FEDERATED SIMULATION OF NETWORK PERFORMANCE USING PACKET FLOW MODELING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

TURAN DEMİRCİ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

FEBRUARY 2010

Approval of the thesis:

FEDERATED SIMULATION OF NETWORK PERFORMANCE USING PACKET FLOW MODELING

submitted by **TURAN DEMİRCİ** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. İsmet Erkmen
Head of Department, **Electrical and Electronics Engineering**

Prof. Dr. Semih Bilgen
Supervisor, **Electrical and Electronics Engineering Department**

Examining Committee Members:

Assist. Prof. Dr. Cüneyt Bazlamaçcı
Electrics and Electronics Engineering, METU

Prof. Dr. Semih Bilgen
Electrics and Electronics Engineering, METU

Assoc. Prof. Dr. Halit Oğuztüzün
Computer Engineering, METU

Assoc. Prof. Dr. Murat Erten
İNNOVA Ltd.

Assist. Prof. Dr. Şenan Ece Schmidt
Electrics and Electronics Engineering, METU

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: TURAN DEMİRCİ

Signature :

ABSTRACT

FEDERATED SIMULATION OF NETWORK PERFORMANCE USING PACKET FLOW MODELING

Demirci, Turan
Ph.D., Department of Electrical and Electronics Engineering
Supervisor : Prof. Dr. Semih Bilgen

February 2010, 103 pages

Federated approach for the distributed simulation of a network, is an alternative method that aims to combine existing simulation models and software together using a Run Time Infrastructure (RTI), rather than building the whole simulation from scratch. In this study, an approach that significantly reduces the inter-federate communication load in federated simulation of communication networks is proposed. Rather than communicating packet-level information among federates, characteristics of packet flows in individual federates are dynamically identified and communicated. Flow characterization is done with the Gaussian Mixtures Algorithm (GMA) using a Self Organizing Mixture Network (SOMN) technique. In simulations of a network partitioned into eight federates in space parallel manner, it is shown that significant speedups are achieved with the proposed approach without unduly compromising accuracy.

Keywords: federated communication network simulation, discrete event simulation, federated simulation, packet flow modeling, gaussian mixtures

ÖZ

BİLGİSAYAR AĞLARININ PAKET AKIŞ MODELLEMESİ İLE FEDERELİ BENZETİMİ

Demirci, Turan
Doktora, Elekrik Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Semih Bilgen

Şubat 2010, 103 sayfa

Haberleşme ağlarının dağınık benzetimi için federeli yaklaşım, benzetim yazılımının sıfırdan geliştirilmesi yerine, halihazırda bulunan benzetim modellerinin ve yazılımlarının bir alt yapı ile bir araya getirilerek tüm benzetim yazılımının oluşturulduğu alternatif bir yaklaşımdır. Bu çalışmada, haberleşme ağlarının federeli benzetim yoluyla çözümlenmesi esnasında federeler arası haberleşme yükünü önemli düzeyde azaltacak bir yöntem önerilmiştir. Bu yöntem federeler arasında paket düzeyinde konuşmak yerine, paket akışlarının ayırıcı özelliklerinin dinamik olarak bulunarak haberleşilmesi esasına dayanmaktadır. Akışların ayırıcı özellikleri Gauss karışımı algoritmasının kendinden organizeli karışım ağları methoduyla çözümlenmesiyle bulunmuştur. Benzetimlerde ağ yapısı uzay ekseninde sekize bölünmüş ve benzetim sonuçlarında çok az hata ile ciddi hız kazanımları gösterilmiştir.

Anahtar Kelimeler: Federeli Haberleşme ağları benzetimi, ayrık olay benzetimi, federeli benzetim, paket akış modellemesi, Gauss karışımı

To My Family

ACKNOWLEDGMENTS

I would like to express my gratitude to my supervisor, Prof. Dr. Semih Bilgen, whose expertise, understanding, and patience, added considerably to my graduate experience. It is an honor for me to have an opportunity to study with him and learn knowledge and skills that will guide me entire my life. In addition, he was always accessible and willing to help his students with their research.

Assist. Prof. Dr. Cüneyt Bazlamaçcı and Assoc. Dr. Halit Oğuztüzün deserve a special thanks as my thesis committee members and advisors. The associated experience broadened my perspective on the thesis subject.

In particular, I would like to thank Prof. Dr. Muammer Ermiş and Prof. Dr. Işık Çadırcı for their help and understanding in the past eight years that made things easier for me. In addition, all my work buddies at the Tubitak Uzay made it a convivial place to work. They showed a great patience on my daily complainings. Thanks.

I would like to express my deepest gratitude to my family, without whose continuous prayers, support and encouragement, this thesis would not have been possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURES

NOMENCLATURE

ALSP:	Aggregate Level Simulation Protocol
API:	Application Programming Interface
CWND:	Congestion Window Size
DIS:	Distributed Interactive Simulation
EM:	Expectation Maximization
GMA:	Gaussian Mixtures Algorithm
HLA:	High Level Architecture
ISP:	Internet Service Provider
LBTS:	Lower Bound Time Stamp
LP:	Logical Process
LRC:	Local RTI Component
PDF:	Probability Density Function
RTI:	Run Time Infrastructure
RTT:	Round Trip Time
SOMN:	Self Organizing Mixture Network
STD:	Standard Deviation

TCP:	Transmission Control Protocol
TSO:	Time-stamp-ordered
UDP:	User Datagram Protocol

CHAPTER 1

INTRODUCTION

1.1 Background

Simulation is a widely used technique which provides to researchers and developers a fast, inexpensive, controlled and reproducible environment to analyze performance of communication networks under different conditions. It is extensively used by researchers while developing and testing new protocols or improving existing ones. Simulation is also used by network developers for comparison of different alternative topologies or technologies and test their performance under different traffic conditions without sizeable investment. Network operators and maintainers can also use simulation for identifying problems and for finding appropriate solutions for their networks without interfering with the operation of the live network.

In the context of packet networks, although there exist different approaches for network simulation, only packet level simulation enables detailed protocol modeling and performance investigation while providing good accuracy as the simulation objects such as packets, queues, buffer locations, communication links, etc. can be modeled as they were in operation in the real network. Because of the level of detail, packet level simulation may require much processing power and memory space. It is well known that processing power required for packet level network simulation increases more than linearly with the intensity of packet traffic to be simulated (e.g. [1],[2],[5]). So problems may arise when the number of simulated nodes, the simulated traffic volume or both get large. On the other hand, memory required for keeping routing information and keeping objects' state information increases at least linearly with the number of nodes to be simulated. So for a large scale network (e.g. more than 1000 nodes), these requirements can not be practically met using a single board computer for which resources are limited.[1]

A logical way for gaining required processing power and memory for a large scale packet level simulation is to partition the simulation in space parallel manner (i.e. partition the topology) and simulate each partition, on a different computer on a different logical process (LP). There exist two main approaches while building such a distributed simulation, namely traditional parallel distributed simulation and federated distributed simulation[2][6][14]. Traditional simulation approach creates optimized simulations built for a specific aim from scratch, where federated approach mainly aims to increase code reuse by combining readily built simulators together to form whole simulation. Time and effort required for traditional approach is much higher than federated approach so that federated approach is usually preferred when models are available in a different simulator.

When a simulation model is partitioned and processed on different LPs, called federates in the federated simulation case, on different machines, it is obvious that some coordination in simulation time becomes necessary between submodels to satisfy the causality constraint. This constraint stipulates that in a distributed simulation, LPs should process all the events in increasing time order. Without compromising causality, LPs have to process local events and foreign events generated by other federates. Local events that occur in individual LPs are stored in event queues and can be processed in time order, but any time during the simulation it is possible that a foreign event can be generated with a time stamp earlier than an LP's local time, thus violating the causality constraint. So a time management policy should be agreed and used between LPs[13][6]. In the literature different time management policies have been proposed such as time driven, event driven, conservative, optimistic etc. for the time synchronization of LPs but as it is easy to realize and implement, most of the network simulators have chosen to use conservative time management policy which strictly avoids causality violation.

Optimistic time management algorithm relaxes the constraints on LPs by allowing increasing their times independently but it is required that a violation of causality should be detected and repaired in terms of rollback operations. So realization and implementation of such a time management approach is harder than conservative time management approach. Also extra memory space required to save multiple of old object states for rollback operations, may increase the memory requirements in the orders of sequential counterpart, that makes optimistic time management approach unpreferred for large scale packet level network simulation. Optimistic time management approach is more suitable for the cases where infrequent communication is required between LPs. For such a case, optimistic time management approach possibly require less rollback operations and achieves high parallelism. But in the case of distributed packet level simulation of network, LPs are packet level dependent on each other. So highly number of violation of causality may occur and lots of rollback operations might be required with optimistic time management approach. These rollback operations may decrease simulation performance significantly. This is another reason why conservative time management approach is usually chosen over optimistic time management.

Conservative time management approach has also one major drawback. When an LP reaches a synchronization point, that is the time up to which it is safe to process local events without violating causality, it waits for other LPs to reach the same point. The problem is, if the mean time between synchronization points decreases, LPs get tightly coupled, decreasing the parallelism. This is exactly the case for network simulation in space parallel distributed manner where it is usual that some of the packet flows being simulated will have routes that are simulated in different LPs. So when a packet belonging to such a flow finishes its part of the route in one LP, some communication and synchronization is required between LPs so that the packet will continue the remaining part of its path in the next LP. When the number of such packets increases, very tight coupling of LPs occurs, decreasing parallelism drastically.

