AD HOC DISTRIBUTED SIMULATION:
A METHOD FOR EMBEDDED ONLINE SIMULATIONS

A Dissertation
Presented to
The Academic Faculty

by

Ya-Lin Huang

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computational Science and Engineering, College of Computing

Georgia Institute of Technology
August 2013

AD HOC DISTRIBUTED SIMULATION: A METHOD FOR EMBEDDED ONLINE SIMULATIONS

To my Mother and Father

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratefulness to my advisor, Dr. Richard Fujimoto, for his continuous guidance and support. It is truly my honor to conduct research under his supervision; I am especially thankful for his flexible management and timely, patient advice that allow me to pursue my own research interests without lost in the darkness. Also, he has always provided and encouraged me to participate in every opportunity to widen my horizon of visions and imagination. Besides research, his insightful suggestions regarding the career planning and full supports to my decisions have led me to a bright future that I am looking for. I feel I am one of the luckiest graduate students as he has made my study in Georgia Tech more than a spectacular journey that, I believe, everyone would envy me and that I will cherish forever.

Also, I would like to thank Dr. Christos Alexopoulos for assisting me with the theories in statistics and data analysis, and Dr. Michael Hunter for sharing useful experiences and constructive opinions. Working with them has broadened my profession both horizontally and vertically that I have the opportunities to analyze myriad applications from both the theoretical and the practical aspects. I thank Dr. George Riley and Dr. Richard Vuduc for serving as my committee members and providing unique recommendations in improving this thesis.

A special thank goes to my colleagues and friends for their selfless support professionally and spiritually: Dwayne Henclewood, Hoe Kyoung Kim, Ying Li, Qi Liu, Robert Pienta, Wonho Suh, and George Vulov. I thank the friends in GT Taiwanese Student Association, who are warm and kind that make the homesick less severe. In particular, I must mention the following best buddies, without whom I cannot imagine how to sustain these years: they are Gigi Fan, Hsini Huang, Lillian Liu, and Chunwan Yen. Moreover, I appreciate all the assistance from the School of Computational Science and Engineering and the College of Computing; I thank those who also ease my confusions to the new culture: Lisa Guethlein, Lometa Mitchell, Holly Rush, Michael Terrell, Arlene Washington, and Carolyn Young.

Finally, I owe a great deal to my parents, my elder brother, sister-in-law, nieces, aunts, uncles, cousins, and all my family. Thank them for being there unconditionally, for knowing the least of what I am working on and yet holding the most faith in me. Without their love, this thesis can never be possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The continual growth of computing power in small devices has motivated the development of novel approaches to optimizing operational systems efficiently and effectively. These optimization problems are often so complex that solving them analytically may be difficult, if not prohibited. One method for solving such problems is to use online simulation. However, challenges in using online simulation include the issues of responsiveness (e.g., because of communication delays), scalability, and failure resistance. To tackle these issues, this study proposes embedding online simulations into a network of sensors that monitors the system under investigation.

This thesis explores an approach termed "ad hoc distributed simulation," which is based on embedding online simulations into a sensor network and adding communication and synchronization among simulators to model operational systems. This approach offers several potential advantages over existing approaches: (1) it can provide rapid response to system dynamics as well as efficiency since data exchange is local to the sensor network, (2) it can achieve better scalability to incorporate more sensors, and (3) it can provide better robustness to failures because portions of the system are still under local control. This research addresses several statistical issues in this ad hoc approach: (1) rapid and effective estimation of the input processes at model boundaries, (2) estimation of system-wide performance measures from individual simulator outputs, and (3) correction mechanisms responding to unexpected events or inaccuracies within the model.

This thesis examines ad hoc distributed simulation analytically and experimentally, mainly focusing on the accuracy of predicting the performance of open queueing networks. First, the analytical part formalizes the ad hoc approach and evaluates its accuracy at modeling certain class of open queueing networks with regard to the steady-state system performance measures. This work concerning steady-state metrics is extended to a broader class of networks by an empirical study, which presents evidence to show that the ad hoc approach can generate predictions comparable to those from sequential simulations. Furthermore, a “buffered-area” mechanism is proposed to substantially reduce prediction bias with a moderate increase in execution time.

In addition to those steady-state studies, another empirical study targets the prediction accuracy of the ad hoc approach at open queueing networks with short-term system-state transients. This study demonstrates that, with slight modification to the prior design of the ad hoc queueing simulation method for those steady-state studies, system dynamics can be well modeled. The results, again, support the conclusion that the ad hoc approach is competitive to the sequential simulation method in terms of prediction accuracy.

CHAPTER 1

INTRODUCTION

1.1 Background

Many operational problems such as real-time traffic modeling for reducing congestion are sufficiently complex that they cannot be solved using analytical methods alone. Simulation offers a potentially more effective way to solve such problems. Online simulation tackles these problems by creating a model to mimic the underlying physical system (called the "system under investigation" or SUI in the literature) and driving the model with real-time field data and problem-specific configurations.

Simulation serves as an economical problem-solving approach in many fields. In chip design, for example, modeling circuit behavior before fabrication saves time and money by detecting errors early in the development process [1-3]. Faster-than-real-time simulation enables the analysis of operational systems and strategies to manage operations and/or systems. For instance, weather forecasting simulations [4] have been used to provide early warnings of dangerous weather. Simulation can be applied to other emergency situations such as the spread of wildfires [5] or oil spills [6], and the evacuation during natural or man-made disasters [7]. In some cases, simulation may be the only practical means of analyzing situations because field deployment may be too costly or hazardous, as in the case of transportation management or the spread of disease [8].

Computer simulation is widely used in virtually all science and engineering disciplines in academia, industry, and government [9]. The applications typically fit into one of the following three categories:

1. **System analysis.** Simulation facilitates the analysis, design, and optimization of systems. It may be performed before building a system in order to reduce design errors or to optimize the design. It can also be used after deployment to tune system parameters for better operational performance. Example applications include weather prediction systems, computer systems, and logistics, among many others.
2. **Training and games.** Simulations embedded in a virtual environment mimic the behavior of the modeled objects or phenomena with which participants interact. The simulation models respond dynamically based on the input from participants. In most cases, participants are human beings for the training or entertainment purposes. Examples of the former category include the military and emergency training exercises while video games belong to the latter.
3. **Testing and evaluation.** Simulation is also an approach to testing and evaluating physical components in a well-controlled and easy-to-measure environment. Compared to field testing, this approach typically costs less and is much safer. For example, simulation can be more economical to test and exercise the launch of missiles as opposed to filed tests.

This thesis focuses exclusively on the simulations used for the system analysis and management of operational systems.

1.1.1 Online Simulation

An online simulation is a predictive computational model that utilizes the data pertaining to the current state of a SUI to project future system states. It is typically used to manage and optimize operational systems in real time. Providing real-world, up-to-date data to an executing simulation enables the model to take into consideration the current operation conditions of the SUI. In other words, this allows model adaptation so that in principle the SUI can be modeled more accurately. As a consequence, the accuracy of predictions is potentially improved, which facilitates system monitoring and control. Note that a successful online simulation must execute faster than real time to be useful for predicting future system states.

Online simulation is also referred to as dynamic data-driven application systems [10], symbiotic simulation [11], and cyber-physical systems [12] in the literature. These approaches have been widely studied and applied to various science and engineering disciplines for a myriad of purposes [13]. One typical application is to optimize the operations of a physical system. For example, in an emergency situation, alternate evacuation scenarios may be modeled and evaluated in order to minimize evacuation time. The evacuation plan may need to adapt as the evacuation evolves when unforeseen events arise. Other online simulation applications include planning path for unmanned aerial vehicles [14], tuning parameter for computer networks [15], the management of semiconductor manufacturing systems [16], and the optimization of surface transportation systems [17, 18]. Furthermore, online simulation helps us better understand and gain insights into the physical systems that are difficult or impossible to observe. Example applications include identifying accidents using cell phone data [19, 20] and determining the boundary conditions of fluid-thermal systems [21, 22].

Many existing approaches to online simulation are centralized: the sensor data are transmitted to a specific location for simulation, and the simulation results (instructions or stimuli) are then sent back to the field to optimize the physical system. This centralized paradigm can cause a number of problems. For one, communication by sensors to a central site can be problematic in so far as it consumes energy and may incur large delays. For some sensor nodes, the ratio of the power consumption for communication compared to that for computation can be up to 10,000 to 1 [23]. Also, communication failures may impact the effectiveness of online simulation; failures must be expected in sensor networks, especially those operating under harsh environmental conditions (e.g., inclement weather resulting in radio interference). Another concern is scalability—in the centralized paradigm, modeling a large physical system may require an excessive amount of computing resources and communication bandwidth in order to produce results in a timely fashion. All these problems motivate this thesis work, which will involve replacing the centralized method with one that embeds online simulations within the sensor network itself.

1.1.2 Parallel and Distributed Simulation

Parallel and distributed simulation systems carry out a simulation execution using multiple processors [24]. These processors may be on the same machine (e.g., a multi-core or multi-processor machine) or on a number of machines connected via a computer network. Distinguishing a parallel simulation from a distributed one is largely based on the computer system that executes the simulation. Parallel simulations typically run on a computer system in which the processors are tightly coupled, such as a computer cluster composed of homogeneous processors confined to a physically-bounded area (say, a room) and connected through a fast, customized communication network. By contrast, distributed simulations are often (but not always) executed on heterogeneous processors, and the communication takes place via general-purpose networks such as local area networks, wide area networks, or the Internet. Although the simulation methodology proposed in this thesis focuses on distributed simulations, the fundamental mechanisms can be applied to parallel simulations; that is, the computer system is of secondary importance.

Parallel and distributed simulation benefits the applications that are difficult or impossible to be carried out by a single processor (referred to as "sequential simulations" in the literature). It offers several potential advantages:

1. **Reduced execution time.** Subdividing a simulation execution into *n* smaller computations may speed up the execution by up to a factor of *n*. Fast execution enables simulation to be used for the applications where the time to complete the simulation would otherwise be prohibitive. It is also important in the situations where real-time response is required, e.g., in virtual environment applications.
2. **Model integration.** Parallel and distributed simulation can integrate the small models that are not co-located on a single computer; each model is executed by a different simulator and they cooperatively model a larger physical system. One example where this may arise is when the participating simulators are in geographically separated locations and the relocation is not practical, e.g., due to the specialized resources at certain locations. For example, the simulators for the ships and aircraft involved in a simulated battle may be placed at different military bases. Another reason to "connect" simulators is that porting software to a new platform can be costly because it may require extensive code modification. Moreover, porting may not be possible for commercial simulation software when the source code is not available.
3. **Failure resistance.** Distributed computing enables parallel and distributed simulation to tolerate certain failures. For example, processor failures can be

detected so that the affected executions can be transferred to other processors, given the mechanisms to detect and recover from failures are implemented to realize this capability.

1.1.2.1 Synchronization Mechanism

A parallel/distributed simulation contains a collection of sequential simulators (called "logical processes" or LPs in the literature), each modeling a portion of the SUI over time. The portion of the SUI modeled by an LP is referred to as a "physical process." The interactions among physical processes are modeled by exchanging messages among the corresponding LPs. Each message results in one or more events being scheduled at the receiving LP at the simulation time(s) specified in the message. Each LP must process all its events, including those triggered by messages, in the order of the time stamps associated with the events (or the "time-stamp order"). Failure to comply this order may induce errors, referred to as "causality errors," and the challenge to ensure this time-stamp-order event processing is referred to as the "synchronization problem."

The solutions to the synchronization problem are typically categorized into two—conservative and optimistic. In conservative synchronization algorithms, an event can be processed only if one can guarantee that another event will not be scheduled with a time stamp prior to the current simulation time (i.e., the time stamp of the event that is being processed). These algorithms can be further classified as asynchronous or synchronous, which differ according to whether or not a global synchronization mechanism is used. Asynchronous algorithms contain no such global synchronization point. An example is the well-known Chandy/Misra/Bryant method [25, 26], in which null messages are used to prevent LPs from entering deadlock states. Other examples tackle the deadlock issue by deadlock detection and recovery [27, 28]. By contrast, synchronous algorithms separate synchronization activities from the simulation computation over time. That is, LPs process and generate events/messages only in simulation phases, but not during synchronization phases. A synchronization phase may be defined using barriers [29] or time windows [30-33], during which a set of events that can be processed in the next simulation phase is identified. The methods for identifying such events include the bounded lag algorithm [34] as well as the algorithms taking the advantage of the distance information regarding the modeled objects [35].

In optimistic synchronization algorithms, LPs process events without concerning that messages may arrive in the past. If such message-in-the-past incidents occur, a recovery mechanism is activated. The most influential optimistic synchronization algorithm is the Time Warp method developed by Jefferson [36]. In this method, the recovery mechanism relies on rollbacks and, hence, this method is also referred to as a "rollback-based method." The Time Warp method contains two parts: local control and global control. The local control concerns the recovery mechanism, which is performed independently on each LP, while the global control requires all LPs to participate in some way to deal with issues such as memory reclamation.

The local control (i.e., the recovery mechanism) in the Time Warp method works as follows. It is triggered when an LP receives a message in its past, i.e., the message has a smaller time stamp than the current simulation time of the LP. The LP rolls back to a point in time equal or prior to the time-stamp value on the receiving message so that the processing of the events with larger time stamps is revoked. This entails: (1) restoring the local state of the LP and (2) "un-sending" the messages that were sent by the LP for those rolled-back events. The state restoration may be based on copy state saving, where a copy of an LP's state variables is made prior to processing every event, or incremental state saving, where modifications to state variables are logged so that they can be undone. Also, the state can be reconstructed by performing the inverse computation for the rolled-back events [37]. To un-send messages, Time Warp introduces "anti-messages." An anti-message is a copy of a previously sent message that differs from the original one only in a flag bit indicating it is an anti-message. Un-sending a message is accomplished by sending the corresponding anti-message to cancel the effect of the original message (or called the "positive message"). When an LP receives an anti-message, it deletes the corresponding positive message. If this positive message has already been processed, the LP must roll back the processing of the message. In this case, one rollback may result in many other LPs rolling back, and this phenomenon is referred to as "cascaded rollbacks."

The global control of the Time Warp method computes a value known as "global virtual time" (GVT). GVT is a lower bound on the simulation time of any future rollback. Hence, the state information pertaining to the simulation prior to GVT can be discarded, which enables the Time Warp programs to reclaim the memory. Also, GVT serves as the reference time point regarding the execution of certain computations that cannot be rolled back, e.g., I/O operations. These operations are committed as GVT advances past the time points associated with the operations. Computing GVT can be done asynchronously and the two well-known algorithms are the Samadi's method, which tags acknowledge messages to ensure correctness [38], and the Mattern's method, which relies on two-phase cuts [39].

1.1.2.2 High Level Architecture

The High Level Architecture (HLA), developed in the 1990's by the U.S.

Department of Defense, is an approach to integrating different simulators (or “federates”) into a single distributed simulation system (or a “federation”). It aims at simulation interoperability and reusability. These two properties are beneficial to the construction of future simulations for new purposes because reusing existing simulators can dramatically reduce the development cost.

The HLA is not specific to the defense applications. Instead, it has been standardized [40-42] by the Institute of Electrical and Electronics Engineers (IEEE) with the objective to encompass simulations of any purpose, in particularly those for system analysis, virtual environments, and testing and evaluation. To accomplish this, the HLA provides sufficient freedom to federates regarding their individual simulation tasks while guards the interactions among them with a general, yet compact, set of rules. Specifically, federates can be implemented by any software or programming language, and run on any platform. They interact, e.g., for message exchange or synchronization, by invoking the services that are implemented by a run-time infrastructure (RTI) software. However, the specifics of the RTI implementation are not part of the HLA.

The HLA includes three components: rules [40], object model template (OMT) [41], and interface specification [42].

1. **Rules.** The rules are ten principles that guide the development of federates and federations. For example, one rule specifies that a federation must document the objects shared among the federates by defining a federation object model (FOM), which must follow the OMT format. Another rule requires that each federate must document what objects it can share in a simulation object model (SOM), which also must comply with the OMT format. Furthermore, all information exchanges are regulated to the services defined in the interface

specification. Note that since the services are independent of any simulation application, the instances of objects reside within federates (which implement the simulation application), not in the RTI software (which implements the services).

2. **Object model template.** All shared objects must be documented, including their attributes and the interactions/relationships among each other. The HLA only regulates the format, that is the OMT format, for describing the information, but nothing regarding the content.
3. **Interface specification.** The HLA provides a number of services for federates to interact with each other. These services are defined in the interface specification and implemented based on the RTI. The services are categorized into six classes: (1) "federate management" defines how federates create, join, and leave federations; (2) "declaration management" allows federates to announce their intentions to publish or subscribe object data; (3) "object management" concerns the creation, deletion, and modification of object instances and their attributes; (4) "ownership management" is responsible for transferring the ownership of object attributes; (5) "time management" controls the advance of simulation time, or, in other words, provides synchronization services to federates; and (6) "data distribution management" supports the routing of data from publishing federates to subscribing federates.

The HLA has been approved as an IEEE standard in 2000. Since then, modifications have been developed concerning a myriad of aspects such as scalability, flexibility, supporting new technologies, and fault tolerance. The latest standard (version 2010) is based on the "HLA Evolved," and the important updates on the standard are summarized in [43].

1.1.3 Ad Hoc Distributed Simulation

Ad hoc distributed simulation is an approach to real-time system monitoring, analysis, and operation optimization that involves embedding online simulations into the sensor network covering the physical system of interest. An ad hoc distributed simulation contains a collection of autonomous LPs, each modeling a partial physical system, referred to as the "coverage area" or "modeling area." The LPs select individual modeling areas based on their own local objectives while they collectively model the entire physical system. In this way, an ad hoc distributed simulation is regarded as being constructed in a bottom-up fashion, as opposed to the top-down approach used by traditional distributed simulations where physical systems are partitioned into non-overlapping segments. As to the synchronization among LPs, this ad hoc approach adopts an optimistic, rollback-based approach. The features of the ad hoc approach will be elaborated in the rest of this subsection along with an illustrative example depicted in Figure 1. In the figure, the grid represents the physical system; the LPs are co-located with the sensors in the centers of individual modeling areas, which are shown in rectangular boxes with curved corners.

The designation of the area(s) modeled by each LP can be arbitrary. In some cases, an LP may select its modeling area in order to perform some local monitoring tasks such as predicting the travel time for a particular vehicle in a transportation network. The work by Hunter et al. serves as an example in which the LPs reside in vehicles while the sensors may be co-located with road-side cabinets or traffic signals [17, 18]. In other cases, an LP may simply model the area covered by the sensor which the LP co-locates with for predicting local future states. Figure 1 shows such an example where each LP models the designated rectangular region covered by the corresponding sensor. This configuration distributes computational load and potentially reduces data transmission costs.

This feature of arbitrarily assigning modeling areas may lead to the situations in which some parts of a physical system are modeled by multiple LPs (e.g., V_a in Figure 1) while others are left uncovered (e.g., V_b in Figure 1). The former case introduces redundancy, which offers the potential for greater robustness (or resilience to failures); this differentiates the ad hoc approach from traditional distributed simulations, in which the physical system is perfectly partitioned so that each segment is modeled by exactly one LP. Redundancy may also increase prediction accuracy because multiple LPs can provide state predictions. Furthermore, widely varying predictions by different LPs may be indicative of a malfunctioning LP (e.g., caused by incorrect model assumptions or inaccurate sensor data) or an LP that has detected changes in system behavior in advance of other LPs.

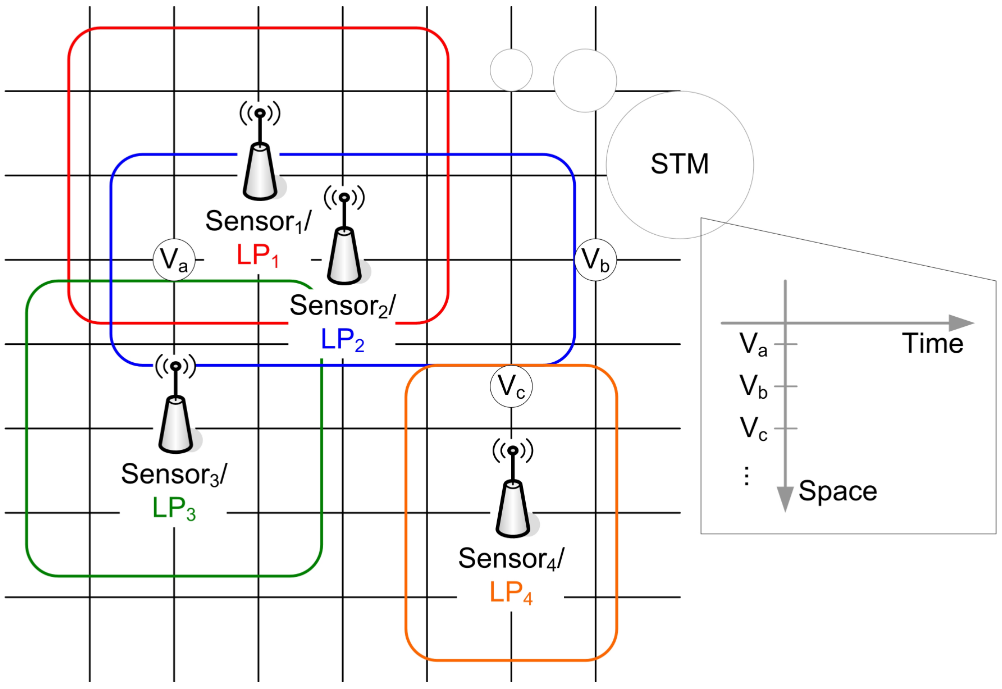


Figure 1: Illustration of an Ad Hoc Distributed Simulation

The ad hoc approach allows LPs to change their modeling areas during simulation execution, as might be the case for the applications involving mobile components. For example, in an application of monitoring a surface transportation system, an LP deployed in vehicles or handheld computing devices (e.g., GPS devices or smart phones) may adjust its modeling area based on vehicle movements and/or driver's desire for the predictions concerning the remainder of the planned route.

In an ad hoc distributed simulation, LPs share current and future state predictions via a construct called "space time memory" (STM); see Figure 1 [44, 45]. The STM holds time-stamped system state updates from different LPs; the time interval associated with an update specifies when the update is valid. Hence, to read a system state, LPs specify not only the name of the desired variable but also a time stamp. Figure 1 depicts several examples of read/write operations: both LP_1 and LP_2 update (i.e., write) V_a while LP_3 reads V_a. This method of information exchange is different from that used in traditional distributed simulations where the LPs are notified by the events that are exchanged through messages. Note that since multiple LPs may generate state updates for the same variable, the STM should aggregate these state updates. In addition, realization of the STM depends on the physical system in which an ad hoc distributed simulation is embedded. Ideally, the STM should be distributed over the simulation infrastructure, but could, in principle, be implemented in a centralized manner.

The optimistic synchronization method in the ad hoc approach allows LPs to advance individual simulation executions without necessarily waiting for other LPs. An example situation would be the one in which certain desired state information of an LP is not available upon request; instead of waiting, the LP may approximate the missing state information based on the past state information or that of similar variables/objects, and then advance the simulation. Since the approximation could be inaccurate, LPs have to detect the disagreement between the used data and the predictions later produced by other LPs or the field data from sensors, and initiate a rollback if necessary. Similarly, if the data received previously by an LP is determined to be in error, a rollback mechanism should be used to correct the erroneous data. The rollback mechanism is similar to that in the Time Warp method. That is, it rewinds the target LP back to the time prior to when it used the invalid data, restores the previous system state, and restarts the simulation with the "correct" data according to currently-available updates. Furthermore, the predictions produced by the LP prior to this rollback might be contaminated by the invalid data; these predictions should be revoked, which results in re-computing the projections. The re-computation may further cause other LPs to be rolled back if the revoked predictions indeed contaminated the data that those LPs have used. As a consequence, cascaded rollbacks are possible.

1.2 Problem Statement and Research Challenges

Ad hoc distributed simulation introduces a number of analysis issues. This section organizes the discussion of these issues around the life cycle of a modeling and simulation study, which includes the following steps: (1) problem formulation, (2) conceptual model development/validation and data collection, (3) simulation program development, verification, and validation, (4) experimental design and execution, (5) output analysis, and (6) result documentation [46]. The following discussion particularly focuses on steps 2, 4, and 5 where the choice of simulation analysis methodologies influences the design. The issues pertaining to input data analysis (data collection and data model construction) are discussed in Section 1.2.1. Section 1.2.2 explores those in experiment design; Section 1.2.3, model execution and adaptation; and Section 1.2.4, output analysis.

1.2.1 Input Data Analysis

Each LP in an ad hoc distributed simulation is essentially a conventional sequential simulation with the exceptions that its input processes must be estimated using real-time data and it is subject to rollbacks. Since the input data used to construct appropriate input process models are generated by LPs or sensors, they may have not only complex autocorrelation structures but also cross-dependencies owing to the overlapping nature of LPs' modeling areas. Hence, a central issue concerns the definitions of these input processes.

The applications of transportation systems [17, 18, 47] apply an intuitive yet practical approach to input approximation in ad hoc distributed simulation. In these studies, the input flow rates at the boundaries of LPs' modeling areas are estimated based on the aggregated traffic flow-rate predictions within a fixed-length, rolling time window. In addition, the studies list three possible data sources for input process estimation: (1) the projected state information from LPs, (2) the real-time traffic data from sensors, and (3) the behavioral patterns based on traffic history.

The operations research literature presents several state-of-the-art methods for estimating dependent input processes [48, 49] as well as the methods for generating sample paths from such processes [50, 51]. These methods are typically time-consuming and are not designed for data sets that are generated from statistically dependent simulations. Therefore, their applicability in the dynamic setting of ad hoc distributed simulations is challenging, if not prohibitive.

This thesis will explore an input approximation approach to modeling open queueing networks with independent routings. In the approach, the arrival processes of the links between LPs' modeling areas are approximated by renewal processes with gamma-distributed interarrival times; the distribution parameters are estimated based on the data observed within a fixed-length, rolling time window [52, 53]. This approximation design is from the paradigm of Whitt [54, 55] and is limited to open queueing networks. Its applicability to closed queueing systems, queueing networks with state-dependent routings, and transportation systems is the subject of ongoing research.

Another issue of input approximation relates to the sampling mechanisms and the sizes of pertinent data sets. If a simulated system is in steady state, extending sampling periods allows for improving estimation accuracy. However, a longer sampling period introduces challenges, including (1) higher computational requirements for input data analysis and (2) slower response to system transitions away from stable conditions. The latter is a consequence of an LP only modeling a partial physical system; that is, changes to a specific part of a physical system would not be revealed to other LPs until the corresponding LPs share the information. This slow transient tracking may be undesirable and error-prone especially for the applications intended to capture the changes in system behavior or the anomalous events.

In order to obtain more observations within a fixed-length sampling period, multiple LPs can be deployed to model the same area, and the estimation of a state variable is then derived by aggregating the data from these LPs. One advantage of this method is that the amount of data grows in proportion to the number of LPs. However, the overlaps between LPs' modeling areas would potentially result in correlated data streams; such correlations must be handled with carefulness. A counter example would be aggregating the data using some simple methods based on the mean or median because they are not suitable in general. In addition, regardless of the aggregation method, predictions from various LPs may be weighted differently. Greater weight could be given to the LPs projecting with higher fidelity, which can be affected by sensor capability, computing power, and modeling details, to mention a few.

Finally, recall that ad hoc distributed simulations are fundamentally online simulations embedded in sensor networks. That is, to this ad hoc approach, prediction accuracy is an objective, but speedy (i.e., faster-than-real-time) execution is essential. Both are important in evaluating an input approximation method. Since execution efficiency is controlled by, for example, the method complexities and the available computational resources, the best approximation to the underlying data model is determined by not only the data quality but also the effectiveness of the input analysis method under such computational resource constraints.

1.2.2 Experiment Design

The first issue encountered in designing the experiments using the ad hoc approach concerns the assignment of coverage areas to LPs. This is referred to as the "system partitioning problem." One solution is to assign each LP the area where sensor data are locally available. This is cost effective in terms of communication as sensor data are consumed locally; long-distance transmission is not required. However, local data alone may be insufficient for making useful predictions or recommendations. By contrast, instead of separating a physical system geographically, some applications may benefit from the methods based on component similarities. One such application is to model the systems that involve a large number of interactions among the components of the same category but very few across categories. Examples include modeling the spread of computer viruses in the Internet and the information diffusion in social networks. In addition, a more general consideration should involve the computational resource restrictions imposed by the environments in which simulations are executed. The balance between the resource restrictions, the LP deployment/configuration complexities, and the desired output measures create a complex dependency among each other, which is not yet completely understood.

A related issue to the system partitioning problem concerns the choice of shared information. Ideally, an LP would share as much information as possible. However, this ideal configuration is impeded by the communication and computational limitations (as the simulation must execute faster than real time in most applications). The information to be shared must be sufficient in terms of driving the input processes of individual simulations, and must allow identifying state changes so that rollbacks, when necessary, may be utilized efficiently to catch up system dynamics.

An issue pertaining to how many LPs are required to model a segment of a physical system is referred to as the "system coverage problem." Solutions to this problem have the potential to significantly influence the output measure accuracy. Intuitively, a large number of LPs are preferable as this should increase the rate at which a system transient is recognized. However, a key to the success of deploying multiple LPs is the ability to distinguish between the LPs that indicate the changes in system states versus those "outliers" that do not reflect a true system trend.

1.2.3 Model Execution and Adaptation

Ad hoc distributed simulations are most likely to be carried out in sensor networks. However, sensor networks are highly constrained environments with significant limitations and deficiencies. For example, the limited battery energy necessitates a balance between the sensing and simulation tasks as well as a balance between the computation and communication within each of the tasks. Another impediment comes from the CPU clock speed and the available memory for they may be less powerful compared to modern computers, which would then prevent real-time responses. Wireless communication, which is widely adopted in sensor networks, is also an obstacle owing to the limited bandwidth, the potential large latencies, and the likelihood of errors. Last but not least, LP failures may be frequent when simulations are for emergency purposes; the failures can be caused by a myriad of possible damage arising in the embedded environments, e.g., floods, wildfires, or earthquakes. Therefore, replicates are important in improving the robustness of the ad hoc approach.

The rollback-based optimistic synchronization mechanism in the ad hoc approach allows LPs to advance simulations as fast as possible without being held back by slower LPs. It helps recover from the usage of incorrect input models by triggering rollbacks. Recall that if a requested system state (as input to a simulation) is unavailable, the requesting LP may approximate the input rather than wait. The approximation can be inferred from the available historical or real-time data. While each approximation method has different impacts on the accuracy of the interim system state predictions, the final prediction relies heavily on how the invalid input models are identified and corrected by a rollback mechanism, or specifically rollback triggering criteria.

An effective rollback triggering criterion must differentiate between the uses of invalid input data and the normal statistical fluctuations because a rollback is unnecessary if the distinction between the adopted input model and the corresponding system state results from pure randomness or expected fluctuations. Therefore, one major issue in deriving such criteria is to quantify the difference and set the bounds of acceptable differences. A straightforward method is to specify a tolerance range with a fixed width. However, the appropriate width varies by cases; a general guideline can hardly be formulated. A more reliable alternative is to set the width in a relative sense (e.g., allowing a 10% deviation). Another type of methods applies statistical hypothesis tests to distinguish one model from the other. Without loss of generosity, consider the case in which the null hypothesis states that the two models in comparison are statistically identical. Then, increasing the tolerance can be accomplished by choosing a lower significance level, which is the probability of rejecting the null hypothesis. However, doing so would raise the probability of the type II errors. That is, it becomes harder to reject the hypothesis, which is actually false. In addition, regardless of the rollback criterion, it may be helpful to evaluate the sensitivity of output metrics to the variations in input data, as a prior step.

Resource constraints (on computation and communication, or more generally power consumption) add complexities to designing a rollback mechanism. When power is a limiting factor, it must be considered how to achieve the best mapping of an ad hoc distributed simulation over the available computing resources. In some cases, restricted optimism may be applied to sacrifice execution efficiency in order to prevent resources from being excessively consumed by potential rollbacks.

1.2.4 Output Analysis

Statistical analysis of the output data from ad hoc distributed simulations introduces several challenges that are not present in conventional simulations. First, the overall predictions from an ad hoc distributed simulation are tightly coupled with the complex mechanisms involved in the input process approximations and the rollback mechanism. While the simulation literature contains several methods for adjusting the point estimates and confidence intervals of the output measures from conventional simulations in order to account for uncertainties in input parameters [49, 56], these procedures are not suitable for the dynamic nature of ad hoc distributed simulation.

The rollback mechanism in the ad hoc approach complicates the problem of removing initial transient effects in steady-state analysis. In conventional simulations, this problem can be addressed by batching [57]. However, this batch method is not directly applicable to ad hoc distributed simulations because the output processes can exhibit temporary departures from the stationary state owing to, e.g., the communication failures and the incident-driven changes in field data.

Estimating metrics spanning across multiple LPs is another challenge. For example, estimating the distributions of travel times in an ad hoc traffic/queueing network simulation involves various complications. Consider estimating the mean travel time of a unit across a route, which is a relatively straightforward problem for conventional simulations. In an ad hoc distributed simulation, the route may cross the areas covered by different and potentially overlapping LPs. If the route is acyclic, the mean travel time may be estimated by forming an optimal linear combination of the estimators from the corresponding LPs. On the other hand, if the route contains one or more cycles, as in the queueing networks with probabilistic routings, this estimation problem becomes even more difficult.

1.3 Research Contributions

This thesis addresses several issues in designing ad hoc distributed simulations, particularly focusing on the accuracy of predictions regarding the performance of open queueing networks. The principal contributions are as follows:

1. **Formalization of ad hoc distributed simulation.** To facilitate the analysis of ad hoc distributed simulation, I have formalized the approach using the notion of functions and sets to describe the behavior of the involved components (e.g., LPs and STM) and the mechanisms (e.g., rollback criteria). Also, I have delivered the pseudo codes that help convert the high-level, conceptual description of the ad hoc approach into an implementation.
2. **Theoretical analysis of steady-state prediction accuracy for open queueing networks.** I have proved that the ad hoc approach can yield steady-state results being statistically equivalent to those from sequential simulations in modeling the open queueing networks under some conditions. The open queueing networks that satisfy these conditions are identified and documented using Kendall's queueing model notation [58].
3. **Experimental analysis of steady-state prediction accuracy for open queueing networks.** Since most open queueing networks do not satisfy the conditions mentioned in the previous bullet, the ad hoc approach cannot guarantee the prediction accuracy at modeling them. However, some applications allow a minor bias in estimation in exchange for other performance gains such as scalability, flexibility, and fault tolerance. Therefore, to study the steady-state prediction accuracy of the ad hoc approach at modeling general open queueing networks, I have constructed experiments with various network configurations (e.g., the service-time distributions). In the experiments, overestimation is observed but the estimation biases are tolerable,

which supports the conclusion that the ad hoc approach can provide the steady-state predictions comparable to those from sequential simulations. Furthermore, I have studied the overestimation issue through another set of experiments to show that the issue is due to the input approximation method. Moreover, the results confirm that the issue becomes more apparent when the service-time coefficient of variation (CV) increases.

4. **An accuracy improving mechanism.** While the overestimation issue can be solved directly by an input approximation method that produces the models with higher fidelity to the underlying data, this method is not computationally efficient for ad hoc distributed simulations as they are expected to run under constrained computing resources. Instead, I have proposed a "buffered-area" mechanism, which builds upon the idea of enlarging the "distance" between where the imperfect input approximations are made and where the system performance estimates are measured. The empirical evidence is presented to show that this mechanism substantially reduces the estimation bias with a moderate increase in the execution time.
5. **Performance evaluation of system partitioning methods.** I have explored the system partitioning problem in the ad hoc approach at modeling open queueing networks, and have focused on the influence of partitioning methods over the prediction accuracy. In total 11 partitioning methods are evaluated experimentally, and the results indicate that these partitioning methods do not appear to significantly affect the prediction accuracy as long as the entire modeled queueing network is covered by a sufficient number of LPs.
6. **Experimental analysis of prediction accuracy for open queueing networks**

with short-term system-state transients. I have extended the empirical studies further to demonstrate the potential of the ad hoc approach regarding instantly capturing system dynamics. This work starts with examining the ad hoc queueing simulation method in the aforementioned steady-state study and identifies a design flaw, which causes a delayed response. I have fixed the issue and showed the effectiveness of the new design in various conditions: the experiments evaluate 6 different network configurations with the system dynamics introduced by both increasing and decreasing the external arrival rates. Moreover, the ideas behind the new design are elaborated with counter examples to reveal the possible impacts from the inappropriate usage of the ad hoc method.

