min, and average number of events processed by each LP across top 10 busiest nodes

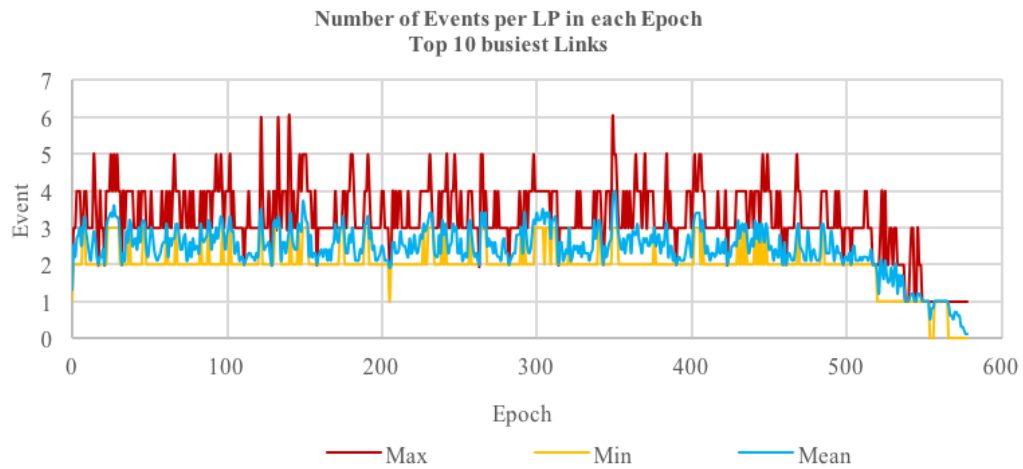


Figure 9 – Epoch-by-epoch measure of the max, min, and average number of events processed by each LP across top 10 busiest links

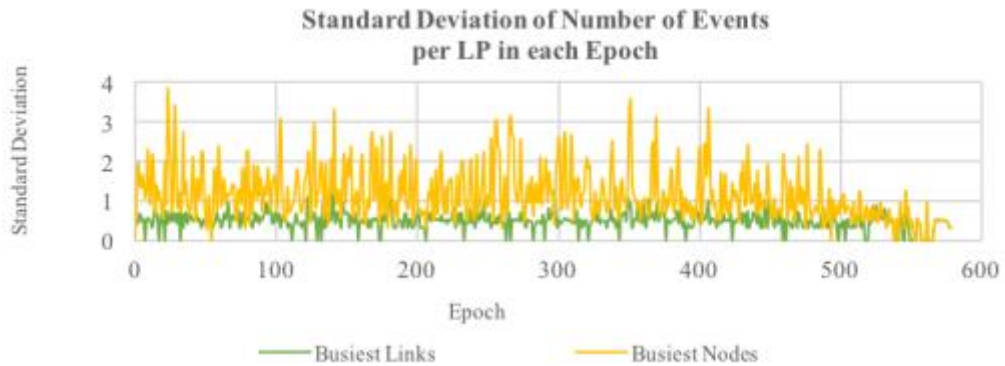


Figure 10 – Standard deviation of the numbers of events processed by each LP across top 10 busiest LPs

We repeated the above experiments with an interarrival time of $\lambda = 5$ (lower traffic rate) and obtained similar results. Better parallelism and event distributions were obtained in the link-partitioned model. The parallelism graphs shown earlier indicate better event distributions are obtained across different traffic rates.

3.3 Impact of Lookahead

In this section, we explore the impact of lookahead on the performance of the two models. It is well known that a small lookahead will negatively impact the performance of conservative synchronization algorithms. The node-partitioned model derives its lookahead from link propagation and transmission delays while the link-partitioned model derives its lookahead from delays within the router. As such, one would expect the node-partitioned model to have better lookahead. We will examine how this impacts the baseline results presented in the previous section for different traffic loads.

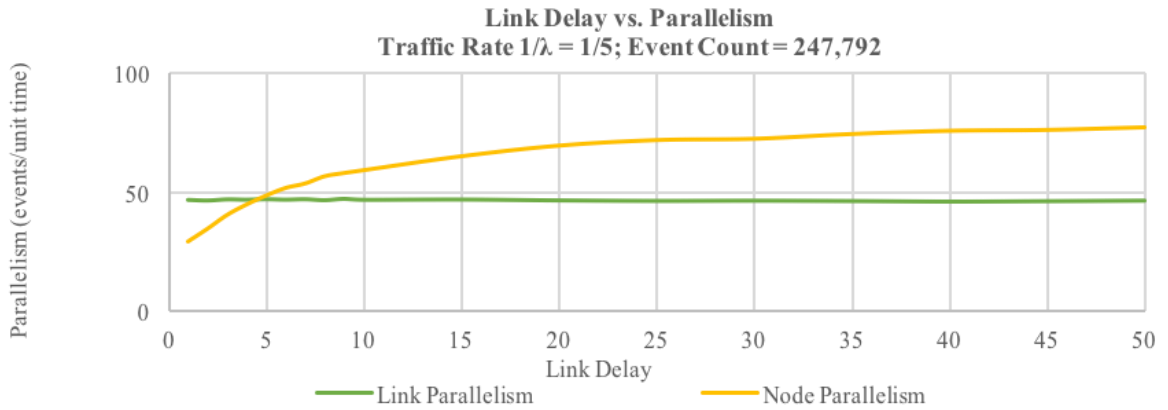


Figure 11 – Parallelism obtained under different link delay values and average interarrival time $\lambda = 5$

The graph in Figure 11 shows the amount of parallelism of the two models for an experiment where we keep the node delay remains fixed at 1 unit of time while increasing the link delay from 1 to 50 units of time. The traffic flow is set at 0.2, i.e., the average interarrival time λ is set to 5.

The link delay is equal to the lookahead in the node model. As the link delay increases, the node model’s parallelism increases, as expected. The link model’s parallelism does not change because the link model’s lookahead is derived from the node

delay which is kept constant. The node model's parallelism increases with the increase of the link delay, and is seen to exceed the link model's parallelism if the link delay is five times that of the node delay.

However, a different result is seen at higher traffic loads. Figures 12, 13 and 14 show parallelism as link delay is increased at traffic loads of 0.33, 0.5, and 1.0, respectively. At higher traffic rates, even with more lookahead the node-partitioning approach does not provide as much parallelism as that of the link-partitioning approach. These results suggest that for lower traffic rates, node-partitioned model parallelism benefits from large lookahead, but at higher traffic rates, this advantage is not sufficient to overcome the bottleneck that occurs in the hub nodes.