A further source of overhead for federated simulation is the highly renowned High Level Architecture (HLA) which is a general purpose architecture for distributed computer simulation systems. Using HLA, computer simulations can communicate with other computer simulations regardless of the computing platforms. Communication between simulations is managed by Run Time Infrastructure (RTI). The RTI provides a programming library and an application programming interface (API) compliant to the interface specification [10][21]. While HLA and RTI provide powerful standards and facilities for distributed simulation, all communication between federates must be realized through RTI mechanisms, requiring significant overheads if invoked at a packet level.

1.2 Thesis Objective and Scope

In this study a novel approach for distributed simulation of a network is proposed, based on packet level flow modeling that loosens LPs by reducing communication and synchronization requirements. The idea is, instead of performing the necessary synchronization for each of these packets, obtain characterizing parameters for each flow and send these characteristics to the receiving LP. At the receiving LP side, a packet flow that possesses the same characteristics as the original flow will be generated and simulation will continue. This would drastically reduce the overhead incurred by synchronizing individual packets between different federates. So an accurate and time efficient way should be found to represent these packets at the sender side and a way to generate packets similar to the original ones using these parameters at the receiver side.

The Gaussian Mixtures Algorithm (GMA), which is used for capturing packet flow characteristics in this study, is a popular method that is used for probability density function estimation of a given sample set[29][35]. As the aim in this study is to achieve speed up without unduly sacrificing simulation accuracy, GMA must be implemented to operate in a fast and accurate manner. In the literature different methods for solving GMA are proposed. Within the scope of the present study, Expectation Maximization (EM)[34] and Self Organizing Mixture Network (SOMN)[31] approaches for GMA implementation were investigated and a comparison was made on their accuracy and time performance. As SOMN algorithm was seen to have important advantages over EM on both time and accuracy, it is chosen as the method to be implemented in the simulation. Studies on SOMN showed that learning rate methodology and determining a stopping condition have great effects on SOMN accuracy and time results[32][36]. In this study, a novel learning rate methodology and stop condition for SOMN training that provides more accurate results without increasing computation time much are also proposed and are shown to perform better than other tested approaches.

In short the major contribution of this study is packet flow modeling for inter-federate packets. A lesser contribution is enhancing SOMN with a novel learning rate adaptation mechanism.

Although flow modeling can be applied to the traditional approach of distributed simulation which aims to speed up simulation by developing optimized models from scratch and using fast communication libraries, in the present study, we preferred to use the federated approach which has the significant benefit of providing a high level of interoperability among existing simulations and possibly live actual systems. In general, it is accepted that federated simulation greatly reduces the cost, time and effort of large scale simulation development[16].

To see the accuracy and time performance of the proposed flow modeling approach, three fundamental approaches to the simulation of a packet network were compared. These were (i) classical sequential simulation, (ii) federated simulation with each inter-federate packet being synchronized between federates, and (iii) federated simulation with the proposed approach that computes and communicates per-flow characteristics between federates. Correct representation of inter-federate packet characteristics is critical as it directly affects packet drop rates and packet inter-arrival characteristics. To be accurate and fast, flow characteristics were modeled using GMA whose parameters were estimated with SOMN method using the new learning rate methodology and stop condition proposed.

To show that the proposed approach works correctly and accurately for different traffic types under different network conditions, first, a UDP-like non-feedback protocol has been implemented and a group of simulations were compared with sequential simulation for various packet traffic characteristics (Exponential, Pareto, Hyper-exponential) and network conditions (i.e. limited/unlimited router buffers). It is shown that the proposed flow modeling approach highly reduces the inter-LP synchronization and communication overhead and significantly speeds up simulation, without unduly compromising per flow packet interarrival mean and standard deviation (STD) accuracy. Next, a TCP-like feedback based protocol is simulated where data packet flows are modeled with GMA and protocol packets (i.e. ACKs) are sent individually between federates. Comparison of the three fundamental simulation approaches (sequential simulation, federated simulation without flow modeling, federated simulation with flow modeling) on per flow packet interarrival mean and STD, show that federated simulation with flow modeling achieves comparable accuracy to classical federated simulation in much shorter time, especially as the network size increases. Further speed up was shown to be achieved by also modeling ACK packet flows.

1.3 Thesis Organization

This thesis is organized as follows: In chapter 2, fundamental concepts and the relevant literature on communication network simulation, packet level simulation, large scale network simulation, distributed simulation and also time management algorithms for distributed simulation are reviewed. In Chapter 3, federated network simulation developed using HLA is summarized. An overview of the RTI mechanism and functions are given and their usage in a federation life cycle are presented. Also the kernel of the federated simulator developed within the scope of this study is presented. Chapter 4 presents the proposed novel approach and its evaluation. First the problem of tight synchronization is introduced and resulting issues are illustrated on a two-federate network simulation. Next, the novel idea of flow modeling is presented and the possible different approaches that can be used for flow modeling, namely, modeling by a simple mean, modeling according to the distribution type and general packet modeling are investigated. A group of simulations are presented based on these approaches. Simulation results, especially the ones where routers have limited memory, show that for the purposes of simulation accuracy, a method that can model a general distribution should be deployed. Next, GMA and the two methods EM and SOMN for evaluating GMA parameters are summarized. The chapter continues by presenting the accuracy and speed up obtained with the proposed modeling approach with the SOMN training method. Next, flow modeling for a TCP-like feedback protocol is investigated. The chapter ends by presenting the accuracy and speed up obtained for the proposed modeling approach for TCP-like feedback based protocol and comparing the achievements with both sequential and packet level federated simulation without flow modeling. Chapter 5 concludes the thesis, summarizing the contributions of the study and its limitations. Suggestions for future work on this subject complete this final chapter.

CHAPTER 2

COMMUNICATION NETWORK SIMULATION

2.1 Packet Level Simulation

Simulation is a widely used technique for modeling a physical system and investigate its behaviour characteristics for various scenarios. In a simulation, the physical system that is under investigation is represented in terms of a set of states and its behavior is predicted from the behavior of the modeled (simulated) system as the states change over time. As such, simulation has to model the states and state changes through state transitions over time for the physical system and also should have a representation of time. As an example, from the simulation of a communication network, the number of packets waiting in a router can be selected as the state where a packet arriving to that router can be defined as a state transition by simply increasing the state value by one at time t. Time can be represented at any convenient level of detail starting at zero to the end of simulation time.

Two main approaches prevail in network simulation. The first approach is based on queuing theory that may yield fast solution of network states in a memoryless fashion. But the complexity of modern networks combined with the inability to apply simplifying assumptions in many situations (e.g., it is well known that Markovian traffic assumptions are often inappropriate and can lead to misleading results) makes it usually less detailed, less accurate and less flexible than packet level simulation, rendering it useless for many practical protocol level analyses.[22][23]

The second approach, packet level discrete event simulation that is used by most popular network simulators (i.e. Ns2[7], Glomosim, Omnet++, Opnet, Qualnet,PDNS[8]), fit well for simulation of communication networks. In this approach communication protocols, network entities are modeled by state machines in packet level detail. Network traffic is generated as packets and simulated packet by packet along the route from source to destination according to the modeled protocols under investigation. In packet level discrete event simulation, state changes related to the network traffic like packet formations, packet movements, packet drops are represented as a sequence of events occurring at discrete points in simulation time and stored in a time ordered event queue. The simulator engine selects the event with earliest timestamp and removes it from the event queue. Next, it advances simulation local time to the timestamp of the event and processes the event. This processing may cause one or more state changes, generation of new events or cancellation/postponing of future events. Simulation continues to process the events until predefined simulation time is achieved or there is no event to be processed in the event queue. Figure 2.1. shows a schematic for a simple discrete event simulator.

```
While ( ( simulation_time < simulation_end_time )
        &&  ( event_queue.size() > 0 ) )
{
    Event event;
    event = Remove_earliest_event_from_event_queue();
    Update_simulation_time(event.timestamp);
    Process_the_event(event)
    Change_states_of_appropriate_objects;
    Schedule_new_events_if_required;
    Cancel_future_events_if_required;
    Postpone_future_events_if_required;
}
```

Figure 2.1: Discrete Event Simulator Engine

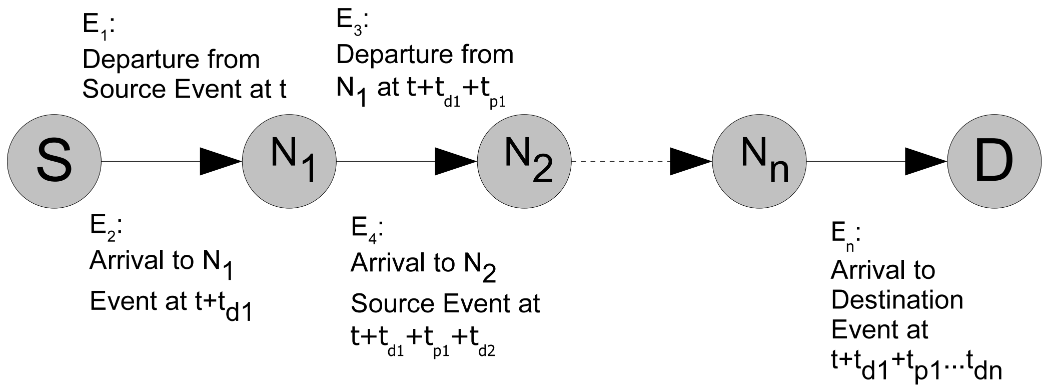
In Figure 2.2, a simple diagram that represents a typical life cycle of a packet from generation to the arrival to the destination in a packet level discrete event simulation is demonstrated.