CHAPTER 2

PRINCIPLES OF AD HOC DISTRIBUTED SIMULATION

An ad hoc distributed simulation of a physical system is constructed by employing multiple logical processes (LPs), each modeling a portion of the system, and adding communication among the LPs via a synchronization service to collectively model the entire system of interest. Without the synchronization service, these LPs are essentially independent sequential simulations. That is, they do not coordinate individual modeling tasks, including the modeling areas, the shared information, as well as the required input data to the individual simulations. The synchronization service "links" these simulations together as a comprehensive modeling task, and the design of the involved mechanisms (e.g., rollback triggering criteria) is important in terms of the effectiveness of the ad hoc distributed simulation.

In this chapter, I will formalize the ad hoc approach, particularly the synchronization service, and deliver the pseudo codes for better understanding the service. Also, I will prove that, in modeling some open queueing networks, the ad hoc approach can produce high-quality system-performance estimates, which are statistically equivalent to those from conventional sequential simulations.

2.1 Formalism

A modeled physical system can be represented as a state G, which is a set of M objects, denoted as $G = \{G_1, G_2, \ldots, G_M\}$. Each object G_g $(1 \le g \le M)$ stands for a portion of the physical system. These objects are shared among LPs, which implies that an LP can read or write the value of any object through the synchronization service. To facilitate the following formulation, the number of LPs involved in an ad hoc distributed simulation is assumed to be N and the LPs are denoted as LP_1, LP_2, …, LP_N. Furthermore, the local state of LP_{lp}'s modeling area is L_{lp} ($1 \leq lp \leq N$). For clarity, here g is used to index objects and lp to index LPs.

An LP writes the value of an object with a time value t. Typically, the value is a prediction of the object at that time. It is derived by invoking a function, which maps the LP's local state to an appropriate representation with respect to the object. Hence, the function, named Local2Global, can be formulated as

$$\text{Local2Global}_{lp,g}\text{: } \{L_{lp}\} \times \{t\} \rightarrow \{v_g\},$$

where v_g is the value of G_g at time t that LP_{lp} shares with other LPs. The subscripts, lp and g, next to the function name indicate that the specific functionality depends on the LP and the object. Similarly, a function Global2Local converts the value of an object to an input for the simulations that model the object. This function, which is generally triggered after an LP reads the value of an object, is defined as

$$\text{Global2Local}_{lp,g}\text{: } \{v_g\} \times \{t\} \rightarrow \{in_g\},$$

where v_g is the value of G_g with respect to time t, and in_g is the input that is fed into LP_{lp} for simulating G_g.

The synchronization service of the ad hoc approach allows LPs to read and write values of objects, and also to "undo" these read/write operations in case errors are detected in the read or the written values. The synchronization service is supported by a logical construct called space time memory (STM), which hosts a collection of records. The records are created through read/write operations and are removed by anti-write (i.e., undo) operations; I will elaborate on these operations after discussing the capability of the STM.

A record in the STM is a 5-tuple that follows the format:

$$(rw, lp, g, v, t).$$

The first parameter *rw* indicates the type of the operation corresponding to this record; it can be either "**R**" for read operations or "**W**" for write operations. The next two parameters, *lp* and *g*, are the indexes of the invoking LP and the requested object, respectively; *lp* is an integer between 1 and N while g is between 1 and M, inclusive in both cases. The read/written value of G_g is v, which is the fourth parameter. The last parameter t is a time interval that specifies when v takes effects. The time interval is bounded by a closed starting point t_{s} and an open end point t_{e}, denoted as $t = [t_{\mathrm{s}}, t_{\mathrm{e}})$. The meaning of a record depends on *rw*: if *rw* is **R**, the record indicates that LP_{lp} reads v as the value of G_g and will use the value v in its simulation with respect to t; otherwise, if *rw* is **W**, the record implies that LP_{lp} predicts the value of G_g to be v with respect to t.

The STM provides an interface to the synchronization service for accessing records. Considering data integrity and software modularity, a proper design must not allow simulation applications to directly manipulate the records in the STM; this STM interface should be visible to the synchronization service only. The STM interface defines three functions and their pseudo codes are shown in Figure 2.

1. **STM_Add(*rw*, *lp*, *g*, *v*, *t*).** This function adds to the STM the record with the corresponding arguments; see lines 01–04 in Figure 2.
2. **STM_Erase(*rw*, *lp*, *g*, *v*, $t = [t_{\mathrm{s}}, t_{\mathrm{e}})$).** This function removes from the STM the records with the corresponding arguments; see lines 05–13 in Figure 2. Note that comparing two time intervals requires special treatment. For example, considering an existing record $r = (rw, lp, g, v, t_r = [t_{r\mathrm{s}}, t_{r\mathrm{e}}))$, the relationship between the two intervals t and t_r belongs to one of the following six cases:

   i. $\boldsymbol{t_{r\mathrm{e}} \leq t_{\mathrm{s}}}$**.** In this case, the two time intervals do not overlap. Hence, this record is not deleted.

ii. $\boldsymbol{t_{r\mathrm{s}} < t_{\mathrm{s}} < t_{r\mathrm{e}} \leq t_{\mathrm{e}}}$**.** The two time intervals overlap between t_{s} and $t_{r\mathrm{e}}$. That is, one part of the record with respect to the overlapping time interval $[t_{\mathrm{s}}, t_{r\mathrm{e}})$ should be removed. However, the other part of the record with respect to the rest time interval should remain. As a consequence, the time interval of the record is shortened into $[t_{r\mathrm{s}}, t_{\mathrm{s}})$.

iii. $\boldsymbol{t_{r\mathrm{s}} < t_{\mathrm{s}} < t_{\mathrm{e}} < t_{r\mathrm{e}}}$**.** Similar to Case ii, the two time intervals overlap at $[t_{\mathrm{s}}, t_{\mathrm{e}})$ and the record with respect to $[t_{\mathrm{s}}, t_{\mathrm{e}})$ should be deleted. Unlike Case ii, $[t_{\mathrm{s}}, t_{\mathrm{e}})$ breaks the time interval of the record into two non-overlapping time intervals: $[t_{r\mathrm{s}}, t_{\mathrm{s}})$ and $[t_{\mathrm{e}}, t_{r\mathrm{e}})$. Hence, the time interval of the record is modified to be $[t_{r\mathrm{s}}, t_{\mathrm{s}})$ and, in addition, a new record is added with the same arguments as the original one except that the time interval is set to $[t_{\mathrm{e}}, t_{r\mathrm{e}})$.

iv. $\boldsymbol{t_{\mathrm{s}} \leq t_{r\mathrm{s}} < t_{r\mathrm{e}} \leq t_{\mathrm{e}}}$**.** In this case, the two time intervals overlap between $t_{r\mathrm{s}}$ and $t_{r\mathrm{e}}$, which results in deleting the entire record.

v. $\boldsymbol{t_{\mathrm{s}} \leq t_{r\mathrm{s}} < t_{\mathrm{e}} < t_{r\mathrm{e}}}$**.** The two time intervals overlap at $[t_{r\mathrm{s}}, t_{\mathrm{e}})$, which is similar to Case ii. As a consequence, the time interval of the record is modified to be $[t_{\mathrm{e}}, t_{r\mathrm{e}})$.

vi. $\boldsymbol{t_{\mathrm{e}} \leq t_{r\mathrm{s}}}$**.** Same as Case i, the two time intervals do not overlap; the record remains untouched.

For simplicity, a notation $t_1 = t_2$ is used in the pseudo code to indicate that the two time intervals $t_1 = [t_{1\mathrm{s}}, t_{1\mathrm{e}})$ and $t_2 = [t_{2\mathrm{s}}, t_{2\mathrm{e}})$ overlap. In other words, $t_1 = t_2$ implies $t_{1\mathrm{s}} \leq t_{2\mathrm{s}} < t_{1\mathrm{e}}$ or $t_{2\mathrm{s}} \leq t_{1\mathrm{s}} < t_{2\mathrm{e}}$.

```
01: function STM_Add($rw$, $lp$, $g$, $v$, $t$)
02: begin
03:     STM $\leftarrow$ STM $\cup$ $\{(rw, lp, g, v, t)\}$
04: end

05: function STM_Erase($rw$, $lp$, $g$, $v$, $t = [t_s, t_e)$)
06: begin
07:     $\forall\ r = (rw, lp, g, v, t' = [t_s', t_e'))$: $t = t'$ AND $t_s' < t_s$
08:         STM $\leftarrow$ STM $\cup$ $\{(rw, lp, g, v, [t_s', t_s))\}$
09:     $\forall\ r = (rw, lp, g, v, t' = [t_s', t_e'))$: $t = t'$ AND $t_e < t_e'$
10:         STM $\leftarrow$ STM $\cup$ $\{(rw, lp, g, v, [t_e, t_e'))\}$
11:     $\forall\ r = (rw, lp, g, v, t' = [t_s', t_e'))$: $t = t'$
12:         STM $\leftarrow$ STM \ $\{r\}$
13: end

14: function STM_GetRecords($rw$, $lp$, $g$, $v$, $t$)
15: begin
16:     $R \leftarrow \Phi$
17:     $\forall\ r = (rw, lp, g, v, t')$: $t = t'$
18:         $R \leftarrow R \cup \{r\}$
19:     Return $R$
20: end
```

Figure 2: Pseudo Codes of STM Interface

3. **STM_GetRecords(*rw*, *lp*, *g*, *v*, *t*).** This function returns a set of the records with the corresponding arguments; see lines 14–20 in Figure 2. Similar to the last function, the comparison of two time intervals must consider both the start and end points. In this function, a record is included in the set if its time interval overlaps *t* and, of course, all the other four arguments match.

The synchronization service is the core of the ad hoc approach, and provides LPs with the ability to communicate and share information. This service is realized through an interface, which defines six functions. Three functions are to be invoked by LPs to perform the read, write, and anti-write (i.e., undo) operations; they are referred to as the "operational functions." The remaining three are callback functions—simulation applications must implement them so that the synchronization service can request for certain application-dependent decisions or enforce LPs to execute rollbacks. Details of the operational functions (the pseudo codes of which are in Figures 3 and 4) as well as the expected behaviors of the callback functions are as follows:

1. **Read(*lp*, *g*, *v*, *t*).** LP_{lp} invokes this function to read the value of G_g with respect to time t. This value v' is computed based on the currently-available predictions by invoking the callback function AggregateValues(); see lines 03–04 in Figure 3. In line 03, the asterisks "*" are used to represent the notion of "all." That is, line 03 states that the set R is composed of the write records corresponding to G_g with respect to t, regardless who wrote the value and what value was written. The notation "*" will be used throughout the whole pseudo codes with this same meaning. After the value of G_g with respect to time t is computed, the follow-up operation depends on the parameter v:

   i. If v is not specified (or "NULL" by our convention), a read record is created in the STM with the computed value, i.e., v', which will be returned via the function parameter v. In this case, the function returns FALSE to notify the function caller that a new value may be inserted into v. One special case is that if the corresponding predictions (i.e., write records in the STM) are not sufficient for estimating the value of G_g, v may be NULL. In this case, the read operation is not logged.

   ii. If v is not NULL, the next step is to determine if v is acceptable compared against those currently-available predictions. This is done via the callback function AcceptDifference(), the details of which will be revisited soon. If *v* is within tolerance or if the corresponding predictions are insufficient, a read record with *v* is created in the STM and the function returns TRUE, which indicates that *v* is accepted. Otherwise, the function sets *v* to the computed value *v*′, creates a read record with *v*′ in the STM, and returns FALSE.

```
function Read($lp$, $g$, $v$, $t$)
begin
    $R$ ← STM_GetRecords(W, *, $g$, *, $t$)
    $v'$ ← AggregateValues($R$)
    if $v$ = NULL AND $v'$ = NULL then
        Return FALSE
    else if $v$ = NULL AND $v'$ ≠ NULL then
        $result$ ← FALSE
    else if $v$ ≠ NULL AND $v'$ = NULL then
        $result$ ← TRUE
    else
        if AcceptDifference($lp$, $g$, $v$, $t$, $R$) then
            $result$ ← TRUE
        else
            $result$ ← FALSE
        end
    end
    if $result$ = FALSE then
        $v$ ← $v'$
    end
    STM_Add(R, $lp$, $g$, $v$, $t$)
    Return $result$
end
```

Figure 3: Pseudo Codes of Synchronization Service Interface (I)

2. **Write(*lp*, *g*, *v*, *t*).** LP_{lp} calls this function when it predicts that v is the value of G_g with respect to time t and it wants to share this information with other LPs. A write record is created in the STM, which is followed by a procedure to check if rollbacks are necessary to some LPs, or more specifically, to those LPs who had read the value of G_g with respect to time t. The checking process (lines 16–24 in Figure 4) is based on individual read records. That is, for a read record pertaining to G_g with respect to time t, the value in the record is compared against the currently-available predictions. If the value is unacceptable, the LP associated to the record needs to be rolled back. The rollback notification piggybacks a new value (computed via AggregateValues()) for the LP to replace the invalid one.
3. **AntiWrite(*lp*, $t = [t_s, \infty)$).** LP_{lp} invokes this function if it is rolled back to the time point t_s and it needs to remove all the possibly-contaminated read and write records. Since removing read records does not affect the projection of the modeled physical system, the read records can be deleted with no further concern. However, this situation does not apply to removing write records. Specifically, removing a write record necessitates examining the LPs that had read the corresponding object; see lines 09–12 and 14 in Figure 4. Hence, the entire procedure of removing write records involves (1) finding all the read records that might be affected due to those to-be-deleted write records, and (2) after those write records are deleted, checking each affected read record with respect to rollbacks, which is the same as that performed in write operations.

```
function Write($lp$, $g$, $v$, $t$)
begin
    STM_Add(W, $lp$, $g$, $v$, $t$)
    $R_{\text{affectedReads}} \leftarrow$ STM_GetRecords(R, *, $g$, *, $t$)
    CheckRollbacks($R_{\text{affectedReads}}$)
end

function AntiWrite($lp$, $t = [t_s, \infty)$)
begin
    $R_{\text{toBeDeletedWrites}} \leftarrow$ STM_GetRecords(W, $lp$, *, *, $t$)
    $R_{\text{affectedReads}} \leftarrow \Phi$
    $\forall\ r = (W, lp, g, v, t_r): r \in R_{\text{toBeDeletedWrites}}$
        $R_{\text{affectedReads}} \leftarrow R_{\text{affectedReads}} \cup$ STM_GetRecords(R, *, $g$, *, $t_r$)
    STM_Erase(*, $lp$, *, *, $t$)
    CheckRollbacks($R_{\text{affectedReads}}$)
end

function CheckRollbacks($R$)
begin
    $\forall\ r = (R, lp, g, v, t): r \in R$
        $R_r \leftarrow$ STM_GetRecords(W, *, $g$, *, $t$)
        if NOT AcceptDifference($lp$, $g$, $v$, $t$, $R_r$) then
            $v_r \leftarrow$ AggregateValues($R_r$)
            Rollback($lp$, $g$, $v$, $t$, $v_r$)
        end
end
```

Figure 4: Pseudo Codes of Synchronization Service Interface (II)

4. **AggregateValues(*R*).** User applications must implement this callback function to aggregate the predictions in the parameter *R* into a value. *R* is a set of the write records corresponding to the same object. This function should either return the aggregated value or NULL if the write records in *R* are insufficient to generate an estimate because, for example, the application-defined confidence

level is not satisfied.

5. **AcceptDifference(*lp*, *g*, *v*, *t*, *R*).** User applications must implement this callback function to determine if the value v is within the acceptable range based on R, which is the set of the currently-available predictions pertaining to G_g with respect to time t. The decision may take into account the LP who had used v (i.e., LP_{lp}). This function should return TRUE if v is acceptable, otherwise FALSE.

6. **Rollback(*lp*, *g*, v_{invalid}, *t*, v_{new}).** User applications must implement this callback function so that the synchronization service can notify LP_{lp} to rollback. This rollback request is based on the fact that LP_{lp} had used v_{invalid} in modeling G_g with respect to time t. However, as now v_{invalid} turns out to be in error, LP_{lp} should use v_{new} for simulating G_g at t. This function does not have a return parameter.

2.2 Accuracy on Modeling Open Queueing Networks

This section derives a sufficient condition for the ad hoc distributed simulations of open queueing networks to produce asymptotic results to be statistically equivalent to those produced by traditional, replicated sequential simulations. In modeling queueing networks, a common objective is to derive the estimation accuracy regarding the system performance measures in steady state. Hence, the following analysis focus on the accuracy of point estimates and the confidence-interval (CI) coverage of the steady-state system performance measures at each queueing station. The analysis further assumes that the open queueing networks under study are stable.

This analysis relies on partitioning a modeled open queueing network into several non-overlapping subnetworks so that each LP models one of them. The internal arrivals for a subnetwork are the departures from the queueing stations surrounding this subnetwork. The links along which internal arrivals enter a subnetwork are referred to as the “internal input links;” these links are also the links along which internal departures leave another subnetwork, and hence they are also referred to as the “internal output links.” The data models that capture arrival processes on internal input links are crucial to the accuracy of system performance measures, which are produced by the corresponding simulations of the subnetworks. Typically, one cannot know a priori true arrival process on every internal input link; approximations must be made. As a consequence, to guarantee the desired accuracy performance of ad hoc open queueing network simulations, the condition is derived as follows.

Proposition 1. Suppose that (1) the subnetworks modeled by all LPs form a perfect partition of the entire modeled network; (2) the steady-state flow on every internal output link forms a renewal process, and the inter-departure times on these links are used to build and update the corresponding empirical distributions; and (3) the arrivals on internal input links are generated following those periodically-updated empirical distributions. Then, as the simulation run length increases, the ad hoc approach yields more accurate point estimators and confidence intervals (CIs) for the steady-state performance measures at every queueing station. These point estimators and CIs will be statistically equivalent to those produced by a sufficiently-long execution of sequential simulations.

Proof: Recall that the external arrivals feeding the modeled network are generated from the appropriate theoretical data models, which is the same in both the ad hoc approach and the sequential simulation method. What distinguishes the two simulation methodologies is the mechanism that models internal arrivals. In the ad hoc approach, the internal arrivals are generated following the periodically-updated empirical distributions. Since these empirical distributions are constructed based on inter-departure times, they will converge weakly, with the probability 1, to the respective theoretical distributions after a sufficiently-long transient phase (according to Section 20 in [59]). Then, given that the interior of each subnetwork is simulated properly, under some mild continuity assumptions, point estimates for mean performance measures will converge to the respective steady-state means [60]. Also, under appropriate mixing or ergodicity conditions [61], one can establish the asymptotic normality of point estimates and the equality of underlying variance estimates with those from sequential simulations. ■

The aforementioned assumptions are clearly onerous; the one that requires the renewal property is one such example. As Table 1 in [62] shows, very few queueing stations have renewal departure processes. By contrast, stationary arrival and departure processes at queueing stations in the networks with complex routes are in general dependent processes with complex multivariate distributions [63-66]. Fitting multivariate models to such processes and generating realizations from the fitted models is a hard problem, especially under the expected computational constraints in ad hoc distributed simulations.

According to Table 1, the two types of networks that meet the above assumptions are acyclic Jackson networks and the tree-like networks that are composed of GI/D/*m* queueing stations along with state-independent, unidirectional routing. In the former case, departures at each queueing station are routed to a destination with a probability, which is irrelevant to the network state. That is, this forms several independent Poisson departure processes at that queueing station. Hence, the steady-state input process at every queueing station is a superposition of Poisson processes. In the latter case, the routing mechanism at each queueing station results in delayed (albeit dependent) renewal flows. Moreover, the acyclic nature of both the classes of networks allows the establishment of asymptotic validity of estimators for additional performance measures, such as the mean travel times of units across paths.

Table 1. Queueing Stations Producing Renewal Departure Processes in Steady State

Queueing Station	**Departure Process**	**Description**
M/M/*m*	Poisson	Poisson arrivals, exponential service times, and *m* identical servers
M/0/*m*/*L*	Poisson	No service time (0) with system capacity *L*
M/GI/1/1	Renewal	General independent-and-identically-distributed service times with no waiting capacity (excluding the one in service)
M/D/1/1	Renewal	Constant service times
M/GI/∞	Poisson	Infinite number of servers
M/GI/*m* with PS service discipline	Poisson	Process-sharing (PS) service discipline
M/GI/*m*/*L* with LCFS-PR service discipline	Poisson	Last-come-first-serve (LCFS) preemptive-resume (PR) service discipline
GI/D/*m*	Renewal	Renewal arrivals

Note: All systems have infinite system capacity, infinite calling population, and first-come-first-serve service discipline, unless otherwise specified.

2.3 Conclusions

This chapter focused primarily on the theoretical aspects of the ad hoc approach. First, this approach was formalized and modularized for software engineering considerations. The interface between LPs and the synchronization service provided by the ad hoc approach were defined as simple but also as comprehensive as possible, which improves flexibility while assures the correctness in synchronization. In addition, the separation of the STM from the synchronization service mainly considers the software reuse and replacement. This design allows the STM implementation to be arbitrary, such as reusing existing software or designing specifically for a user application for optimizing execution performance. Furthermore, modularization makes it possible to replace the old STM implementation should the advanced data management techniques become available in the future.

Secondly, this chapter explored the theoretical capability of the ad hoc approach on modeling open queueing networks. It was proven that the estimates produced by an ad hoc open queueing network simulation can be as accurate as those generated by the sequential counterpart for the queueing networks satisfying some conditions. However, in practice the conditions are rather limiting. Also, it is challenging to further relax these conditions while hold the theoretical accuracy, not to mention that it might be infeasible if additional requirements are to be enforced, such as restricting computation resource consumption and achieving desired execution efficiency. As a consequence, Chapter 3 will address the issue of prediction accuracy in practical applications by experimenting on various designs of the ad hoc approach as well as different queueing network configurations.

CHAPTER 3

AD HOC QUEUEING NETWORK SIMULATIONS

Although ad hoc distributed simulation is motivated in constructing real-time, microscopic in-vehicle transportation simulations for urban areas [47], this approach offers the potential for use in other applications, which motivates the work described next. To explore the capabilities of this approach, a typical procedure involves applying this approach to a general, abstract model that can capture a myriad of practical realizations. Hence, this chapter focuses on applying the ad hoc approach to simulate queueing networks as they are used to model a variety of industrial systems, ranging from computer networks, communication systems, to supply chains. In this context, each simulator (logical process, or LP) in an ad hoc distributed simulation could model potentially overlapping portions of a global supply chain, which consists of production facilities (e.g., semiconductor fabs), warehouses, and transportation systems (e.g., vessels, planes, and trucks). More specifically, each LP can involve various geographical areas or facilities.

The remainder of this chapter is organized as follows. Section 3.1 describes the software architecture for the ad hoc queueing network simulations in the following sections. Section 3.2 elaborates on the mechanisms in these simulations while the experiments for the performance study are described in Section 3.3. Section 3.3 also includes the experiment results and analyzes some deficiencies observed in these results. To address these deficiencies, a buffered-area mechanism is proposed in Section 3.4, along with an empirical analysis to show its effectiveness. Finally, Section 3.5 concludes this chapter.

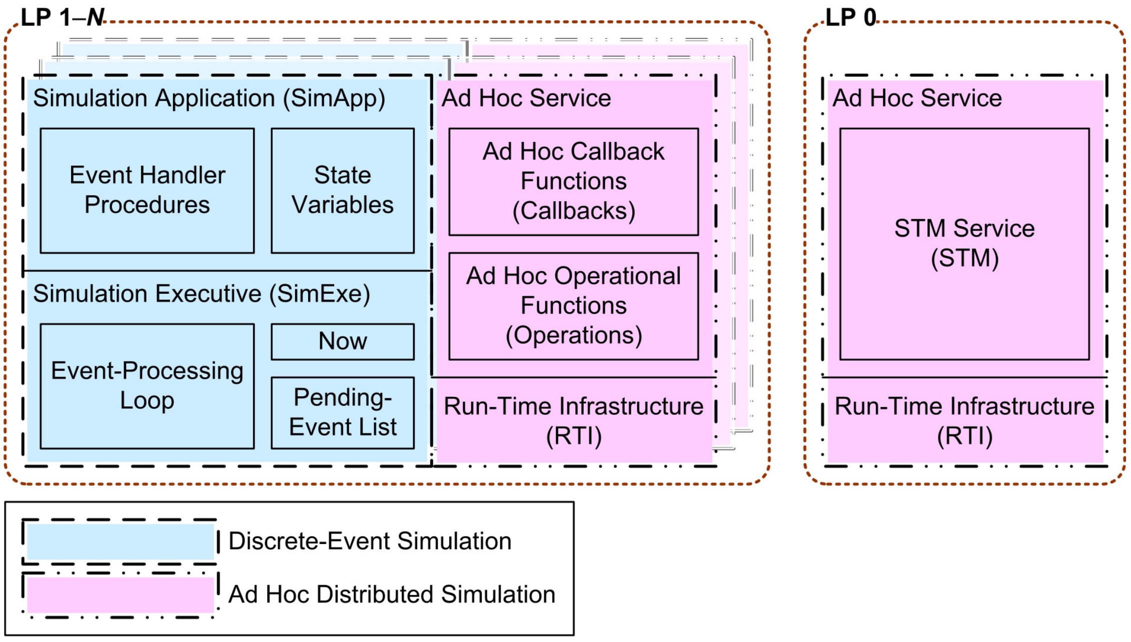

LP 1–N
Simulation Application (SimApp)
Event Handler Procedures
State Variables
Simulation Executive (SimExe)
Event-Processing Loop
Now
Pending-Event List
Ad Hoc Service
Ad Hoc Callback Functions (Callbacks)
Ad Hoc Operational Functions (Operations)
Run-Time Infrastructure (RTI)
LP 0
Ad Hoc Service
STM Service (STM)
Run-Time Infrastructure (RTI)
Discrete-Event Simulation
Ad Hoc Distributed Simulation

Figure 5: Primitive Software Architecture for Ad Hoc Distributed Simulation Based on Discrete-Event Simulation

3.1 Implementation Architecture

Ad hoc distributed simulation is an approach to predicting future states of operational systems by “connecting” autonomous LPs, so it can be regarded as more a synchronization mechanism rather than a modeling methodology, such as the time-stepped simulation approach or the discrete-event simulation approach. In other words, the modeling method adopted by each LP is in principle irrelevant although the choice of which would affect the software implementation that “glues” the synchronization mechanism and the modeling method. Figure 5 depicts a design of the software architecture for the LPs modeling a system under investigation (SUI) using the discrete-event simulation approach. That is, the operations of a SUI are modeled by a sequence of time-ordered events.

An implementation of a discrete-event simulation typically contains two parts: the simulation application and the simulation executive; see Figure 5. This design separates the SUI-related code from those independent of the SUI. Specifically, the simulation application models a SUI by maintaining the state variables that represent the SUI's state and providing event handler procedures for events to mimic the operations of the SUI through modifying those state variables as well as scheduling future events if needed. By contrast, the simulation executive manages the simulation time and future events, which are stored in a pending-event list; the procedure is common to any SUI. That is, a standard implementation of the simulation executive repeatedly loops through a block of codes, within which the event associated with time closest to the current simulation time is retrieved from the pending-event list, the simulation time is updated accordingly, and the respective event handler procedure in the simulation application is invoked to process the event. Moreover, the simulation executive provides an interface for the simulation application to schedule future events into the pending-event list.

Similar isolation of the components that are independent of the SUI can be adopted in implementing the ad hoc approach. One implementation is to isolate the space time memory (STM), as shown in Figure 5. Recall that, as described in Section 2.1, the STM holds a collection of data and provides an access to these data through the following function calls: STM_Add(), STM_Erase(), and STM_GetRecords(). Although it is suggested in Section 1.1.3 that the STM should be implemented as a distributed set of objects, the implementation and experiments described here use a simple approach where the STM is implemented centrally as a single LP.

While the STM is separated as an independent component, the rest of the ad hoc approach needs to be closely coupled with the modeling part (including the simulation application and the simulation executive) because it might affect the model. For example, when a rollback is triggered, the simulation time must be rewound, the state variables have to be restored to their previous values, and the pending-event list requires reconstruction. Hence, as shown in Figure 5, the ad hoc approach is implemented as a component that can access both the simulation application as well as the simulation executive. The ad hoc implementation at each LP is further decomposed into the ad hoc operational functions: Read(), Write(), and AntiWrite(), and the ad hoc callback functions: AggregateValues(), AcceptDifference(), and Rollback(). Recall from Section 2.1 that applications must develop their specific callback functions as these functions are invoked by the operational functions for resolving application dependent issues. In addition, the operational functions would call the STM services for accessing the records of previous read/write operations. In this implementation, this is realized via the run-time infrastructure (RTI), which is not unlike that used in the High Level Architecture (HLA) to interconnect simulations.

The ad hoc implementation in Figure 5 allows for customization of LPs as they can have their own implementations of the callback functions. An LP may adjust the adopted mechanisms based on its available resources (i.e., capability), the desired prediction accuracy or execution efficiency, and other conditions. Regardless, in the following experiments in this chapter, all LPs employ the same mechanisms for the callback functions. That is, the functions for the ad hoc operations and the ad hoc callbacks can be extracted out of LPs, and they are incorporated into the centralized code that drives the STM; see Figure 6. This implementation strategy is mainly for programming convenience although it also reduces the communication overhead between the ad hoc operations and the STM services (the study of which is outside the scope of this chapter). The only part of the callback functions that is left at each LP is for rollback handling since each LP has different system states to be reconstructed.

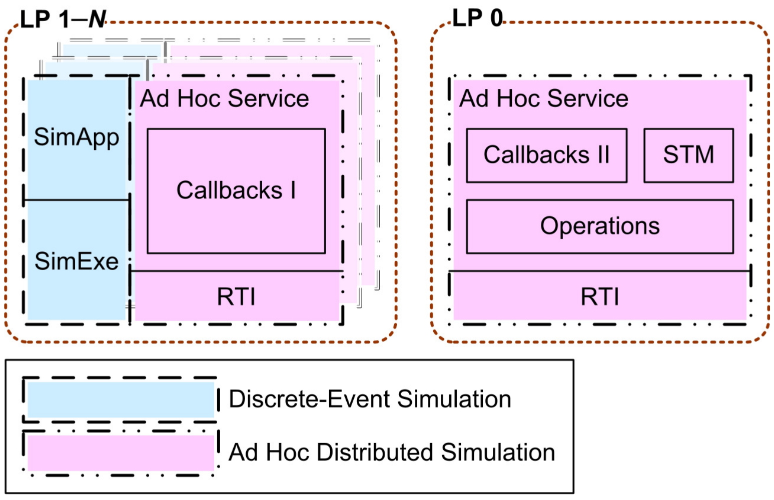


Figure 6: Adopted Software Architecture for Ad Hoc Queueing Network Simulations

All the ad hoc queueing network simulations in this chapter are executed in a computing system with a cluster of tightly-coupled processors. These processors do not share memory spaces, and instead the underlying communication in the RTI is through the message passing interface (MPI).

3.2 An Open-Queueing-Network Model

The remainder of this chapter examines the ad hoc approach on modeling open queueing networks, in which the queueing stations (nodes) are arranged in an $m \times n$ rectangular configuration. The primary focus is the 8 × 8 grid network shown in Figure 7. Each node in the network contains a single server with an unlimited buffer and the first-come-first-served (FCFS) service discipline. Service times of all the servers are independent and identically distributed (IID) with the mean equal to one second, and the service-time distribution varies by experiments.

Arrivals to a node in the network in Figure 7 are twofold: external (from outside the network) and internal (from any node in the network). External arrivals to each node form a Poisson process with the rate 1 / 6 per second. By contrast, internal arrivals to a node are the processed units from the neighboring nodes based on the routing probabilities illustrated in Figure 8, which differ by node types. Nodes are classified into three types: the corner nodes (i.e., nodes 0, 7, 56, and 63), the edge nodes, (e.g., nodes 1, 2, and 3), and the interior nodes (e.g., nodes 9, 10, and 11). Figure 8(a) shows the routing probabilities at a corner node: a processed unit leaves the network with the probability 0.6 or moves to any of its neighboring nodes with the equal probability 0.2; similarly, Figures 8(b) and 8(c) depict the routing probabilities at an edge node and an interior node, respectively. Note that the processed units are routed independently of each other.

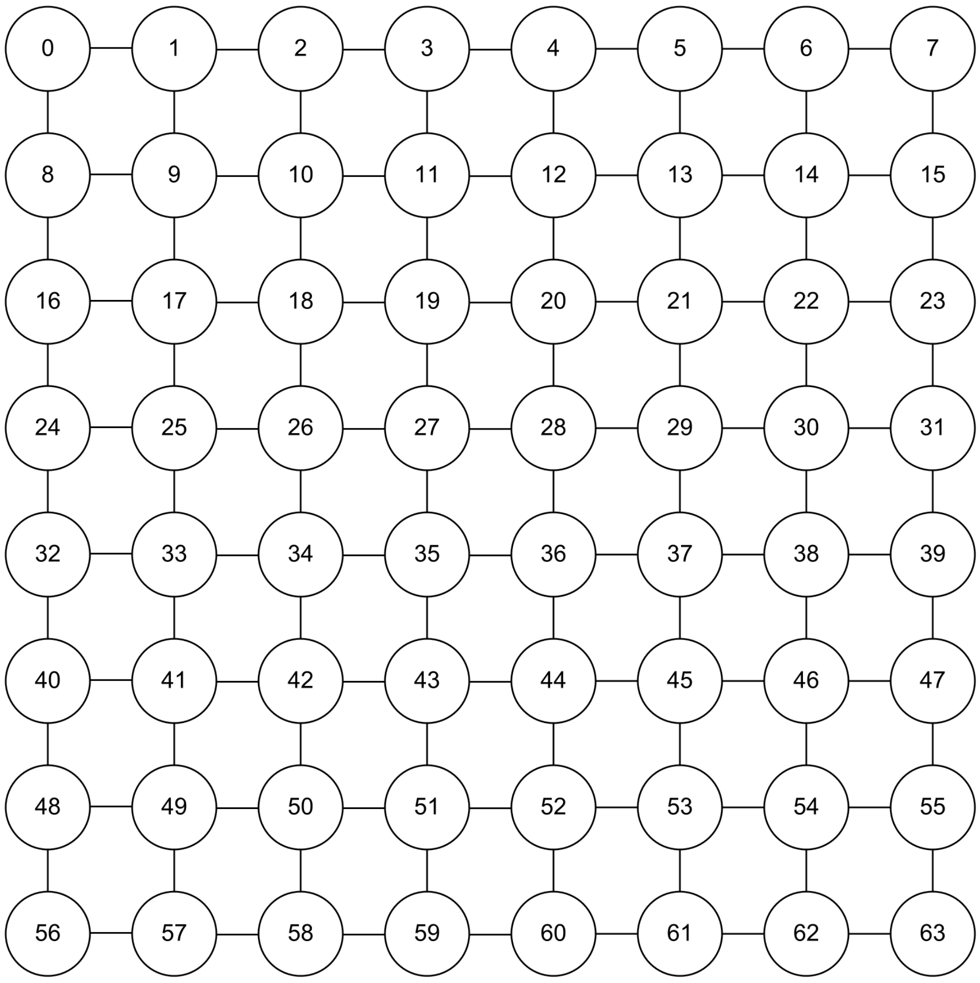


Figure 7: An 8 × 8 Grid Network

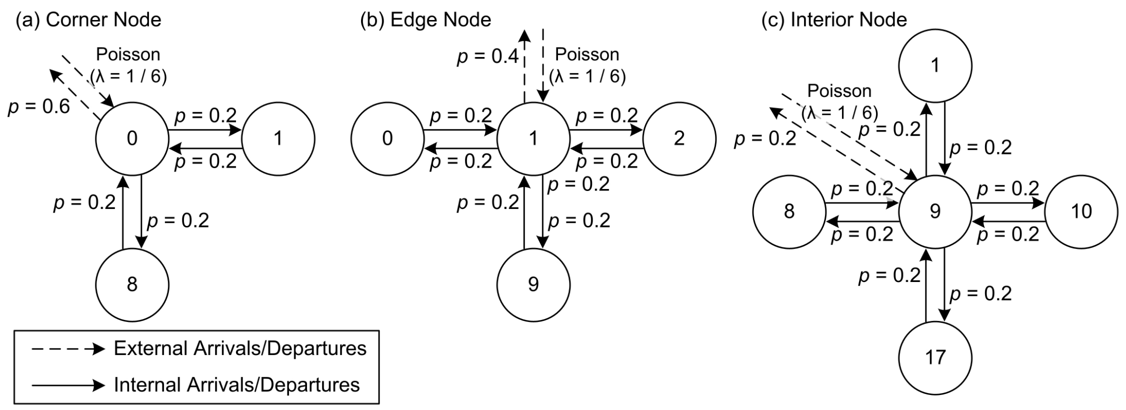


Figure 8: Routings at Nodes

This 8 × 8 grid network in Figure 7 is quite large and it contains many cycles, which leads to complex potential routings. Also, the traffic intensity of a node can be up to 0.8; that is, a server can be heavily loaded. As a consequence, modeling such a network is no easy task—it has the potential to expose deficiencies of the current, early-stage design of ad hoc queueing network simulations.