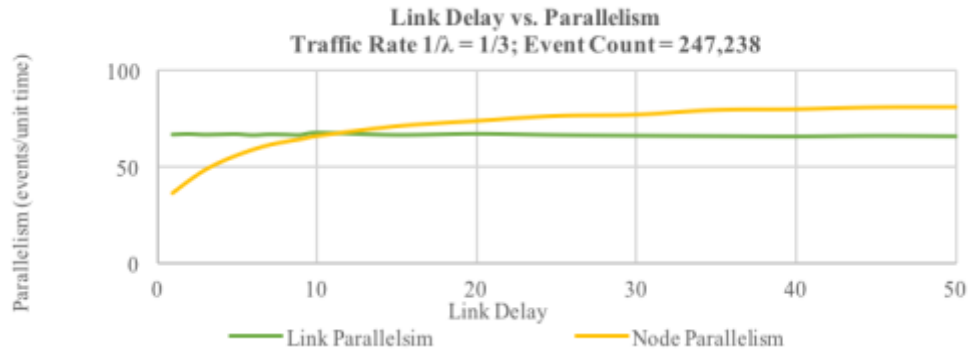


Figure 12 – Parallelism obtained under different link delay values and average interarrival time $\lambda = 3$

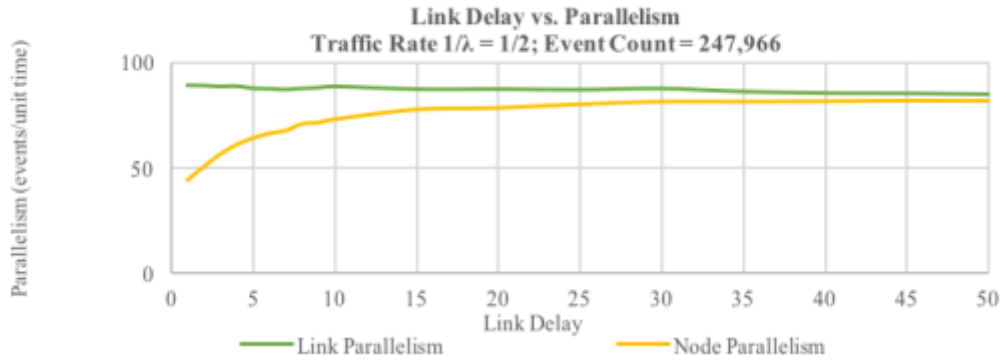


Figure 13 – Parallelism obtained under different link delay values and average interarrival time $\lambda = 2$

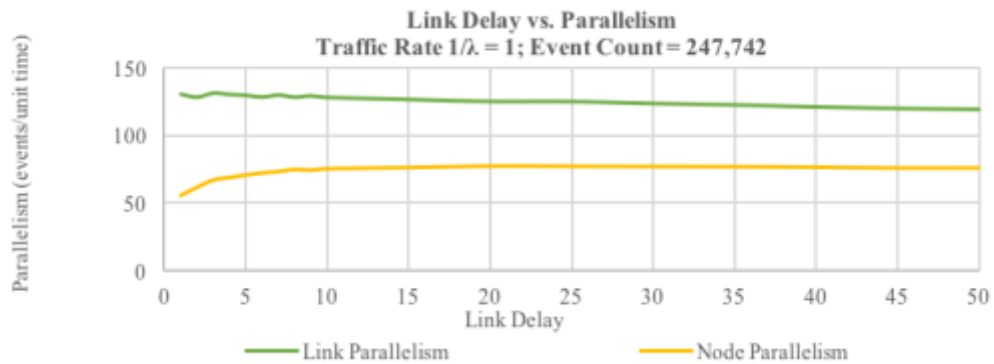


Figure 14 – Parallelism obtained under different link delay values and average interarrival time $\lambda = 1$

3.4 Performance Evaluation Using Critical Path Analysis

In this section, we compare the potential performance of the node- and link-partitioned models using critical path analysis, a parallel simulation performance prediction technique pioneered by Berry and Jefferson [31]. Srinivasan and Reynolds, Jr. also showed that discrete event simulation can be described as an acyclic dependency graph, which turns out to be similar to those referred to as project planning graphs used in operation research [32]. Thus, the well-known techniques of PERT-CPM can be used to determine the critical path of this graph, which will give the lower bound on the execution

time of the simulation. In a study done by Jefferson and Reiher it has been shown that all conservative protocols have a lower bound on execution time and that this lower bound is essentially the time required to traverse the critical path in the simulation. Therefore, by performance critical path analysis on the models, we will be able to determine and compare the lower bounds on the execution time of the PDES using the link approach versus the node approach.

The experiment configuration in this section is the same as the one used in previous sections (i.e., 1,000 nodes, 19,792 links and 50,000 packets.)

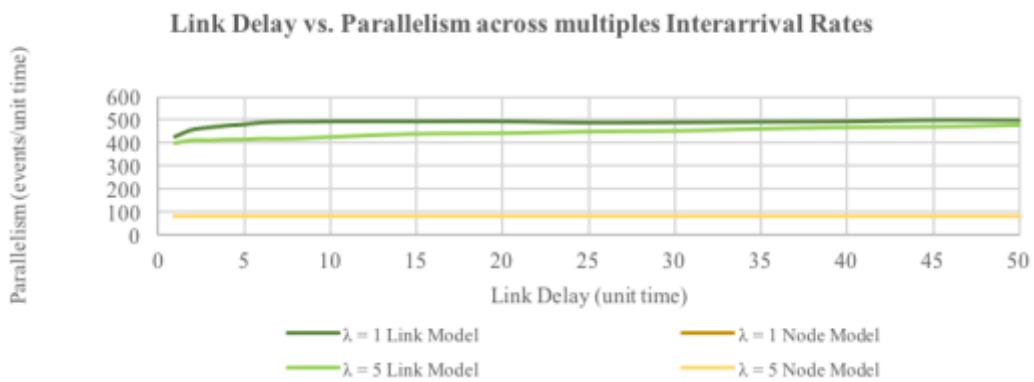


Figure 15 – Parallelism achieved by the Link and the Node models under different interarrival rates (traffic loads) and link delays

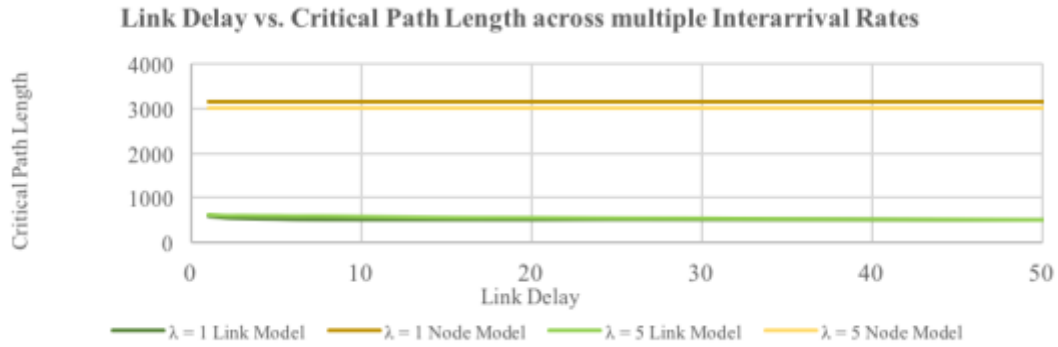


Figure 16 – Critical Path Length of the Link and the Node models under different interarrival rates and link delays

Figure 15 and Figure 16 show the critical path lengths and the amount of parallelism as the link delay as varied. As can be seen, the amount of parallelism, as measured by critical path analysis, in the link model is consistently larger than that of the node model across all lookahead values and traffic loads. Parallelism is computed by the total number of events divided by the execution time of the simulation.

Figure 17 and Figure 18 show the amount of parallelism of the link and the node models as measured based on the YAWNS protocol compared to that measured using CPA. The results indicate that there is much more parallelism available than can be exploited by YAWNS, especially in the link-partitioned model. This is perhaps due to the model's smaller lookahead that diminishes YAWNS performance. We, hypothesize that the link model may yield significantly better performance using another synchronization protocol.