If we briefly explain Figure 2.2 in time sequence:

1. At time t, a packet is generated at the source and an arrival event to the intermediate node N1 at time $t + t_{d1}$ (t_{d1} refers to link delay from source to intermediate node) is scheduled.
2. At time $t+t_{d1}$, N1 makes required processing for the protocols under simulation, makes appropriate state changes and schedules a departure event for time $t + t_{d1} + t_{p1}$ (t_{p1} to

queuing and processing delay in the source node).

3. At time $t + t_{d1} + t_{p1} + d_2$ N2 receives the packet and makes required processing.
4. Packet traverses route to the destination as described in 2-3 until it reaches the destination. And finally, destination produces an appropriate response packet to be forwarded to the receiver or other destination.



E_1: Departure from Source Event at t
E_3: Departure from N_1 at $t+t_{d1}+t_{p1}$
S
N_1
N_2
N_n
D
E_2: Arrival to N_1 Event at $t+t_{d1}$
E_4: Arrival to N_2 Source Event at $t+t_{d1}+t_{p1}+t_{d2}$
E_n: Arrival to Destination Event at $t+t_{d1}+t_{p1}...t_{dn}$

Figure 2.2: Typical Life Cycle of a Packet in Packet Level Network Simulation

During simulation, some statistics about flows (i.e. number of packet arrived, number of packets dropped, throughput, packet interarrival variances) or about router buffer utilizations, etc. should be collected as results while packets traverse intermediate nodes or when they reach the destination.

Looking at the above discussion, modeling and implementation of packet level discrete event simulation for a communication network is not so difficult. One only needs to

1. model the network protocols,
2. model the behavior of the network entities,
3. find a way to represent network topology under simulation,
4. develop a simulator engine to process the generated events,
5. develop some observing mechanisms to get simulation results.

2.2 Large Scale Network Simulation

It is usually enough to simulate for small to medium network sizes to get familiar with the behavior of most network protocols. But to investigate the performance and problems of the protocols that may operate on an actual existing network, the number of network nodes that must be considered easily run in the thousands or possibly more. There is also a need for simulating large number of network nodes for special cases like investigation of a virus attack, simulation of peer to peer networks or investigating the effect of enabling multicast traffic in an Internet Service Provider (ISP)[4]. Furthermore, statistics about network capacities show that used bandwidth tends to double approximately every two years [1][3]. Together with the increase of network nodes and bandwidth to be simulated, large numbers of packets must be simulated in typical simulation. This leads to large-scale simulations that require high processing power and memory resources.

It is well known that the execution time of a packet level network simulation grows faster than linearly with the network size [2][5]. Theoretical studies show that for a network size of order O(n), the sequential discrete event simulation time includes terms which are of order O(n . log(n)), that corresponds to keeping the event queue of the simulator sorted in time so that the order of processing of events corresponds to their simulation time[2][5].

Also memory requirement increases at least linearly with the number of simulated nodes. The memory size is mainly defined by centralized network configuration and routing information required during construction of the simulated network. Additionally, the needed memory increases also with the intensity of traffic flows that dictate the size of the event list of the simulator. As the network size increases, after some point, processing power and memory requirements increase to a great extent so that it becomes impossible to achieve required memory for the simulation using a single board computer for which memory capacities can be increased up to some fixed point [2][8][12]. E.g., for the simulation of a campus network having 3000 nodes connected together with 100Mb/s connection using ns-2 may require more than one hour to simulate one minute of operation on a powerful computer [1]. Also even on a simulation system with large amounts of physical memory (2 GBytes for example), the maximum size of the network that can be represented is usually on the order of a few thousand nodes for famous simulators such as ns-2. [1]

Efficient simulation techniques for such networking models have become an important issue. Many methods have been proposed to speed up network simulation. These methodologies can be categorized into three different types: computational power; simulation technology; and simulation model[22]. In the direction of computational power, simulations can be speeded up by using faster and more powerful machines. Trend in this way is to increase the number of CPU cores rather than increase processor speeds because of the limitations on the hardware manufacturing technology. So for a simulation that is developed to be run on a single CPU could not be speeded up much using this approach.

In the simulation technology direction, new enhanced algorithms for implementing the simulation can further speedup simulation. Algorithms such as the calendar queue algorithm[28], which is used in ns-2 [7], and splay tree algorithm have been proposed in order to improve the efficiency of event list manipulation. It is very well known that such algorithms are enhanced for specific assumptions. They will speed up the simulation for the case they are developed for, but may perform worse for different scenarios. An example, a calendar queue's performance depends on how evenly events are distributed across its buckets. If the bucket width is too big comparing to the average inter-event gap, many events fall into few buckets. On the other hand, if the bucket width is too small, events are sparsely populated and finding the next event requires scanning many buckets. [23].

A third approach is to use models with a higher level of abstraction, simplifying the simulation and improving its efficiency. The tradeoff in this case, is the accuracy of the desired measures of interest obtained by the more abstract model. For example, the packet-train simulation technique models a cluster of closely-spaced packets as a single ”packet-train” [23]. The cost reduction comes at the price of introducing a measurable degree of approximation into the resulting metrics. Such coarse grain simulators run quickly, as it deals with packet-trains instead of individual packets, but cannot represent the details of the underlying physical system.[22]

Another modeling technique making simplified assumptions about the real system is the fluid model, which was first proposed by Anick et al. in [39] to model data network traffic. In the fluid simulation paradigm, network traffic is modeled in terms of a continuous fluid flow, rather than discrete packet instances. A cluster of closely-spaced packets may be modeled as a single fluid chunk with a constant fluid rate, with small time-scale variations in the packet stream being abstracted out of the model.

A fluid simulator keeps track of the fluid rate changes at traffic sources and network queues. An equivalent packet-level simulator would keep track of all individual packets in the network. In fluid simulation, the higher level of abstraction suggests that less processing might be needed to simulate network traffic. This is not surprising as a large number of packets can be represented by a single fluid chunk[24][27]. For simple network components, where traffic flows do not compete for resources, the fluid simulator outperforms the packet-level simulator. An example would be a link that connects two nodes and never experiences queuing; this component only introduces a constant propagation delay. The major drawback of a fluid model is that the accuracy of the interest measures is compromised due to the abstraction.[22]

Under light or moderate traffic conditions, the number of events raised in flow-level simulators is typically much less compared to packet-level simulators. However, it has been shown that under heavy traffic conditions and when several flows share the same available resources, one change in the sending rate of a single flow may influence the sending rate of many other flows. This can cause an avalanche of sending rate updates with a dramatic impact on the running time of the simulation. This effect is known as the ”ripple effect” [40], and may lead to drastic performance degradation, such that packet-level simulation will outperform flow-level simulation.

A high precision, fast and logical way of achieving the required computational power and large memory is to partition the network into small sub-networks, and then simulate each partition on a different computer in a coordinated fashion. In the literature numerous studies have been devoted to this subject and the solutions are centered under two approaches, namely, traditional and federated simulation.

The traditional approach to distributed simulation is already used by well known simulators like TeD, SSFNet, GloMoSim, TaskKit, and ROSSNet [1]. In this approach, the simulation model is partitioned into small models and each partition is simulated by a copy of the same process in coordination with the others. Here, simulation components are designed and implemented to be optimized with the rest of the simulation and communicate through special fast communication libraries to achieve information update and time synchronization. The main advantage of this approach is the simulation speed achieved as the simulation kernel and simulation models are specifically developed and optimized. The main drawback of the approach is the enormous level of effort required to develop the simulation system (i.e. Simulation engine, simulation models, development and analysis tools) from scratch. Also the effort required to validate and verify the whole simulation system is usually prohibitively high.

The second approach, federated simulation, aims to combine readily built simulation systems together to maximize code reuse. High Level Architecture (HLA) is a widely adopted international standard that defines the principles of distributed simulation and the interfaces among different partitions. The simulation model is partitioned into small parts and each sub-model is simulated by a different simulator, called a federate, that has proficiency for the sub-model it is assigned. The resulting simulation system is called a federation. Federates do not communicate or synchronize directly but all the communication and synchronization between federates are realized by the Run Time Infrastructure (RTI). The main advantage of this approach is, if a property (i.e. Network protocol) is unimplemented for some simulator but implemented in the other one, one can federate these simulators to form a complete simulation for the desired properties. As the approach combines simulators that are readily available, tested and validated, the effort required to form the simulation is drastically decreased.

The concept of federated simulation in not new. It has been successfully applied especially for the case of military simulation applications. The SIMNET project and the Distributed Interactive Simulation (DIS) protocols, Aggregate Level Simulation Protocol (ALSP) are some examples of the use of this approach. The parallel simulation community has also applied this technique to parallel queuing network simulations [1][14][18].