3.2.1 Partitioning

The first set of experiments aims at evaluating the ad hoc queueing network simulations with a symmetric partition. Specifically, the modeled grid network is evenly partitioned into portions (subnetworks) while, at the same time, an overlap may exist between any two adjacent portions. This is referred to as the "regular partitioning." The specifics about applying this partitioning method to the modeled network will be discussed in Section 3.3.1.

Considering the regular partitioning as a baseline, the second set of experiments focuses on the potential impact of the partitioning methods to the accuracy of system performance estimates in ad hoc queueing network simulations. The network under investigation is partitioned so that LPs model areas with different sizes and shapes. This "irregular partitioning" will be elaborated in Section 3.3.2.

Regardless of how the modeled network is partitioned, an LP models the arrivals and departures of the units circulating in its modeled subnetwork using the discrete-event simulation approach. This leads to the problem that the processes capturing the units arriving at the boundary of a subnetwork are unknown at the beginning of a simulation. For example, an LP does not have prior knowledge of the arrival process from node 4 to node 3 (in the network in Figure 7) if the latter node is in its modeled subnetwork while the former one is not. Therefore, these processes are preconfigured as Poisson processes with the rate 1 / 6 per second, and they will be adjusted dynamically based on the information exchanged among LPs during the simulation execution.

As in typical cases, an ad hoc queueing network simulation starts in an empty and idle state. Then, LPs start exchanging information through the space time memory (STM) after five minutes in simulation time, which is primarily for preventing the shared data from being influenced by initial transients. The details pertaining to the information exchange and other involved mechanisms will be described in the following subsections. Note that in the remainder of this chapter all time values refer to the simulation time, rather than the wall-clock time.

3.2.2 Information Exchange

The objects shared among the LPs in an ad hoc queueing network simulation are the flow states of all links in the modeled network, where a link is a connection between two nodes with the direction being from the departing network to the arriving one. The value of a flow state is defined to be a set of the statistics of interdeparture times on a link.

The statistics include the number of observed interdeparture times and the first two sample moments; grouping them together forms a "prediction," denoted by (n_{lp}, $m_{1,lp}$, $m_{2,lp}$) in which the subscript *lp* indicates the LP that provides this prediction. The observation period is set to be the last five minutes; this five-minute rolling window is primarily to prevent the statistics from becoming overly sensitive to the statistical "fluctuations."

After the five-minute transient period at the beginning of a simulation, every 30 seconds each LP produces a prediction for every link it models with respect to the following 30 seconds. Specifically, let t_{now} be the time point an LP produces a prediction, say (n_{lp}, $m_{1,lp}$, $m_{2,lp}$). The data constitute to this prediction are observed in $[t_{\text{now}} - 300, t_{\text{now}})$ while the prediction is regarded as the projection of the corresponding flow state with respect to $[t_{\text{now}}, t_{\text{now}} + 30)$. Hence, a write record with value $v = (n_{lp}, m_{1,lp}, m_{2,lp})$ and time $t = [t_{\text{now}}, t_{\text{now}} + 30)$ will be created in the STM, as the result of a write operation invoked by the LP. This write operation also involves checking if rollbacks need to be triggered at the LPs who had requested the corresponding flow state with respect to $[t_{\text{now}}, t_{\text{now}} + 30)$ before this new prediction is generated. This aspect will be elaborated in Section 3.2.5.

Every 30 seconds, an LP queries the flow state of every link that originates at a node outside but connects to some node inside its modeled subnetwork. The query provides a value the LP would like to use for the next 30 seconds; the value is set to the last statistics the LP has used for the previous 30 seconds. The provided value may be granted so that this LP keeps using it, or it may be denied while a new value is returned. This LP then replaces the provided value with the new one. The validity of the provided value is evaluated according to the same criteria for determining if a rollback is necessary (see Section 3.2.5). As to the returned new value, it is computed based on the currently-available predictions; the details are described in Section 3.2.3. It is noted that this approach is somewhat different from the traditional read operations where an LP issues a request and waits for the value in response.

3.2.3 Estimation Aggregation

Given a set of predictions, denoted $\{(n_{lp}, m_{1,lp}, m_{2,lp})\}$, aggregating them may be necessary when, for example, an LP queries the respective flow state with which these predictions are associated. To fit the format of a prediction, the aggregated prediction is also presented in a 3-tuple: $(n, \hat{m}_1, \hat{m}_2)$; n is the number of predictions involved in calculating the aggregated prediction (i.e., $n = |\{(n_{lp}, m_{1,lp}, m_{2,lp})\}|$), and $\hat{m}_1$ and $\hat{m}_2$ are the estimated first and second moments, respectively. The estimated first moment is a random sample from the predictions while the estimated second moment is calculated from the pooled variance.

Specifically, the estimated first moment (i.e., $\hat{m}_1$) and the first moments from the predictions (i.e., $m_{1,lp}$'s) are regarded as the samples of an unknown random variable. The density function of this random variable is estimated using a standard Gaussian kernel [67]. The estimated first moment is generated by randomly picking one of the samples and then sampling a normal variate centered at the chosen sample. This kernel-density-estimation method is chosen because it is non-parametric and it requires a very small computational effort. Given the estimated first moment, the estimated second moment is calculated so that the estimated variance is equal to the pooled variance of the predictions.

The estimated first moment is expressed in Equation (3.1), in which h is the bandwidth based on Silverman's suggestion [67] and ε is a standard normal random variate. Since the first-moment estimates might form a distribution other than a uni-modal one (e.g., a bimodal one), the bandwidth is set to $h = 1.06An^{-1/5}$ with A being the minimum between the sample standard deviation (S_1 in Equation (3.2)) and the sample inter-quartile range divided by the value 1.3.

$$\hat{m}_1 = \text{RandUni}\left(\left\{m_{1,lp}\right\}\right) + h\varepsilon \tag{3.1}$$

$$S_1 = \sqrt{\frac{1}{n-1}\sum_{lp}\left(m_{1,lp} - \overline{m}_1\right)^2} \text{ with } \overline{m}_1 = \frac{1}{n}\sum_{lp} m_{1,lp} \tag{3.2}$$

The estimated second moment is shown in Equation (3.3), the derivation of which is based on equating the estimated variance S^2 in Equation (3.4) and the pooled variance of the predictions $S_{(p)}^2$ in Equation (3.5).

$$\hat{m}_2 = \frac{\sum_{lp} n_{lp} - 1}{\sum_{lp} n_{lp}} \times \frac{\sum_{lp}\left(n_{lp} - 1\right)S_{lp}^2}{\sum_{lp}\left(n_{lp} - 1\right)} + \hat{m}_1^2 \tag{3.3}$$

$$S^2 = \frac{\sum_{lp} n_{lp}}{\sum_{lp} n_{lp} - 1}\left(\hat{m}_2 - \hat{m}_1^2\right) \tag{3.4}$$

$$S_{(p)}^2 = \frac{\sum_{lp}\left(n_{lp} - 1\right)S_{lp}^2}{\sum_{lp}\left(n_{lp} - 1\right)} \text{ with } S_{lp}^2 = \frac{n_{lp}\left(m_{2,lp} - m_{1,lp}^2\right)}{n_{lp} - 1} \tag{3.5}$$

3.2.4 Data Resolution Conversion

Although LPs model unit arrivals and departures, they do not share every interdeparture time, considering the possible communication overhead. Instead, as discussed in Section 3.2.2, the first two moments of interdeparture times are published. This implies that an LP will not obtain the detailed information more than the first two moments upon requests. Given only the first two moments, arrivals can be generated by various approaches. Two methods are adopted: the first addresses the cases in which service times follow exponential distributions and the second covers the remaining cases.

In the cases where all service times follow exponential distributions, each LP approximates the arrival processes on the links across its modeled subnetwork as Poisson processes. For a specific Poisson arrival process, the mean is set to the given first moment while the given second moment is discarded; the interarrival times are generated by random sampling. On the other hand, when the service times are non-exponential, the arrival processes are modeled as renewal processes with gamma interarrival times. This approximation is based on Whitt’s methodology [54, 55] where the shape parameter α and the scale parameter β of a gamma distribution are estimated using the method of moments; see Equations (3.6) and (3.7). In these equations, m_1 and m_2 are the first two moments, respectively.

$$\alpha = \frac{m_1^2}{m_2 - m_1^2} \tag{3.6}$$

$$\beta = \frac{m_2 - m_1^2}{m_1} \tag{3.7}$$

3.2.5 Rollback Detection

Producing or removing a prediction may change the “big-picture view” of the corresponding flow state that this prediction is associated with. If the change is beyond a certain range, the previous (old) state is considered invalid. That is, the LPs that had used the old state need to be notified so that they can roll back and restart their simulations with a more “update-to-date” value based on the current state. The procedure for determining if a rollback is necessary is referred to as the “rollback detection mechanism,” which is mainly about comparing the old state against the currently-available predictions. Note that a previously-shared prediction may be removed when the sharing LP triggers a rollback to invalidate the prediction, which will be detailed in Section 3.2.6.

The first part of the rollback detection mechanism compares the value an LP once used for modeling a flow state against the currently-available predictions of the respective flow state. The comparison is as follows where only the first moments of the predictions are taken into account. Let $\overline{m}_1$ and S_1^2 be the sample mean and the sample variance of these first moments, respectively; see Equations (3.8) and (3.9) as a repetition of Equation (3.2). In these equations, n is the number of the predictions. In the cases with n less than 2, this rollback detection mechanism aborts since the predictions are insufficient to derive a meaningful conclusion with regards to the validity of a value. Otherwise, with $n \geq 2$, by following a traditional quality control paradigm, estimates within the range $\overline{m}_1 \pm q\sqrt{S_1^2/n}$ are considered acceptable. (The effect of the potential correlation between the predictions is ignored for now.) In general, the constant q can be set to 3, which corresponds to a 99% confidence level. However, this assignment would result in many unnecessary rollbacks, wasting a substantial amount of computing resources without improving prediction accuracy. Hence, q is set to 4 based on the intuition that a larger q value leads to fewer rollbacks but less accurate results. The relationship between the q value, the computational cost, and the prediction accuracy is another direction for future work.

$$\overline{m}_1 = \frac{1}{n}\sum_{lp} m_{1,lp} \tag{3.8}$$

$$S_1^2 = \frac{1}{n-1}\sum_{lp}\left(m_{1,lp} - \overline{m}_1\right)^2 \tag{3.9}$$

The second part of the rollback detection mechanism is conducted if the value an LP once used (say v) is outside the range $\overline{m}_1 \pm q\sqrt{S_1^2/n}$. In this case, the LP will be notified to roll back its simulation back to when it started using v. Also, the notification will carry a new value of the flow state that v is associated with. This new value is calculated based on the currently-available predictions by invoking the aggregation mechanism described in Section 3.2.3.

3.2.6 Rollback Handling

A rollback notification to an LP contains an invalid value the LP had used, the time period associated with the usage (say, $t = [t_s, t_e))$, and a new value as a suggestion that the invalid one should be replaced with (say, $v_{new} = (n, m_{1,0}, m_{2,0}))$. Upon receipt of such a notification, an LP first revokes all the read and write operations it had performed with respect to the time period $[t_s, \infty)$ by invoking an anti-write operation with time t_s. In other words, all the read/write records in the STM after time t_s and also associated with this LP are erased. Such a revocation in turn triggers the rollback detection mechanism discussed in Section 3.2.5.

After the revocation, the LP reconfigures its simulation with the new value. Instead of rolling back to time t_s, the LP rolls back further so that the potential "mismatches" do not result in abrupt changes in predictions. Specifically, the LP rolls back to five minutes earlier (i.e., $t_s - 300$ seconds), and the input data within the period $[t_s - 300, t_s)$ are interpolated. This period is also referred to as the "coast-forward phase." The computation in this period is local to the executing LP. Hence, in this phase, LPs do not share predictions.

The specifics concerning the input data interpolation are as follows. The first moments are linearly interpolated. Let the first moment used at time $t_s - 300$ be $m_{1,300}$. Since $m_{1,0}$ should be used at time t_s, at time $t_s - d$ the LP uses $m_{1,d} = m_{1,0} - d\,(m_{1,0} - m_{1,300})$ / 300 as the first moment. The second moments are not linearly interpolated; instead, the coefficients of variation (CVs) are linearly interpolated. After then, the second moments are calculated from the CVs and the first moments since the CV at time $t_s - d$ can be represented by the first two moments $m_{1,d}$ and $m_{2,d}$, respectively; see Equation (3.10).

$$CV_d = \frac{\sqrt{m_{2,d} - m_{1,d}^2}}{m_{1,d}} \tag{3.10}$$

3.3 Experiments and Results

The ad hoc queueing network simulations introduced in Section 3.2 are evaluated under three scenarios. In the first scenario, the service times follow the exponential distribution with the mean equal to one second. In the second and third scenarios, the service times follow the gamma distributions with shape parameter α and scale parameter β or (α, β) = (2, 0.5) and (0.25, 4), respectively; in these two cases, the mean is αβ (which is 1) and the variance is $\alpha\beta^2$ (which are 0.5 and 4, respectively).

The evaluation compares the results from the ad hoc queueing network simulations against those from the corresponding traditional sequential simulations as well as the known theoretical results, where available. Since the first scenario induces a Jackson network, numerous steady-state network performance measures can be calculated by treating each node as an independent M/M/1 queueing station [68]. Nevertheless, for the rest two scenarios, deriving the theoretical results requires approximations, and hence the evaluation only relies on the estimates obtained from sequential runs, which are the runs where the entire network is modeled by a single simulator.

The evaluation focuses on two steady-state network performance measures: server utilizations and mean queue lengths. Since the modeled network is symmetric in structure, only the following nodes are considered: nodes 0, 1, 2, 3, 9, 10, 11, 18, 19, and 27 (see Figure 7).

Since the ad hoc queueing network simulations start in an empty and idle state, the effect of the initial transient period should be excluded from estimating the steady-state performance measures. Furthermore, computing estimates with a given (absolute or relative) precision necessitates a long simulation. An intuitive principle is to increase the run length of the simulations as the service-time variance grows. Hence, the following configuration varies by scenarios in order to deal with the above two issues: in the first two scenarios, in which the service-time CVs are both less than or equal to 1, the data collection starts after two hours in simulation time and lasts for 10 hours; in the third scenario, with the service-time CV equal to 2, the data collection starts after 10 hours and lasts for 30 hours. This configuration applies to both the ad hoc experiments and the sequential runs.

The simulation results are averaged over multiple simulation runs using different random number seeds. For each ad hoc queueing network simulation, 10 independent replications are performed. The point estimate and the approximately 90% confidence interval (CI) of a specific performance measure is calculated as follows. Since the ad hoc approach allows one node to be modeled by several LPs, one replication may generate multiple (possibly-correlated) predictions for the same performance measure. The representing estimate of a replication is defined to be the average of all the predictions from the r-th replication, that is, X_r in Equation (3.11). In the equation, $X_{r,lp}$ is an estimate produced by one LP and k is the number of all such estimates (i.e., $k = |\{X_{r,lp}\}|$); the value k is the same across replications. Following this definition, the point estimate is $\overline{X}$ in Equation (3.12) and the 90% CI is derived by considering RV_X in Equation (3.13) as the Student's t-distribution with 9 degrees of freedom; the parameter μ in the equation represents the true (unobservable) value of the performance measure. Note that Equation (3.11) also implies that the current design treats every LP equally.

$$X_r = \frac{1}{k}\sum_{lp} X_{r,lp} \quad (3.11)$$

$$\overline{X} = \frac{1}{10}\sum_{r=1}^{10} X_r \quad (3.12)$$

$$RV_X = \frac{\overline{X} - \mu}{S_X / \sqrt{10}} \text{ with } S_X^2 = \frac{1}{9}\sum_{r=1}^{10}\left(X_r - \overline{X}\right) \quad (3.13)$$

The comparison between the estimates from the ad hoc approach and those from sequential simulations is based on the condition that every node is modeled by the same number of LPs/simulators in both approaches. Hence, for a specific performance measure, if the respective node is simulated by *k* LPs in one ad hoc replication, the estimate from the sequential approach requires 10*k* independent runs (for 10 is the number of the ad hoc replications). Let Y_r be the estimate from the *r*-th sequential run. Then, the point estimate is $\overline{Y}$ in Equation (3.14) and the 90% CI is derived by considering RV_Y in Equation (3.15) as the Student's *t*-distribution with $10k - 1$ degrees of freedom; the parameter μ in the equation denotes the true value of the performance measure, same as that in Equation (3.13).

$$\overline{Y} = \frac{1}{10k}\sum_{r=1}^{10k} Y_r \quad (3.14)$$

$$RV_Y = \frac{\overline{Y} - \mu}{S_Y / \sqrt{10k}} \text{ with } S_Y^2 = \frac{1}{10k-1}\sum_{r=1}^{10k}\left(Y_r - \overline{Y}\right) \quad (3.15)$$

3.3.1 Regular Partitioning

This subsection focuses on the regular partitioning in ad hoc queueing network simulations. The modeled 8 × 8 grid network is partitioned into five overlapping portions: the top left, the top right, the bottom left, the bottom right, and the center portions (Figure 9). Each portion is a 4 × 4 subnetwork and is simulated by eight LPs. Hence, in total 40 LPs are deployed.

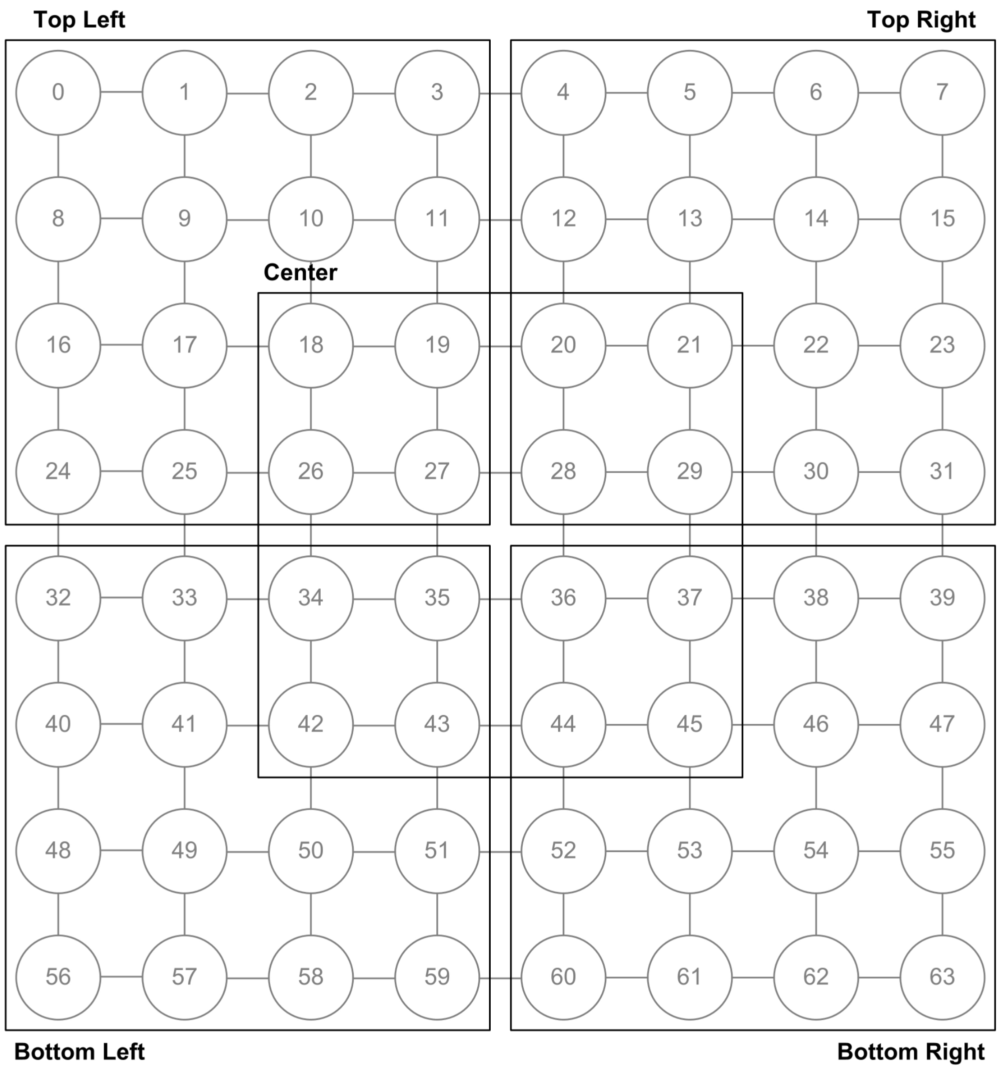


Figure 9: Regular Partitioning on 8 × 8 Grid Network

3.3.1.1 <u>Scenario 1: Exponential Service Times</u>

The configuration of exponential service times results in the modeled queueing network being a Jackson network so that the expected server utilization and the expected mean queue length of each server can be analytically obtained. Figure 10 depicts the relative differences between the utilization estimates from both simulation approaches and the exact values. In the figure, the horizontal axis is labeled with node identifications (IDs) along with the expected utilizations in parentheses. The relative differences are calculated by following Equation (3.16), in which $\bar{X}$ is an estimate from either the ad hoc approach or the sequential approach and μ is the corresponding exact value. This figure demonstrates that this ad hoc queueing network simulation performs well; it generates the estimates comparable to those from the respective sequential simulations. The relative differences for the mean queue-length estimates show similar trends, but rage wider with the maximum relative queue-length difference being 1.14% (see Figure 11).

$$\frac{\bar{X}-\mu}{\mu}\times 100\% \qquad (3.16)$$

Figures 12 and 13 contain the point estimates and the approximately 90% CIs for the steady-state server utilizations and mean queue lengths, respectively. In these figures, squares indicate the point estimates while ×'s mark the exact values. The results from both simulation approaches are close to each other and the CIs contain most of the exact values.

3.3.1.2 Scenario 2: Gamma(2, 0.5) Service Times

The service times in this scenario follow the gamma distribution with the shape and scale parameters (α, β) = (2, 0.5), denoted as Gamma(2, 0.5). The service times are less variable than those in Scenario 1 (with CV equal to $\sqrt{0.5}$ versus 1). In the absence of the analytical results, the formula for calculating the relative differences is adjusted to Equation (3.17), which describes the relative difference between an estimate from the ad hoc approach ($\bar{X}$ in the equation) and the respective one from the sequential approach ($\bar{Y}$ in the equation).

$$\frac{\bar{X}-\bar{Y}}{\bar{Y}}\times 100\% \qquad (3.17)$$

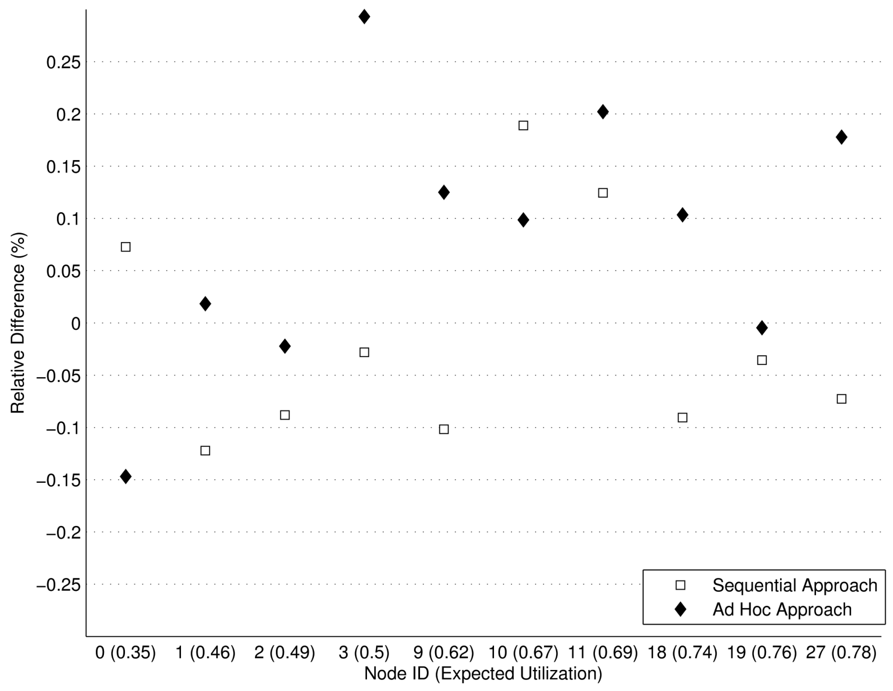


Figure 10: Relative Differences for Utilization Estimates with Regular Partitioning under Scenario 1

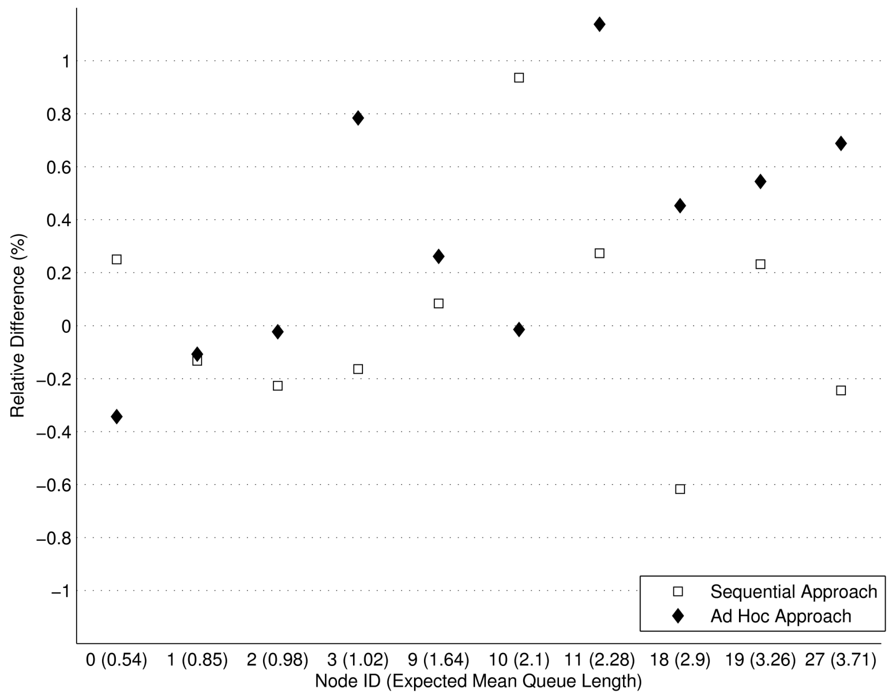


Figure 11: Relative Differences for Mean Queue-Length Estimates with Regular Partitioning under Scenario 1

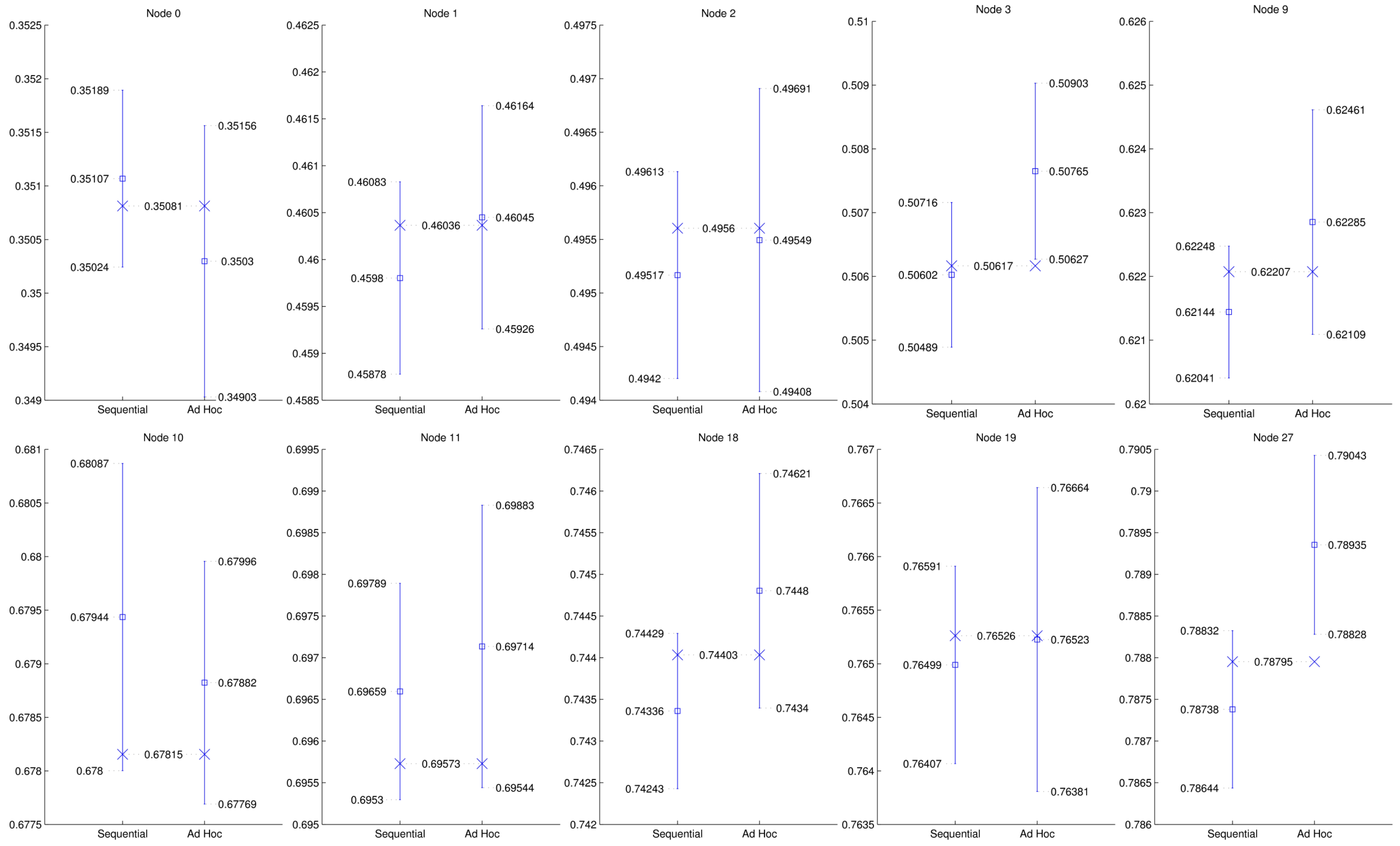


Figure 12: Point Estimates and 90% CIs for Utilization Estimates with Regular Partitioning under Scenario 1

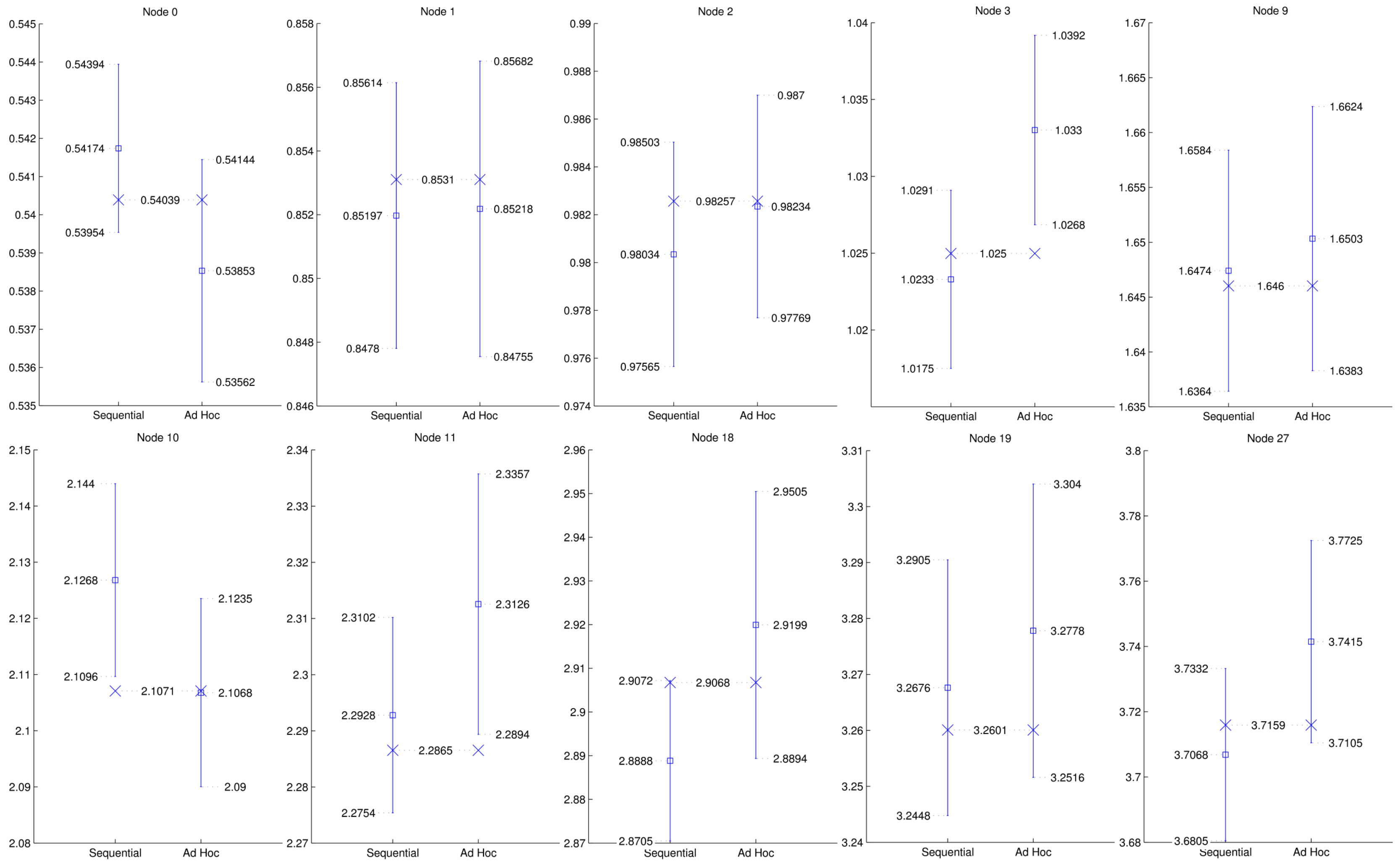


Figure 13: Point Estimates and 90% CIs for Mean Queue-Length Estimates with Regular Partitioning under Scenario 1

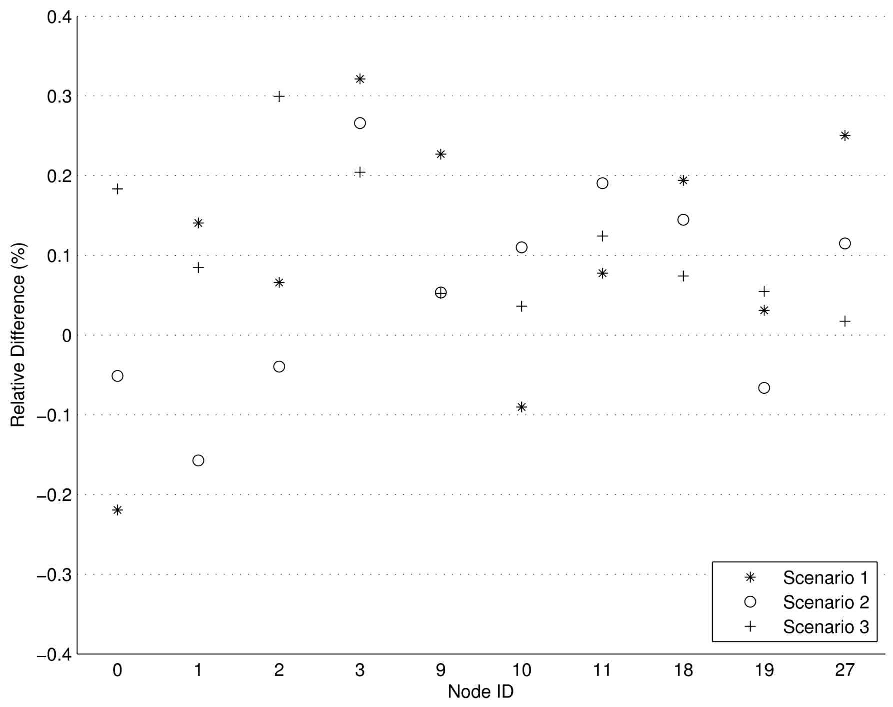


Figure 14: Relative Differences for Utilization Estimates with Regular Partitioning

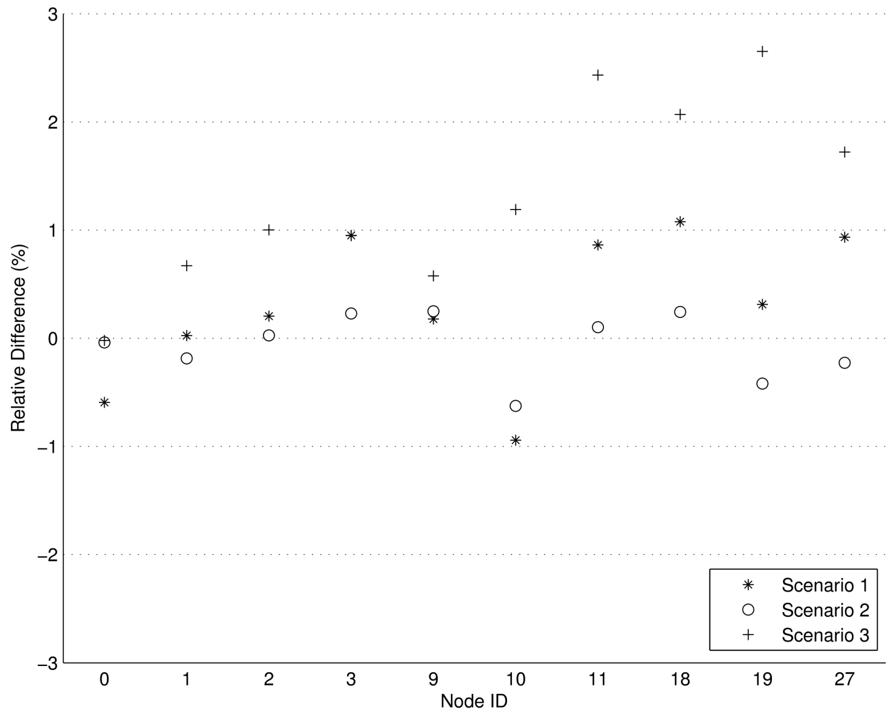


Figure 15: Relative Differences for Mean Queue-Length Estimates with Regular Partitioning

Figures 14 and 15 plot the relative differences for the utilization and the mean queue-length estimates, respectively, along with the results from Scenario 1 for comparison. For both scenarios, the relative differences for the utilization estimates hover around ±0.2% and are within the range ±0.35%. The relative differences for the mean queue-length estimates range wider but are typically within ±1%.