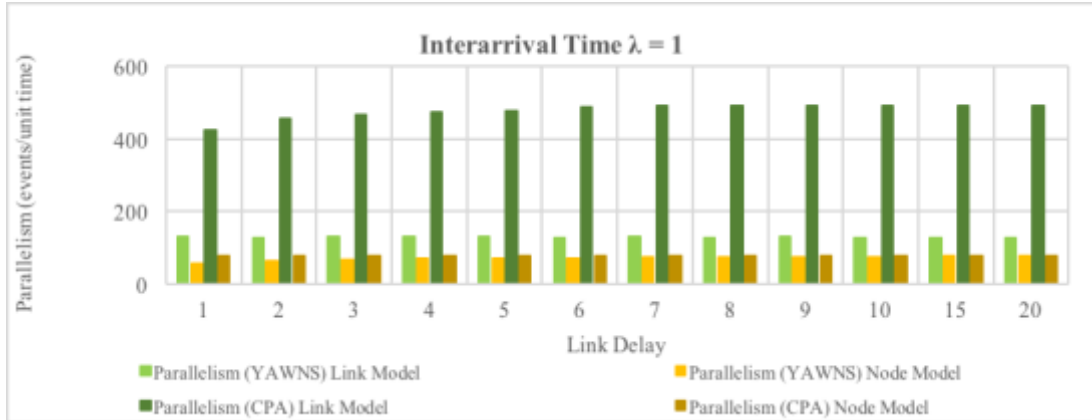


Figure 17 – Parallelism at average interarrival time $\lambda = 1$

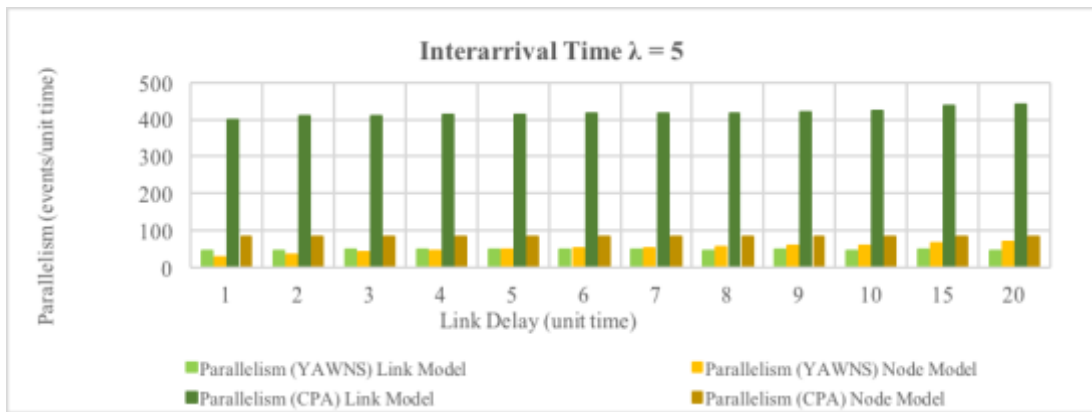


Figure 18 – Parallelism at average interarrival time $\lambda = 5$

3.5 Discussion

The baseline results using the same lookahead for the two models demonstrate that the link model results in better parallelism. This can be attributed to two factors. The first is the fact that the link model simply has more LPs; most networks have more links than nodes. However, the more important factor is that the link model alleviates the bottleneck caused by hub nodes with high degree. LPs modeling hub nodes in the node model must process many more events than most other LPs. The LPs in the link model benefit from a more even distribution of events across the LPs.

When using the YAWNS synchronization protocol, the link model yields more parallelism only when there is sufficient traffic in the network. Due to its inherently small lookahead value, the link model's performance can be worse than the node model at lower traffic rates. On the other hand, at higher traffic rates, the link model proves superior to the node model even when the link model has a much smaller lookahead for the test case examined here.

It may be noted that these results do not exploit lookahead enhancement techniques such as that discussed earlier when static routing and FCFS queues are used. When using this optimization the link-partitioned model would be able to exploit lookahead derived from link delays, the same as the node-partitioned model. In this case, we anticipate link-partitioning will dominate node-partitioning in terms of providing greater parallelism.

Finally, by comparing the parallelism measured using the YAWNS synchronization protocol with that obtained using a critical path analysis, it can be seen that other synchronization protocols may be able to exploit more parallelism in both the link and node models.

CHAPTER 4. PARALLEL PERFORMANCE EVALUATION

This chapter presents experimental results from a PDES implementation to verify the results shown in the previous chapter. Recall that synchronization and communication overheads were assumed to be negligible in previous chapter. The parallel implementation shows the results when these restrictions are relaxed.

We first establish a baseline by measuring parallelism obtained by the link and node models using YAWNS assuming they have the same lookahead. Next, we examine the impact of lookahead on the performance of both models

4.1 Experiment Setup

4.1.1 *Parallel Discrete Event Simulator*

These experiments were conducted using a parallel discrete event simulator implemented in C++ and MPI. The synchronization protocol used is YAWNS.

All parallel experiments were conducted using 4 processors. In both the link and the node models the LPs are assigned randomly to processors. No effort was made to reduce inter-processor communication although this can be improved in future versions of the simulator.

Figure 19 gives a high-level overview of the simulation architecture. Traditionally, each LP has its own future event list (FEL). However, this multi-FEL approach is not required in our experiments. Therefore, our implementation only utilizes one FEL on each processor. This FEL stores all future events for all LPs residing on that processor. Further,

our simulator uses the priority queue from C++ Standard Template Library (STL) as the data structure for the FELs which uses a heap data structure.