2.2.1 Model Partitioning

While developing a distributed/federated simulation, an important issue to be resolved is how to partition the simulation model into subparts and assign them to the processes. Here we will summarize three partitioning approaches, namely space parallel, time parallel and space-time parallel partitioning. [6]

In Space-parallel approach, state variables are partitioned and assigned to the LPs for the whole simulation duration. Each LP is responsible for computing the values of the assigned state variables during whole simulation in time and information parallel with other LPs. Distributed simulation of a communication network is a good case to apply space parallel partitioning approach as the network topology can easily be divided into small parts. Each LP will calculate the states of the subnetwork (i.e. Queue lengths of nodes, average bandwidth achieved for the flows) it owns in parallel from start to the end of simulation. As the topology is divided and each part is assigned to a LP, obviously some state information should be exchanged between LP‘s and also some time synchronization mechanism should be implemented to achieve causality. The advantage of using space parallel approach for parallel simulation of a network is the ease of realization and implementation. The main drawback of the approach is, depending on the model under simulation, tight synchronization of LPs may be required to achieve causality. This results in LPs to wait for each other, decreasing the speed up advantage of parallelism.

Time parallel partitioning approach divides the simulation time axis into non overlapping simulation time intervals. Each time window is assigned to a LP which has to compute all the state variables of the simulation in that duration. At each pass, LPs exchange final values of the state variables achieved at the end of the period they simulate and start a new pass. Iterations continue until a fixed point is reached. The benefit of the time parallel approach is its massively parallel nature of execution because of loose synchronization between LPs. But, it is harder to realize and implement the simulation, and accuracy of results may depend on the initial values guessed. Furthermore, all LPs have to store whole state variables making it useless for large scale simulations where memory requirements may become formidable.

Space-time parallel partitioning can be thought as a combination of space and time parallel approaches. In this approach space-time domain is divided into non-overlapping regions and each region is assigned to a logical process. In this approach as state variables can be divided between LPs, larger memory for the whole simulation can possibly be achieved. Furthermore, this approach enables a relaxation in time synchronization, at the cost of number of iterations. To list the disadvantages, it is harder to realize and implement the simulation and similar to the time parallel approach to distributed simulation, accuracy of results may depend on the initial values guessed.

2.2.2 Time Management in Parallel Simulations

Causality constraint stipulates that, ”a discrete-event simulation, consisting of logical processes (LPs) that interact exclusively by exchanging time stamped messages obeys the local causality constraint if and only if each LP processes events in non-decreasing time stamp order.” [6]

If LPs process local events independently of other LPs, a violation of causality constraint may arise as late or early foreign events from other LPs can possibly be delivered. This kind of a violation may change the course of events and bring forth invalid simulation results that are different from sequential counterpart. In addition, if local times of LPs‘ are incremented independently, there can also be deadlocks in the system that may arise if objects that are simulated in different LPs wait for each other for inputs before producing an output. So some time management should be done between LPs to avoid violation of causality and deadlocks. Figure 2.3 shows a classification of approaches that have been proposed to manage time for distributed simulation to address these problems. [6][9][13]

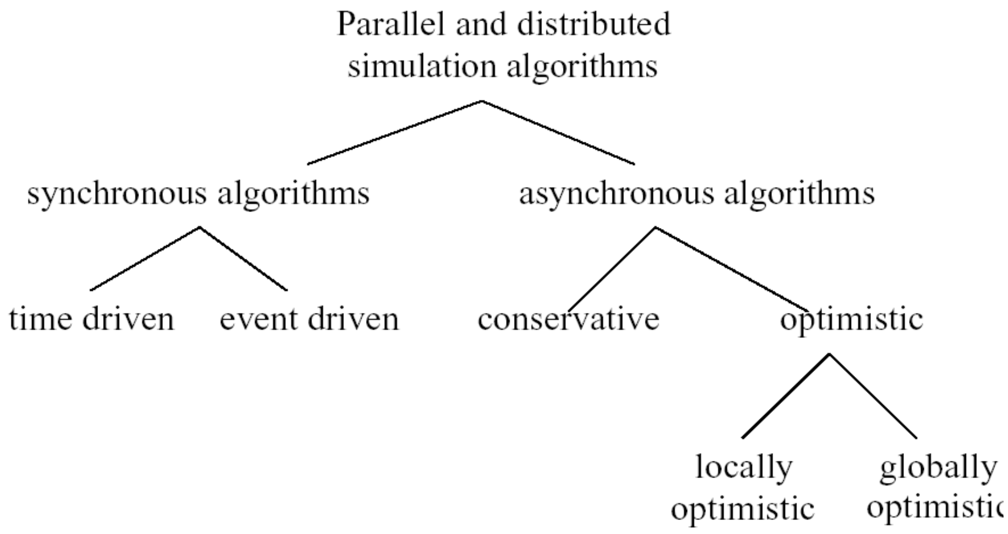


Figure 2.3: Classification of Time Management Algorithms[9]

Synchronous algorithms use a global timing mechanism to manage time and event delivery.

LPs are allowed to advance their times up to the global time. Here no deadlock can occur as global timing mechanism is used. In the time driven case, global time is advanced by constant time intervals and all objects‘ inputs, outputs and new states are computed at each time advance. Computed state variables are exchanged at the end of each time advance. In the event driven case global time is advanced to the time of the next event with the smallest time. State variables are again updated at every global time advancement. The event driven scheme reduces the need of computing objects‘ state and number of synchronization messages compared to the time driven scheme as it omits calculation and synchronization for the time intervals between two events in which no change occurs in objects’ inputs, outputs and states.

Asynchronous algorithms are developed to achieve better parallelism than synchronous ones by allowing LPs to process local events without computing a global time. A synchronization of the local clocks of the LPs is required whenever inter-LP communication occurs. In the conservative case, the main idea is avoiding violation of the causality constraint by guaranteeing that no foreign event will arrive having time less than local time of the LP. So processing of local events and foreign events in proper time sequence is achieved.

Unlike the conservative case, optimistic synchronization does not guarantee causality. LPs process local events as if no foreign event would arrive. When a foreign event having a time earlier than local simulation time arrives, it is named a straggler event, and must be detected and repaired by using a ”rollback” operation. So the LPs should remember all the state changes that occurred after a safe point and also should annihilate any outputs sent from the straggler event‘s time to its local simulation time. Annihilation of the messages is done by sending anti-messages to the corresponding formerly sent messages. When a message and its anti-message are found in the same queue, they cancel each other. If a message corresponding to the anti-message has already been processed, the station should also rollback to that time. So a rollback operation can result in cascade rollbacks.

Because of the ease of realization and implementation, most distributed simulations use the conservative time management approach. For the simulation of a network, the conservative approach performs better than synchronous approaches and also, implementation is relatively simple in comparison to optimistic approaches. For this reason, in the scope of this study, the conservative approach to time management has been preferred.

CHAPTER 3

FEDERATED NETWORK SIMULATION USING HIGH LEVEL ARCHITECTURE

The High Level Architecture (HLA) is a general purpose architecture for distributed computer simulation systems[19]. HLA provides a common architecture supporting reuse and interoperation of simulations regardless of the computing platforms.

The first key components of the system are the simulations themselves, or more generally, the federates. A federate can be a computer simulation, or a supporting utility (such as a viewer or data collector). All modeling and computation is in the federates. HLA imposes no constraints on what is represented in the federates or how it is represented, but it does require that all federates incorporate specified capabilities to allow the objects in the simulation to interact with objects in other simulations through the exchange of data supported by services in the Run Time Infrastructure (RTI). The second functional component of HLA is RTI. RTI provides a set of services that support the simulations in carrying out these federate to federate interactions and federation management support functions. These services will be discussed later. All interactions among the federates flow through the RTI. The third functional component is the runtime interface. The HLA runtime interface specification provides a standard way for federates to interact with the RTI, to invoke the RTI services to support runtime interactions among federates, and to respond to requests from the RTI. This interface is implementation independent and is independent of the specific object models and data exchange requirements of any federation.[41]

3.1 Run-Time Infrastructure (RTI) Overview

RTI is a software that provides common services to federates in the HLA standard [19][20][21]. RTI supports many types of functions that are required for operating federates having different simulation architectures (i.e. time management, object management) in parallel. RTI handles federate to federate communication over TCP/IP network so federates can be executed on a standalone workstation or over an arbitrarily complex network. This allows federates to be run from arbitrary geographic places where it may be impossible or too costly to bring simulation components together (i.e. A battlefield training simulation whose components are in two or more countries).