3.3.1.3 Scenario 3: Gamma(0.25, 4) Service Times

This scenario sets the service-time distribution to Gamma(0.25, 4), with CV equal to 2, which implies the service times are more variable than those in both Scenarios 1 and 2. Figures 14 and 15 also include the results from this scenario. The relative differences for the utilization estimates maintain a level of closeness similar to the previous results, but are all positive. This pattern is more pronounced with the mean queue-length estimates, especially on the nodes with high server utilizations. Specifically, the mean queue-length estimates from the ad hoc approach can be up to 3% larger than the respective estimates from the sequential approach. This issue also has surfaced in Scenario 1 but is not apparent under Scenario 2 with the lower service-time CV. A further study of the issue is detailed in Section 3.3.3.

3.3.2 Irregular Partitioning

This subsection examines the effect of the irregular partitioning on the estimation accuracy of ad hoc queueing network simulations.

3.3.2.1 Center-Weighted Node Coverage

The term "node coverage" refers to the pattern of LPs collectively modeling a queueing network. In the previous experiments in Section 3.3.1, the center part of the modeled 8 × 8 grid network is covered by twice as many LPs as those modeling the bordering nodes. This is referred to as the "center-weighted node coverage."

For the sake of comparison, the first set of experiments evaluates the irregular partitioning along with this center-weighted node coverage. That is, within one replication that contains 40 LPs, each of the "central" 16 nodes is modeled by 16 LPs while every other node is covered by 8 LPs. Many partitioning layouts satisfy this requirement and Figure 16 provides eight of them. For example, Layout 1, depicted in Figure 16(a), partitions the network into five overlapping portions: each portion is denoted by a rectangle with a distinguishing line style. The numbers in rectangles are node IDs. All the eight layouts in Figure 16 involve five portions. Since in total 40 LPs are deployed in one replication, each portion is modeled by eight LPs. Furthermore, Layout 9 (not depicted) is constructed by mixing all the portions in Layouts 1–8. As each layout contributes five portions, Layout 9 contains 40 portions, which leads to each portion being simulated by exactly one LP in a replication.

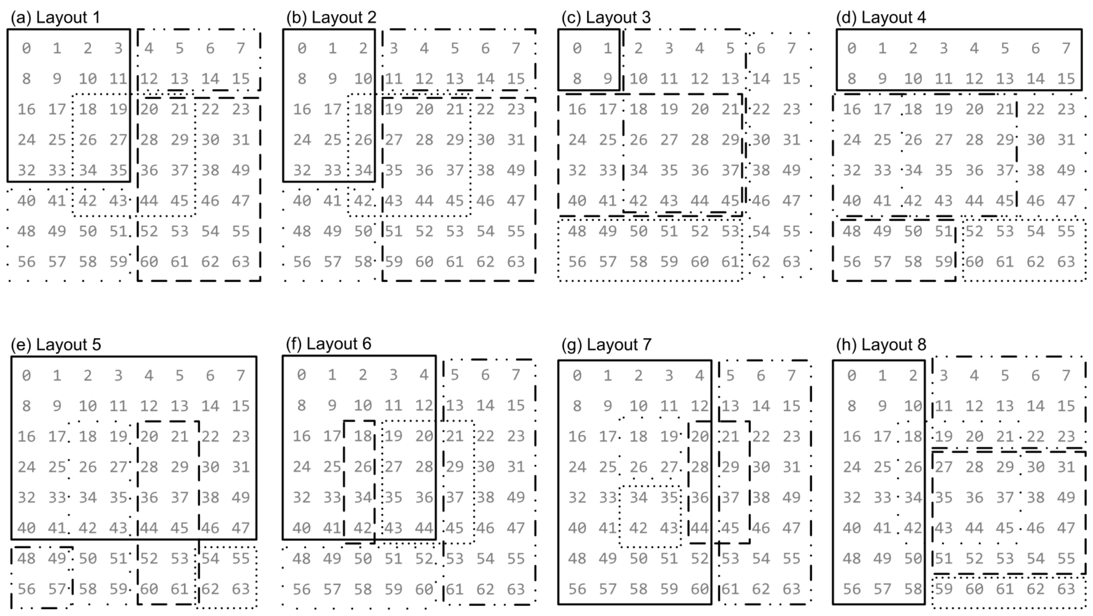


Figure 16: Irregular Partitioning Layouts on 8 × 8 Grid Network with Center-Weighted Node Coverage

All the nine layouts are evaluated. The experimental settings are the same as those in Section 3.3.1, except the partitioning. For brevity, the focus is solely on the mean queue-length estimates.

The first experiment applies the configuration of Scenario 1; that is, the service times follow the exponential distribution with the mean equal to one second. Figure 17 shows the relative differences between the mean queue-length estimates from the ad hoc approach and the corresponding exact values (i.e., the analytical results). The maximum relative difference is 1.88% from Layout 4. Although this value is slightly larger than the largest relative difference from Section 3.3.1 (1.14%), it does not seem to indicate that any specific partitioning layout deteriorates the estimation accuracy.

Moreover, all the nine layouts are evaluated with the configurations in Scenarios 2 and 3, and the results are presented in Figures 18 and 19, respectively. These figures plot the relative differences between the mean queue-length estimates from the ad hoc approach and those from the sequential approach. The minor decrease in accuracy suggests that these partitioning schemes do not appear to be a major issue. Under Scenario 2, the maximum relative difference is 1.3% from Layout 3 (compared to 0.63% under the regular partitioning scheme). Under Scenario 3, the maximum relative difference is 3.93% (from Layout 8) while it is 2.65% under the regular partitioning scheme. In addition, the overestimation issue observed in the previous experiments resurfaces (see Section 3.3.3 for further discussion).

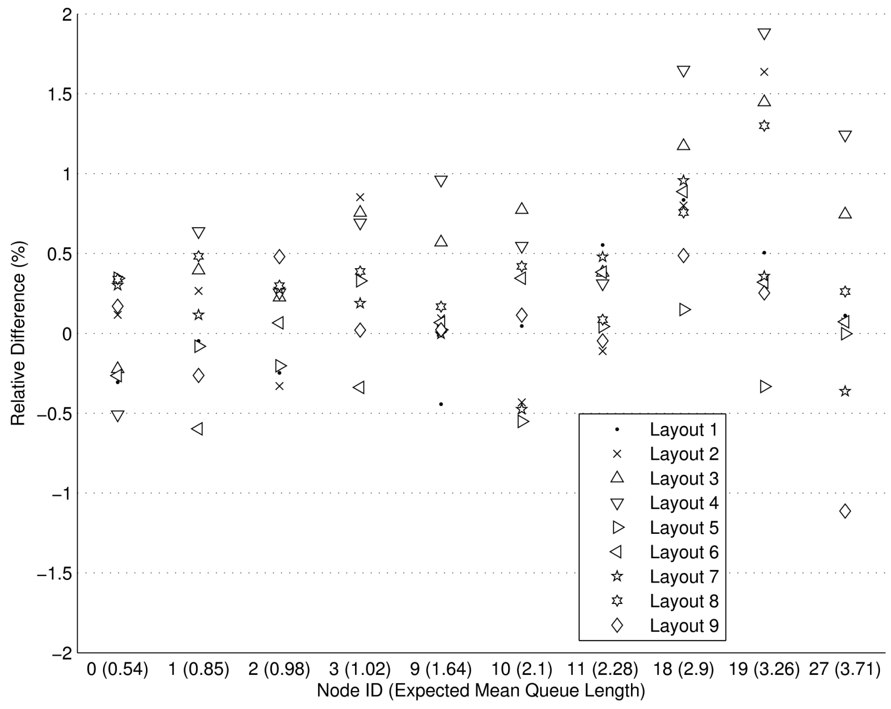


Figure 17: Relative Differences for Mean Queue-Length Estimates with Irregular Partitioning and Center-Weighted Node Coverage under Scenario 1

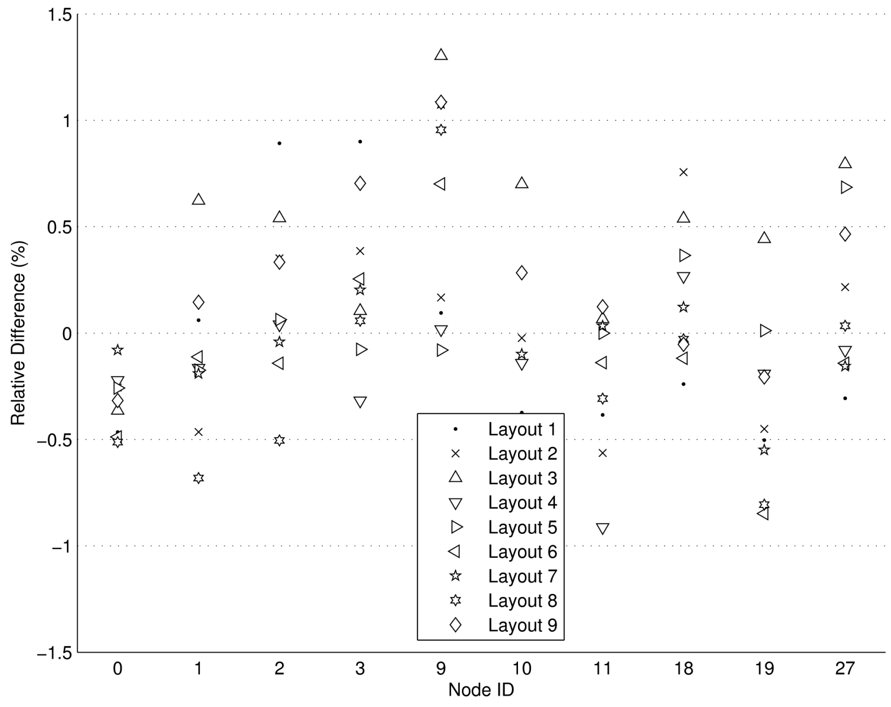


Figure 18: Relative Differences for Mean Queue-Length Estimates with Irregular Partitioning and Center-Weighted Node Coverage under Scenario 2

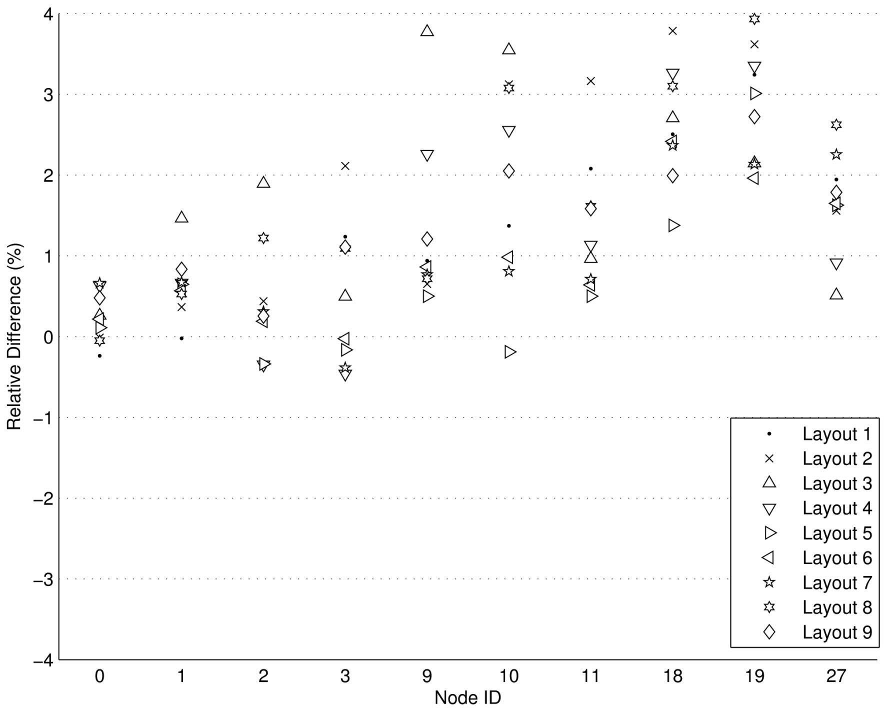


Figure 19: Relative Differences for Mean Queue-Length Estimates with Irregular Partitioning and Center-Weighted Node Coverage under Scenario 3

3.3.2.2 Balanced Node Coverage

As opposed to the center-weighted node coverage, the term "balanced node coverage" refers to the situation in which every node in the modeled network is simulated by the same number of LPs. In designing the experiments with this balanced node coverage, a further question arises that concerns how many LPs should be deployed to model each node. In general, the answer depends on not only the characteristics of a SUI but also the goals and constraints of a modeling and simulation task. Intuitively, a small number of LPs should be avoided because insufficient predictions may introduce additional variation, which would lead to unnecessary rollbacks. Hence, the following experiment borrows the experience from the previous ones and constructs a layout in which each node is modeled by eight LPs. The influence of this coverage redundancy on the estimation accuracy and variability requires further exploration.

A layout with the balanced node coverage is shown in Figure 20; each solid rectangle represents a portion while the dashed squares denote the modeled network for assisting in positioning the portions. The size of a portion is specified on the top of the corresponding square. Since each replication deploys 40 LPs and this layout contains 40 portions, each LP models exactly one distinct portion. Same as those layouts in Section 3.3.2.1, this layout is evaluated under all the three scenarios, and the experimental settings are the same as well, except the partitioning.

Since Scenario 1 can be easily analyzed theoretically, as mentioned earlier, the relative differences between the mean queue-length estimates from both simulation approaches and the analytical values are depicted in Figure 21. By contrast, Figure 22 plots another evaluation metric, the relative differences between the mean queue-length estimates from the ad hoc approach and the respective values from the sequential approach for all the three scenarios. All these differences are close to those from the previous experiments: they are within the same order of magnitude. Although slight increases in difference are observed, this partitioning method does not appear to significantly affect the prediction accuracy of the ad hoc approach. In addition, the data support the claim that once a node is modeled by a sufficient number of LPs, the marginal benefit of additional coverage is minimal.

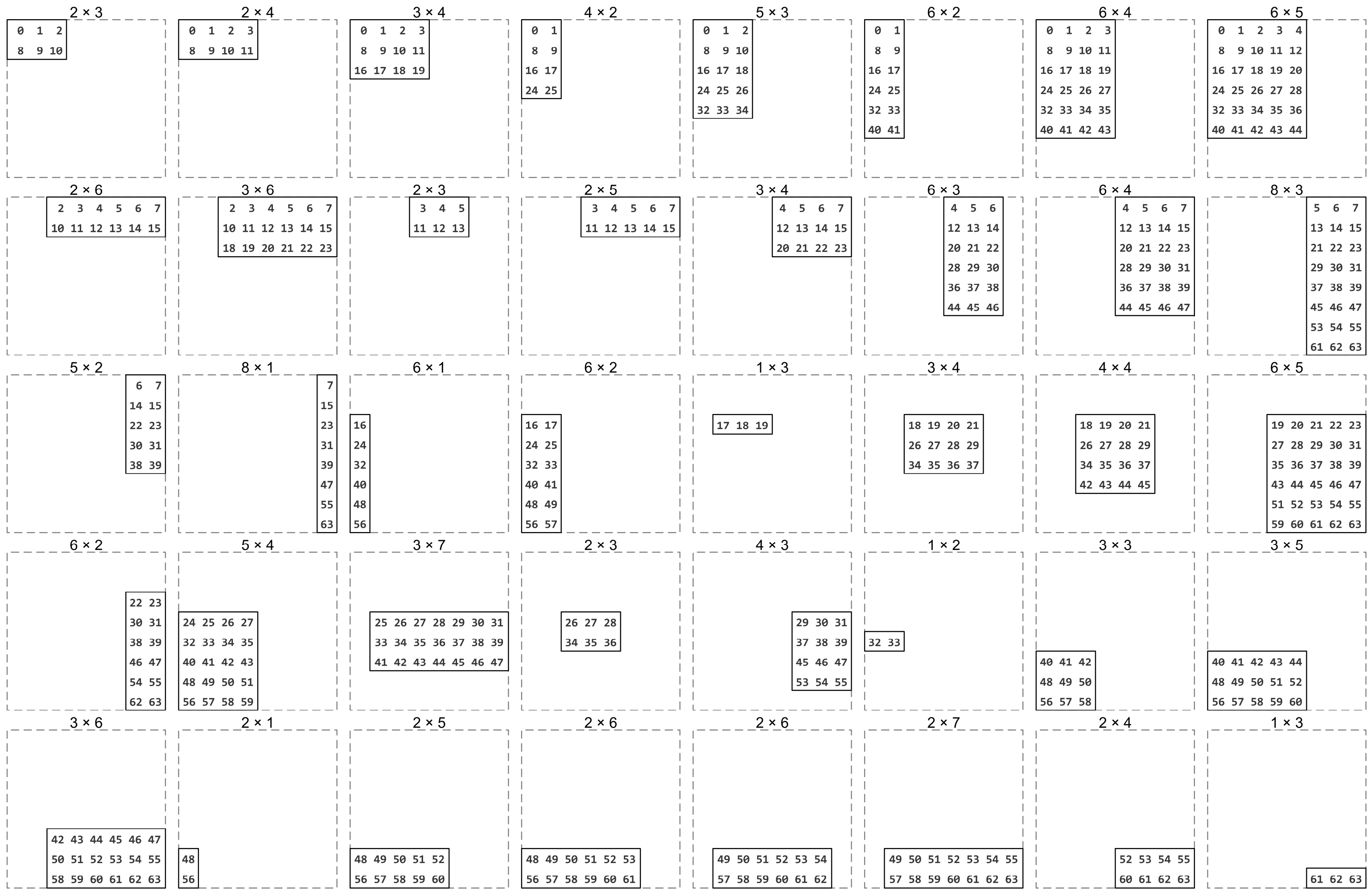


Figure 20: An Irregular Partitioning Layout on 8 × 8 Grid Network with Balanced Node Coverage

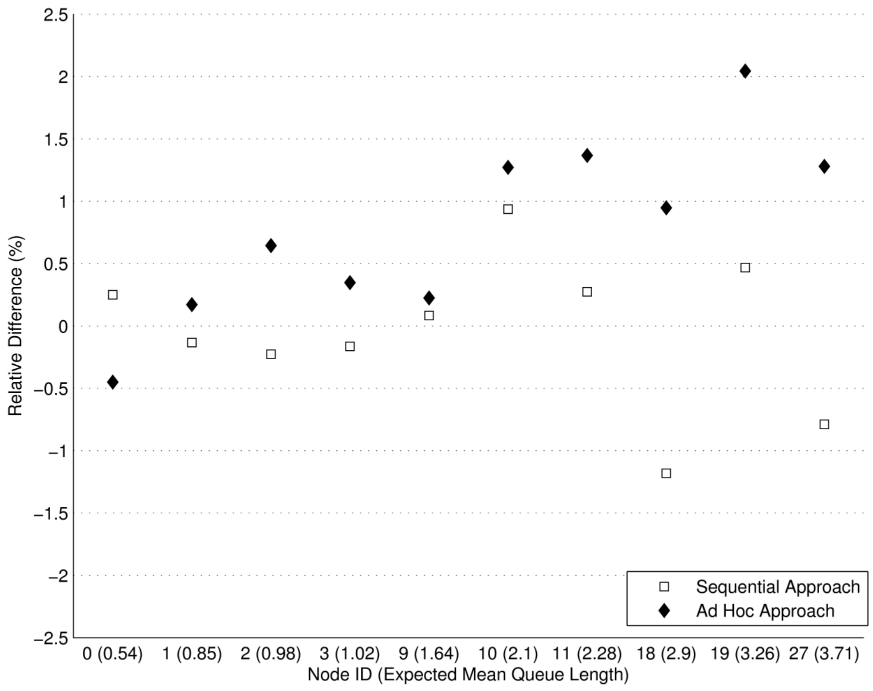


Figure 21: Relative Differences for Mean Queue-Length Estimates with Irregular Partitioning and Balanced Node Coverage under Scenario 1

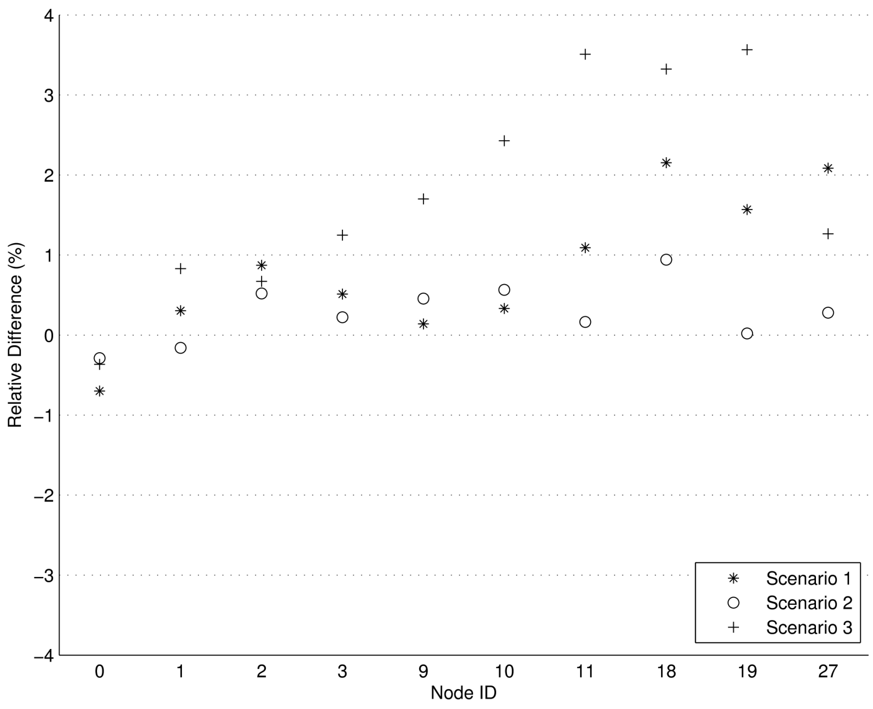


Figure 22: Relative Differences for Mean Queue-Length Estimates with Irregular Partitioning and Balanced Node Coverage

3.3.3 Overestimation

This subsection explores the overestimation issue observed in the mean queue-length estimates from the ad hoc approach. It appears that this issue is a result of the modeling assumptions for the arrival processes across two subnetworks; further, the overestimation becomes more apparent as the service-time CV increases. Recall that the arrivals are modeled with the following approximations: (1) the interarrival times on one link are IID from either an exponential or a gamma distribution (with the parameters estimated dynamically), and (2) the arrival processes on different links are independent. The following experiments attempt to verify the conjecture that the overestimation errors are because of these two assumptions.

This study uses sequential simulations to mimic the behavior of unit arrivals across two subnetworks in ad hoc queueing network simulations. Specifically, the first half of the study involves a sequential simulation of the original (unmodified) queueing network; during the simulation, the interarrival times are collected on a subset of links that represent the boundary links of potential LPs (see Figure 23). Then in the second half, another sequential simulation models the modified queueing network in which those boundary links are removed. The arrivals on the removed links are generated using the collected interarrival times from the first simulation. In generating unit arrivals, the collected interarrival times on each of those links are treated as IID samples from an unknown distribution, so the arrivals are generated based on random sampling with replacement.

Firstly, the experiment described next intends to show that the unmodified and modified networks are practically the same (e.g., generating numerical results that are close to each other) when the arrival process at a boundary link is renewal. To achieve this, the external arrival rate to node 28 is set to 0.5 (instead of 1 / 6 for other nodes). This leads to extremely-high traffic intensity at node 28, which in turn results in the IID interdeparture times of the processed units moving from node 28 to node 27, given the IID service times of node 28. In other words, approximating the flow on link 136 (see Figure 23) should not induce a significant error on the mean queue-length estimate of node 27, which is confirmed by the results: the relative difference between the point estimate from the modified network (12.512) and that from the original network (12.427) is –0.68%. Also, the 90% CIs are (12.312, 12.542) and (12.407, 12.617) for the modified and original networks, respectively. Note that this experiment servers as a basis for the next one.

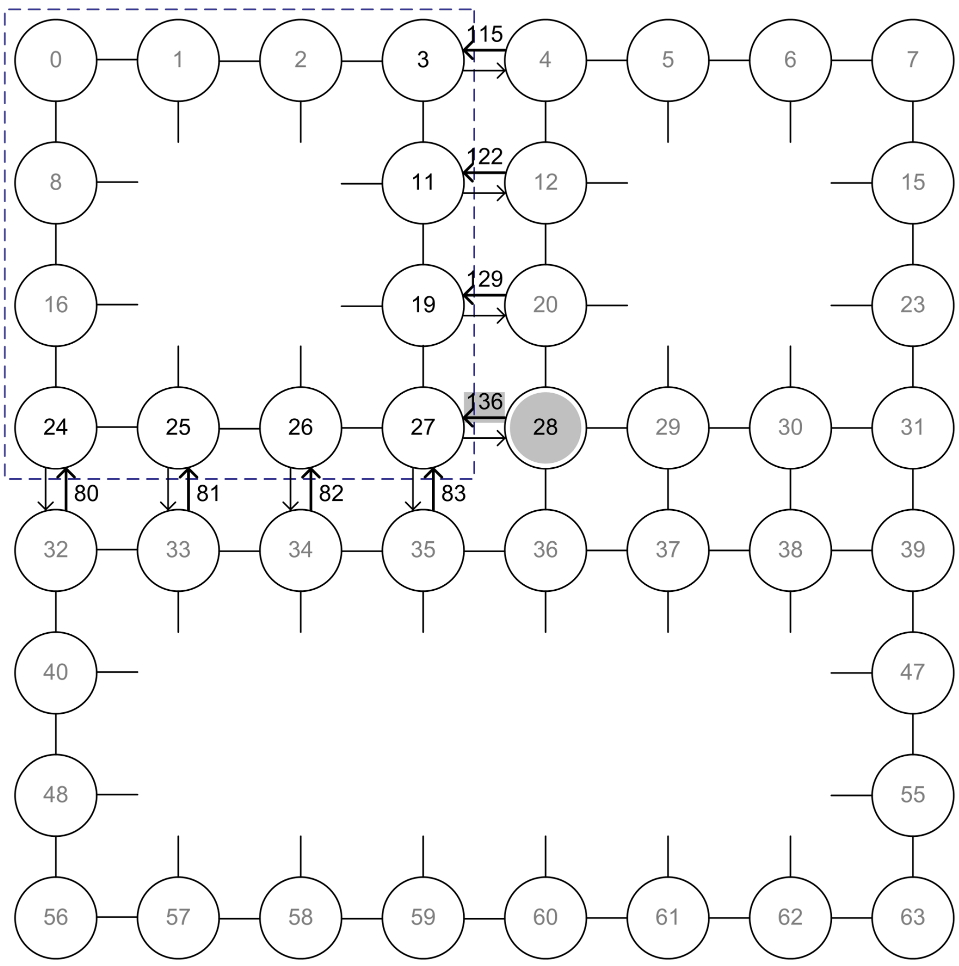


Figure 23: An 8 × 8 Grid Network for Imitation of Ad Hoc Experiments

Given that similar interdeparture time processes produce similar results, the impact of the approximated renewal arrival processes against the actual arrival processes can be quantified based on the relative differences between the estimates from the modified and original networks. Without loss of generosity, the next experiment focuses on the overestimation of the mean queue-length estimate for node 27 under Scenario 3, and attempts to imitate the LPs that model the top left portion in the regular partitioning. That is, the arrival processes of interest are those on links 80, 81, 82, 83, 115, 122, 129, and 136 (see Figure 23). These links are removed with substituting (approximated) arrival processes in the modified network.

The results from this experiment confirm the conjecture mentioned at the beginning of this subsection. The relative difference between the point estimate from the modified network (8.348) against that from the original network (7.967) is 4.79%, and the 90% CIs are (8.288, 8.408) and (7.916, 8.018), respectively. This significant relative difference and the non-overlapping CIs provide a firsthand demonstration of the relationship between the overestimation issue and the renewal arrival process assumption in the previous ad hoc queueing network simulations. In addition, it is noted from the three scenarios that as the service-time CV increases, the overestimation issue becomes more significant. This is intuitively reasonable since the arrival processes have positive autocorrelation. Also, this issue is consistent with the findings by Lester [69]. A positive conclusion based on these results is that the ad hoc approach can perform significantly better if the arrival processes can be modeled more precisely, without a substantial increase in computational requirements.

3.4 Buffered-Area Mechanism

This section proposes a method to reduce the estimation bias owing to the renewal approximation in the current design of ad hoc queueing network simulations. This method, called the "buffered-area mechanism," is based on "filtering" units before they reach boundary nodes. The details of the method are in Section 3.4.1, and Section 3.4.2 evaluates its effectiveness on three queueing networks with various configurations.

3.4.1 Principles of Method

The goal of the buffered-area mechanism is to improve estimation accuracy, which is achieved based on the fundamental idea to "link" a subnetwork with its neighbors. Specifically, given an LP modeling a subnetwork, the LP incorporates an extra portion at the border of this subnetwork into its modeling area. This enhancement is expected to reduce the impact from the approximations made at boundary input links. Take the bidirectional tandem queueing network in Figure 24 as an example. Each node in the network represents a queueing station and the processed units flow via the links connecting two nodes. Suppose that an LP is interested solely in the performance measures of nodes G, H, and I. Instead of simulating only these three nodes, this LP enlarges its modeling subnetwork by adding nodes F and J so that the arrival process approximation is made at the input links to nodes F and J (the arrows with a filled head). Furthermore, this LP shares the predictions (i.e., the statistics of departure processes) pertaining to the output links of nodes G, H, and I (the arrows with a hollow head), but not those of nodes F and J.

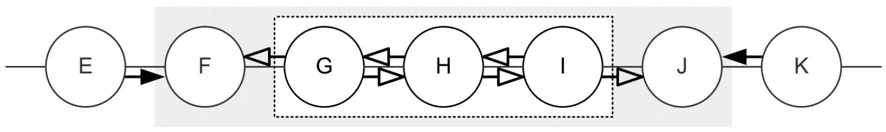


Figure 24: A Bidirectional Tandem Queueing Network

The effectiveness of the buffered-area mechanism depends on the extent to which the approximation bias can be “washed away” by increasing the distance between where output measures are taken and where input approximations are made. The longer the distance, the larger the buffered area, which in turn implies higher simulation cost.

3.4.2 Effectiveness Analysis

The accuracy gain through the buffered-area mechanism depends on several factors, including network topologies, connectivity, and unit routing mechanisms. This subsection studies the accuracy gain with respect to the sizes of buffered areas on three open queueing networks: an 11-node bidirectional tandem network, an 11×11 grid network, and an 8×8 grid network (Figure 25). The last one extends the study in Section 3.3.1.3 in order to illustrate that the buffered-area mechanism lessens the overestimation issue that has arisen there.

The following ad hoc queueing network simulations adopt the design in Section 3.2 and the same experimental settings as those in Section 3.3, unless otherwise specified. Some highlights are as follows. Every 30 seconds, LPs update the statistics of interdeparture times as well as query predictions for generating the arrivals on boundary input links. Aggregating multiple predictions (because of several LPs modeling the same area) is based on the kernel-density-estimation approach. On modeling the arrival processes at boundary input links, at the beginning of a simulation they are configured as Poisson processes with rate λ, but thereafter are assumed to be renewal processes with gamma interarrival times. The shape and scale parameters of these gamma distribution (α and β, respectively) are dynamically adjusted using the data from either periodically-invoked read operations or rollbacks. The rollback detection mechanism is based on an acceptable range, which is constructed following a quality control paradigm.