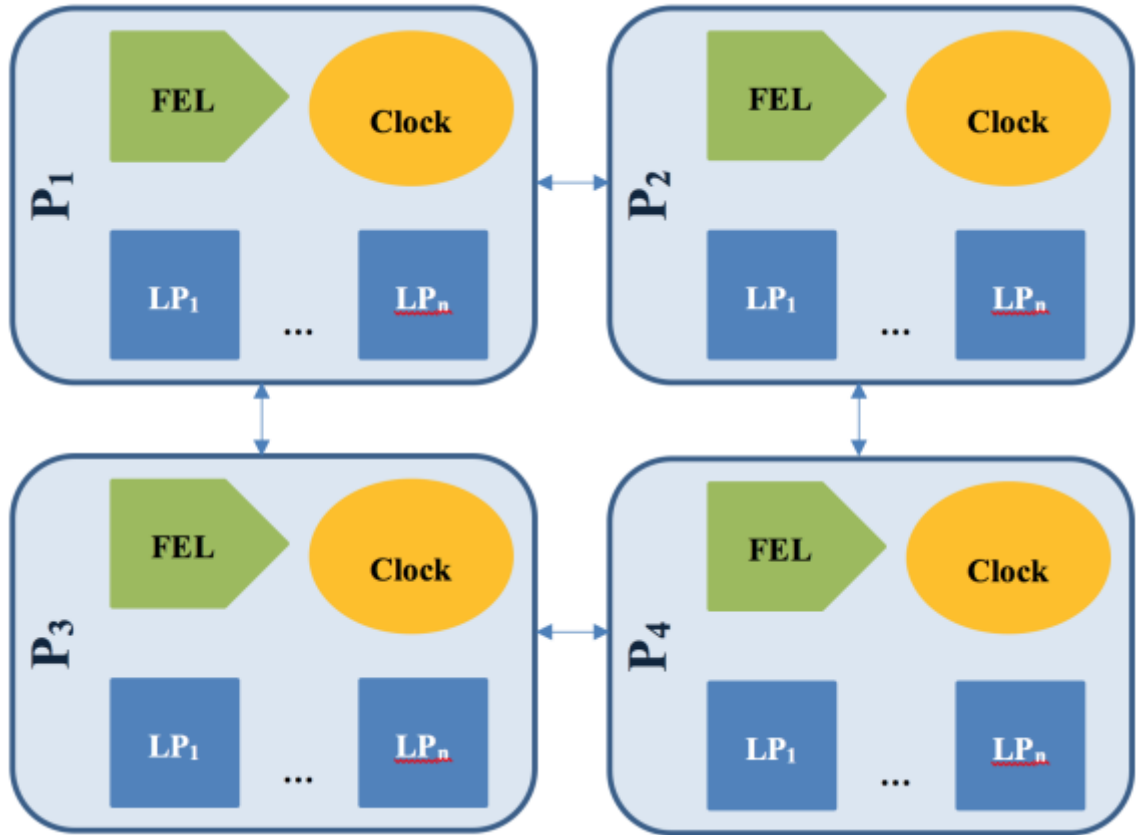


Figure 19 – High-level System Architecture

The simulator's event processing loop (on each processor) can be described as follows.

```

1 while FEL is not empty
2     local_LBTS = timestamp of next event in FEL
3     compute global_LBTS using MPI_Allreduce
4     process all events with timestamp < global_LBTS

```

Figure 20 – Event Processing Loop

4.1.2 *Network & Traffic Generation.*

The network and traffic configuration used here is similar to what described in Section 3.1.2. The network modeled is scale-free consisting of 1,000 nodes and 19,792 links with minimum node degree of 20. The simulation routes 50,000 packets through the network.

4.2 Baseline Parallelism Using YAWNS

To establish the baseline performance for the two models, we measure the number of events processed per second assuming the lookahead is set to 20 milliseconds. We are interested in the performance of the link and node models across different traffic rates. Figure 21 verifies the simulated result shown in Figure 4. In this set of experiments, we varied the interarrival time λ from 1 millisecond to 5 milliseconds. That is a new packet enters the network every 1 to 5 milliseconds.

First, we compare the performance of the node and link models when executing sequentially. Figure 21 shows the event rates obtained from the two models under different traffic rates in a sequential execution of the simulation. As expected, there is no difference between the two models.

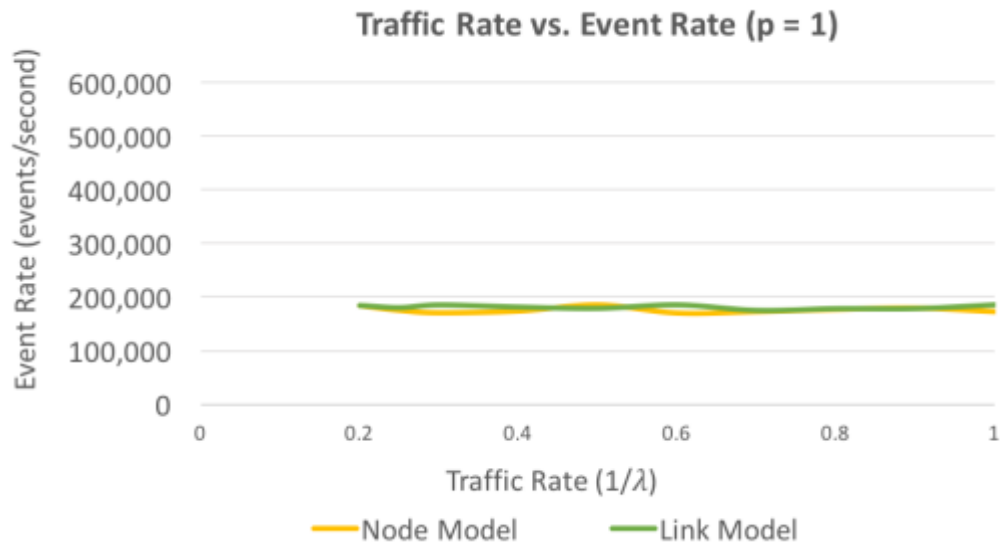


Figure 21 – Event rates across different traffic rates (p = 1)

One the other hand, the performance of the two models differ considerably in parallel execution. Figure 22 shows the event rate of the two models across different traffic rates in a parallel execution of the simulation. At smaller traffic rates, there is not enough traffic to distribute across all LPs during each epoch, especially for the case of the link model where there are 19,792 LPs. The results suggest more parallelism can be exploited when there is enough traffic, or workload, in the network. They also explain the increased performance that is obtained for high levels of traffic.

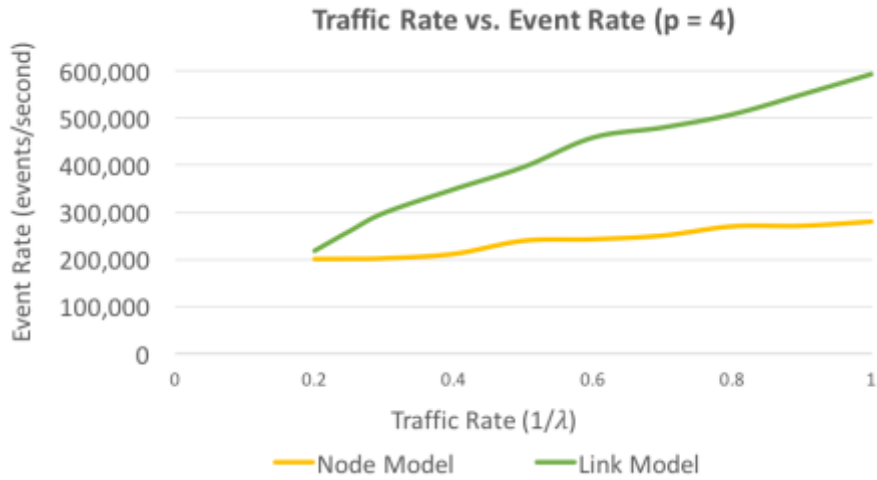


Figure 22 – Event rates across different traffic rates (p = 4)

Next, we attempt to verify our hypothesis that the link model yields a higher event rate due to a better distribution of events across LPs. We examine the performance of the link and the node models during each epoch. In these experiments, the interarrival time λ is set to 1 millisecond ($\lambda = 1, \frac{1}{\lambda} = 1$). Again, the results are consistent with the results discussed in Chapter 3.

As shown in figures 23 and 24, the maximum number of events that each LP has to process in each epoch is higher in the node model compared to the link model. Equivalently, the amount of time needed to complete each epoch is larger in the node model. As the result, the event rate is higher in the link model. This can be attributed to the larger number of LPs that the link model has. However, as we suggested in chapter 3, the more important factor is the distribution of event across LPs.