RTI software is currently composed of the RTI Executive process (RtiExec), the Federation Executive process (FedExec) and the libRTI library [19][21] as shown in figure 3.1

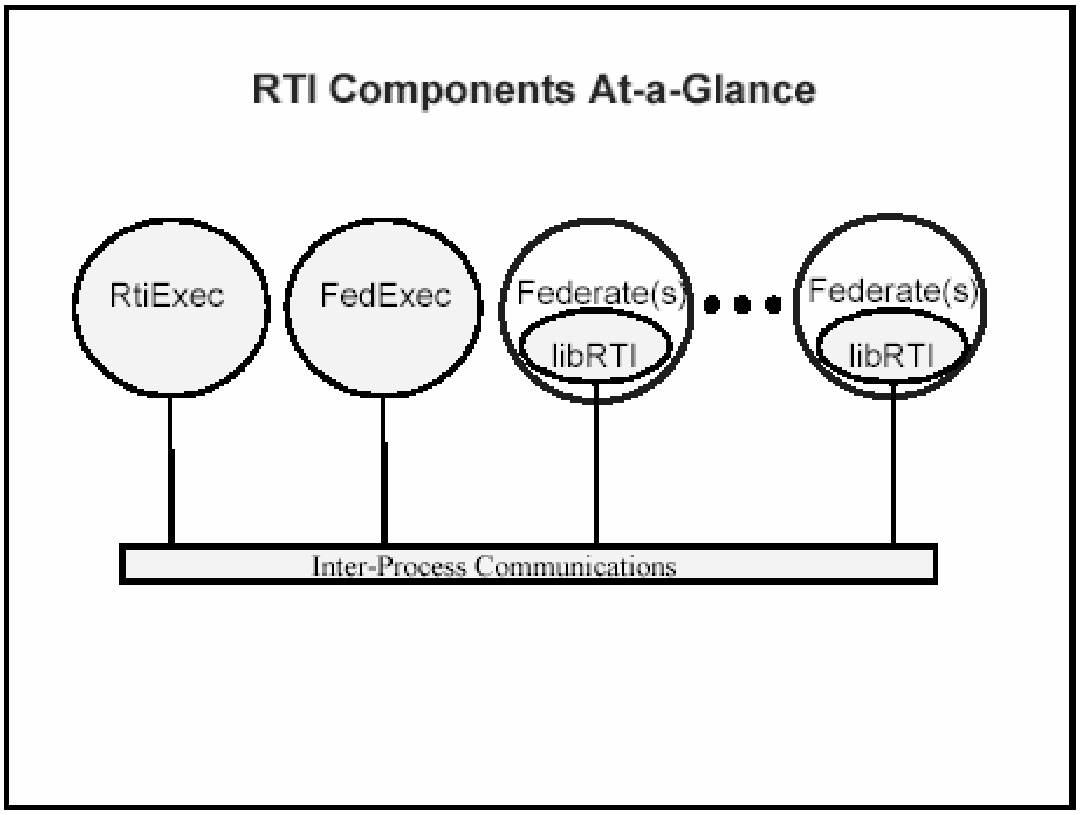


Figure 3.1: View of an HLA Federation[21]

3.1.1 RtiExec

The RtiExec is a globally known process. Each federate communicates with RtiExec to initialize RTI components. The RtiExec's primary purpose is to manage creation and destruction of FedExecs for each federation and also manage multiple federation runs in the same network.

3.1.2 FedExec

When the first federate connects to RtiExec process, a FedExec process that is unique for each federation is created and federation is formed. FedExec manages all the federates' join and resign actions to the federation, and facilitates time synchronization and data exchange between participating federates.

3.1.3 libRTI

libRTI is the C++ library [18][21] provided for developers that implements RTI services specified in the HLA Interface Specification document [19]. Federates includes this library to handle all the communication required for achieving HLA services that take place in a federation through LibRTI. LibRTI provides two classes to the developers namely RTIAmbassador and FederateAmbassador. The services provided by the RTI to the federates are provided under the class RTIAmbassador. RTIAmbassador class implements all the requests (i.e. sign/resign from federation, data exchange, time synchronization etc.) made by a federate through its life cycle in the federation. FederateAmbassador is an abstract class that implements callback functions used for passing RTIs messages to federate. An important point to underline is that FederateAmbassador is an abstract class; developers should implement the internal of the functions declared in FederateAmbassador class.

3.1.4 RTI Management Functions

There are six main classes of operations defined between federates and RTI.

Federation Management: implements support for creating and operating federations. HLA defines join/resign methods for the federates to be used while joining to or resigning from a

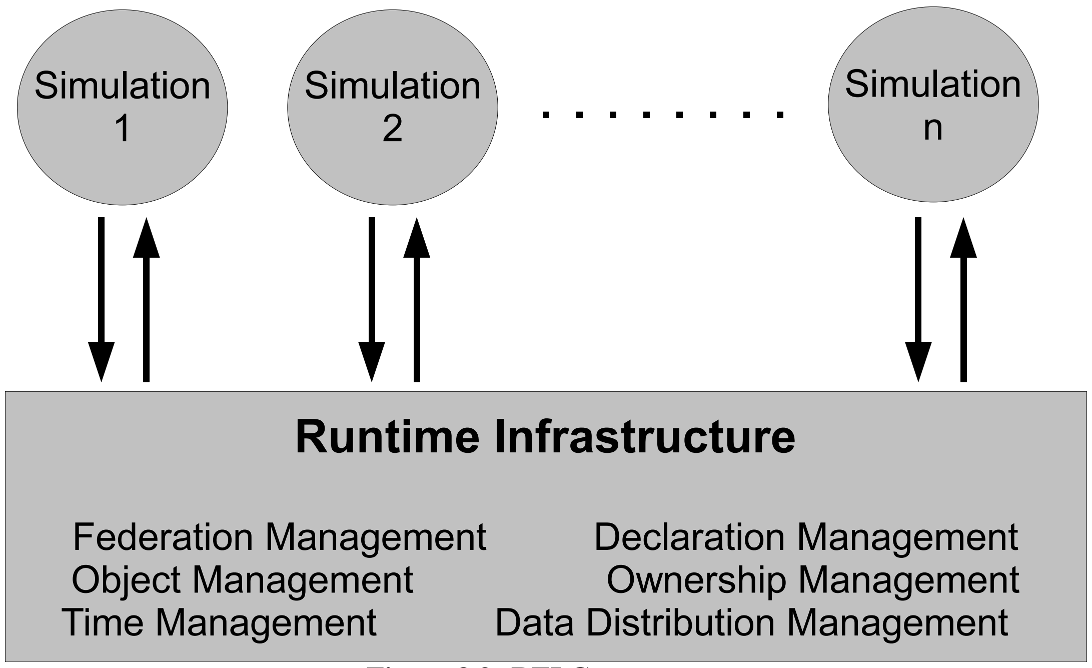


Figure 3.2: RTI Components

available federation. Also methods for creation/destruction of a federation. Federation Management also implements methods for coordination of federation checkpoint (saves) among federates.

Declaration Management: implements services for declaring and registering data to be exchanged between the federates. At any time during the federation execution, a federate can declare it's willing to produce data for an object or generate an interaction of a kind. It can also declare it's interested in getting data for an object or getting interaction of a kind. RTI will handle the data updates or interaction deliveries according to these declarations.

Object Management: implements services for exchanging data and interactions, which are declared by federates using declaration management functions. Using services defined in Object Management, federates register and update their own data objects, send/get interactions to/from other federates, discover and get updates of newly registered object instances. Two types of data exchange mechanism are present in HLA. First, data can be exchanged in terms of objects having attributes. Whenever an attribute change occurs, updateAttributeValues() function is called to inform federates who has registered for data delivery of the object that attribute belongs to. Alternatively data can be exchanged using interactions with parameters. Whenever an event which should be informed to another federate who is in interest occurs, an interaction that represents the event with appropriate parameters is send using sendInteraction() function. Upon receiving the interaction with receiveInteraction() callback function, federates will do the required local processing. The difference between objects and interactions is, objects are persistent where interactions are one timed.

Ownership Management: In HLA architecture, every object instance should be owned by a single federate to guarantee a consistent operation. Only the owner can update the attributes of the object. When needed, Ownership Management functions provide dynamic change of ownership of objects between federates to support updating of the attributes by different federates during the execution.

Data Distribution Management: provides delivery of the correct message to the intended receiver at the correct time. It also provides efficient routing of data among federates by defining regions. Using the information gathered from declaration management functions, Data Distribution Management aims to reduce bandwidth required for data exchange between federates by delivering the information only to the intended federates. For example in a network simulation of wireless network, geographic partitions can be defined and assigned to the federates. As the network nodes can only hear the nodes in the same partition and some of the nodes from the neighboring partitions, each federate may request available data within its own and neighboring partitions. Only object updates or interactions of the objects in the interested regions will be delivered to the federates minimizing the data delivery in the federation.

Time Management: The focus of time management is to implement time management policies and data delivery policy between federates. RTI has support for federates with different time management policies together to form the federation. Time management plays a vital role while running federates concurrently. Time advances must be coordinated with object management services so that information is delivered to each federate in a causally correct and ordered fashion. If time advances between federates are not negotiated correctly, inconsistent simulation results may occur.

Because of its importance, next we will look into time and data delivery mechanisms in more detail.

3.1.5 RTI Time Management and Event Delivering Mechanisms in Detail

It is important to understand time management and message delivering mechanisms of RTI and select appropriate functions for correct operation of federates.

RTI has support for various time management and data delivery services that are supplied to the developers to combine federates with different requirements together. For example, a federate can increase its local time either in coordination with federation or totally freely. It can also either generate or be willing to receive foreign events with or without timestamp information. That is only related with the implementation of the federate. At the beginning of the federation execution, federates declare their interest on how they will increase their local times and how they will sent/receive foreign events.