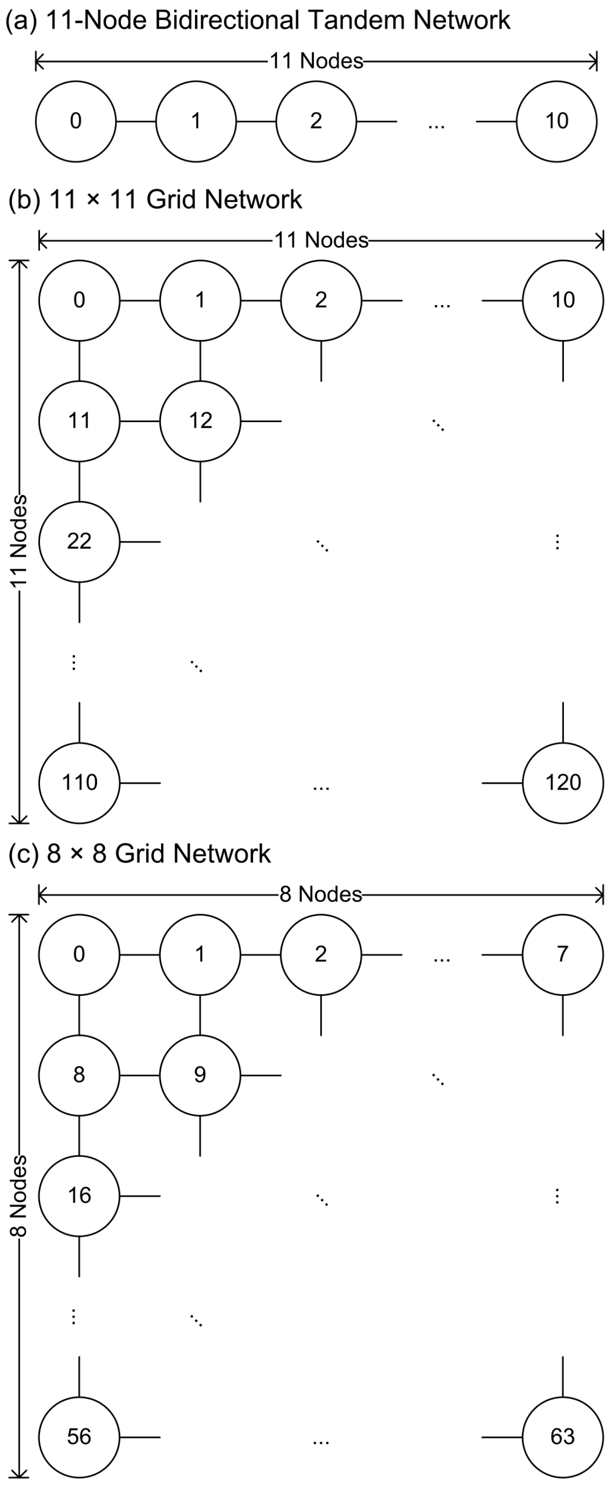
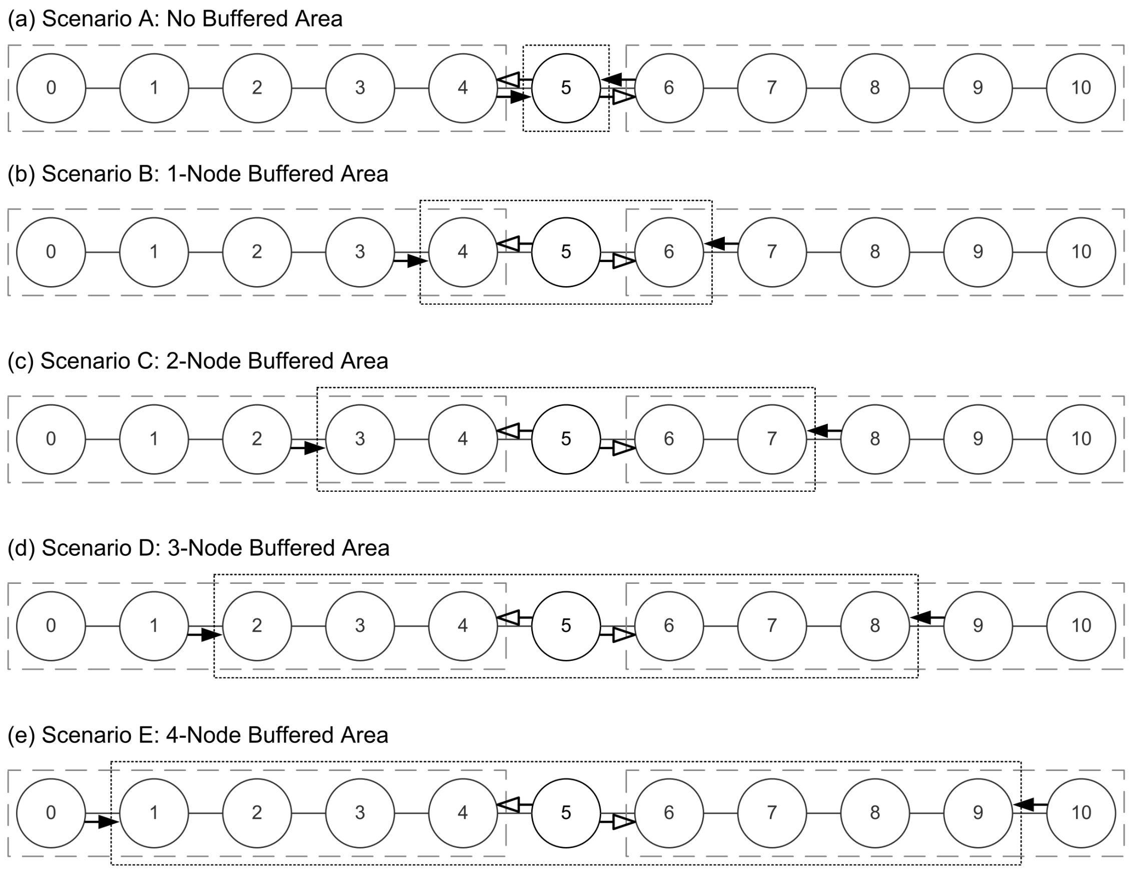


Figure 25: Open Queueing Networks

3.4.2.1 Case 1: 11-Node Bidirectional Tandem Network

This subsection focuses on the queueing network with 11 nodes in tandem (Figure 25(a)). Each node in the network is a single-server, infinite-capacity queueing station; the external arrivals follow a Poisson process with the rate λ per second. All service times are IID from a gamma distribution with the mean equal to 1 second. The served units at internal nodes may move to one of their neighboring nodes with the equal probability p or leave the network (with probability $1 - 2p$), while those at border nodes (i.e., nodes 0 and 10) would continue to a neighboring node with the probability p or leave the network (with probability $1 - p$).



(a) Scenario A: No Buffered Area
(b) Scenario B: 1-Node Buffered Area
(c) Scenario C: 2-Node Buffered Area
(d) Scenario D: 3-Node Buffered Area
(e) Scenario E: 4-Node Buffered Area
0
1
2
3
4
5
6
7
8
9
10

Figure 26: Application of Buffered-Area Mechanism to 11-Node Bidirectional Tandem Network

To simplify the analysis, the entire queueing network is partitioned into three portions: (1) nodes 0–4, (2) node 5, and (3) nodes 6–10. Only the LPs modeling the middle portion (the one with node 5) adopt the buffered-area mechanism. The sizes of buffered areas differ across scenarios; see Figure 26. These LPs update the statistics of the departure processes of node 5 (the arrows with a hollow head in Figure 26) while request the arrival information of different input links varying by scenarios (the arrows with a filled head in Figure 26).

The steady-state mean queue-length estimate of node 5 is the measure of interest. To derive both the point estimate and the 90% CI of this measure, multiple independent replications are needed. The number of replicated runs is set to 10 in the following ad hoc queueing network simulations. Within one replication, each portion is modeled by 10 LPs. Hence, as a basis for comparison, the estimate from the corresponding sequential simulations is based on 100 independent replications in order to equate the number of the replications that are allocated to node 5. Note that for both the simulation approaches, they start collecting the data pertinent to the measure after 3 hours in simulation time and the data collection lasts for 10 hours; this is for alleviating the effect of the initial transient.

Three sets of experiments are performed. The first two involve a heavily-loaded network with traffic intensity approximating 0.81 at node 5; these two cases intend to show that the buffered-area mechanism can mitigate severe bias issues. The third experiment concerns a network with low traffic intensity at node 5 (about 0.28); this configuration is to demonstrate that the buffered-area mechanism does not deteriorate the prediction accuracy.

Case 1(a). This heavily-loaded queueing network has external Poisson arrivals with the rate $\lambda = 1 / 6$ per second to each node. The routing probability p is 0.4 and the service times are from Gamma(0.25, 4). The service-time variation is high with the CV equal to 2. The overestimation resulting from the renewal approximation is anticipated [69].

Figure 27 plots the point estimates and the 90% CIs for the steady-state mean queue length of node 5. The overestimation is apparent under Scenario A, in which the buffered-area mechanism is not activated. The relative difference between the point estimate from Scenario A and that from the sequential approach is 10.71%. This severe estimation bias is reduced when one or more neighboring nodes on each side of node 5 are included in the buffered area. For example, in Scenario B the relative difference is 2.92%, which shows a substantial improvement at a moderate extra cost for modeling two auxiliary nodes. This estimation accuracy can be considered adequate since the 90% CI from Scenario B overlaps with that from the sequential approach. Further augmentation of the buffered area increases the accuracy in a diminishing manner: the relative differences for Scenarios C through E are 1.79%, 1.92%, and 0.78%, respectively. (The increase from 1.79% to 1.92% is likely owing to random error.)

Case 1(b). This study focuses on the same heavily-loaded queueing network as Case 1(a) albeit with low-variable service times: Gamma(4, 0.25) with CV = 0.5. The results are similar to Case 1(a), in which the expected underestimation according to [69] are observed. The relative differences for Scenarios A through E are –9.65%, –1.54%, –1.25%, –0.73%, and –0.41%, respectively; see Figure 28. Cases 1(a) and 1(b) together conclude that the buffered-area mechanism is capable of reducing the estimation bias with little extra effort.

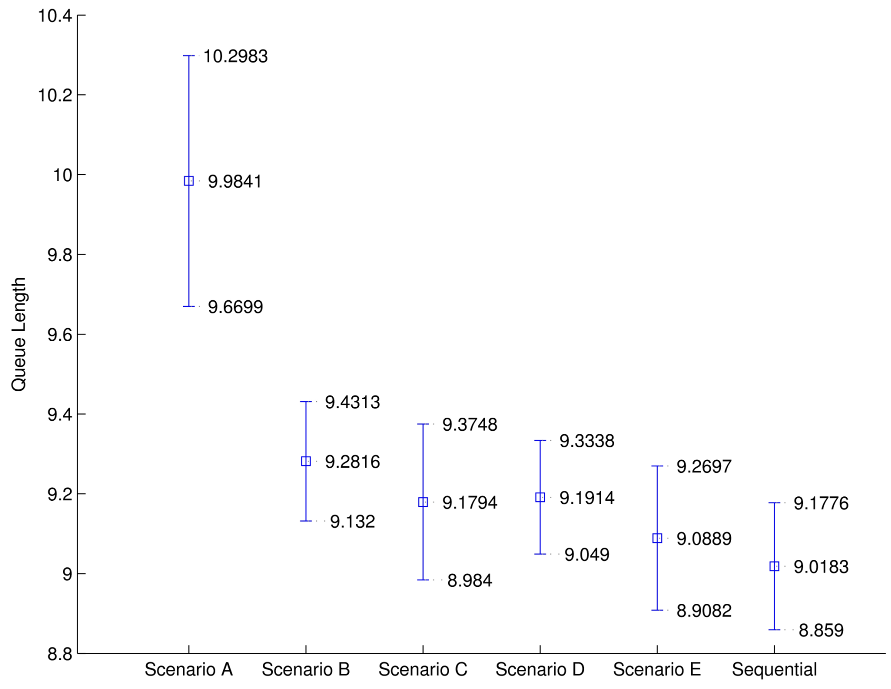


Figure 27: Point Estimates and 90% CIs for Mean Queue-Length Estimates of Node 5 under Case 1(a)

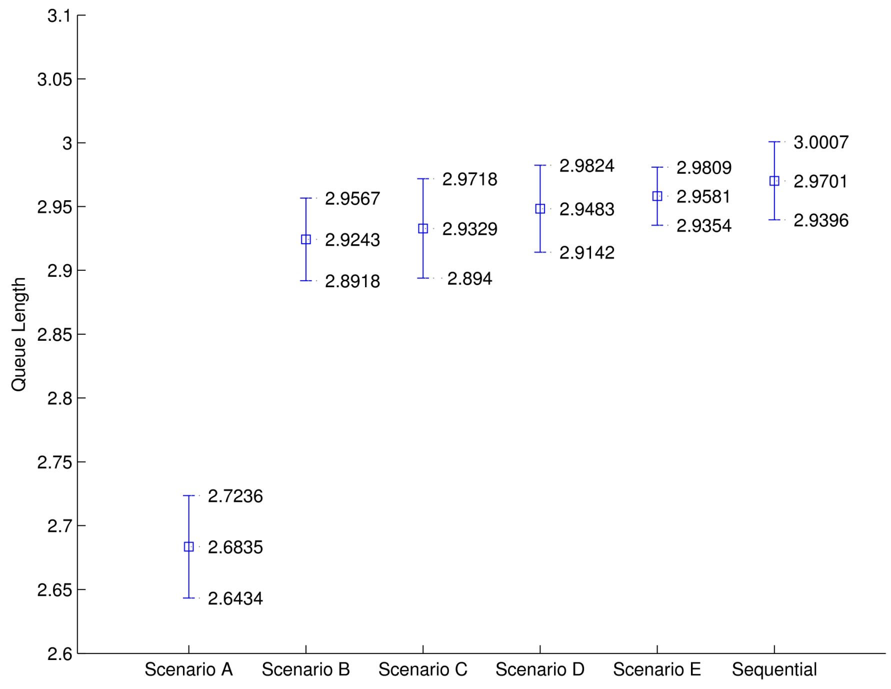


Figure 28: Point Estimates and 90% CIs for Mean Queue-Length Estimates of Node 5 under Case 1(b)

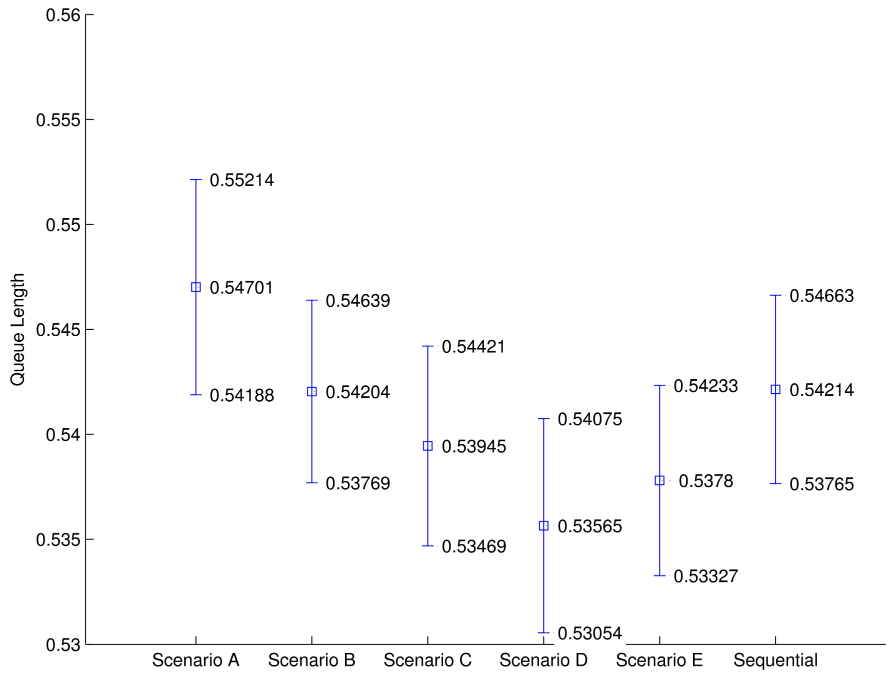


Figure 29: Point Estimates and 90% CIs for Mean Queue-Length Estimates of Node 5 under Case 1(c)

Case 1(c). The light network in this study reveals a less explicit overestimation issue than that in Case 1(a). The network configuration is as follows: $\lambda = 1 / 6$, $p = 0.2$, and Gamma(0.25, 4) service times. Figure 29 plots the results. The 90% CI from Scenario A overlaps with that from the sequential approach despite the small, positive relative difference 0.9%. Although the benefit of applying the buffered-area mechanism is modest, the mechanism does not cause any significant negative effect.

3.4.2.2 Case 2: 11 × 11 Grid Network

This subsection explores the performance of the buffered-area mechanism on a more complex open queueing network: an 11 × 11 grid network, which is a two-dimensional (2D) expansion of the network in Section 3.4.2.1. The nodes in this network are labeled from 0 to 120 starting at the upper-left corner and going across from left to right; see Figure 25(b). Hence, node 60 is the middle one. Each of these nodes contains a single-server, infinite-capacity queueing station with the external Poisson arrivals of the rate $\lambda = 1 / 6$ per second; the IID service times follow Gamma(0.25, 4). Since the served units are configured to move to one neighboring node with the equal probability $p = 0.2$ or leave the network, the traffic intensities range from 0.35 to 0.82 (at nodes 0 and 60, respectively). Similar to Case 1, the steady-state mean queue-length estimates of the middle node (i.e., node 60) is of particular interest as the middle node is expected to have the highest server utilization, which implies that the steady-state mean queue length is potentially to be greatly overestimated.

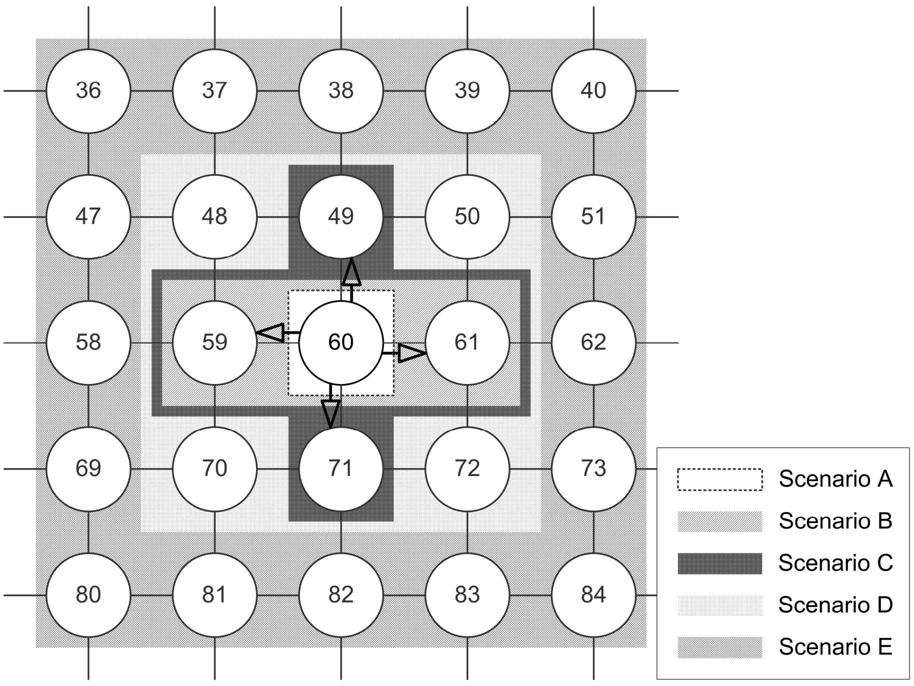


Figure 30: Application of Buffered-area Mechanism to 11 × 11 Grid Network

Given the focus on node 60, the 11 × 11 grid network is partitioned into two portions: one consisting of the entire network but node 60 and the other with node 60 only. The buffered-area mechanism is adopted by the LPs modeling the latter portion. The sizes and shapes of buffered areas vary in scenarios as illustrated in Figure 30. Scenario A includes no buffered area. In Scenario B only two neighboring nodes of node 60 are added to the buffered area while in Scenario C all the four neighboring nodes are included (making the buffered area cross-shaped). Scenario D extends the buffered area in Scenario C into a square buffered area; this design is intended to show that a square modeling area may be convenient for some simulation implementations despite the little gain in accuracy and the increase in simulation cost compared against Scenario C. In Scenario E, the square buffered area is expanded with one more node in every direction.

The experimental settings of the ad hoc queueing network simulations on these five scenarios are similar to those in Case 1, which are as follows. For the ad hoc approach, ten independent replications are performed; within every replication, one portion is modeled by 10 LPs. The results from the ad hoc approach are compared against those from 100 independent runs of the corresponding sequential simulation. Furthermore, since this 11×11 grid network is more complex than the bidirectional tandem network in Case 1, a longer transient period is expected. Hence, the data collection starts after 10 hours in simulation time and lasts for 30 hours.

Figure 31 depicts the point estimates and the 90% CIs for the steady-state mean queue length of node 60. The relative difference between the point estimate from Scenario A and that from the sequential approach is 10.05%, as expected. The relative difference drops to 5.14% with partial neighboring nodes included in the buffered area (Scenario B), and it drops further to 0.93% when all the four neighboring nodes are in the buffered area (Scenario C). As anticipated, Scenario D reveals the similar estimation accuracy as Scenario C where the relative difference for Scenario D is 1.14%. (The increase from 0.93% to 1.14% is likely from random error.) Additional evidence to support this claim is that the 90% CIs from both Scenarios C and D overlap. Also, the two CIs both overlap with that from the sequential approach although some positive bias remains. Further expansion of the buffered area improves the estimation slightly as the relative difference for Scenario E is 0.57%. To conclude, based on Cases 1 and 2, it appears that the buffered area should contain at least all the neighboring nodes of modeled subnetworks.

3.4.2.3 Case 3: 8 × 8 Grid Network

This subsection extends the study in Section 3.3.1.3 to demonstrate the effectiveness of the buffered-area mechanism in terms of bias reduction on modeling the 8 × 8 grid network with Gamma(0.25, 4) service times. The regular partitioning is adopted in this case, which ends up five subnetworks with one located in the middle and each of the other four covering a corner; see Figure 7.

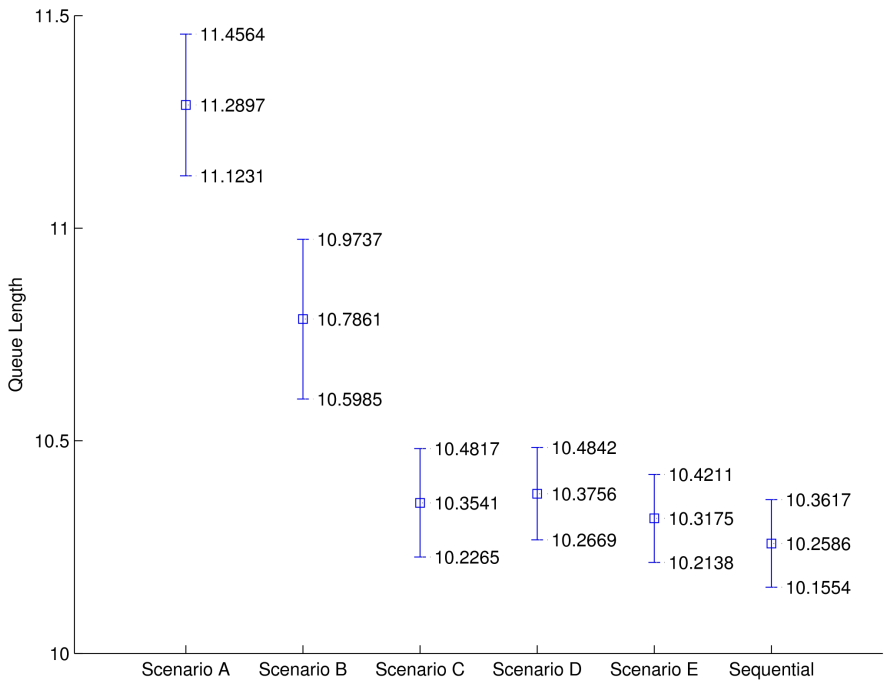


Figure 31: Point Estimates and 90% CIs for Mean Queue-Length Estimates of Node 60 under Case 2

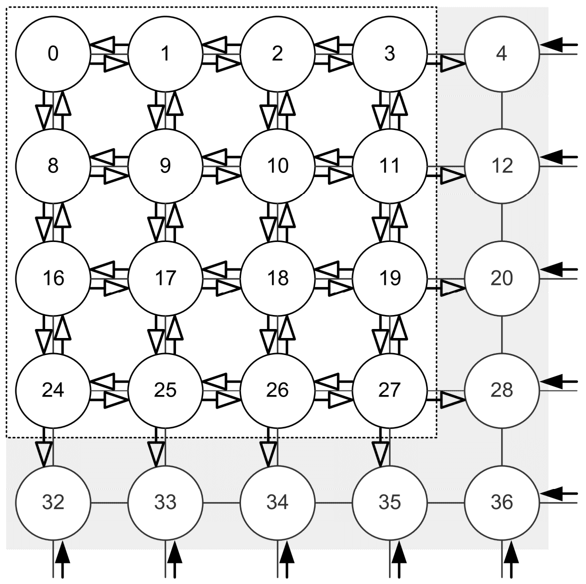


Figure 32: Application of Buffered-area Mechanism to 8 × 8 Grid Network—Top Left Portion

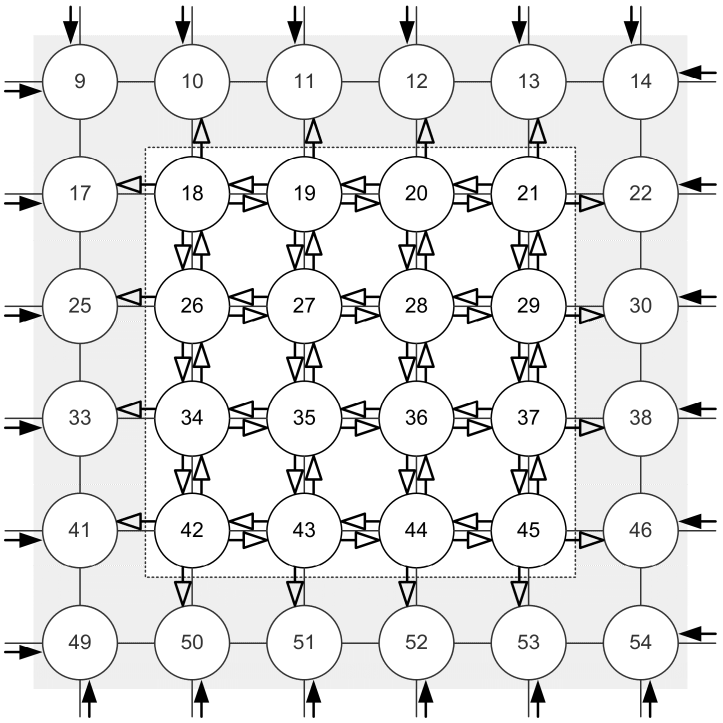


Figure 33: Application of Buffered-area Mechanism to 8 × 8 Grid Network—Center Portion

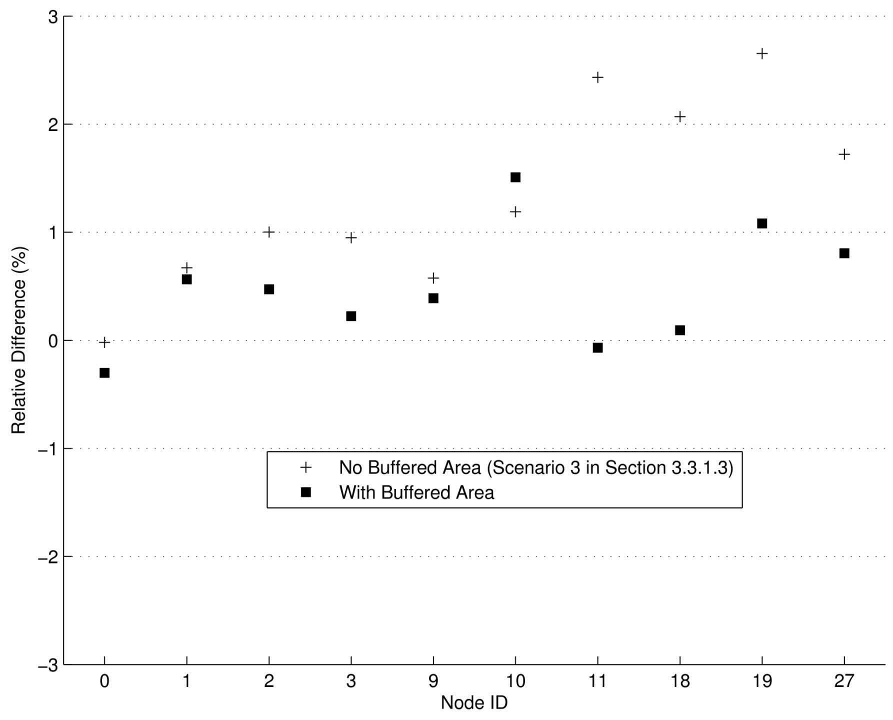


Figure 34: Relative Differences for Mean Queue-Length Estimates under Case 3

Based on the conclusion in Section 3.4.2.2, the buffered-area mechanism is implemented by enlarging the five subnetworks to encompass only the nodes immediately surrounding themselves, as illustrated in Figures 32 and 33. In both figures, the areas in white are the original modeled subnetworks while those in grey represent the buffered area. The buffered area of the top-left portion is in Figure 32, which serves as an example of the four corner portions. The buffered area for the center portion is in Figure 33.

Figure 34 plots the relative differences for the steady-state mean queue-length estimates with and without the buffered-area mechanism. The accuracy gain is apparent, especially for the nodes with high traffic intensities (e.g., nodes 18, 19, and 27). For example, the relative difference for the mean queue length at node 27 drops from 1.92% to 0.83%; at node 19, from 2.74% to 1.12%; and at node 18, from 1.93% to 0.12%. Although the overestimation issue remains, the bias is reduced substantially—more than 100%

improvement is observed in some nodes.

3.5 Conclusions

The performance of ad hoc queueing network simulations is a good indicator of the capability of the ad hoc approach since queueing networks are a widely-used, abstract model for a myriad of industrial systems. This chapter explored a preliminary design of ad hoc queueing network simulations by discussing the software implementation and the design of the involved mechanisms (e.g., the rollback detection mechanism), and illustrating some of the underlying statistical issues that come along with the design. The experiment results showed that the ad hoc queueing network simulations appear to be competitive to their sequential counterparts with respect to the accuracy of the steady-state network performance measures, such as server utilizations and mean queue lengths. However, it was also observed that the mean queue lengths tend to be slightly over-estimated, especially on the nodes with high traffic intensities.

In addition, this chapter examined the influences of partitioning methods on prediction accuracy. By employing the symmetric regular partitioning (with a “highlight” on the center part of the modeled network) as a baseline, nine partitioning layouts based on the irregular partitioning methodology were evaluated. The results showed that the nine layouts do not affect the estimation accuracy in any particular way; that is, the point-estimate biases are of the same order and the expected overestimation is revealed. Note that all these layouts involve more LPs in modeling the center part of the network (which happens to have heavier loads than other parts). To study the impact of this “central-highlight” design, another experiment was conducted on a partitioning layout where every node in the network receives the same amount of “attention” from LPs. The experiment result was compared to that from the baseline case, and the anticipated phenomena, as those in the last experiment, were observed. Overall, the simulation is not affected by partitioning layouts as long as the entire modeled network is modeled by a sufficient number of LPs.

To solve the overestimation issue, this chapter further analyzed the problem and proposed a buffered-area mechanism. The analysis concluded that the overestimation is caused by the imperfect approximations of the arrival processes at nodes on the boundary of modeled subnetworks. Hence, to reduce the effect of those input approximations, the proposed mechanism extends modeling areas to include the extra nodes at boundaries. In such case, execution efficiency is traded for prediction accuracy. The estimation bias is substantially reduced with a small increase in computational cost. This statement was supported by the empirical evidence.

CHAPTER 4

ON THE TRANSIENT RESPONSE OF OPEN QUEUEING NETWORKS USING AD HOC DISTRIBUTED SIMULATION

This chapter explores the ability of ad hoc distributed simulation to predict the transient behavior of physical systems. Capturing system dynamics is crucial to online simulations, e.g., to trigger modifications to the configuration of physical systems in response to events. For example, a sudden increase in traffic volume may indicate that changes in traffic signal plans for the responsive transportation system are necessary to help reduce congestion.

Transient-state analysis can be complex and computationally expensive, especially in fulfilling the real-time requirement for online analysis. Transient-state analysis concerns the system state varying over the span of time; it takes into account not only the system state at every time point but also the correlations among the prior and posterior system states. Instead, this study simplifies the analysis in evaluating the ad hoc approach: a sufficient number of time points are considered and the respective state predictions from the ad hoc approach are compared against those from the corresponding sequential simulations. Specifically, the evaluation discretizes the simulation time into small time intervals, and the output measures of interest (e.g., the queue-length estimate) are calculated by averaging the numbers within each interval.

The rest of this chapter is organized as follows. First, Section 4.1 examines the effectiveness of the preliminary ad hoc queueing simulation method introduced in Chapter 3 in the scenario that involves increase in external arrival rates; the method reveals a delayed response in capturing the propagation of the expanded number of arrivals throughout modeled queueing networks. To resolve this delayed-response issue, Section 4.2 proposes a new method and discusses the possible livelock issue that comes along with the new design. The new method, termed "iterative ad hoc queueing simulation method," is evaluated empirically in Section 4.3 under several network configurations; this includes a case with a large increase in arrivals over a short period of time, which leads to rapid increases in queue occupancy. Last, Section 4.4 concludes this study.

4.1 Delayed Responses in the Original Ad Hoc Queueing Simulation Method

The ad hoc queueing simulation method introduced in Chapter 3, which is referred to as the "original" method in the following context, is prone to delayed response to system dynamics because LPs share locally observed current state information as predictions of the future. Specifically, consider the case where a logical process (LP) models an object and shares state information with other LPs that use the information as input. As shown in Figure 35, at simulation time t, the LP computes a value based on the behavior of the object over the time period $[t - \Delta, t)$. The value becomes public as a predicted value of the object with respect to $[t, t + \delta)$, rather than $[t - \Delta, t)$. As a consequence, observations are not immediately reflected to the simulation model that requires the information, which results in a delayed response.

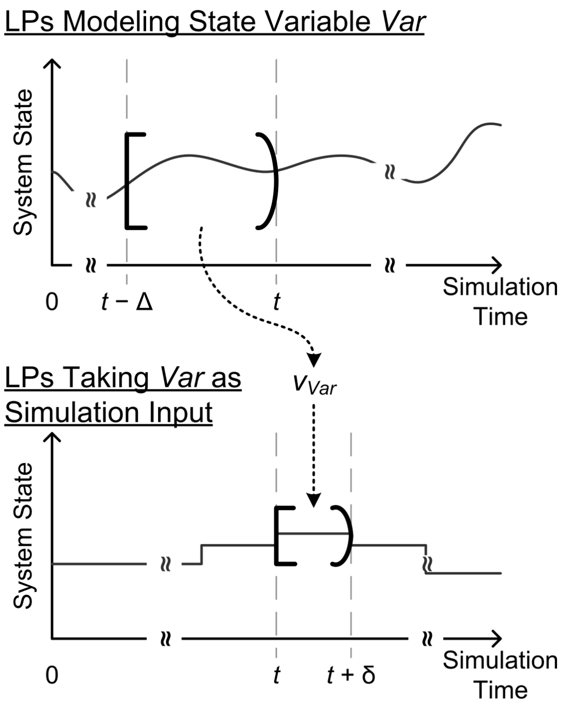


Figure 35: Information Sharing Mechanism Leading to Delayed Response

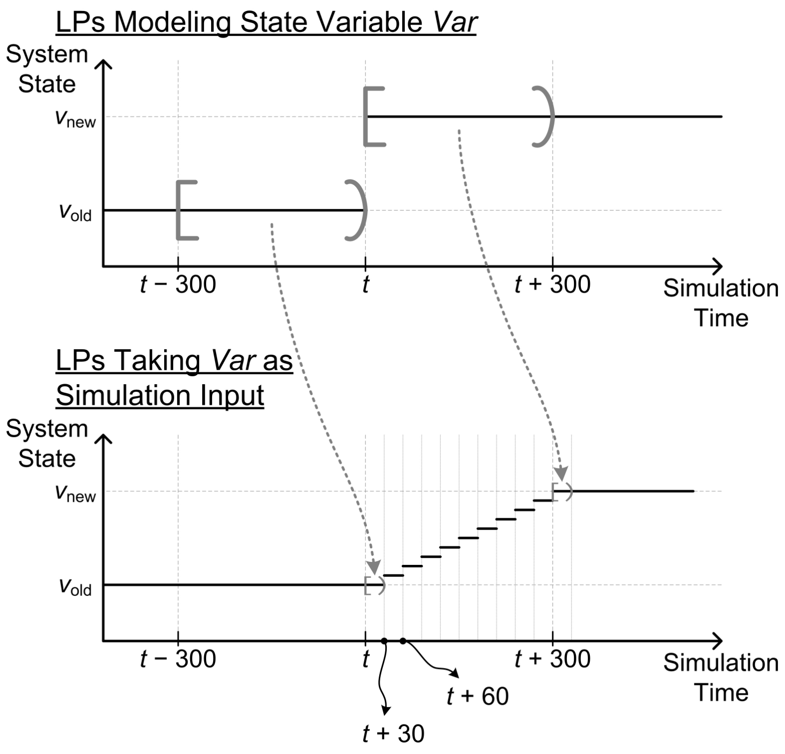


Figure 36: A Delayed-Response Example

Consider the previous design in Chapter 3 as an example where $\delta = 30$ and $\Delta = 300$, and the shared state values are computed by taking averages; see Figure 36. Assume the state of an object prior to the simulation time t is v_{old}, but it becomes v_{new} at t and remains at the level afterwards. Then, v_{new} is revealed for the first time in the prediction with respect to $[t + 30, t + 60)$, which is $0.9v_{\text{old}} + 0.1v_{\text{new}}$. More generally, the prediction with respect to $[t + 30i, t + 30i + 30)$ is $(1 - 0.1i) \times v_{\text{old}} + 0.1i \times v_{\text{new}}$ where $i = 1, 2, \ldots, 9$. As a consequence, v_{new} is not completely reflected until $t + 300$, which is 300 seconds beyond when the change occurred.