The link model results in a more uniform distribution of events as shown by the smaller standard deviation in Figure 25. On the other hand, the node model shows more

imbalance in event distribution. This implies there are some LPs with many events to process while most of the LPs are idle.

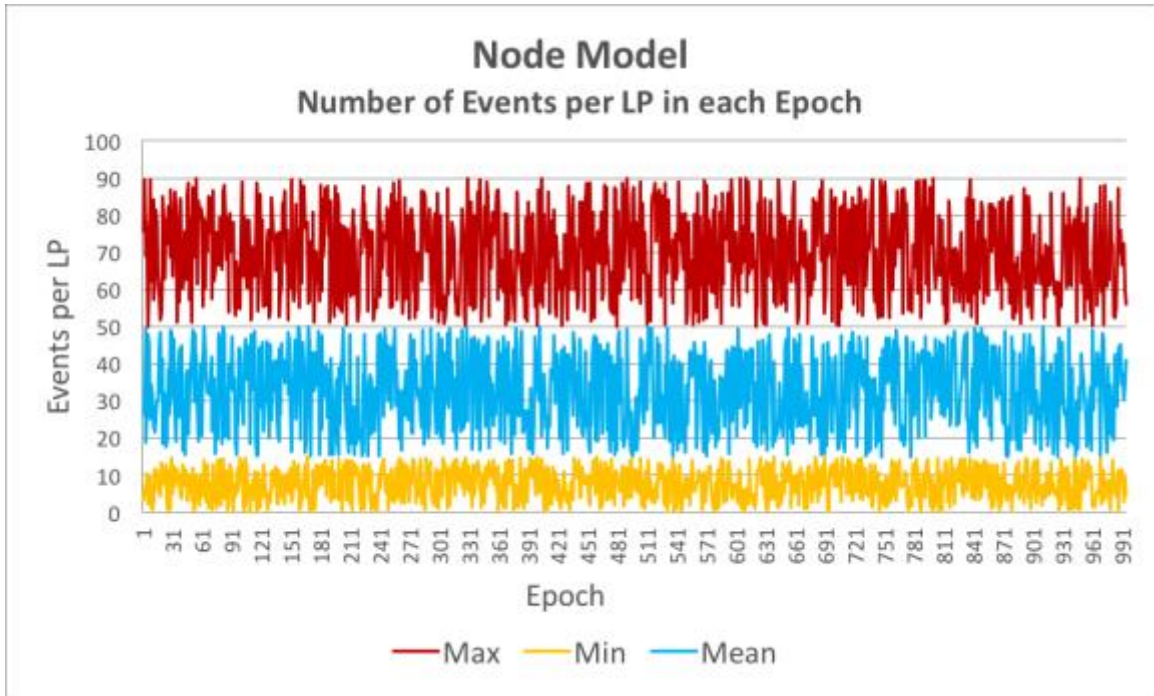


Figure 23 – Epoch-by-epoch measure of the max, min, and average number of events processed by each LP in the Node Model

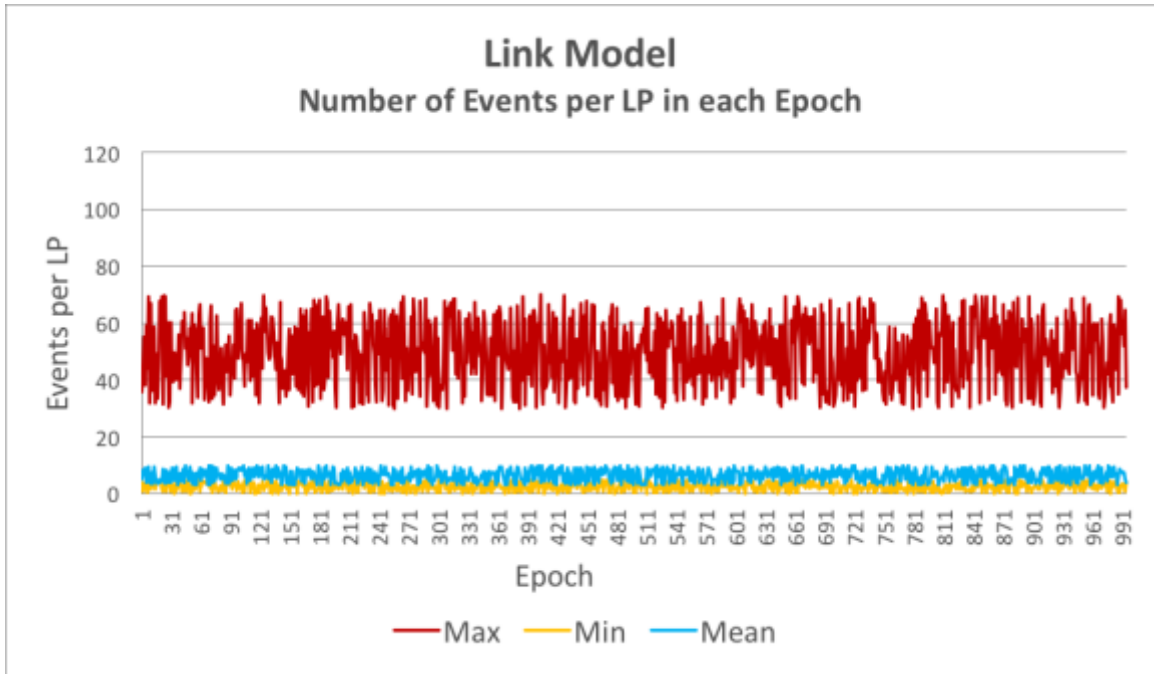


Figure 24 – Epoch-by-epoch measure of the max, min, and average number of events processed by each LP in the Link Model