Time-stamp-ordered (TSO) event is an event that has a time-stamp information and occurs at some time in federation time frame. A federate that generates TSO events is called regulating. As event times are related with federation time, regulating federates should increase their local time in coordination with RTI. A federate need not be regulating if it does not produce TSO events. Events generated by a non-regulating federate are delivered to the other federates in receive-order.

A federate that wants to receive TSO events is called constrained. There are two queues maintained for each constrained federate. The first one stores TSO events, and the second is used to keep non-TSO events. A non TSO event is available to the federate as soon as it enters the queue where TSO events are available to the federate as soon as its timestamp is less than the time requested by the federate. When a federate requests time advance, it is not given time advance grant until all the TSO events before the requested time are delivered.

A federate can be unconstrained if it is not interested in receiving TSO events. Events are delivered to the unconstrained federates in receive order and they are free to increase local time without any constraint.[15][21] A federate can be regulating, constrained, regulating-constrained or neither regulating nor constrained. This depends on the design of the federates.

Regulating and constrained federates advance their local time and data delivery in the supervision of RTI. Not to violate causality, RTI has to give time advance grant to the federates correctly and coordinate in time order data delivery. Federates are given time advance grant by RTI using two parameters, federates local time and lookahead. These parameters should be declared by each regulating federate. Lookahead is defined as the time duration that is promised by the regulating federate, in which no event will be delivered to RTI. Using these parameters, RTI computes a lower bound time stamp (LBTS) for each constrained federate. LBTS is the earliest time stamped event that constrained federate can receive. It is determined by RTI looking at three parameters, the federation-wide available TSO events, regulating federates local times and their lookahead values. A constrained federate can not advance time beyond its LBTS. That is because RTI only guarantees that no TSO event will come until LBTS violating causality. A higher lookahead value results in a higher LBTS. So parallelism increases with the increase of lookahead. But lookahead should be chosen such that no causality violation occurs.

3.1.6 Federated Simulation of a Network Using RTI

To get familiar with the RTI functions which can be used while developing a federated simulation of a network, a two federate, event-based simulation of a wired network was developed. The routes for the individual flows is partitioned between the federates. That is every flow has some part of its route in both federates. So communication between federates should occur when a packet traverses the part of route in the first federate and should continue to the next part of the route in the second federate. This kind of packets, called inter-federate packets, are represented as interactions between federates. In this architecture, as each federate will have some TSO messages to be delivered and received to/from the other one, the federates were designed and developed as regulating and constrained.

RTI is developed to support different kinds of simulations, it has more facilities than needed in this particular example. We need to chose the functions to be used for creating the federation, joining a federation, publishing interaction classes, registering for interaction classes, time management, resigning from the federation and destroying the federation for event based regulating and constrained federates.

Figure 3.3 shows the kernel of the conservative federated simulation developed in this study. The sequence of events in federated simulation over RTI are shown in Figure 3.4.

At the start of operation first federate creates federation execution by calling createFedera-

```
Main()
{
   if (!Federation exists)
   {
      createFederationExecution;
   }
   else
   {
      joinFederationExecution;
   }
   getInteractionClassHandles;
   getParameterHandles;
   subscribeInteractionClasses;
   publishInteractionClasses;
   enableAsynchronousDelivery;
   enableTimeConstrained;
   setLookahead;
   enableTimeRegulation;
   granted_time=0;
   while(!granted_time<end of simulation
   {
       get event from event queue
       if (event_time>granted_time+Lookahead)
       {
           Next_Event_Request(event_time);
           while(granted_time<event_time)
           {
               RTI_Tick();
           }
           Process the event;
           sent_interaction_if_required;
        }
        else
        {
           Process the event
           sent_interaction_if_required;
        }
   }
}
```

Figure 3.3: Kernel of the Conservative Federated Simulation

tionExecution()function. Once the federation is created, other federates joins this execution by calling joinFederationExecution(). Next, federates use publishInteractionClass() and subscribeInteractionClass() functions to declare interests in getting and generating the specified type of interaction. Then RTI is informed about the time management policies of the fed-

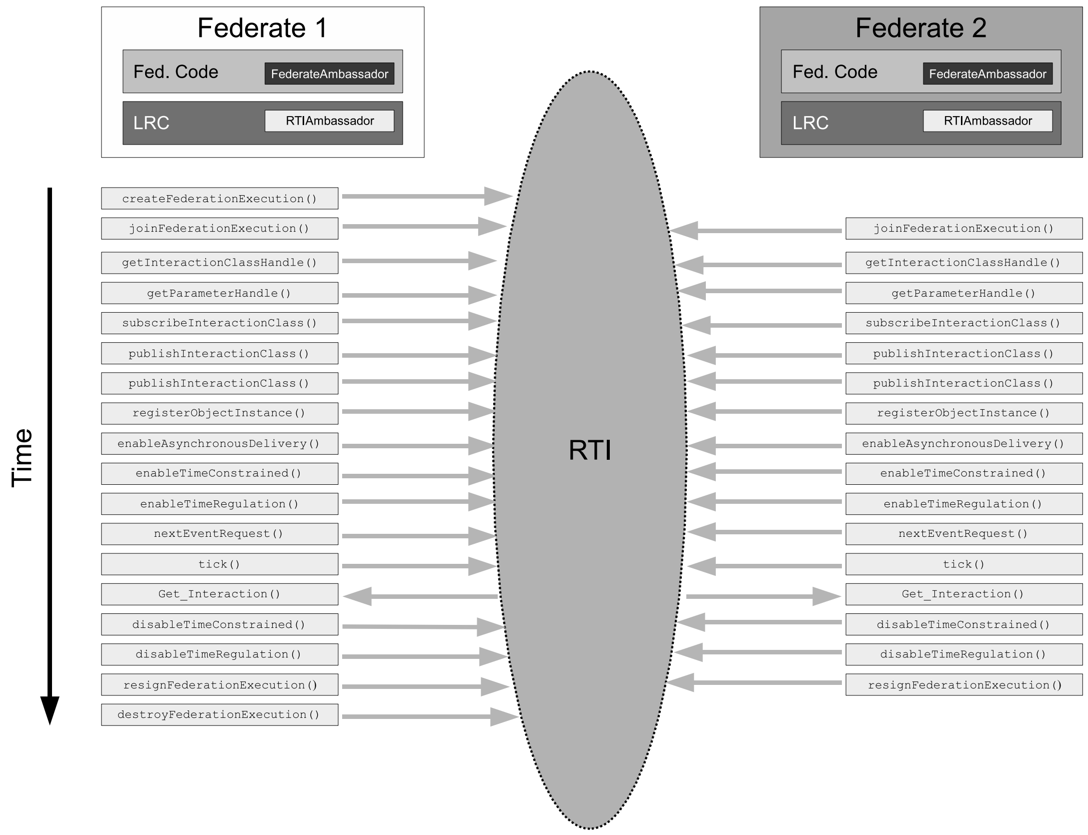


Figure 3.4: Usage of RTI Functions in federate Life Cycle

erates by calling enableTimeConstrained() function, which enables advancement of federate time with federation time and enables delivery of timestamp ordered events to the federate, and by calling enableTimeRegulation() function to informs federation to take federate time information into account while managing federation time. Also lookahead value of the federate is informed by calling setLookahead() method. After these settings made between federate and RTI, until the end of simulation, the event having the earlier timestamp is taken from federate's event queue. If its time is larger than $federate\_time + lookahead$, not to violate causality, federate requests federate time to the next available event time in the federation by calling nextEventRequest(). tick() function is called by federate to supply required time for Local RTI Component(LRC). During tick LRC delivers messages to the federate maintains time and various RTI internal functions. Time advance grant is given when all TSO events less than requested time are released for the federate. Then federate is free to process the local event surely that violation of causality is avoided. During the simulation, if a communication is required between federates, sendInteraction() functions is used to inform RTI about the message. It is RTI‘s responsibility that correct message is delivered to the correct federate at the correct time. When the federate reaches the end of simulation, it calls disableTimeConstrained(), disableTimeRegulation(), resignFederationExecution() methods to end its relation with RTI. The last federate calls destroyFederationExecution() to completely remove the federation execution.

CHAPTER 4

FEDERATED SIMULATION OF NETWORK PERFORMANCE USING PACKET FLOW MODELING

This study deals with the tight synchronization problem of a federated simulation of a large scale network at packet level of detail. A method that loosens federate coupling using flow modeling is proposed. Although the proposal is applicable to both traditional and federated simulation approaches, the scope of this study is limited to conservative federated packet level simulation only.

4.1 Problem Definition

As discussed extensively in chapters 2 and 3, an efficient and simple way of achieving processing and memory resource effectiveness for large scale network simulation is to partition the simulation model in a space-parallel manner and distribute and compute sub models on different computers in a coordinated fashion. This approach has three advantages over sequential simulation:

1. Larger memory can be used than a single board computer. So larger topologies can be simulated.
2. Processing power of more than one CPU can be used to speed up simulation.
3. More interestingly, as event queues are divided into smaller ones, overall processing power required for keeping event queues in time order is reduced compared to maintaining a single event queue.