The following experiments illustrate the effects of δ and Δ. Three configurations are evaluated, all with $\delta = \Delta = 60$, 300, and 600. The rest of the simulation model is the same as that in Chapter 3, and the fundamental mechanisms are as follows. Each LP models an arbitrary queueing subnetwork using a sequential, discrete-event simulation. Every Δ seconds, these LPs update the mean interdeparture times on the links they simulate and request (or estimate if the data is unavailable upon request) the same information on the input links entering their modeling networks. The arrivals on those input links are modeled as Poisson processes. At the beginning of the simulation, the arrival rates are all set to λ, which is the same as the external arrival rate to every queueing station; thereafter the rates are dynamically estimated using the data from rollbacks. The rollback detection function is based on an acceptable range, which is constructed following a quality control paradigm. Since there may be several predictions from the LPs that model the same link, the data retuned to the requesting LPs are generated using a kernel-density-estimation approach.

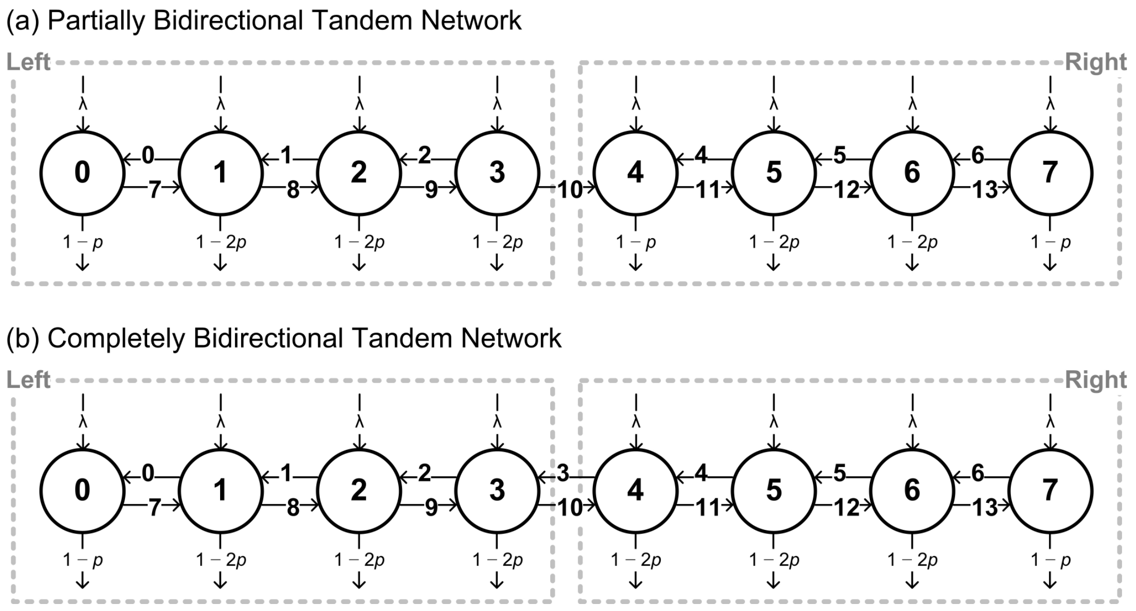

(a) Partially Bidirectional Tandem Network
Left
Right
λ
0
1
2
3
4
5
6
7
8
9
10
11
12
13
1 − p
1 − 2p
(b) Completely Bidirectional Tandem Network
Left
Right

Figure 37: 8-Node Tandem Queueing Networks

Furthermore, the experiments involve two open queueing networks with the intention of showing various degrees of impact due to the delayed-response issue; see Figure 37. The network in Figure 37(a) is an 8-node partially bidirectional tandem network. Each node represents a single-server queueing station with an unlimited buffer, first-come-first-served service discipline, and independent-and-identically-distributed (IID) exponential service times with the mean equal to 1 second. Each node has external Poisson arrivals with the rate λ. The probability of a processed unit moving to another node is *p*. Hence, a processed unit leaves the network with the probability $1 - p$ (at nodes 0, 4, or 7) or $1 - 2p$ (otherwise). The network in Figure 37(b), which is referred to as a "completely bidirectional tandem network," is almost the same to the former one with an exception that the processed units at node 4 may leave for node 3 with the probability *p*.

The experiments on both the networks involve 20 LPs in each replicate run: ten LPs modeling the leftmost 4 nodes and the remaining ten modeling the rightmost 4 nodes. In the simulations of the partially bidirectional tandem network, the shared information always goes from the left subnetwork to the right subnetwork. The right subnetwork "learns" the system dynamics in the left one through the changes in the flow rate (or, equivalently, the mean interdeparture time) of link 10. It is anticipated that the larger the Δ, the later the dynamics are detected. This phenomenon is expected to be worse in the simulations of the completely bidirectional tandem network since both the left and right subnetworks require information from each other.

The evaluation of δ and Δ considers two metrics over 10 IID ad hoc runs: the arrival rate across link 10 and the mean queue length at node 4. In one run, since the 10 LPs modeling node 4 (where link 10 enters) may use different arrival rates with respect to the same simulation time, these rates are averaged into one value. Then, the mean of the 10 values from 10 IID runs represents the point estimate. A similar calculation is performed to determine the mean queue length of node 4. Moreover, the mean queue length is estimated every 60 seconds.

The results from the corresponding traditional sequential simulations serve as ground truth. These results are based on 100 replicate sequential runs because 10 IID ad hoc runs deploy a total of 100 LPs to model one node. Note that these sequential simulations do not model the arrivals on link 10 as a Poisson arrival process with the rate being dynamically estimated (as is done by ad hoc simulations). Instead, the arrivals are the departures from node 3 filtered by the probability p. For the output, the rate is estimated by the corresponding mean interarrival time, which is estimated every 60 seconds based on the arrivals that appear since the last computation.

4.1.1 Case 1: Partially Bidirectional Tandem Network

The first experiment focuses on the partially bidirectional tandem network with $p =$

0.45 and, for the first 4 hours in simulation time, λ = 1 ⁄ 8 per second; the steady-state traffic intensities of the nodes range from 0.36 (nodes 1 and 4) to 0.66 (node 6). Afterwards, the external arrival rate of every node increases to λ = 1 ⁄ 6 per second, which leads to the steady-state traffic intensities growing to between 0.48 (nodes 1 and 4) and 0.87 (node 6).

Figure 38 plots the arrival rates of link 10 estimated by four different simulations: traditional sequential simulations and the ad hoc queueing network simulations with δ = Δ = 60, 300, and 600 seconds. This plot focuses on the transient period and hence the prior and the later parts are removed for now. The results match the expectation that larger Δ values give rise to longer delay in incorporating state changes. The length of the delay is approximately Δ except in the case where Δ = 60, when the delay is slightly larger. This is because the predictions have higher variation as they are based on smaller amount of data.

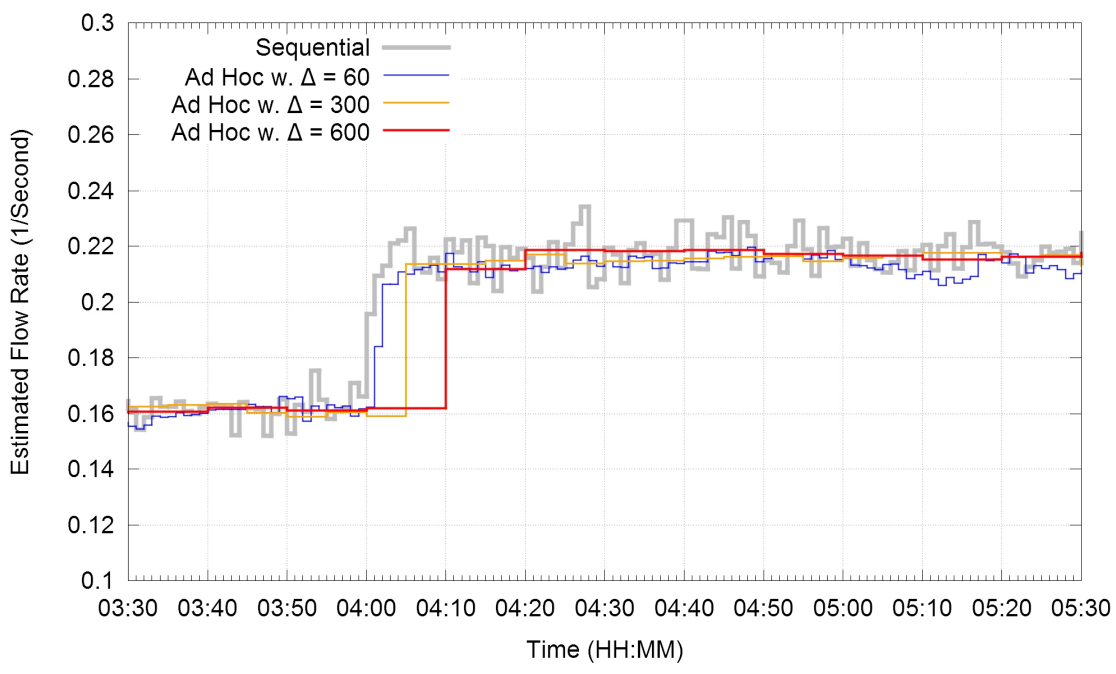


Figure 38: Estimated Arrival Rates across Link 10 under Case 1

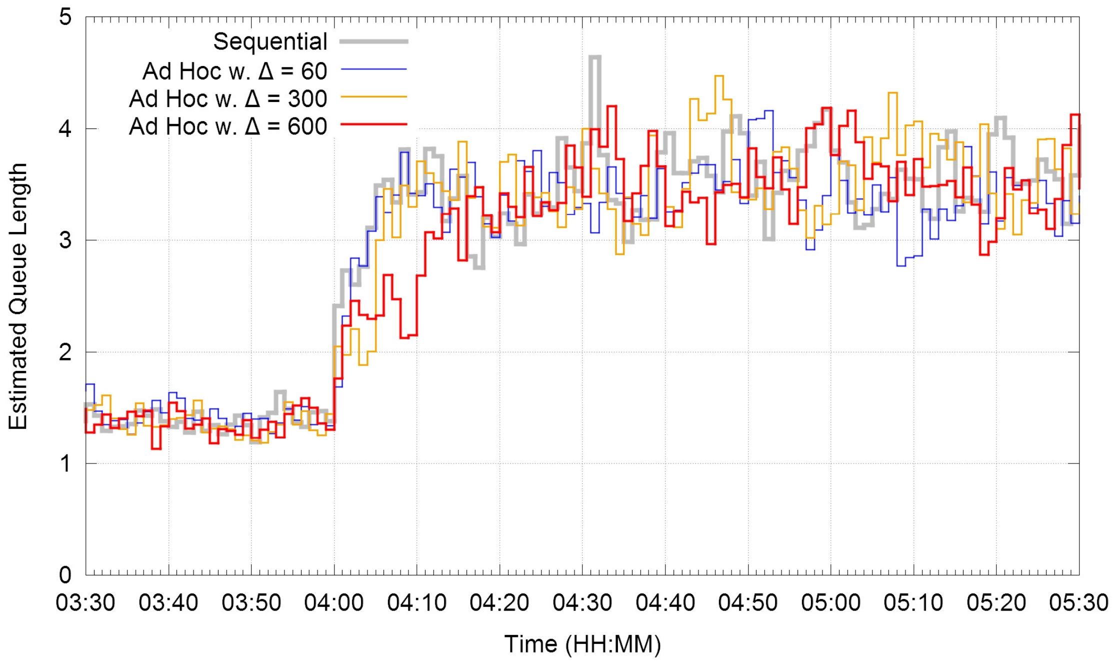


Figure 39: Estimated Mean Queue Lengths at Node 4 under Case 1

Figure 39 shows the estimated mean queue lengths of node 4 in the same simulation settings. Although the queue length is partly influenced by the arrivals on link 10, the discrepancy between the sequential runs and the ad hoc runs is noticeable, albeit less severe.

4.1.2 Case 2: Bidirectional Tandem Network

This case concerns the bidirectional tandem network with $p = 0.4$. Similar to the previous case, the external arrival rate of every node increases from $\lambda = 1/8$ to $1/6$ per second after 4 hours in simulation time. Before the transition, the steady-state traffic intensities range from 0.31 (nodes 1 and 7) to 0.57 (nodes 4 and 5); following the rate change they range between 0.41 (nodes 1 and 7) and 0.76 (nodes 4 and 5).

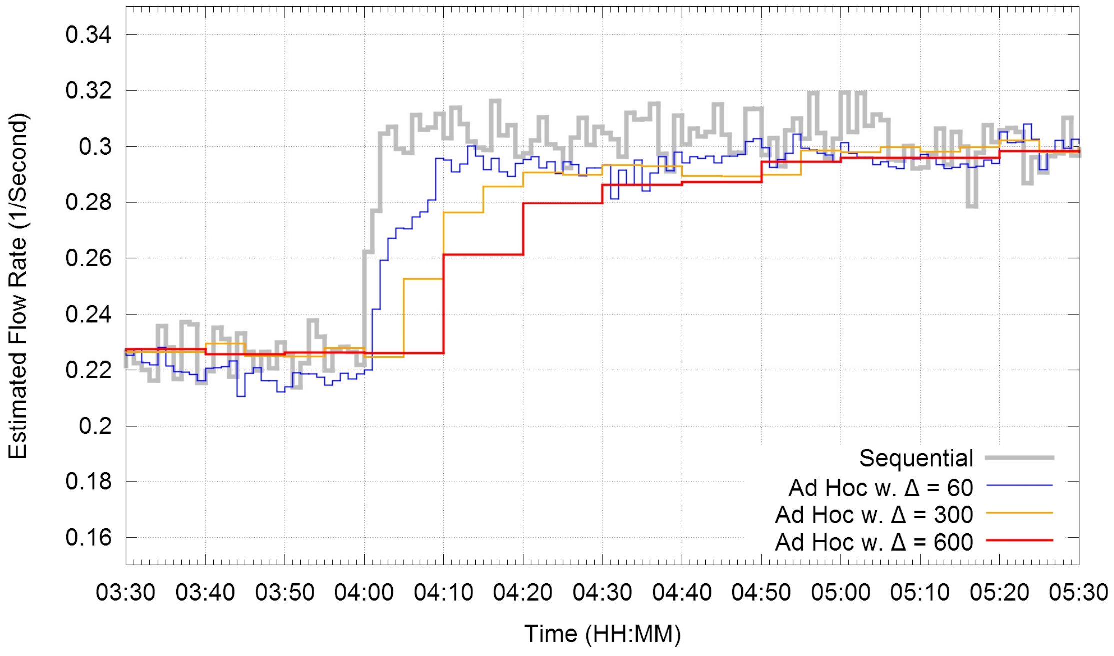


Figure 40: Estimated Arrival Rates across Link 10 under Case 2

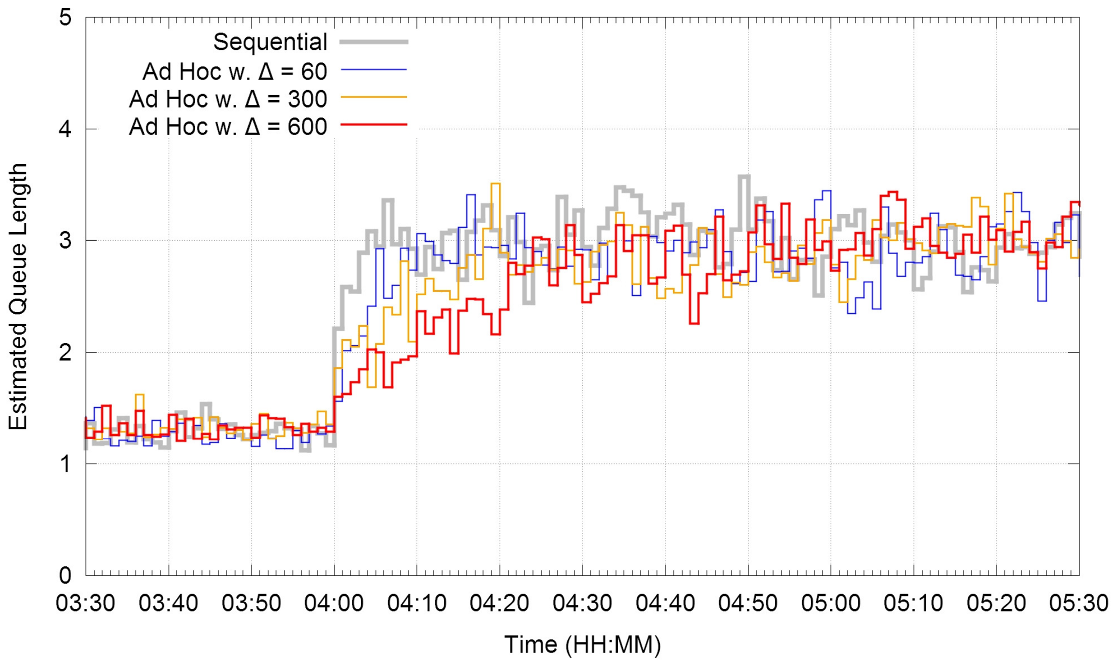


Figure 41: Estimated Mean Queue Lengths at Node 4 under Case 2

Figures 40 and 41 show the estimates of the arrival rates across link 10 and the mean queue lengths at node 4, respectively. Compared to those in Case 1 (Figures 38 and 39), the ad hoc runs in this case take longer to pick up the state change. For example, the ad hoc runs with $\Delta = 600$ do not fully reach the expected state until approximately one hour after the change has occurred. This prolonged delay results from the "mutual dependence" of the left and the right subnetworks. Specifically, the projected arrival rates of link 3 rely on those of link 10, and vice versa.

4.2 Iterative Ad Hoc Queueing Simulation Method

This section proposes a solution to the delayed-response problem by reassigning a new meaning to the values computed during the simulation execution. These values are considered as representations of the current system state, rather than predictions of the future as in the original ad hoc method. Details of the proposed solution are described in the following subsections followed by a discussion of a particular design aimed at avoiding potential issues such as livelocks. However, this design does not eliminate livelocks in certain incorrect simulation models as will be illustrated by an example.

The proposed iterative ad hoc queueing simulation method inherits most components from the original one, including the partitioning, the local simulation models, the information aggregation, and the rollback-based optimistic synchronization mechanism. These common parts are briefly summarized in the following corresponding subsections to provide a comprehensive view of the method without the focuses deviating from the new design.

4.2.1 Partitioning and Local Simulation Model

As in the original ad hoc method, a queueing network of interest is partitioned into subnetworks of different sizes and shapes with the possibility that these subnetworks may overlap with each other. Every LP models one subnetwork by adopting the discrete-event simulation technique for constructing its local simulation model. The simulation input is the state information of the incoming links to the modeled subnetwork, and the output is that of all modeled links. The specifics regarding the link state information will be revisited in Section 4.2.2.

4.2.2 Information Sharing

Similar to the original ad hoc method, the shared link states are represented in a high level abstraction instead of the exact time points of job arrivals/departures. Specifically, LPs exchange estimated flow rates every Δ seconds where an estimated flow rate of a link is computed by reversing the mean interarrival time over the last Δ seconds on that particular link. For example, let $t_1 \leq t_2 \leq \ldots$ denote arrival times and let $t_i \leq t_{i+1} \leq \ldots \leq t_{j-1}$ be those within the time interval of interest. Then, the estimated flow rate is the multiplicative inverse of the mean interarrival time, i.e., $\hat{r}$ in Equation (4.1). However, generating individual arrival times by reversing the above procedure is not straightforward because the information of the respective interarrival-time distribution is not maintained; nor is the correlation relationship among those arrivals. Here, it is assumed that the arrivals on one link form a Poisson process with rate equal to the corresponding estimated flow rate.

$$\hat{r} = 1 \div \left(\frac{1}{j-i} \sum_{k=i}^{j-1} \left(t_k - t_{k-1} \right) \right) = \frac{j-i}{t_{j-1} - t_{i-1}} \tag{4.1}$$

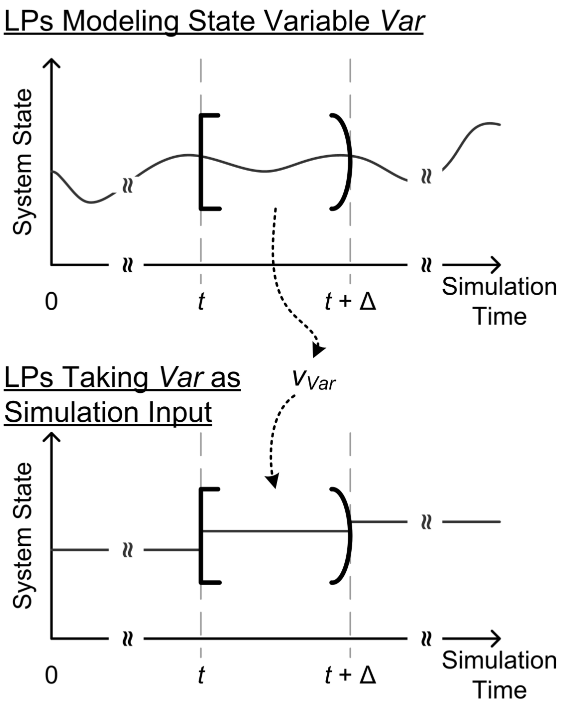


Figure 42: Information Sharing in Iterative Ad Hoc Queueing Simulation Method

Unlike the original ad hoc approach, the iterative ad hoc distributed simulation method aims for LPs sharing link information in a way that state transients are reflected to others momentarily. This is accomplished by imposing that an estimated flow rate over a given time period can only be used to reconstruct the link during that particular time interval. Specifically, consider a value v denoting the flow rate over $[t, t + \Delta)$ on some link l. The LPs that model link l as an incoming link must use v (along with the assumption about the corresponding arrival process) to generate arrivals within $[t, t + \Delta)$; Figure 42 illustrates such design.

This design guarantees that LPs will not progress simulations backwards since the value based on $[t, t + \Delta)$ from one LP affects, at earliest, the simulations of $[t, t + \Delta)$ carried out by other LPs. Or, in the terminology of distributed simulations, the “lookahead” value is zero. Zero lookahead commonly raises the concern of deadlocks, which can be eliminated by optimistic synchronization. However, in this case, livelocks are possible because LPs may fall into a loop within which they keep rolling back each other. Avoiding such livelocks requires careful design of the local simulation models, which will be discussed in Sections 4.2.6 and 4.2.7.

4.2.3 Information Aggregation

This ad hoc method retains the original information aggregation mechanism—each state variable (i.e., the estimated flow rate of one link over a certain time interval) is affiliated with a data model so that aggregating various estimates is equivalent to randomly generating a sample out of the model. Data models are constructed using the kernel-density-estimation (KDE) method with the Gaussian kernel.

4.2.4 Optimistic Synchronization and Rollbacks

The optimistic synchronization in this modified ad hoc method builds upon three ideas: 1) intuitive practices, 2) simple implementation/maintenance, and 3) statistical validity. While the last one is the fundamental idea to the process of determining the necessity of a rollback (referred to as the "rollback criterion"), the former two are pervasive throughout the design. For example, when a desired link flow rate in unattainable, the requesting LP uses the most current rate of the same link, rather than an arbitrary value, by assuming that the link state has not changed. Another example concerns the system state restoration for nonstationary Poisson processes; this will be revisited soon.

The rollback criterion involves a statistical test that evaluates a used value against the average of all shared, estimated rates. Specifically, the confidence interval of the point estimate of the mean rate, in Equation (4.2), serves as the acceptance range:

$$\bar{r} \pm t_{n-1,1-\alpha/2} \frac{S_r^2}{\sqrt{n}} \qquad (4.2)$$

In the equation, $\bar{r}$, S_r^2, and n are the sample mean, sample variance, and sample size, respectively. The significance level is α, which is set to 0.005. The critical value $t_{n-1,1-\alpha/2}$ is based on a Student's t distribution with $n - 1$ degrees of freedom. If the used value falls outside the range, it is rejected and a rollback is triggered. Compared to the original design in Section 3.2.5, this method introduces an additional parameter (i.e., the degrees of freedom) in order to adapt to different variations due to different number of estimates.

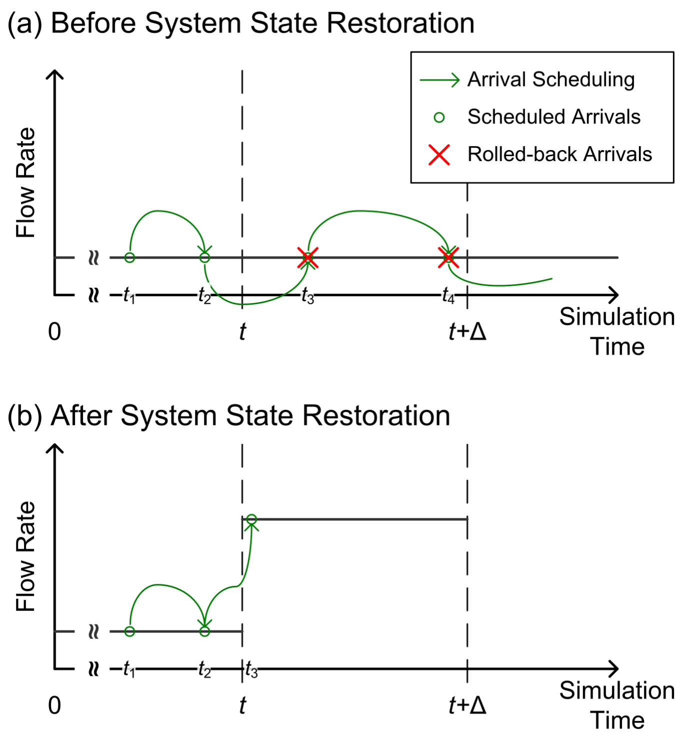


Figure 43: Arrival Scheduling in Rollback Handling

In response to rollbacks, LPs perform system state restoration. As input links involve nonstationary arrival processes, additional state information other than the flow rates is needed. In typical discrete-event-based queueing simulations, processing the current arrival event includes scheduling a new arrival event, the time stamp of which relies on some "future information" (or, in this design, the future flow rates). If any pertaining future rate is later proved incorrect, a rollback is triggered; Figure 43 depicts such a situation. In the figure, an LP is rolled back for using an underestimated flow rate for some link during the simulation time period $[t, t + \Delta)$. Since resetting the system state also removes all the arrivals beyond t, an initial arrival has to be generated on every input link. The generating process must consider the elapsed time from t_2 (when the latest arrival occurred) to t as part of the interarrival time. Note that the new arrival must come after t because a rollback targeting at $[t, t + \Delta)$ cannot affect the system state in prior to t.

The above issue is simplified in ad hoc queueing network simulations where the arrivals on input links are modeled as nonstationary Poisson processes. Two well-known methods for generating nonstationary Poisson arrivals are the thinning method [70] and the inversion method [71]. The thinning method, an acceptance-rejection algorithm, requires the upper bound on a flow rate function, which is generally unavailable. Furthermore, this method may be inefficient when the acceptance rate is low. On the other hand, the inversion method generates the arrivals times $\{t_i\}$ using a sequence of Poisson arrival times at rate 1 $\{t_i'\}$ and the expectation function of a rate function, as defined in Equation (4.3):

$$\Lambda(t) = \int_0^t \lambda(y)dy \tag{4.3}$$

The algorithm is shown in Figure 44. This method is adopted for it being practical and easy to implement by one additional variable for t_{i-1}' and an array data structure for Λ^{-1} (because the rate functions in ad hoc queueing network simulations are step functions).

```
01: $u \sim U(0, 1)$
02: $t_i' = t_{i-1}' - \ln(u)$
03: $t_i = \Lambda^{-1}(t_i')$
```

Figure 44: Inversion Method for Generating Nonstationary Poisson Arrivals

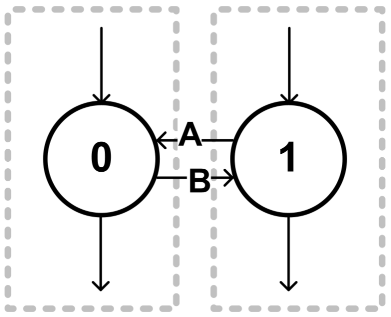


Figure 45: A 2-Node Bidirectional Tandem Open Queueing Network

4.2.5 Naming—“Iterative” Ad Hoc Queueing Simulation Method

The term “iterative” comes from the iterative methods in computational mathematics. These iterative methods solve problems that are formulated into the fixed-value problem $f(x) = x$ where f is a function. To find a solution of such a problem, the typical procedure of an iterative method starts with an arbitrary x_0, which is used in f to obtain $f(x_0)$; then, the value $f(x_0)$ is set to x_1. This procedure of $x_{n+1} = f(x_n)$ is repeated until the sequence $\{x_i\}$ converges. The definition of convergence varies; it can be that the difference between the last two numbers in the sequence are either zero or within a designated scale of error.

A similar phenomenon of iteration can be observed in the ad hoc queueing simulation method. Figure 45 depicts an example where the queueing network is partitioned into two parts, each containing one node. The LPs modeling node 0 generate the state information of link B and request that of link A; by contrast, those modeling node 1 use the information of link B to produce that of link A. The relationship between the desired information and the shared information can be captured by functions: F_0 for the LPs simulating node 0 and F_1 for node 1. Considering $A_{[t,\, t+\Delta)}$ as the state of link A with respect to the time period of $[t, t + \Delta)$ and similarly $B_{[t,\, t+\Delta)}$ for link B, the relations can be written down as Equations (4.4) and (4.5). Combining these two yields Equation (4.6), which has the form $f(x) = x$ where f is $F_1 \circ F_0$.

$$B_{[t,\, t+\Delta)} = F_0(A_{[t,\, t+\Delta)}) \tag{4.4}$$

$$A_{[t,\, t+\Delta)} = F_1(B_{[t,\, t+\Delta)}) \tag{4.5}$$

$$A_{[t,\, t+\Delta)} = F_1(F_0(A_{[t,\, t+\Delta)})) \tag{4.6}$$

4.2.6 Avoidance of Potential Livelocks

To prevent livelocks that result from the zero-lookahead nature in the ad hoc method, it is essential to comprehensively understand the physical system before building the local simulation models and designing those ad hoc components, especially the rollback criteria. A livelock situation arises when two or more LPs are involved in a loop in which their shared values keep invalidating each other's simulation inputs. That is, these LPs roll back each other successively so that, from the global view, the entire simulation execution does not show forward progress. Although the zero-lookahead feature allows LPs to "bring back" others to the same simulation time point, this feature can not be blamed for livelocks. Instead, the causes of such livelocks include unrealistic rollback criteria and incorrect simulation models. This subsection will focus on the former one (and the latter in Section 4.2.7).

A feasible, legitimate rollback criterion must take into account the characteristics of the corresponding state variable, such as randomness. For example, for a state variable with possible values ranging across continuous space (i.e., a continuous stochastic variable), its affiliated rollback criterion needs to be flexible with regard to evaluating the "correctness" of a value. A value should be considered correct if it is within a certain range. This idea applies to discrete stochastic variables as well. However, the consequence would be more severe for continuous random variables because the probability of a continuous random variable being equal to any arbitrary value is zero. This implies that, given an LP that has used a value *v* for some input link, another LP that models the link is highly improbable to generate *v* as a state measure for the link. Hence, the LP using *v* is rolled back. The following example illustrates such a situation based on the queueing network in Figure 45.

The example concerns a simulation of the queueing network in Figure 45 with LP_0 modeling the left part (i.e., node 0 and link B) and LP_1 in charge of the right one. LP_0 requires the state information of link A as input to its local simulation model, and this information is shared by LP_1. Similarly, LP_1 relies on LP_0 for link B. The link states are measured by the flow rates estimated over Δ seconds, and the LPs must use the exact value their corresponding LP generates. Considering the time period $[t, t + \Delta)$, the process used by the two LPs to reach an agreement on the rates is depicted in Figure 46; clearly they fail and fall into a livelock situation. The detailed process is as follows:

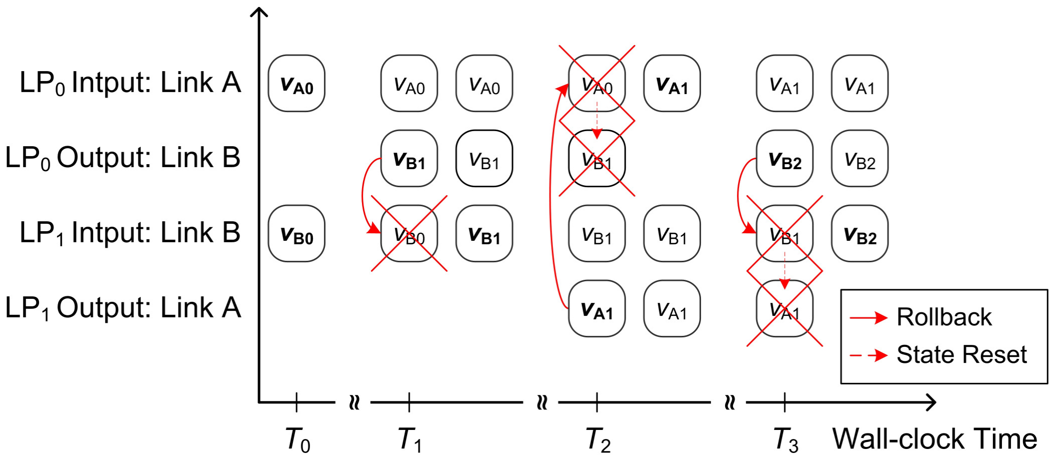


Figure 46: A Livelock Example of Modeling the 2-Node Bidirectional Tandem Queueing Network in Figure 45

1. At wall-clock time T_0, both LP_0 and LP_1 have not generated flow rates for their corresponding links with respect to $[t, t + \Delta)$. As a consequence, they utilize arbitrary values: LP_0 applies v_{A0} for link A and LP_1 uses v_{B0} for link B.
2. At time T_1, LP_0 produces the estimated flow rate of link B, v_{B1}, which rolls back LP_1.
3. Then at T_2, LP_1 observes the flow rate of link A being v_{A1} after using v_{B1} for link B. As a consequence, LP_0 is rolled back and its shared state information about link B, v_{B1}, is revoked.
4. At time T_3, similar situation occurs: LP_0 derives a new value for link B, v_{B2}, due to the usage of v_{A1}. Rolling back LP_1 results in the simulation ending up in a situation resembling that at T_1.

This loop of rollbacks is anticipated to continue because the flow-rate estimates are continuous stochastic variables; it is highly impossible that $v_{Ai} = v_{Ai+1}$ (nor $v_{Bi} = v_{Bi+1}$) for any integer $i \geq 0$.

4.2.7 Livelocks from Incorrect Simulation Models

An incorrect local simulation model may cause livelocks as well. This situation is analogous to the classical livelock problem where the application/model itself is not appropriately defined. This subsection presents such an example where an ad hoc queueing network simulation involves inappropriate network partitioning and incorrect assumptions in designing the local simulation models. This example signifies that it is vital to understand a physical system (e.g., the interactions among the components in the system) before applying the ad hoc approach (or any other simulation method) to model it.