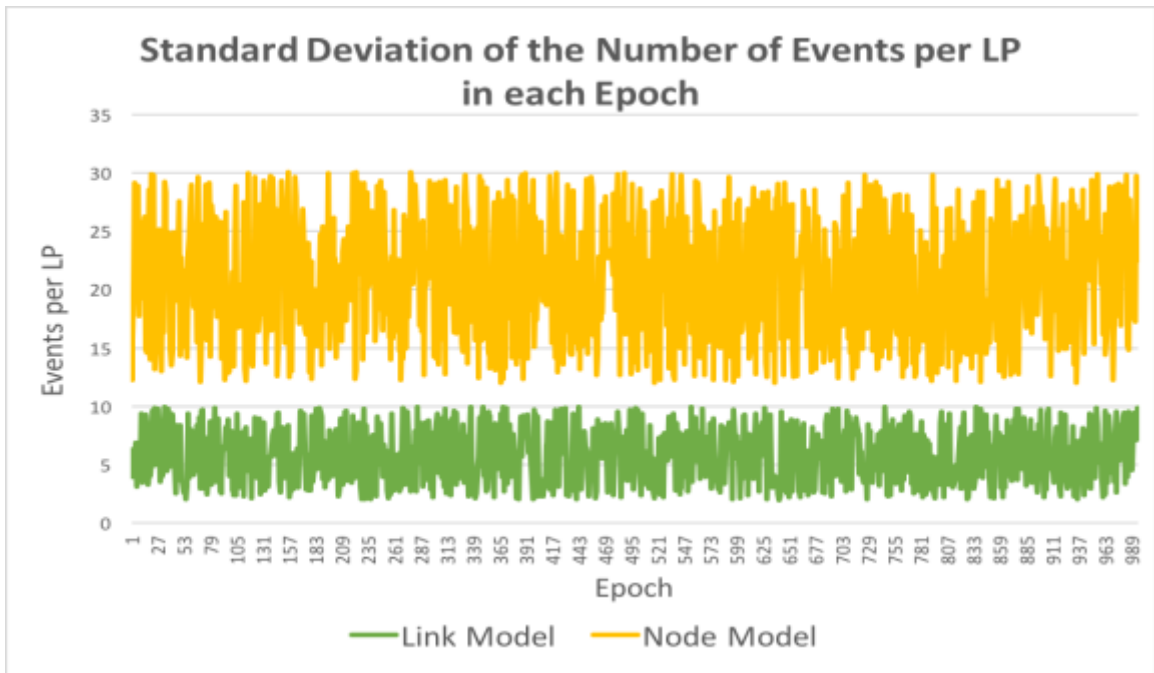


Figure 25 – Standard deviation of the number of events per LP in each epoch

We also examined the behavior of the top ten busiest nodes and links. As shown in the following figures, the maximum number of events per LP per epoch in the link model is consistently smaller than that of the node model. Also, the link model exhibits a smaller standard deviation of the number of events per LP per epoch. In other words, the workload is more equally distributed in the link model than in the node model.

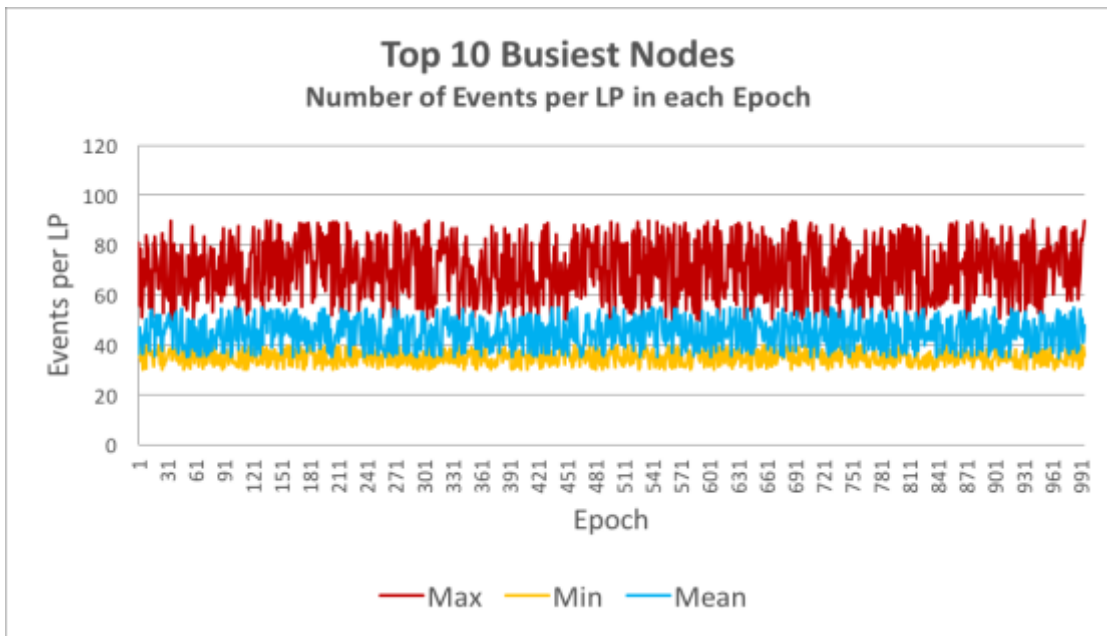


Figure 26 – Number of events per LP per epoch among top 10 busiest nodes

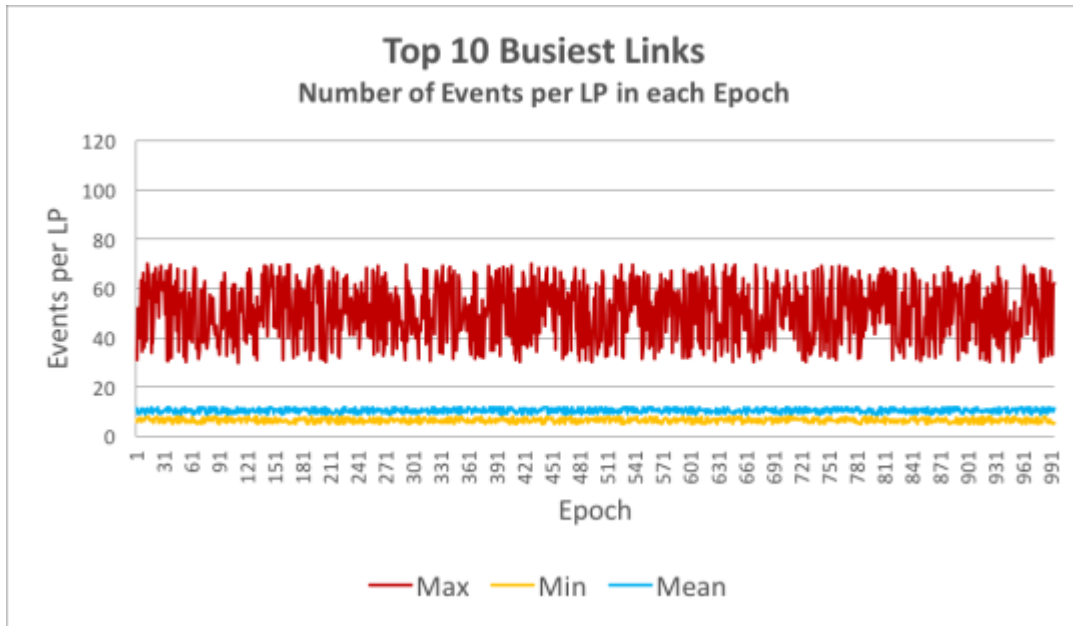


Figure 27 – Number of events per LP per epoch among top 10 busiest links

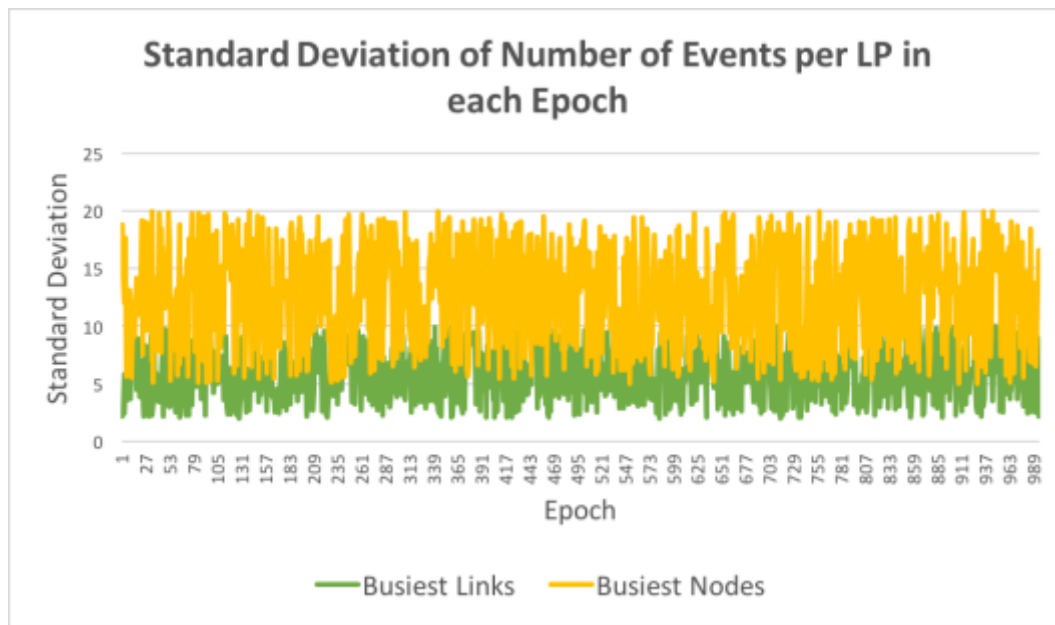


Figure 28 – Standard deviation of the number of events per LP per epoch among the top 10 busiest LPs