It is obvious that if the network to be simulated is partitioned into sub-networks, some communicating node pairs will be simulated in different federates. A simple two-federate network simulation configuration is depicted in Figure 4.1. Inter-federate packets should be communicated through RTI. These packets can be represented by interactions. So federates will generate TSO interactions to be delivered to other federates and also will receive TSO interactions from others. Hence, federates must be both regulating and constrained. This means, for each inter-federate packet that has $event\_time > (federate\_time + lookahead)$, a time advance request and time advance grant should be issued between RTI and federates [17]. The main problem here is that the frequent exchange of inter-federate packets forces to set a small lookahead value to avoid violation of causality. So intervals between synchronization points decreases, resulting in federates waiting for each other until they reach the requested time, decreasing parallelism. Additively, time advance requests and grants introduce communication overhead between federates and RTI that is handled over network interface, which is quite slow in comparison to processor-memory communication, as well as computational overhead to compute LBTS in RTI. So if we can find a way to avoid frequent inter-federate messages, lookahead value can be set to a higher value and a relaxation can be introduced between federates, decreasing communication and RTI computation overheads.

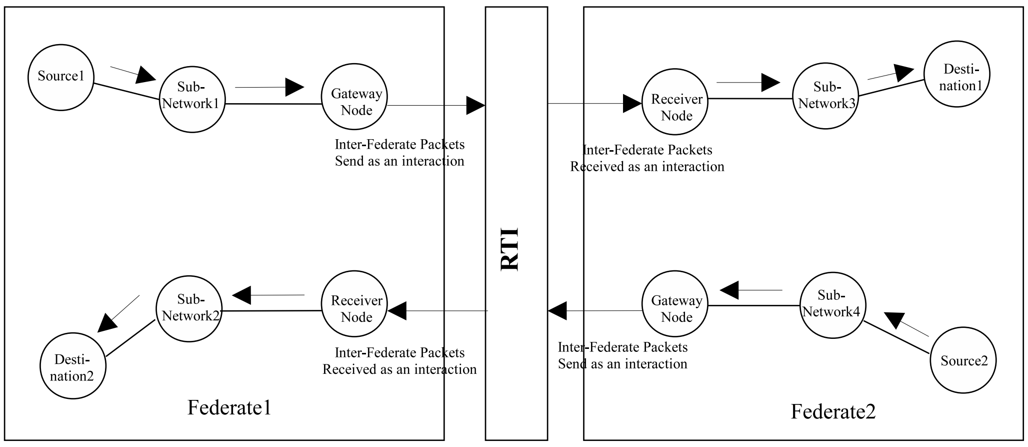

Source1
Sub-Network1
Gateway Node
Inter-Federate Packets Send as an interaction
RTI
Receiver Node
Sub-Network3
Desti-nation1
Inter-Federate Packets Received as an interaction
Sub-Network2
Receiver Node
Desti-nation2
Inter-Federate Packets Received as an interaction
Federate1
Gateway Node
Sub-Network4
Source2
Inter-Federate Packets Send as an interaction
Federate2

Figure 4.1: A Simple Space Parallel Federated Network Simulation

4.2 Time Performance of Federated Network Simulation

First, the performance of a space-parallel federated simulation of a network without flow modeling for UDP-like non-feedback based traffic was investigated. As a representative example, an eight federate network simulation (Figure 4.2) without flow modeling that sends an interaction for every inter-federate packet and its sequential counterpart have been implemented. Packet traffic characteristics obtained with sequential and federated simulation without flow modeling have been compared, as well as elapsed simulation times for different numbers of nodes. Simulated network topology consists of nodes each serving as a source for one flow and destination for another flow. A flow traverses a static route of 8 hops in the source federate and another static 8 hops in the destination federate resulting in 16 hops as in the case of sequential counterpart. A sample path for a flow is given in figure 4.2.

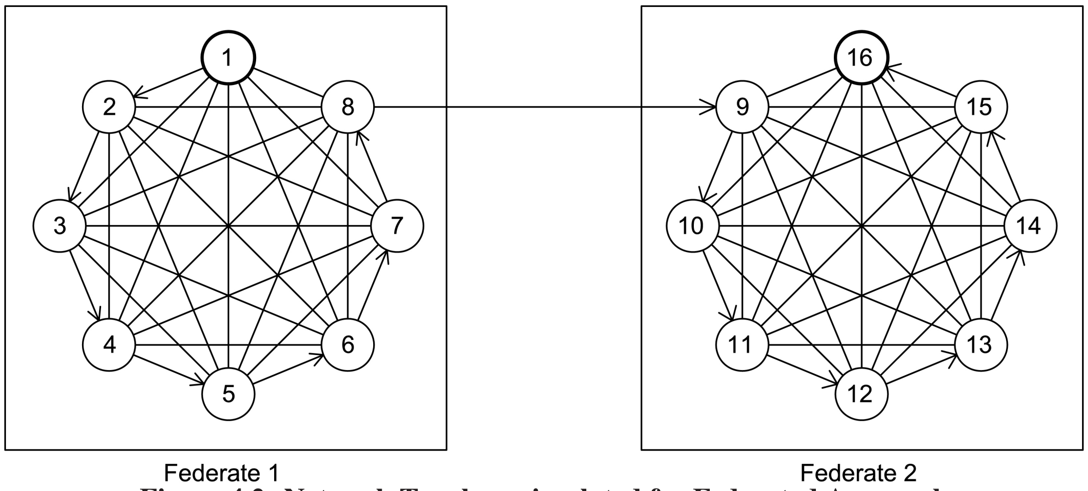


Figure 4.2: Network Topology simulated for Federated Approach

At the source, packets are generated according to exponential distribution having a mean of 200msec. A constant processing and communication delay of 20msec was used at each intermediate node. Lookahead value was chosen to be 10msec. Accuracy results were nearly same as that of sequential simulation as expected. Here a lookahead value of 10msec was chosen carefully so that it will result in fast simulation without decreasing simulation accuracy much. For the simulation of UDP-like non-feedback based traffic, choosing a wrong lookahead value would not have a serious effect on the error of number of packets. But this will have a bigger effect on the error of STD of packet interarrival times. In example, if lookahead value is chosen unacceptably large, the packet interarrival time between some of the two consecutive packets of the same flow will be smaller than lookahead value. So these packets will be delivered to the receiving federate as they were generated at the same time. Here packet interarrival time information is lost, resulting in large error in STD of interarrival packets. Because of the wrong timing of these inter-federate packets, some packet drops may occur that will introduce error on the error of number of packets, but will not be so much.

Results in Table 4.1 show the simulation times required to complete the sequential and federated simulation without flow modeling for the simulation of 900 seconds of the network described above for different number of nodes. Looking at the column corresponding to sequential simulation, computational requirement increases more than linearly with network size, as expected. After 800 nodes it is not feasible to simulate the topology in a sequential way as simulation time becomes prohibitive. The second column presents the time required for federated simulation without flow modeling. As expected, when the network size increases, federated simulation finishes simulation faster than its sequential counterpart.

Table 4.1: Simulation time (sec.) Sequential vs. Federated Simulation

No of. Nodes	Sequential Simulation Time (sec.)	Federated Simulation (without flow modeling) Time (sec.)
80	39	193
200	201	477
400	831	940
600	2015	1809
800	5174	5969
1200	89965	35621
1600	226842	134221
2400	539500	398400

Simulation results also show an interesting phenomenon. Federated simulation without flow modeling, which communicates interactions for each inter-federate packet, is slower than sequential simulation for small network sizes because of the extra cost of RTI communication and synchronization, as expected. When number of nodes increases beyond 400 nodes, it performs better than sequential version showing the advantage of space parallel simulation. After this point on we expect it would perform better than sequential simulation. However, there is a region between 700 and 900 nodes, in which the sequential version performs better again. As the number of inter-federate packets increases with number of nodes, in this region the cost of communicating for each inter-federate packet is increased extremely, resulting in high RTI load and poor parallelism and federated simulation looses its advantage over sequential simulation. Again after 900 nodes, processing power required for keeping event queue in time order dominates and federated simulation without flow modeling performs better than the sequential counterpart, as expected. Although with the increase of number of nodes federated simulation without flow modeling performs better than sequential simulation, the speed up achieved is not even close to the number of CPUs used.

4.3 Proposed Method: Flow Modeling

The simulation results presented in Table 4.1 clearly show that when federates get frequently synchronized, desired speed up can not be achieved even when the number of CPUs are increased. So if one can find a way to decrease the inter-federate communication, this will achieve significant simulation speedups. In many network simulation applications, rather than individual packet behavior, one is more interested in some aggregate metrics, for example, traffic throughput, end-to-end packet delay, packet lost rate, etc. A simulation system only needs to produce these metrics accurately, or with approximations within a satisfactory range, instead of guaranteeing the correct behavior of each individual packet. With this view of network simulation, a distributed simulation system can be considered as a loosely coupled distributed computing system. For flow modeling approach, each federate runs separately performing local computation (simulating the domain assigned to it) with all the information of the network it has at that time, to produce local results as accurately as possible. Federates exchange computation results and update network information among them as soon as a predefined number of packets arrive at federate boundaries and flow characteristic information is extracted. Each federate uses this ”fresh” information to update its own computation and information base. In this way, there is no need to synchronize and exchange packet-level data among simulators.