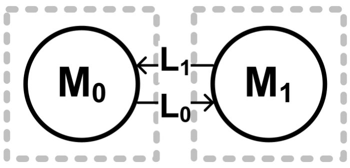


Figure 47: A 2-Node Bidirectional Tandem Closed Queueing Network

The example concerns a simple procedure with two machines M_0 and M_1 cooperating on 1) creating jobs, 2) processing them, and then 3) checking their final status; the first and third tasks are executed by M_0 while M_1 is in charge of the second one. The jobs are transmitted between the machines via links L_0 and L_1 without incurring any latency, as depicted in Figure 47. The detailed operations of M_0 and M_1 are as follows. M_0 conducts three types of operations, all uninterruptable with deterministic operating duration: resting takes 1 second; creating a job, 4 seconds; and checking a job's status, 4 seconds. The checking operation has higher priority than the creating one. That is, if a job has returned from M_1, M_0 checks its status, updates corresponding records, and deletes the job; otherwise, M_0 creates a new job. However, M_0 must rest right after both the creating and checking operations. On the other hand, M_1 is involved in a more straightforward scenario: it idles until getting a job from M_0 when it would immediately process the job with 0.5 second and then send the job back to M_0. In sum, Figure 48 plots the operations of and the interactions between M_0 and M_1 given no job in both machines at time 0.

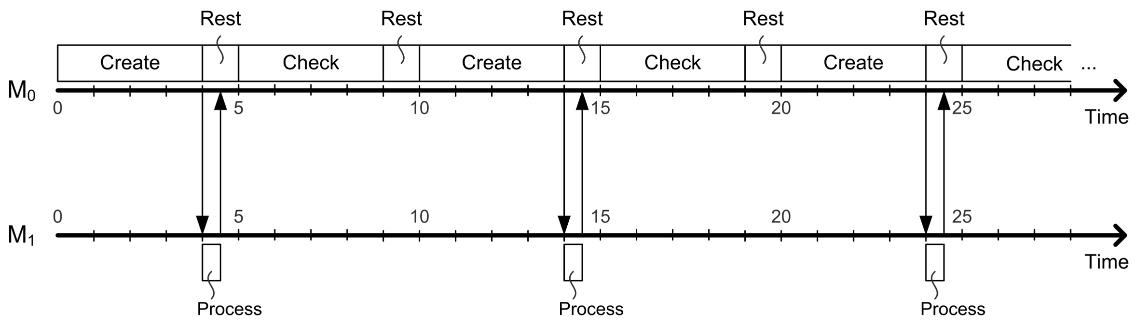


Figure 48: Operations of and Interactions between Machines M_0 and M_1

Table 2. A Livelock Example of Incorrectly Modeling the Behaviors in Figure 48

Iteration	LPs modeling M_0		LPs modeling M_1	
	Input L_1 (M_1 to M_0)	**Output L_0 (M_0 to M_1)**	**Input L_0 (M_0 to M_1)**	**Output L_1 (M_1 to M_0)**
1	*0*		*0*	
	0	*5*	0	*0*
2	0	5	*5*	
	0	5	5	*5*
3	*5*		5	5
	5	*0*	5	5
4	5	0	*0*	
	5	0	0	*0*
5	*0*		0	0
	0	*5*	0	0

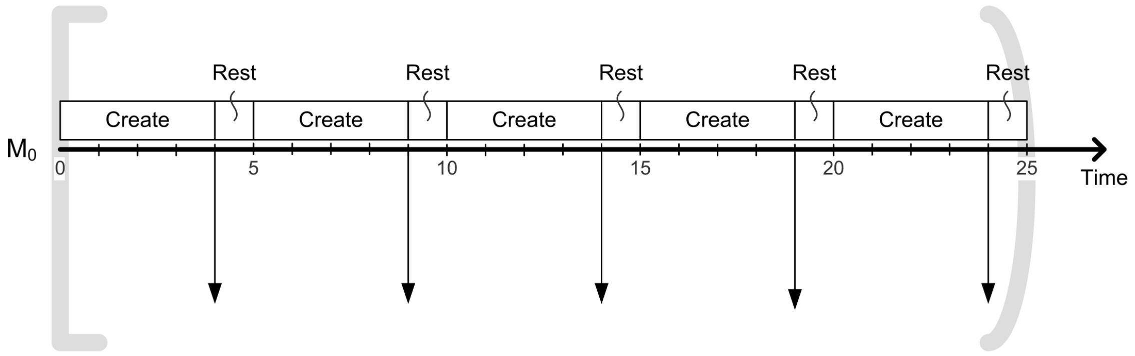
A failed usage of the ad hoc method on modeling M_0 and M_1 is as follows. The partitioning is as shown by the grey boxes in Figure 47 where one set of LPs modeling M_0 and the other set for M_1. In this case, the procedure does not involve stochastic components and hence the number of the LPs with the same modeling area is not an important factor. The two sets of LPs exchange the numbers of jobs sent through link L_0 and L_1 every 25 seconds in simulation time; the LPs use the value 0 if the desired information is unavailable upon request. The LPs would roll back when the corresponding LPs generate values that are different from what they have used. As to generating *n* arrivals within the respective 25 seconds, the arrivals are evenly distributed. For example, considering the simulation time interval [0, 25) with *n* equal to 5, the arrivals occur at simulation time points 0, 5, 10, 15, and 20.

Table 2 lists the first few iterations within a livelock where the two sets of LPs attempt to converge to the numbers of jobs on L_0 and L_1 during some time interval with length 25 seconds. For the illustration purpose, let the time interval be [0, 25). The detailed iterations are as follows:

1. **Iteration 1.** At simulation time 0, both the numbers of jobs on L_0 and L_1 are assumed to be 0. For machine M_0, this implies that it does not need to perform any checking operation and hence would create 5 jobs, as depicted in Figure 49(a). On the other hand, machine M_1 idles during the entire 25 seconds. After simulating this 25-second interval, the LPs modeling M_0 delivers a number 5 since 5 jobs were created and sent through L_0. This number invalidates the value 0 used by the LPs in charge of M_1 (see the shaded numbers in Table 2).
2. **Iteration 2.** The LPs modeling M_1 restart the simulation from time 0 with the input L_0 being 5. Five jobs arrive at M_1 for processing, and are returned through L_1 (Figure 50). As a consequence, the number 5 goes public to stand for the state of L_1.
3. **Iteration 3.** The LPs modeling M_0 are asked to use 5 as the input for link L_1. Figure 49(b) illustrates the situation, in which M_0 spends all applicable time on the checking operations and does not create any new job to put onto L_0.
4. **Iteration 4.** The usage of 5 for the input L_0 is rolled back. Instead, the value 0 should be adopted, which then leads to M_1 idling during [0, 25).
5. **Iteration 5.** This iteration is exactly the same as Iteration 1, indicating that the simulation runs into a loop.

The livelock issue in the above example is the result of an incorrect simulation design by ignoring the relationship between the two links. In other words, such partitioning mechanism and the location simulation models are erroneous.

(a) State of M_0 during [0, 25) with Input Value 0



(b) State of M_0 during [0, 25) with Input Value 5

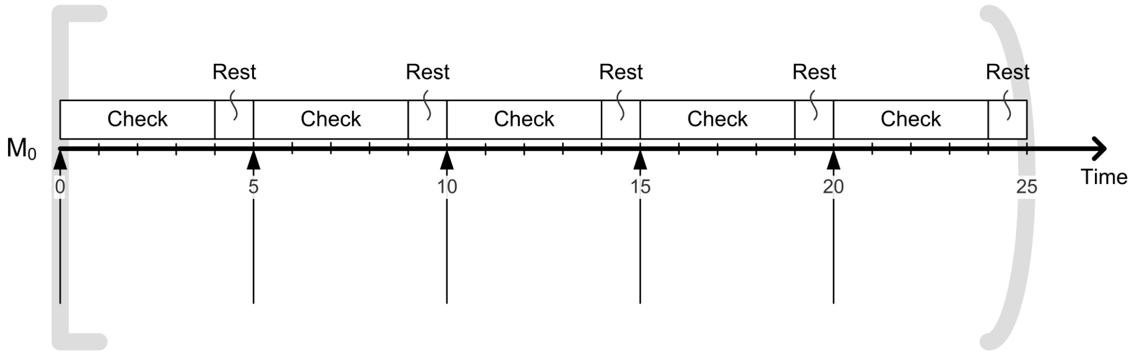


Figure 49: State of Machine M_0 during [0, 25)

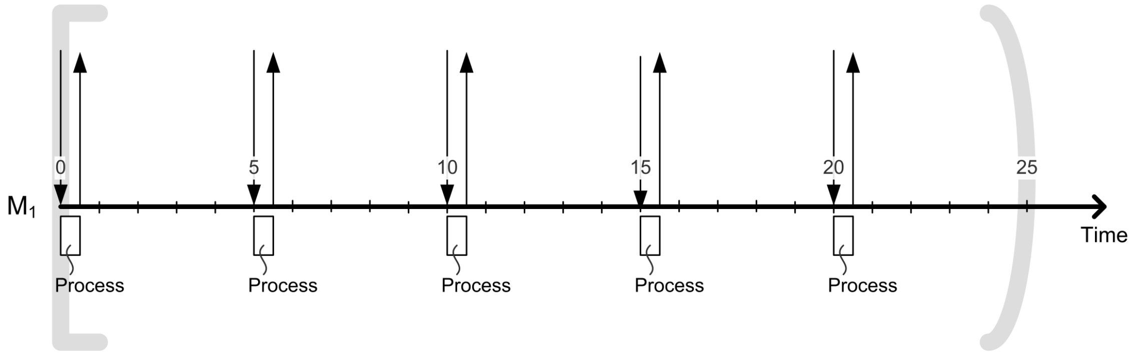


Figure 50: State of Machine M_1 during [0, 25) with Input Value 5

4.3 Experiments and Results

This section evaluates the iterative ad hoc queueing simulation method with six experiments, which involve two types of open queueing networks under various configurations. The experiments will demonstrate that the method is effective in terms of identifying system transients. The metrics for evaluation include the flow rates across input links and the mean queue lengths at nodes. The results are compared against the sequential counterparts where plots will be delivered to show the differences between the averaged numbers (similar to those in Sections 4.1.1 and 4.1.2). Moreover, Welch's *t* test is adopted for a more comprehensive assessment.

The following ad hoc queueing network simulations adopt the design of the ad hoc components described in Section 4.2, and the experiment settings are similar to those in Sections 4.1.1 and 4.1.2. An additional note regarding the number of IID replicate runs is that, for a node covered by n LPs in one ad hoc run, the number of the required sequential-counterpart replications is $n \times m$ where m is the number of ad hoc runs; in particular, $m = 10$ for all the experiments in this section. Moreover, 10 LPs are deployed to model each portion of a queueing network. Since one node may reside in one or more (say k) portions, n can be formulated as $10 \times k$. Examples of such scenarios with $k > 1$ are the experiments in Section 4.3.3 where $k = 2$ for some nodes.

4.3.1 Welch's *t* Test

The following experiments adopt Welch's *t* test to statistically access the (null) hypothesis that the mean values of two samples are equal. This test modifies the well-known Student's *t*-test to consider that two samples may have unequal variances. Its statistic t is defined by the formula in Equation (4.7) and the degrees of freedom ν is estimated by Equation (4.8) where $\overline{X}_i$, S_i^2, and N_i are the i-th sample mean, sample variance, and sample size, respectively. Given a significance level α (e.g., 0.01), the hypothesis is rejected if the derived p-value is below α; the p-value for a two-tailed test based on Student's *t*-distribution is 1 − ($F_\nu(|t|) - F_\nu(-|t|)$) where F_ν denotes the respective distribution function.

$$t = \left(\overline{X}_1 - \overline{X}_2\right) \Big/ \sqrt{\frac{S_1^2}{N_1} + \frac{S_2^2}{N_2}} \tag{4.7}$$

$$\nu = \left(\frac{S_1^2}{N_1} + \frac{S_2^2}{N_2}\right)^2 \Big/ \left(\frac{S_1^4}{N_1^2(N_1 - 1)} + \frac{S_2^4}{N_2^2(N_2 - 1)}\right) \tag{4.8}$$

The two samples in the subsequent hypothesis tests are 1) the data from the ad hoc runs and 2) those from the sequential counterparts. Regardless of the metric, the size of the ad hoc sample (i.e., N_1) is always 10, which is the same as the number of ad hoc replication runs. In other words, the results produced by the LPs in one run are averaged into one value to stand for that particular run. On the other hand, the size of the sequential sample (i.e., N_2) is 100 × *k*, that is, the number of the required sequential-counterpart replications.

4.3.2 Experiments with 8-Node Tandem Networks

This subsection focuses on three 8-node tandem networks, all with similar network topologies as those in Section 4.1. The first two experiments (Sections 4.3.2.1 and 4.3.2.2) revisit those in Sections 4.1.1 and 4.1.2 but use the new iterative ad hoc method. The third experiment loads the previous tandem network with heavy traffic to show how the method is capable of capturing system dynamics when the nodes are under high traffic intensities.

4.3.2.1 Experiment 1: Partially Bidirectional Tandem Network

This experiment concerns the network in Figure 37(a) with all the configurations from Section 4.1.1. The simulation results based on the iterative ad hoc method are shown in Figures 51 and 52. Regardless of the Δ value, the ad hoc simulations perform well on capturing the estimated flow rates and mean queue lengths. Although the small Δ (e.g., Δ =

60) may reveal choppiness in estimates, this is expected due to the randomness in the simulation models. On the other hand, it is anticipated that large Δ may weaken the real-time response as the changes are averaged out across the entire update periods. However such issue is insignificant in this experiment.

Figures 53 and 54 plot the *p*-values from Welch's *t* tests on the same two measurements of interest. These results reaffirm the good performance of the iterative ad hoc method because the vast majority of the *p*-values are above the critical level $\alpha = 0.01$. The three exceptions for the mean queue length at node 4 at simulation time 3:54, 4:16, and 4:32 (see Figure 54) are rather insignificant for two reasons: 1) the rejected hypothesis associates to 3:54, which is in prior to the arrival rate increase at 4:00; and 2) the failed tests are sparse across the entire simulation.

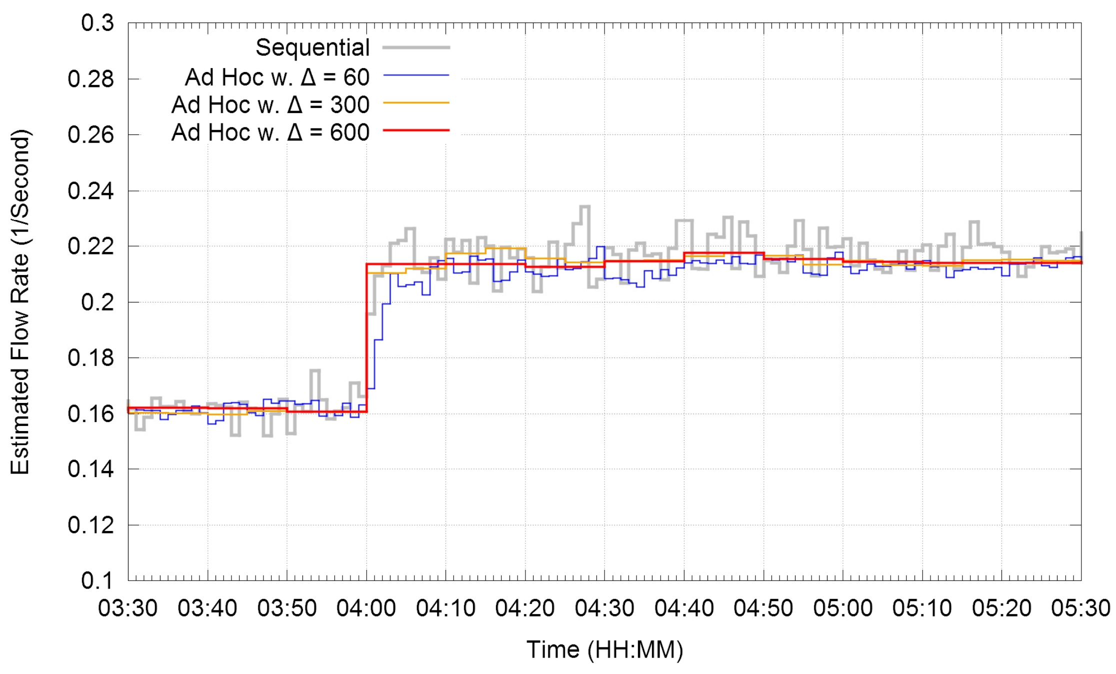


Figure 51: Estimated Arrival Rates across Link 10 in Experiment 1

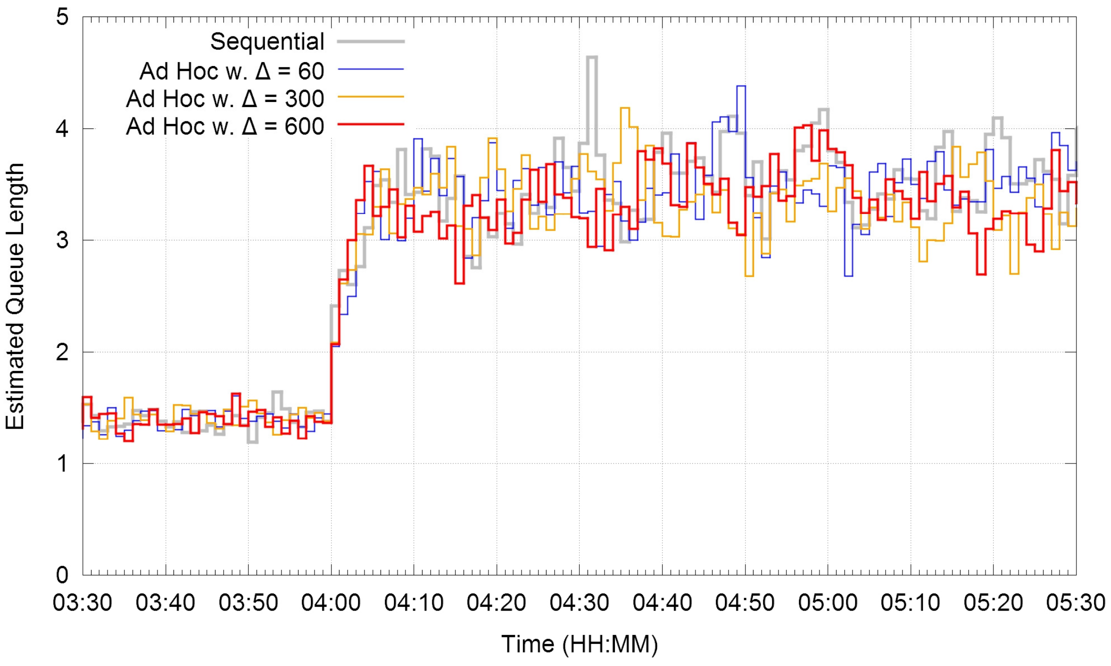


Figure 52: Estimated Mean Queue Lengths at Node 4 in Experiment 1

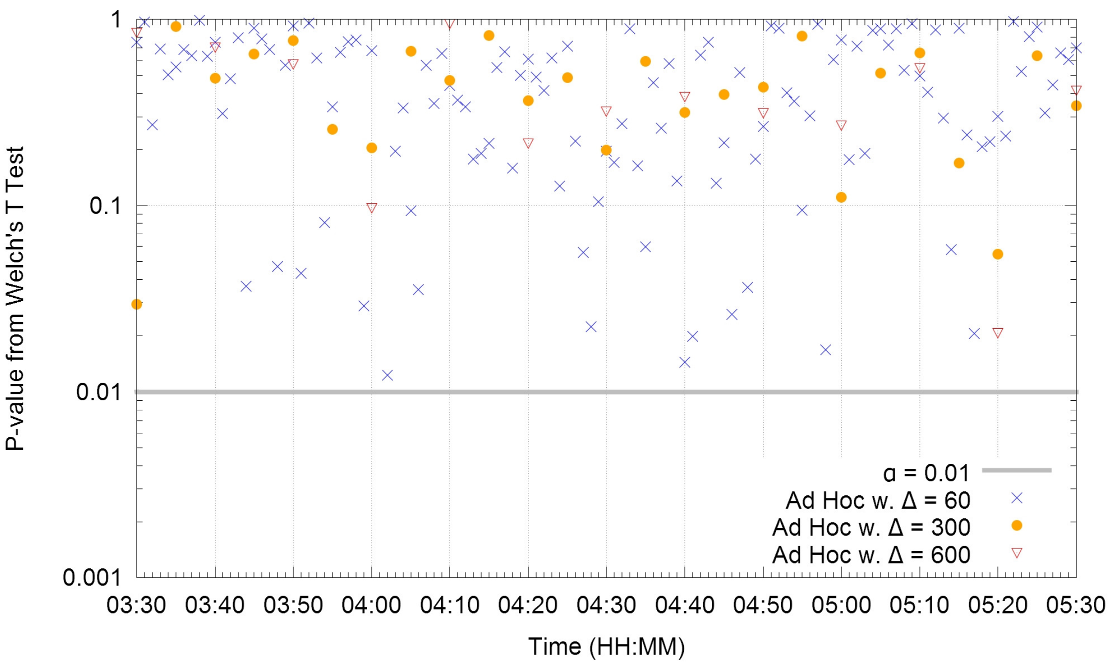


Figure 53: *P*-Values from Welch's *t* Tests on Estimated Arrival Rates across Link 10 in Experiment 1

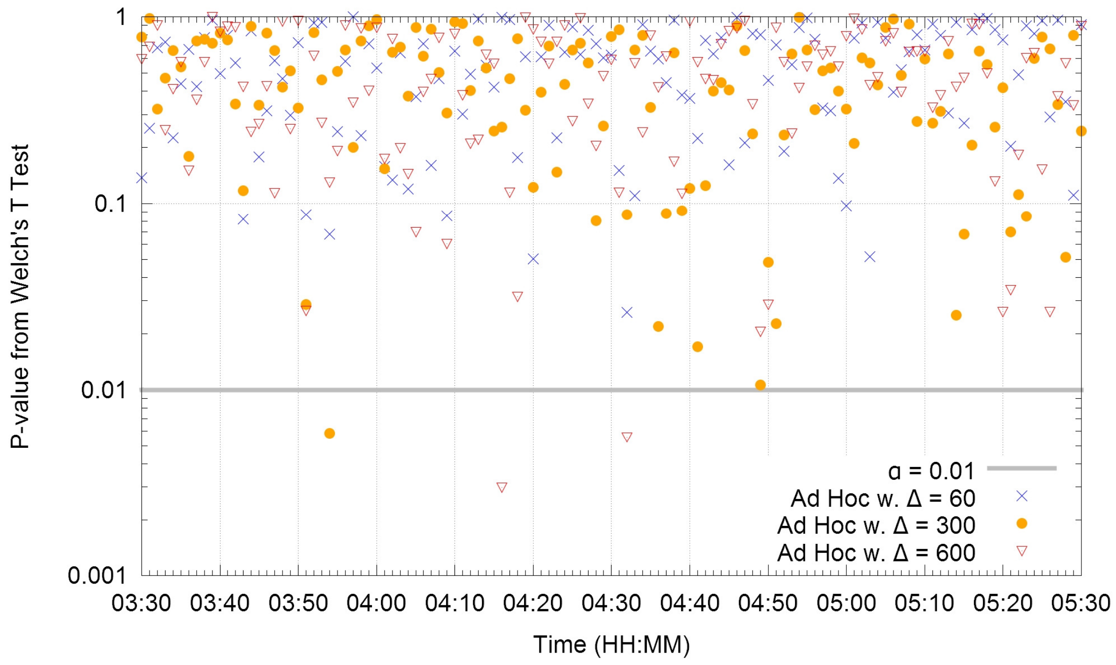


Figure 54: *P*-Values from Welch's *t* Tests on Estimated Mean Queue Lengths at Node 4 in Experiment 1

4.3.2.2 Experiment 2: Bidirectional Tandem Network with Moderate Traffic

This experiment extends that in Section 4.1.2 by replacing the ad hoc queueing simulation method with the new one. The network of interest is in Figure 37(b) and the results are in Figures 55 and 56. Here (and afterwards), the flow-rate metric is skipped as it leads to the same conclusion as the queue-length metric—the iterative ad hoc method performs well. Nevertheless, again, sporadic hypothesis rejections occur (see Figure 56) but they are not major, as those in the previous experiment.

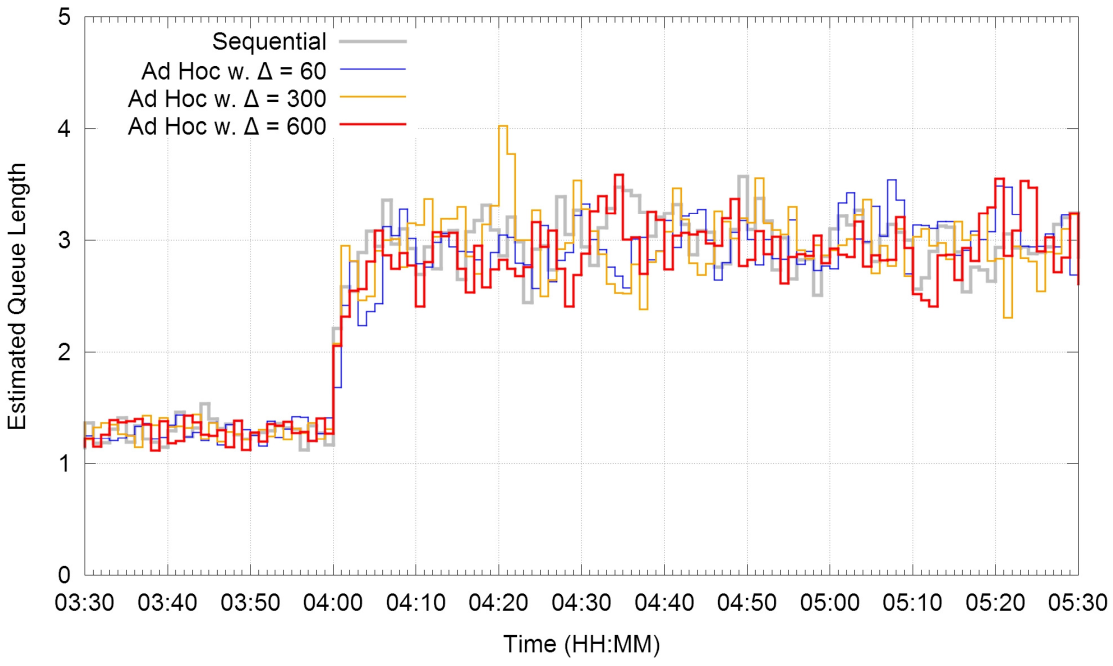


Figure 55: Estimated Mean Queue Lengths at Node 4 in Experiment 2

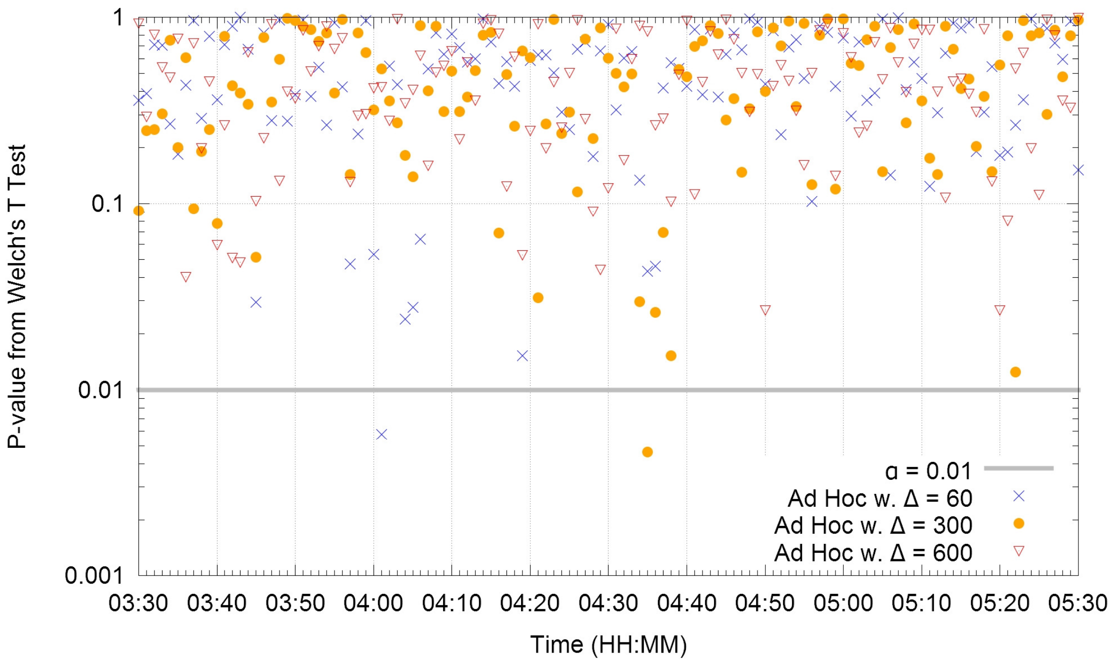


Figure 56: *P*-Values from Welch's *t* Tests on Estimated Mean Queue Lengths at Node 4 in Experiment 2

Experiment 3: Bidirectional Tandem Network with Heavy Traffic

This experiment intends to show that the iterative ad hoc method is capable of promptly responding to sudden, massive state changes. The network of interest is depicted in Figure 37(b) and the system transients are introduced by increasing the external arrivals to each node as depicted in Figure 57. This makes node 3 saturated—an extremely busy server and its queue building up. Node 4 is also saturated with its steady-state traffic intensity reaching approximately 0.97. Since the queues grow rapidly, the system transient period lasts for only 30 minutes. This allows the ad hoc method to demonstrate that it can also capture state changes in the opposite direction (i.e., decrease in flow rates).

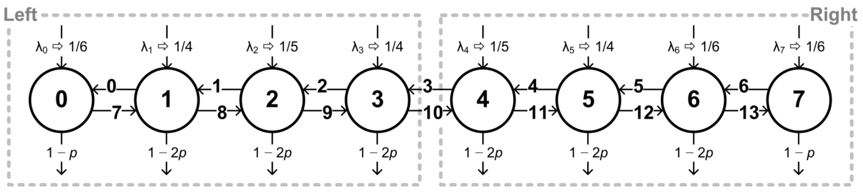


Figure 57: An 8-Node Bidirectional Tandem Network with Heavy Traffic

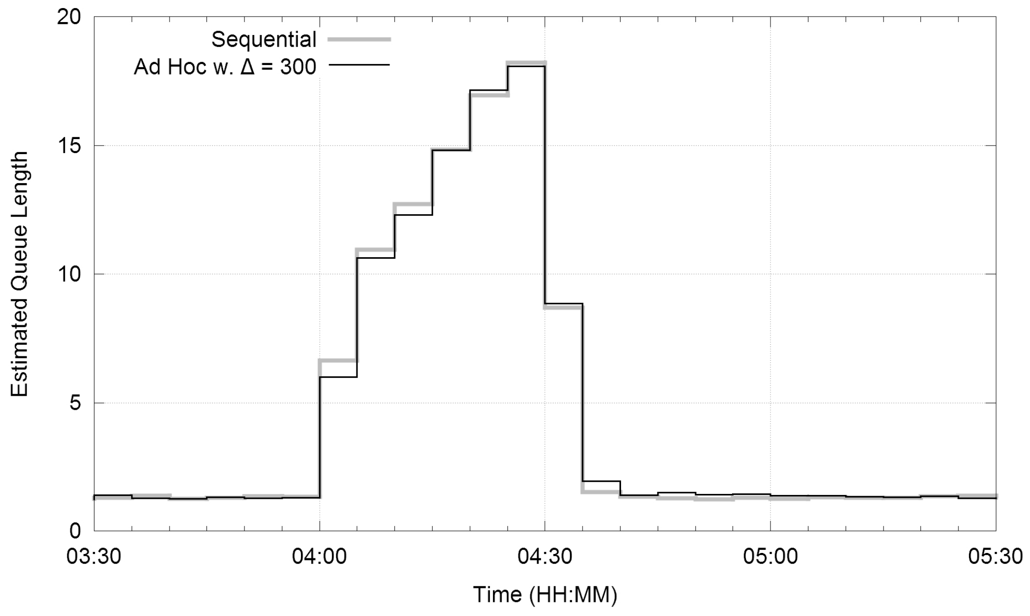


Figure 58: Estimated Mean Queue Lengths at Node 4 in Experiment 3

The simulation results affirm that the ad hoc method is competent in this heavy-traffic scenario; see Figure 58 for the estimated mean queue length of node 4. For clarity, Δ is set to a moderate value (Δ = 300), leaving off the jagged results from small Δ. Furthermore, the respective *t* tests indicate insignificant differences between the ad hoc and sequential simulations (the figure is skipped for space conservation).

4.3.3 Experiments with an 8 × 8 Grid Network

The three experiments in this subsection intend to show that the iterative ad hoc method can handle the state transients in the networks that involve complex, cyclic unit routings. The 8 × 8 queueing network under regular partitioning (i.e., the one in Section 3.3.1, as shown in Figure 59) is adopted. Moreover, the experiments vary in the service processes as well as the respective service-time coefficients of variation (CV), which range from 0.5 to 2.

The modeled scenario is as follows. Throughout the entire simulation, each node in the network is associated with an external Poisson arrival process, starting with the rate equal to 1 / 6 per second. Then, after 4 hours in simulation time, the external arrival processes of nodes 16 and 24 both increase their rates to 1 / 2 per second (see Figure 59); this change lasts for 30 minutes and goes back to the starting condition afterwards. During the 30-minute period, the steady-state traffic intensities of nodes 16 and 24 are anticipated to go up to 0.98 and 0.99, respectively; these two nodes are then the busiest among all.

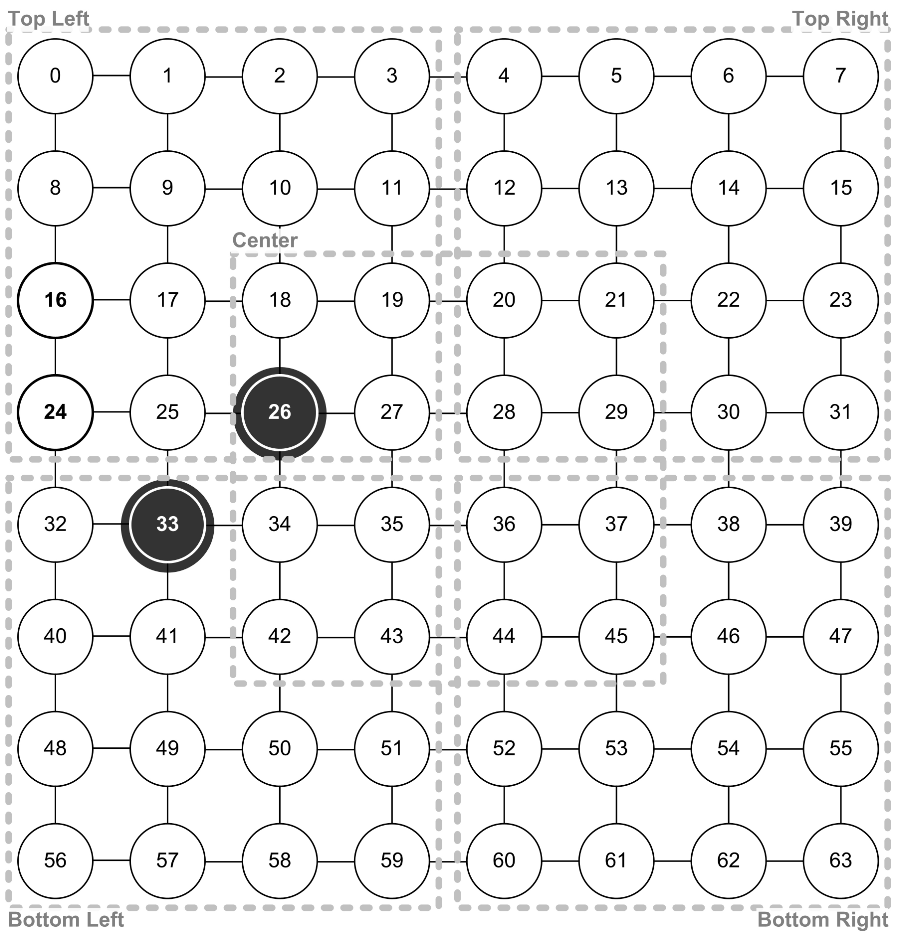


Figure 59: A Regularly Partitioned 8 × 8 Grid Network with Dynamic Flow Rates

The metrics for evaluation are the mean queue lengths of nodes 26 and 33. Node 26 is covered by 20 LPs, 10 of which also model nodes 16 and 24 (where the system dynamics are introduced) while the remaining 10 LPs require the information shared by the former 10 LPs. On the other hand, node 33 resides outside the portion where the state changes initially take off; that is, the 10 LPs modeling node 33 entirely rely on other LPs with regards to the system transients.