4.3 Impact of Lookahead

This section aims to verify the result presented in section 3.3. We will examine the impact of lookahead values on the performance of both models. The experiments were conducted under different traffic rates.

Recall that the node model derives its lookahead value from link propagation and transmission delays (also known as link delay), whereas the link model derives its lookahead from delays within the router (also known as node delay). As the result, the node model has better lookahead since link delays tend to be much larger than node delays.

Figure 29 shows the event rates for the node and link models as the link delay increases from 5 to 40 milliseconds. We keep the node delay constant at 20 milliseconds and interarrival time λ at 5 milliseconds. As the graph shows, the node model's event rate increases as the model's lookahead value increases. Also, since the link model's lookahead value (node delay) is kept constant, it is expected that there is no change in the link model's event rate. Again, this result concur with the finding we showed in section 3.3.

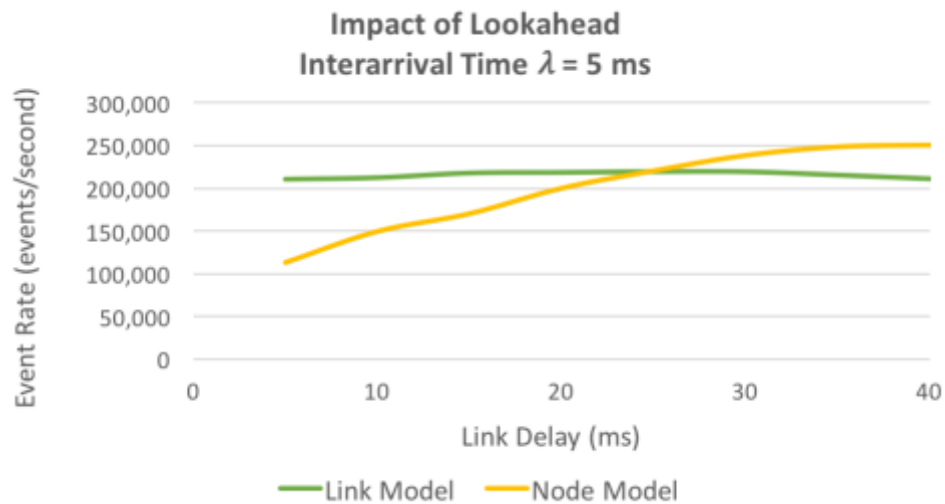


Figure 29 – Impact of lookahead on event rate, $\lambda = 5$ ms

Next, to verify our hypothesis that at higher traffic rates (smaller interarrival times), the node model no longer benefits from large lookahead value, we repeat the above experiments at interarrival time $\lambda = 3$ and $\lambda = 1$.

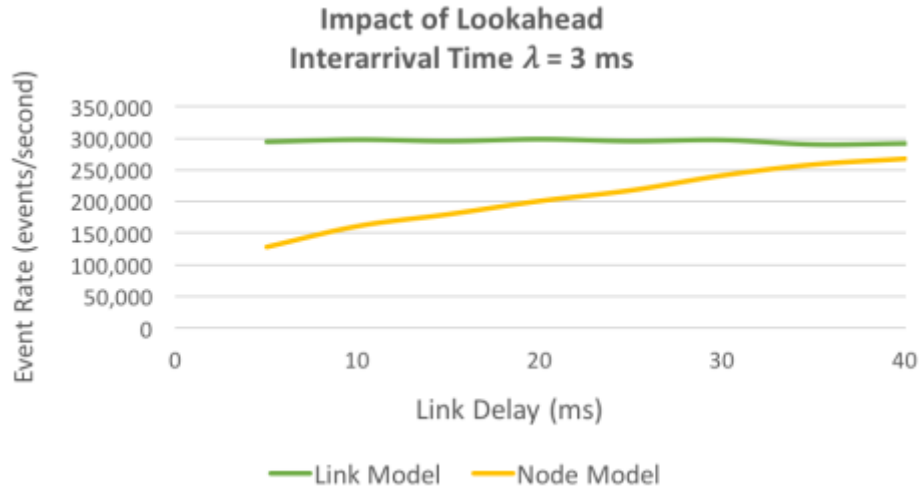


Figure 30 – Impact of lookahead on event rate, $\lambda = 3$ ms

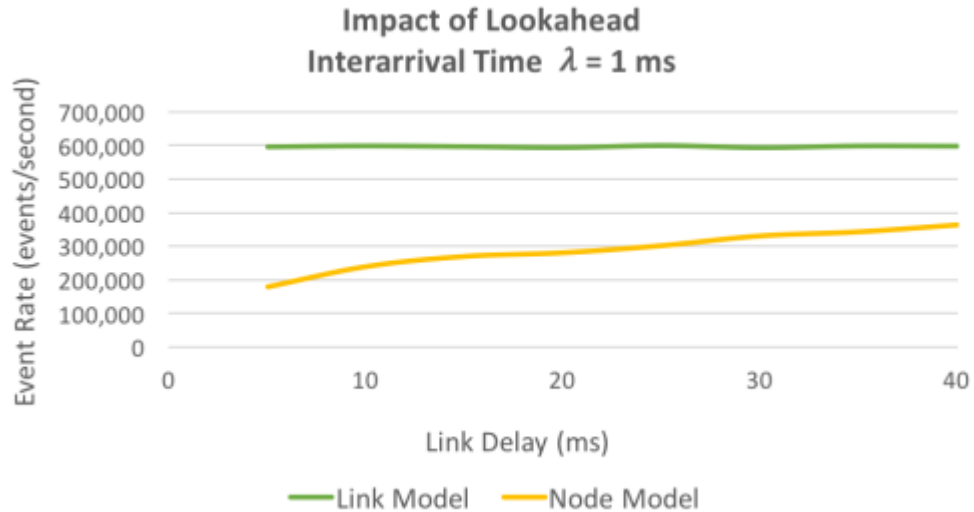


Figure 31 – Impact of lookahead on event rate, $\lambda = 1$ ms

4.4 Discussion

The baseline results in this chapter verifies our hypothesis stated in chapter 3. The link model yields higher event rate, which can be attributed primarily to a more even distribution of workload across LPs.

However, the inherently small lookahead in the link model can hinder the performance of the model. In fact, the link model only proves superior to the node model when there is sufficient traffic in the network

CHAPTER 5. CONCLUSION AND FUTURE WORK

This research proposed and evaluated a technique to improve the performance of parallel discrete event simulation where the application models are scale-free networks. In this section, we summarize the contributions of this work and discuss possible areas of future research.

5.1 Contributions

Motivated by bottlenecks in scale-free networks, we have presented a different approach for modeling networks where each LP represents a network link rather than a node. We have also presented performance comparisons between this approach and the traditional approach where each LP represents a node of the network.

These data suggest that the link model can perform better than the node model although it may suffer from lower lookahead when using YAWNS. On the other hand, this disadvantage is less important when network traffic is high. Finally, data from critical path analysis suggests that much greater levels of parallelism may be obtained using another synchronization protocol.

5.2 Future Work

These results suggest several directions for future research. First, more experimental work is needed to validate that the observations reported here translate to higher parallel performance. Second, more advanced algorithms can be used in partitioning LPs across processors to reduce communication cost. Third, more

extensive experimentation is needed to compare these partitioning approaches across a wider variety of networks and traffic flows. More extensive experiments on different traffic configurations with actual network simulations are also needed in order to verify the results presented here. More broadly, different partitioning approaches for other applications that exhibit scale-free network topologies and other topologies that are not scale-free, but exhibit skewed node degree distributions are needed. Third, analysis of the performance of the link model with other synchronization protocols such as other conservative algorithms or optimistic synchronization protocols are needed to provide better understanding of the link model's actual performance.

REFERENCES

- [1] L. A. Schintler, S. P. Gorman, A. Reggiani, R. Patuelli and P. Nijkamp, "Scale-free Phenomena in Communication Networks: A cross-Atlantic Comparison," in *The 43rd European Congress of the Regional Science Association*, Jyväskylä, Finland, 2003.
- [2] K. M. Chandy and J. Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," *Communications of the ACM - Special issue on simulation modeling and statistical computing*, vol. 24, no. 4, pp. 198-206, April 1981.
- [3] J. K. Peacock, J. W. Wong and E. G. Manning, "Synchronization of Distributed Simulation Using Broadcast Algorithms," *Computer Networks*, vol. 4, pp. 3-10, 1980.
- [4] A. Park, R. M. Fujimoto and K. A. Perumalla, "Conservative Synchronization of Large-Scale Network Simulations," in *Proceedings of the Workshop on Parallel and Distributed Simulation (PADS)*, ACM/IEEE/SCS, 2004.
- [5] B. D. Lubachevsky, "Efficient Distributed Event-driven Simulations of Multiple-loop networks," *Communication of the ACM*, vol. 32, no. 1, pp. 111-123, January 1989.
- [6] R. M. Fujimoto, "Parallel and Distributed Simulation Systems," *Communications of the ACM*, vol. 33, no. 10, pp. 30-53, 10 1990.
- [7] D. R. Jefferson and H. Sowizro, *Fast Concurrent Simulation Using the Time Warp Mechanism*, 1982.
- [8] D. R. Jefferson, "Virtual Time," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 7, no. 3, pp. 404-425, July 1985.
- [9] J. S. Steinman, "Speedes: synchronous parallel environment for emulation and discrete event simulation," in *Advances in Parallel and Distributed Simulation*, 1991.
- [10] J. S. Steinman, "Breathing Time Warp," in *PADS '93 Proceedings of the seventh workshop on Parallel and distributed simulation*, San Diego, CA, 1993.
- [11] P. Reynolds, "A Spectrum Of Options For Parallel Simulation," in *In Proceedings of the 1988 Winter Simulation Conference*, 1988.

- [12] D. West and K. Panesar, "Automatic Incremental State Saving," in *10th Workshop on Parallel and Distributed Simulation*, 1996.
- [13] H. Bauer and C. Sporrer, "Reducing rollback overhead in time-warp based distributed simulation with optimized incremental state saving," in *Simulation Symposium, 1993. Proceedings., 26th Annual*, Arlington, VA, 1993.
- [14] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, Marseille, France, 2008.
- [15] S. Keshav, "REAL: A Network Simulator," 1988.
- [16] A. Park, R. M. Fujimoto, H. Wu, G. F. Riley, M. H. Ammar and K. Perumalla, "Large-Scale Network Simulation: How Big? How Fast?," in *Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, Orlando, Florida, 2003.
- [17] M. Faloutsos, P. Faloutsos and C. Faloutsos, "On Power-law Relationships of the Internet Topology," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication - SIGCOMM '99*, 1999.
- [18] T. Hase, H. Tanaka, Y. Suzuki, S. Nakagawa and H. Kitano, "Structure of Protein Interaction Networks and their Implications on Drug Design," *PLoS Computational Biology* 5, no. 10, no. 2009.
- [19] K. Soramaki, M. L. Bech, J. Arnold and W. E. Beyeler, "The Topology of Interbank Payment Flows," *Physica A: Statistical Mechanics and Its Applications*, vol. 379, no. 11, pp. 317-333, June 2007.
- [20] A. Reka, H. Jeong and A.-L. Barabasi, "Diameter of the World Wide Web," *Nature*, vol. 401, no. 9, pp. 130-131, 9 1999.
- [21] J. Kurose and K. Ross, *Computer Network: A Top-Down Approach*, Pearson, 2013.
- [22] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson and C. Guestrin, "PowerGraph: Distributed Graph-parallel Computation on Natural Graphs," in *In Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation (OSDI'12).*, Berkeley, CA, USA, 2012.
- [23] R. Pearce, M. Gokhale and N. M. Amato, "Scaling Techniques for Massive Scale-Free Graphs in Distributed (External) Memory," in *IEEE 27th International Symposium on Parallel Distributed Processing*, 2013.

- [24] Y. Wu, X. Hou, W. J. Tan and W. Cai, "Efficient Parallel Simulation over Social Contact Network with Skewed Degree Distribution," in *Proceedings of the 2017 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2017.
- [25] G. Kunz, S. Tenbusch, J. Gross and K. Wehrle, "Predicting Runtime Performance Bounds of Expanded Parallel Discrete Event Simulations," in *19th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2011.
- [26] X. Liu and A. A. Chien, "Realistic Large-Scale Online Network Simulations," in *Proceedings of the ACM/IEEE SC 20014 Conference*, 2004.
- [27] G. D'Angelo and S. Ferretti, "Simulation of Scale-free Networks," in *Proceedings of the Second International ICST Conference on Simulation Tools and Techniques*, 2009.
- [28] R. Pienta and R. M. Fujimoto, "On the Parallel Simulation of Scale-free Networks," in *Principles of Advanced and Discrete Simulation*, 2013.
- [29] T. Hruz, S. Geisseler and M. Schöngens, "Parallelism in Simulation and Modeling of Scale-free Complex Networks," in *Parallel Computing 36, no. 8*, 2010.
- [30] J. Leskovec and R. Sosič, "A General Purpose Network Analysis and Graph Mining Library," 2014. [Online]. Available: <http://snap.stanford.edu>.
- [31] O. Berry and D. Jefferson, "Critical Path Analysis of Distributed Simulation," in *Proceedings of the SCS Conference on Distributed Simulation*, 1985.
- [32] S. Srinivasan and P. F. Reynolds, Jr., "On Critical Path Analysis of Parallel Discrete Event Simulations," 1993.