4.3.3.1 Experiment 4: Exponential Service Times

All the service times in this experiment are IID and follow the exponential distribution with the mean equal to 1 second. The CV is 1 since the variance is 1 second squared. After the system transients occur at 4:00, the steady-state mean queue length of node 26 is expected to grow from 3.26 to 4.34 and that of node 33, from 2.29 to 3.13.

Figure 60 illustrates the experiment results: the estimated mean queue lengths averaged over 300-second periods and the *p*-values from the respective Welch's *t* tests. These results show that the iterative ad hoc method is competitive to the corresponding sequential runs under this complex 8 × 8 network with the moderate service-time CV.

4.3.3.2 Experiment 5: Gamma Service Times with Low Variation

This experiment employs the gamma service times with the shape and scale parameters equal to 4 and 0.25, respectively; the resulting CV is 0.5 (since the mean is 1 and the variance is 0.25). The mean queue lengths are expected to be shorter than those in the last experiment due to this smaller service-time CV. This anticipation is confirmed by the results in Figure 61, which also support the claim that the ad hoc method is indistinguishable from the sequential simulation. Unlike the results in Section 4.3.3.1, an extremely rare number of hypothesis rejections hit the claim. However, these incidents are minor unless further evident is presented to against it.

4.3.3.3 Experiment 6: Gamma Service Times with High Variation

As opposed to the experiment in Section 4.3.3.2, here the service times follow the gamma distribution with the shape and scale parameters equal to 0.25 and 4, respectively. Under this configuration, the mean is 1 second and the variance is 4 second squared. That is, the service-time CV is 2, which results in longer mean queue lengths compared to those in Experiment 4 (in Section 4.3.3.1). The experiment results are depicted in Figure 62, and again these positive results show the potential effectiveness of the iterative ad hoc method.

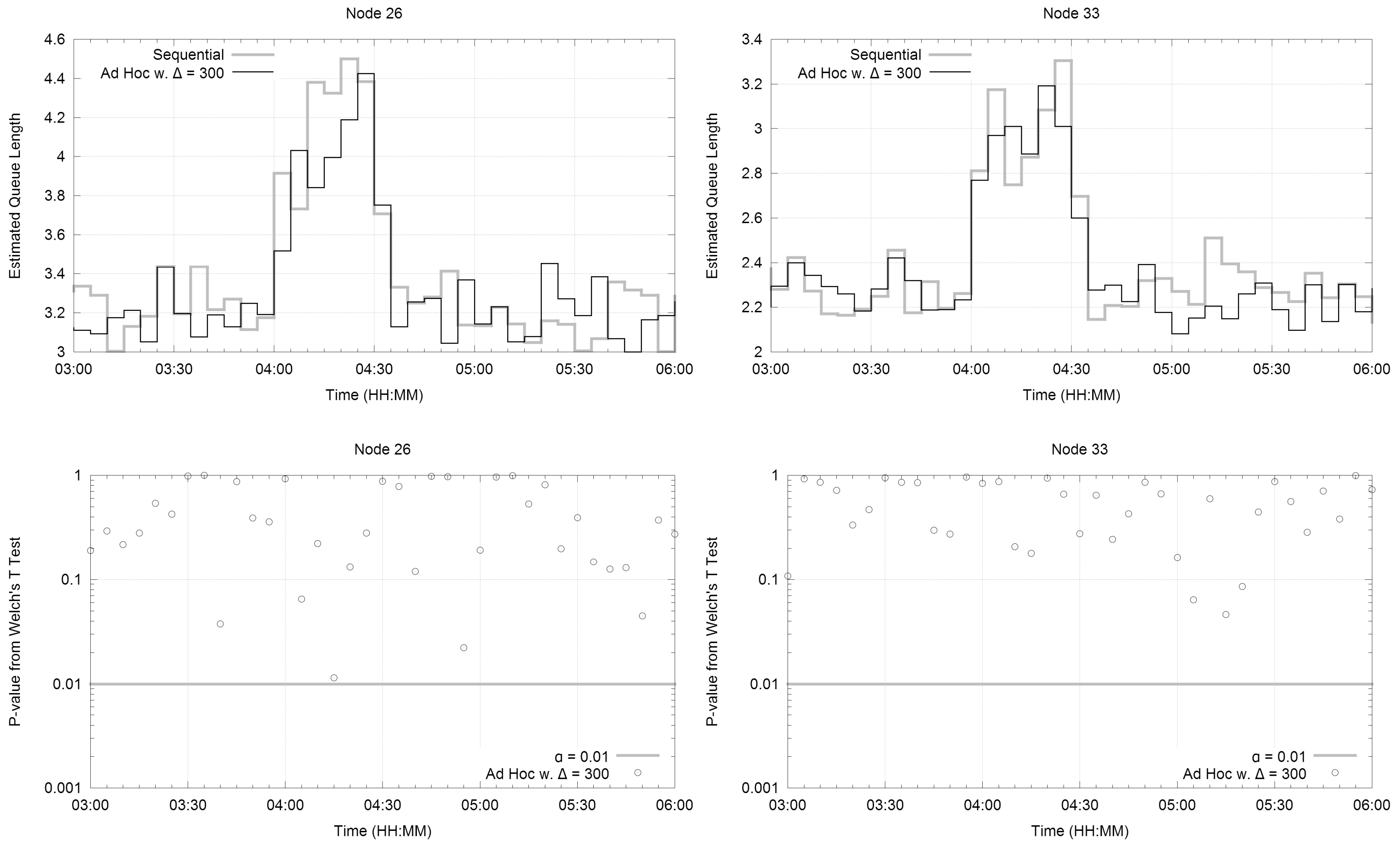


Figure 60: Estimated Mean Queue Lengths and *p*-Values from Welch's *t* Tests in Experiment 4

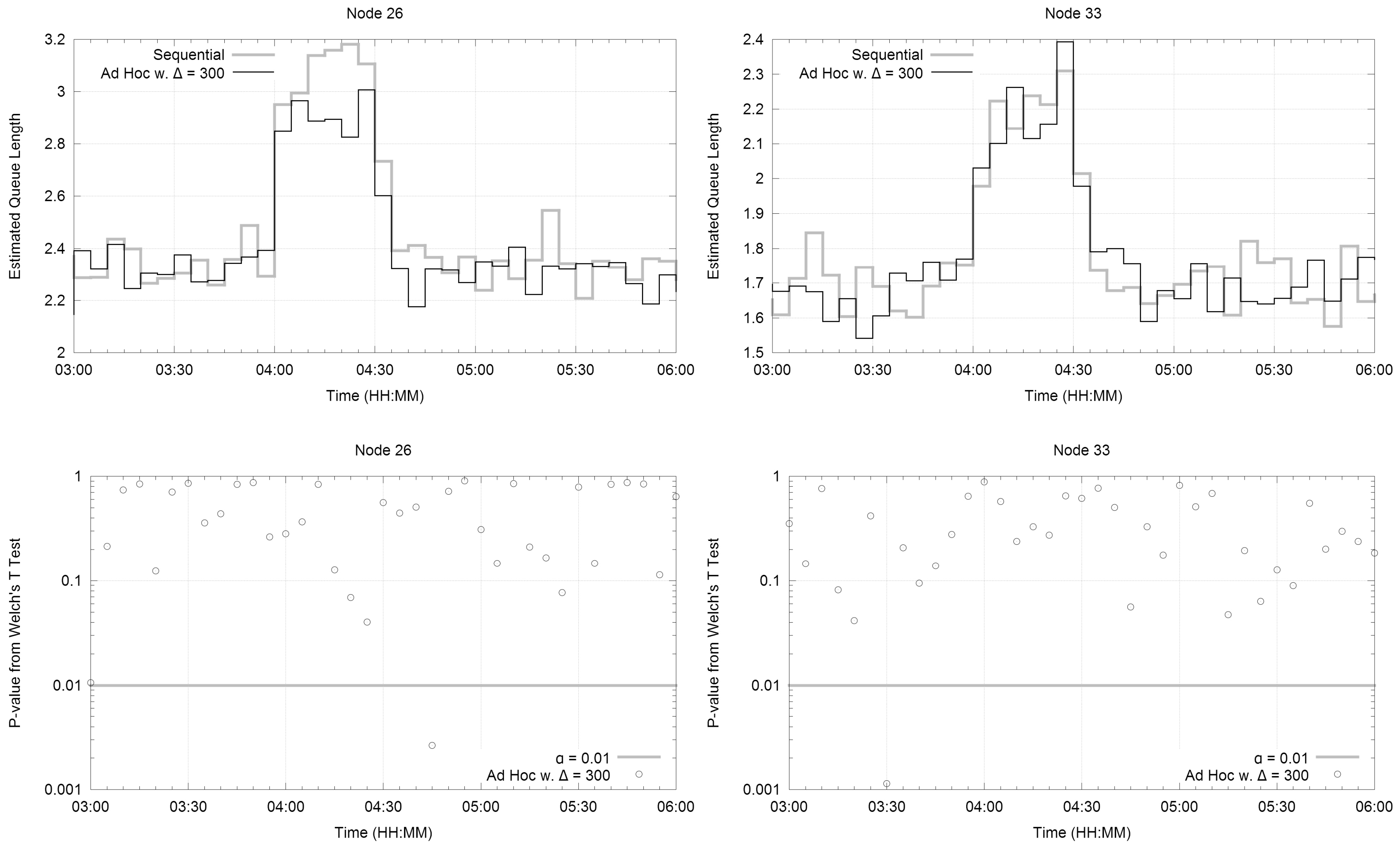


Figure 61: Estimated Mean Queue Lengths and *p*-Values from Welch's *t* Tests in Experiment 5

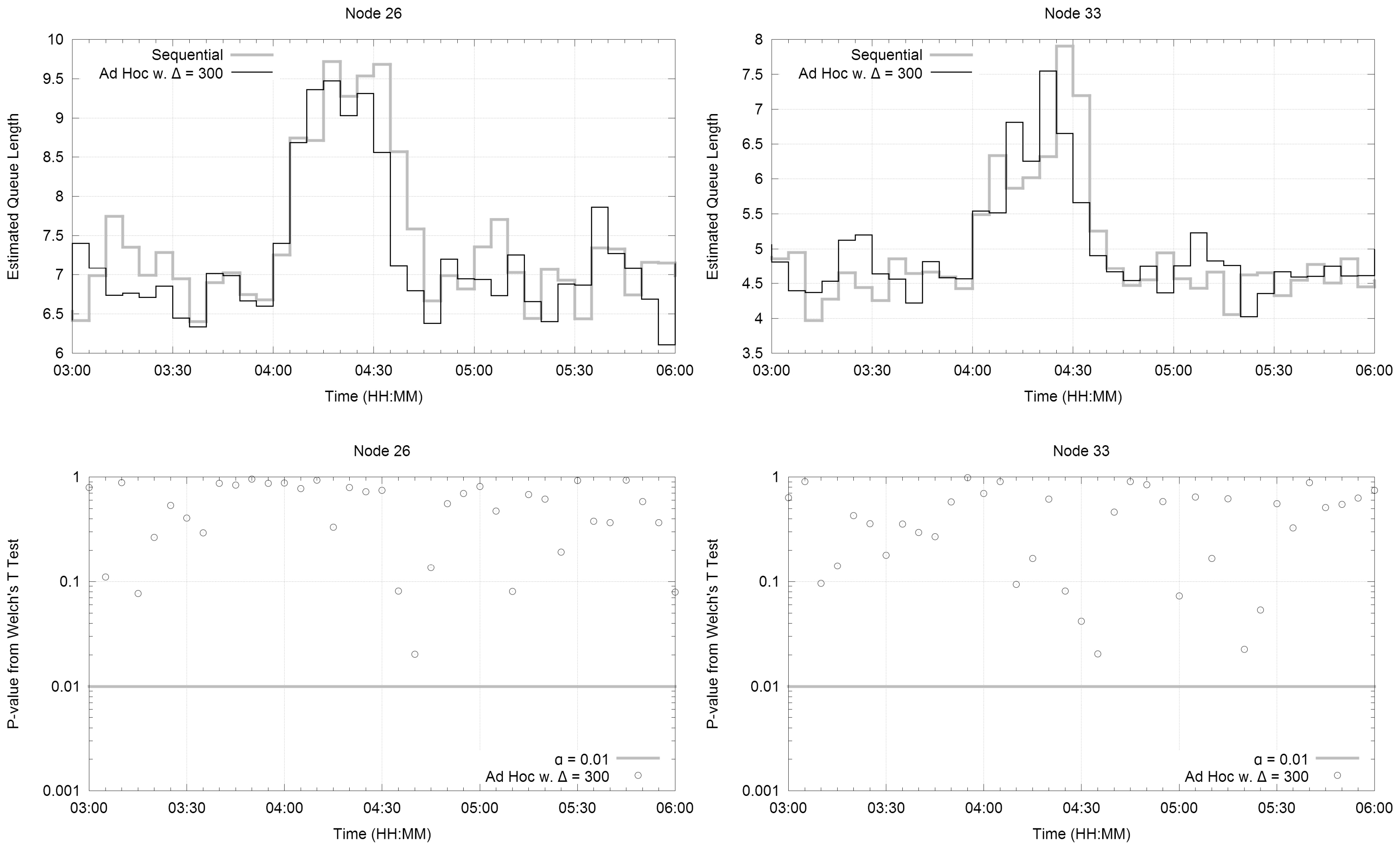


Figure 62: Estimated Mean Queue Lengths and *p*-Values from Welch's *t* Tests in Experiment 6

4.4 Conclusions

By extending the study of the accuracy of steady-state metrics in Chapter 3, this chapter demonstrated the competitiveness of ad hoc distributed simulation on modeling short-term system dynamics in opening queueing networks. First, two experiments with flow rates increasing during simulation executions were conducted to argue that the original ad hoc method in the previous chapter fails in instant reflection of system transients. The extent of those delayed responses relates to the length of the update period Δ. The failures are due to the fact that the state updates shared by individual LPs are regarded as predictions of future system states, rather than reflections of the current state.

To address the delayed-response issue, the iterative ad hoc method was proposed along with detailed discussions on the design of the engaged components as well as the potential issues owing to flawed simulation models; suggestions and negative examples were delivered as guidelines for preventing application builders from running into such issues. The new method was empirically evaluated by six experiments, which demonstrated the potential capability of the proposed iterative approach in terms of capturing system dynamics by comparing the results against the sequential counterparts. The evaluation was based on point estimation, which includes not only the direct comparison of the averaged numbers but also Welch's *t* test to provide more compelling and comprehensive statistical evidence.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

An ad hoc distributed simulation models an operational system by bringing together a collection of autonomous online simulations, each responsible for modeling a portion of the system. These simulators (i.e., LPs, short for logical processes) coordinate with each other via the ad hoc service (i.e., the synchronization mechanism) along with the space time memory (STM), which serves as data storage. The ad hoc service involves several mechanisms, such as the rollback mechanism; a proper design of these mechanisms with respect to a specific application is essential. This thesis work particularly focuses on modeling open queueing networks using ad hoc distributed simulation because queueing models are widely used abstractions for analyzing industrial systems.

First, this thesis mathematically defined the ad hoc approach, primarily for two reasons: (1) facilitating the performance analysis and (2) standardizing the terminology to, for example, improve the communication between application designers and simulation programmers. Furthermore, based upon this formulation this thesis modularized the ad hoc approach and provided the pseudo codes to help translate the conceptual descriptions to programs.

The second part of this thesis explored the capability of the ad hoc approach for the steady-state analysis of open queueing networks. Both analytical and empirical studies were conducted to address the estimation accuracy. The analytical study concerned in particular the queueing networks within which the departures of each queueing station (i.e., node) form a renewal process because renewal processes can be accurately modeled without a significant computational effort. The study identified such networks and proved that the ad hoc approach can model these networks and produce the estimates that are statistically equivalent to those from the sequential counterparts.

On the other hand, three empirical studies were conducted on the queueing networks that do not satisfy the above condition. The first one focused on the steady-state server utilization and mean queue lengths of the nodes in an 8×8 grid network. The network was configured with a stochastic unit-routing plan and various service-time distributions, leading to non-Poisson departures. The experiment results showed that the ad hoc approach is competitive to the sequential simulation approach despite the observable, yet mild and tolerable, overestimation in the queue-length estimators. Furthermore, an extended study revealed that a major cause of such overestimation issue is the input approximation.

The second empirical study examined the impact of partitioning methods on the prediction accuracy by extending the last study to evaluate 10 more partitioning layouts with the same experiment settings. Specifically, every node in the network was modeled by at least 8 LPs. The results did not show significant discrepancy among those layouts. In other words, partitioning methods do not appear to be a critical factor in the estimation accuracy as long as an adequate number of LPs are deployed for each node.

The third empirical study proposed a buffered-area mechanism to improve the prediction accuracy, which was shown to deteriorate due to the imperfect input approximation in the first empirical study. Instead of improving the accuracy though a better, more sophisticated input approximation method (which could be challenging and expensive to compute), the proposed buffered-area mechanism attempted to ease this overestimation problem by inserting buffers to “calm the shock wave” induced by the input approximation. The experiment results demonstrated that a small buffer is sufficient to greatly reduce the estimation biases.

The third part of this thesis broadened the applications of ad hoc distributed simulation to prove its competence in capturing the short-term state transients in open queueing networks. This study started with evaluating the capability of the previous ad hoc queueing simulation method used for the steady-state analysis. This original method showed itself unable to accurately reflect state changes in terms of when the changes happen and how long they last. Specifically, the changes were revealed with delay, positively connecting to the parameter—the length of the update period. To fix this delayed-response issue, the iterative ad hoc queueing simulation method was proposed by considering the observations from individual small simulations as the current state, instead of the future state predictions. The proposed method was discussed in detail, including the ideas behind the design of the involved components and the guidelines to prevent the undesirable livelock issue. Also, some instances of violating these guidelines were delivered to show the possible consequences.

The performance of the iterative ad hoc queueing simulation method with respect to instantaneous system-state reflection was evaluated empirically under 6 different networks. Both tandem and grid networks were included in the experiments, as well as various network conditions such as the server loadings (moderate and heavy) and the service-time variances (low, medium, and high). The experiment results led to the conclusion that this ad hoc method has the ability to model system dynamics accurately with the simulation results being statistically equivalent to those from sequential simulations.

5.2 Future Work

Ad hoc distributed simulation, which is motivated by the emerging embedded online simulations, presents a great number of future directions, for example, with regard to monitoring and optimizing operational systems. While this thesis has explored the work primarily focusing on modeling open queueing networks, many areas merit further investigation. The following lists some possible directions:

1. **Embracing data correlation.** The data processing in the current ad hoc queueing network simulations has been assuming that each observed/shared value is independent. This configuration benefits the implementation simplicity and the execution efficiency while sacrifices the accuracy performance. For example, Section 3.3.3 has shown that the renewal assumption regarding the arrival processes on input links is the cause of the observed estimation biases. Hence, accommodating correlation information into data models may mitigate estimation biases. Moreover, several other mechanisms in the ad hoc method should take data correlation into account as well. The information aggregation is one instance because the LPs modeling the same area may deliver predictions with correlation (considering the input data to their individual simulations may be potentially correlated). Also, the rollback detection mechanism should avoid possibly correlated estimations from destructing rollback criteria, such as widening or narrowing acceptable ranges to favor undesirable values or to reject valid ones.
2. **Handling unreliable data.** When it comes to data processing, it is always one of the critical factors that how much data is sufficient to support a claim with a

certain statistical significance. This concern impacts the ad hoc approach directly in determining the LP-coverage requirements for each portion of a system under investigation. Also, it should be taken into consideration the reliability of LPs as well as the communication among them because any failure and error during simulation executions would lead to data loss.

In addition to data loss, data contamination harms data reliability. Inaccurate data are produced by malfunctioning LPs (resulting from failed computing devices and invalid simulation models) and other data sources (e.g., sensors). These data should be excluded from simulations as soon as possible to avoid further damage. However, identifying such situations can be challenging as a productive detection mechanism must distinguish the errors due to pollution from true system dynamics.

3. **Balancing tradeoffs.** Ad hoc distributed simulation emerges from the issues that traditional simulation methodologies may run into when serving for real-time operational systems—the issues are responsiveness, scalability and failure resistance. In order to tackle these issues, some sacrifices have to be made, and the estimation accuracy is one of them. While the notion of tradeoffs has been "embedded" every bit in the ad hoc method, a general model to quantify the tradeoffs is essential for guiding simulation users throughout all the possible choices of the ad hoc mechanisms. Moreover, this may help develop new methods for various current and, most importantly, future conditions and requirements.
4. **Expanding simulation applications.** Queueing models are popularly-accepted benchmarking applications for evaluating the performance

of a simulation method. While this thesis work has explored the ad hoc method on modeling open queueing networks, an apparent next step is to study closed queueing networks. A closed queueing network, by definition, involves a constant number of units, getting services and looping inside the network; that is, no external arrival or departure exists. In practice, closed queueing networks deserve the same amount of attention as open ones do; particularly, the computer and communication systems are commonly modeled using closed queueing networks.

REFERENCES

[1] C. Toumazou, F. J. Lidgey, and D. G. Haigh, *Analogue IC Design: The Current-Mode Approach*, Repr. with minor corrections June 1990 ed. London: Peregrinus on behalf of the Institution of Electrical Engineers, 1990.

[2] R. J. Baker, *CMOS: Circuit Design, Layout, and Simulation*, 3rd ed. Hoboken, N.J.: IEEE Press/Wiley, 2010.

[3] R. L. Geiger, P. E. Allen, and N. R. Strader, *VLSI Design Techniques for Analog and Digital Circuits*. New York: McGraw-Hill Pub. Co., 1990.

[4] P. Lynch, "The Origins of Computer Weather Prediction and Climate Modeling," *Journal of Computational Physics,* vol. 227, pp. 3431–3444, March 2008.

[5] M. A. Finney, *FARSITE: Fire Area Simulator—Model Development and Evaluation*. Ogden, UT: US Department of Agriculture, Forest Service, Rocky Mountain Research Station, 1998.

[6] M. Reed, Ø. Johansen, P. J. Brandvik, P. Daling, A. Lewis, R. Fiocco*, et al.*, "Oil Spill Modeling towards the Close of the 20th Century: Overview of the State of the Art," *Spill Science & Technology Bulletin,* vol. 5, pp. 3–16, April 1999.

[7] D. Helbing, I. J. Farkas, P. Molnár, and T. Vicsek, "Simulation of Pedestrian Crowds in Normal and Evacuation Situations," in *Pedestrian and Evacuation Dynamics*, M. Schreckenberg and S. D. Sharma, Eds., New York: Springer, 2002, pp. 21–58.

[8] S. Eubank, H. Guclu, V. S. A. Kumar, M. V. Marathe, A. Srinivasan, Z. Toroczkai*, et al.*, "Modelling Disease Outbreaks in Realistic Urban Social Networks," *Nature,* vol. 429, pp. 180–184, May 2004.

[9] L. G. Birta and G. Arbez, *Modelling and Simulation: Exploring Dynamic System Behaviour*. London: Springer, 2007.

[10] F. Darema, "Dynamic Data Driven Applications Systems: A New Paradigm for Application Simulations and Measurements," in *Computational Science – ICCS 2004*, M. Bubak, G. D. v. Albada, P. M. A. Sloot, and J. Dongarra, Eds., Berlin: Springer Berlin Heidelberg, 2004, pp. 662–669.

[11] R. Fujimoto, D. Lunceford, E. Page, and A. M. Uhrmacher, "Grand Challenges for Modeling and Simulation," Schloss Dagstuhl, Dagstuhl, Germany Dagstuhl Seminar Report 350, August 2002.

[12] E. A. Lee, "Cyber Physical Systems: Design Challenges," in *Proceedings of the 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing*, Orlando, FL, 2008, pp. 363–369.

[13] W. J. Davis, "On-Line Simulation: Need and Evolving Research Requirements," in *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*, J. Banks, Ed., New York: Wiley, 1998, pp. 465–518.

[14] F. Kamrani and R. Ayani, "Using On-Line Simulation for Adaptive Path Planning of UAVs," in *Proceedings of the 11th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, Chania, Greece, 2007, pp. 167–174.

[15] T. Ye, H. T. Kaur, S. Kalyanaraman, and M. Yuksel, "Large-Scale Network Parameter Configuration Using an On-Line Simulation Framework," *IEEE/ACM Transactions on Networking,* vol. 16, pp. 777–790, August 2008.

[16] M. Y. H. Low, K. W. Lye, P. Lendermann, S. J. Turner, R. T. W. Chim, and S. H. Leo, "An Agent-Based Approach for Managing Symbiotic Simulation of Semiconductor Assembly and Test Operation," in *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems*, Utrecht, The Netherlands, 2005, pp. 85–92.

[17] M. Hunter, H. K. Kim, W. Suh, R. Fujimoto, J. Sirichoke, and M. Palekar, "Ad Hoc Distributed Dynamic Data-Driven Simulations of Surface Transportation Systems," *Simulation,* vol. 85, pp. 243–255, April 2009.

[18] M. Hunter, J. Sirichoke, R. Fujimoto, and Y.-L. Huang, "Embedded Ad Hoc Distributed Simulation for Transportation System Monitoring and Control," in *Proceedings of the 2009 INFORMS Simulation Society Research Workshop*, Coventry, United Kingdom, 2009, pp. 15–19.

[19] G. R. Madey, G. Szabó, and A.-L. Barabási, "WIPER: The Integrated Wireless Phone Based Emergency Response System," in *Computational Science – ICCS 2006*, V. N. Alexandrov, G. D. v. Albada, P. M. A. Sloot, and J. Dongarra, Eds., Berlin: Springer Berlin Heidelberg, 2006, pp. 417–424.

[20] G. R. Madey, A.-L. Barabási, N. V. Chawla, M. Gonzalez, D. Hachen, B. Lantz*, et al.*, "Enhanced Situational Awareness: Application of DDDAS Concepts to Emergency and Disaster Management," in *Computational Science – ICCS 2007*, Y. Shi, G. D. v. Albada, J. Dongarra, and P. M. A. Sloot, Eds., Berlin: Springer Berlin Heidelberg, 2007, pp. 1090–1097.

[21] D. Knight, T. Rossman, and Y. Jaluria, "Evaluation of Fluid-Thermal Systems by Dynamic Data Driven Application Systems," in *Computational Science – ICCS 2006*, V. N. Alexandrov, G. D. v. Albada, P. M. A. Sloot, and J. Dongarra, Eds., Berlin: Springer Berlin Heidelberg, 2006, pp. 473–480.

[22] D. Knight, Q. Ma, T. Rossman, and Y. Jaluria, "Evaluation of Fluid-Thermal Systems by Dynamic Data Driven Application Systems - Part II," in *Computational Science – ICCS 2007*, Y. Shi, G. D. v. Albada, J. Dongarra, and P. M. A. Sloot, Eds., Berlin: Springer Berlin Heidelberg, 2007, pp. 1189–1196.

[23] F. Zhao and L. J. Guibas, *Wireless Sensor Networks: An Information Processing Approach*. Amsterdam: Morgan Kaufmann, 2004.

[24] R. M. Fujimoto, *Parallel and Distributed Simulation Systems*. New York: Wiley, 2000.

[25] K. M. Chandy and J. Misra, "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs," *IEEE Transactions on Software Engineering,* vol. SE-5, pp. 440–452, September 1979.

[26] R. E. Bryant, "Simulation of Packet Communication Architecture Computer Systems," Master's Thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1977.

[27] 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, pp. 198–206, April 1981.

[28] E. W. Dijkstra and C. S. Scholten, "Termination Detection for Diffusing Computations," *Information Processing Letters,* vol. 11, pp. 1–4, August 1980.

[29] D. M. Nicol, "Noncommittal Barrier Synchronization," *Parallel Computing,* vol. 21, pp. 529–549, April 1995.

[30] D. M. Nicol, C. C. Michael, and P. Inouye, "Efficient Aggregation of Multiple PLs in Distributed Memory Parallel Simulations," in *Proceedings of the 1989 Winter Simulation Conference*, Washington, DC, 1989, pp. 680–685.

[31] D. M. Nicol, "The Cost of Conservative Synchronization in Parallel Discrete Event Simulations," *Journal of the ACM,* vol. 40, pp. 304–333, April 1993.

[32] J. Steinman, "Multi-Node Testbed: A Distributed Emulation of Space Communications for the Strategic Defense System," in *Proceedings of the 21st Annual Pittsburgh Conference on Modeling and Simulation*, Pittsburgh, PA, 1990, pp. 1111–1115.

[33] J. Steinman, "SPEEDES: Synchronous Parallel Environment for Emulation and Discrete Event Simulation," in *Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation*, Anaheim, CA, 1991, pp. 95–103.

[34] B. D. Lubachevsky, "Efficient Distributed Event-Driven Simulations of Multiple-Loop Networks," *Communications of the ACM,* vol. 32, pp. 111–131, January 1989.

[35] R. Ayani, "A Parallel Simulation Scheme Based on Distance between Objects," in *Proceedings of the SCS Multiconference on Distributed Simulation*, Tampa, FL, 1989, pp. 113–118.

[36] D. R. Jefferson, "Virtual Time," *ACM Transactions on Programming Languages and Systems,* vol. 7, pp. 404–425, July 1985.

[37] C. D. Carothers, K. S. Perumalla, and R. M. Fujimoto, "Efficient Optimistic Parallel Simulations Using Reverse Computation," *ACM Transactions on Modeling and Computer Simulation,* vol. 9, pp. 224–253, July 1999.

[38] B. Samadi, "Distributed Simulation, Algorithms and Performance Analysis," PhD Dissertation, Department of Computer Science, University of California, Los Angeles, Los Angeles, 1985.

[39] F. Mattern, "Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation," *Journal of Parallel and Distributed Computing,* vol. 18, pp. 423–434, August 1993.

[40] IEEE, "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Framework and Rules," New York: IEEE Computer Society, 2010, pp. 1–38.

[41] IEEE, "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Object Model Template (OMT) Specification," New York: IEEE Computer Society, 2010, pp. 1–112.

[42] IEEE, "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Federate Interface Specification," New York: IEEE Computer Society, 2010, pp. 1–378.

[43] B. Möller, K. L. Morse, M. Lightner, R. Little, and R. Lutz, "HLA Evolved—A Summary of Major Technical Improvements," in *2008 Fall Simulation Interoperability Workshop*, Orlando, FL, 2008.

[44] R. M. Fujimoto, "The Virtual Time Machine," in *Proceedings of the 1st Annual*

ACM Symposium on Parallel Algorithms and Architectures, Santa Fe, NM, 1989, pp. 199–208.

[45] K. Ghosh and R. M. Fujimoto, "Parallel Discrete Event Simulation Using Space-Time Memory," in *Proceedings of the 1991 International Conference on Parallel Processing*, Austin, TX, 1991, pp. 201–208.

[46] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*, 3rd ed. Boston: McGraw-Hill, 2000.

[47] R. Fujimoto, M. Hunter, J. Sirichoke, M. Palekar, H. Kim, and W. Suh, "Ad Hoc Distributed Simulations," in *Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation*, San Diego, CA, 2007, pp. 15–24.

[48] B. Biller and S. Ghosh, "Multivariate Input Processes," in *Handbooks in Operations Research and Management Science: Simulation*, S. G. Henderson and B. L. Nelson, Eds., Amsterdam: Elsevier, 2006, pp. 123–153.

[49] S. E. Chick, "Subjective Probability and Bayesian Methodology," in *Handbooks in Operations Research and Management Science: Simulation*, S. G. Henderson and B. L. Nelson, Eds., Amsterdam: Elsevier, 2006, pp. 225–257.

[50] L. Devroye, "Nonuniform Random Variate Generation," in *Handbooks in Operations Research and Management Science: Simulation*, S. G. Henderson and B. L. Nelson, Eds., Amsterdam: Elsevier, 2006, pp. 83–121.

[51] L. M. Leemis, "Arrival Processes, Random Lifetimes and Random Objects," in *Handbooks in Operations Research and Management Science: Simulation*, S. G. Henderson and B. L. Nelson, Eds., Amsterdam: Elsevier, 2006, pp. 155–180.

[52] Y.-L. Huang, C. Alexopoulos, M. Hunter, and R. M. Fujimoto, "Ad Hoc Distributed Simulation of Queueing Networks," in *Proceedings of the 24th International Workshop on Principles of Advanced and Distributed Simulation*, Atlanta, GA, 2010, pp. 57–64.

[53] Y.-L. Huang, C. Alexopoulos, R. Fujimoto, and M. Hunter, "On the Accuracy of Ad Hoc Distributed Simulations for Open Queueing Network," in *Proceedings of the 25th International Workshop on Principles of Advanced and Distributed Simulation*, Nice, France, 2011, pp. 163–168.

[54] W. Whitt, "Approximating a Point Process by a Renewal Process, I: Two Basic Methods," *Operations Research,* vol. 30, pp. 125–147, January–February 1982.

[55] W. Whitt, "The Queueing Network Analyzer," *The Bell System Technical Journal,* vol. 62, pp. 2779–2815, November 1983.

[56] F. Zouaoui and J. R. Wilson, "Accounting for Input-Model and Input-Parameter Uncertainties in Simulation," *IIE Transactions,* vol. 36, pp. 1135–1151, November 2004.

[57] C. Alexopoulos and D. Goldsman, "To Batch Or Not To Batch?," *ACM Transactions on Modeling and Computer Simulation,* vol. 14, pp. 76–114, January 2004.

[58] D. G. Kendall, "Stochastic Processes Occurring in the Theory of Queues and Their Analysis by the Method of the Imbedded Markov Chain," *The Annals of Mathematical Statistics,* vol. 24, pp. 338–354, September 1953.

[59] P. Billingsley, *Probability and Measure*, 3rd ed. New York: J. Wiley & Sons, 1995.

[60] C. Alexopoulos, "Statistical Estimation in Computer Simulation," in *Handbooks in Operations Research and Management Science: Simulation*, S. G. Henderson and B. L. Nelson, Eds., Amsterdam: Elsevier, 2006, pp. 193–223.

[61] C. Alexopoulos, D. Goldsman, and R. Serfozo, "Stationary Processes: Statistical Estimation," in *Encyclopedia of Statistical Sciences.* vol. 12, S. Kotz, C. B. Read, N. Balakrishnan, and B. Vidakovic, Eds., 2nd ed. New York: Wiley, 2005, pp. 7991–8006.

[62] R. L. Disney and D. Konig, "Queueing Networks: A Survey of Their Random

Processes," *SIAM Review,* vol. 27, pp. 335–403, September 1985.

[63] D. J. Daley, "The Correlation Structure of the Output Process of Some Single Server Queueing Systems," *The Annals of Mathematical Statistics,* vol. 39, pp. 1007–1019, June 1968.

[64] D. J. Daley, "Queueing Output Processes," *Advances in Applied Probability,* vol. 8, pp. 395–415, June 1976.

[65] J. F. Reynolds, "The Covariance Structure of Queues and Related Processes: A Survey of Recent Work," *Advances in Applied Probability,* vol. 7, pp. 383–415, June 1975.

[66] J. L. Jaina and W. K. Grassmannb, "Numerical Solution for the Departure Process from the GI/G/1 Queue," *Computers & Operations Research,* vol. 15, pp. 293–296, May 1988.

[67] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*. London: Chapman and Hall, 1986.

[68] R. Serfozo, *Introduction to Stochastic Networks*. New York: Springer, 1999.

[69] L. N. Lester, "Accuracy of Approximating Queueing Network Departure Processes with Independent Renewal Processes," *Information Processing Letters,* vol. 16, pp. 43–48, January 1983.

[70] P. A. W. Lewis and G. S. Shedler, "Simulation of Nonhomogeneous Poisson Processes by Thinning," *Naval Research Logistics Quarterly,* vol. 26, pp. 403–413, September 1979.

[71] E. Çınlar, *Introduction to Stochastic Processes*. Englewood Cliffs, NJ: Printice-Hall, 1975.