The available flow modeling literature mainly deals with fluid model simulation. ”Genesis: A Scalable Distributed System for Large-scale Parallel Network Simulation” [5] is a study that applies flow modeling to packet level simulation successfully. It was a long-term research by Yu Liu, Boleslaw K. Szymanski and Adnan Saifee at Department of Computer Science, RPI, supported by DARPA and CISCO Systems Inc.

In Genesis, a large network is decomposed into parts and each part is simulated independently from and simultaneously with the others. Each part represents a subnet or a subdomain of the entire network. These parts are connected to each other through edges that represent communication links existing in the simulated network. In addition, the total simulation time was partitioned into separate simulation time intervals selected in such a way that the traffic characteristics change slowly during most of the time intervals. Each domain is simulated by a separate simulator which has a full description of the flows whose sources are within the domain. This simulator also needs to simulate and estimate flows whose sources are external to the domain but will be routed to or through the domain. The flow delay and the packet drop rate experienced by the flows outside the domain are simulated by the random delay and probabilistic loss applied to each packet traversing simulator boundary. These values are generated according to the average packet delay as well as observed packet loss frequency communicated to the simulator by its peers at the end of simulation of each time interval. Every domain simulator stops its simulation at pre-defined checkpoints, and exchanges data with all the other domain simulators. In the initial (zero) iteration of the simulation process, each part assumes on its external in-links either no traffic, if this is the first simulated interval, or the traffic defined by the packet delays and drop rate defined in the previous simulation time interval for external domains. Then, each part simulates its internal traffic, and computes the resulting outflow of packets through its out-links. In the subsequent $k > 0$ iteration, the in-flows into each part from the other parts will be generated based on the out-flows measured by each part in the iteration $k-1$. Once the in-flows to each part in iteration k are close enough to their counterparts (i.e. based on some pre-defined metrics (end-to-end packet delay, packet loss rate, etc.) and parameters (e.g. Precision threshold)) in the iteration $k - 1$, the iteration stops and the simulation either progresses to the next simulation time interval or completes execution and produces the final results. Global process collects convergence information from all domain simulators and makes global convergence decisions. If some convergence condition is not satisfied, this process will inform some or all domain simulators to roll back and re-iterate. Those simulators which need to roll back will go back to the last checkpoint and re-simulate the last time interval, however, utilizing the data received during the current checkpoint. When all the domain simulators converge, a global convergence is reached and a global process will inform all the domain simulators to go on to the next time interval.

In Genesis, the convergence for UDP traffic was reported to be achieved in 2 to 3 iterations, while for TCP or mixed UDP/TCP traffic the reported convergence was in 5-10 iterations. Simulation accuracy metric was chosen to be the utilization of the routers, which was reported to deviate about %2 from the sequential counterpart only. No investigation was made in that study about the second order moment (variance) of the utilization. Speedup of ˜18 is achieved comparing to sequential counterpart for the simulation of 1024 nodes for non-feedback based protocol where ˜6 of speedup is achieved for a feedback based protocol using 16 CPUs. Although this method performs well in both accuracy and simulation time manner, it has a major drawback: The need for modifying the kernel of a readily built simulation to make it work in an iterative manner is difficult or sometimes even impossible.

In the present study, we propose a solution to the tight synchronization problem, as depicted in Figure 4.3, that can be applied to any simulation easily without modifying the simulation kernel[42]. The idea is refraining from sending an interaction whenever a packet passes though the federate boundary, and instead, profiling these inter-federate packets for each flow individually and sending their characteristics to the receiving federate. Whenever a predefined number of packets arrive at the flow boundary, new characteristics are computed and sent to the receiving federate. At the receiving federate side, packets similar to the original ones will be generated using these characteristics, which is valid for this particular number of packets. This can be thought as an abstraction like using the dead reckoning model for object movement modeling [6]. This approach highly reduces the synchronization and communication required between federates and also gives opportunity to set lookahead to a larger value which increases parallelism. Here we should mention that in a network simulation, packets can belong to two classes: The first class of packets contain important content that affects network traffic, like protocol control information, while the second class of packets carry less important payload like data. Flow modeling is only meaningful for data packets whose content is not important on the protocol's operation. Packets that carry control information should be processed individually[42].

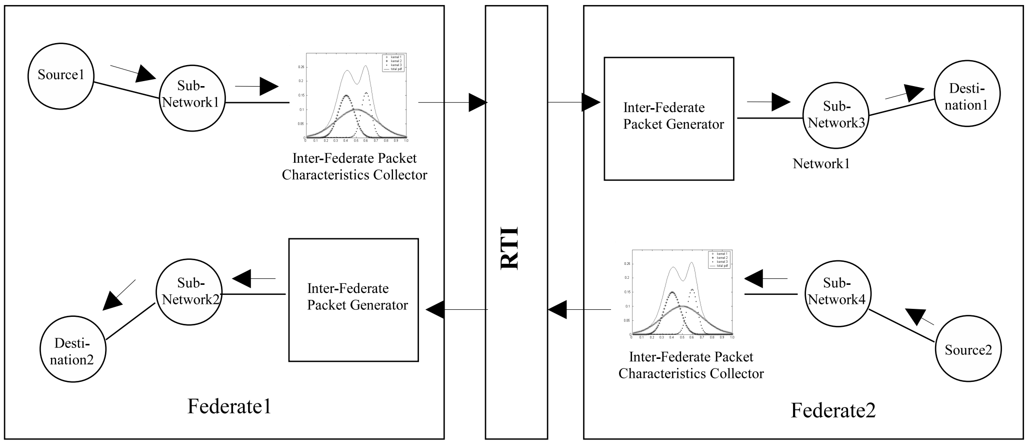

Source1
Sub-Network1
Inter-Federate Packet Characteristics Collector
Sub-Network2
Desti-nation2
Inter-Federate Packet Generator
Federate1
RTI
Inter-Federate Packet Generator
Sub-Network3
Desti-nation1
Network1
Inter-Federate Packet Characteristics Collector
Sub-Network4
Source2
Federate2

Figure 4.3: Proposed Abstraction Method

There may be different methods for capturing the characteristics of the packet interarrivals at the egress federate boundary. In the sections 4.3.1-4.3.3 three possible methods will be investigated and their accuracy and time performance will be discussed.

Accuracy performance of the methods will be tested for 4 different cases where Exponential and Pareto traffic is simulated with intermediate routers having unlimited/limited buffer space. Exponential distribution was chosen for its smooth nature while Pareto distribution was chosen as a representative of bursty traffic. The characteristic of Exponential distribution is controlled simply by its mean m. It has a variance $m * m$. Pareto distribution has two parameters, location parameter x_m and shape parameter k. Mean and variance of the random numbers generated by Pareto distribution can be modified using these parameters. Mean of Pareto distribution is formulated as $\frac{k*x_m}{k-1}$ for $k > 1$ where variance is formulated as $\frac{x_m^2*k}{(k-1)^2*(k-2)}$ If $k < 2$, the variance is infinite.

There is always a trade of between speed and accuracy. In this study, accuracy is investigated for a two federate simulation that uses the topology presented in Figure 4.2, having 20 nodes in each federate, with a total of 40 flows. In this study accuracy was operationally investigated in terms of per flow error of number of packets received during the simulation (directly yielding mean interarrival time) and the error of STD of packet interarrivals compared to the sequential simulation. Per flow error of number of packets received during the simulation (directly yielding mean interarrival time) and the error of STD of packet interarrivals compared to the sequential simulation were the investigated indicators of accuracy. Besides the error of number of packets, the error of packet interarrival STD should also be investigated as real time traffics like voice and video streaming strictly depends on the link delay and delay variances. The acceptable error on the simulation accuracy may change with the traffic under simulation. As an example, for the simulation of a file transfer more error on STD of packet interarrivals can be tolerated comparing with the case of a video streaming.

One thing to mention here without going any further on the concept of simulation with packet times that are generated using a random number generator is the need for repeating the simulation with different seeds. That is because actually random number generators are not fully random. Random numbers are known series generated using a seed that repeats itself after a point. Simulation results may vary according to random series generated by different seeds. So simulations should be repeated with different seeds until simulation results are stabilized. In our case, as the flows and routes are identical and equally distributed, results for each flow can be thought as the results obtained by a single simulation run. So instead of repeating the same simulation scenario with different random seeds until a confidence interval is achieved, percent error of flows with the sequential counterpart are found for each flow and the errors for number of packets received and standard deviation (STD) of packets for %95 confidence interval was calculated using the values obtained from single simulation run using the following relationship[38].

$$y_n - \frac{1.96 * s_n}{\sqrt{n}} < \mu < y_n + \frac{1.96 * s_n}{\sqrt{n}} \tag{4.1}$$

where μ is the real average of the samples, y_n is the mean and s_n the standard deviation of n samples for $n > 30$

4.3.1 Packets modeled by simply a mean

The simplest form of flow modeling can be done using the first moment of the packet interarrival times. At the federate boundary, mean of the interarrival times of inter-federate packets for each individual flow can be calculated and at the receiver side, packets with constant interarrival times can be generated using this parameter. New parameters for each flow are exchanged at every new 200 inter-federate packets.

Table 4.2 and Table 4.4 present the simulation results obtained for Exponential and Pareto traffic with unlimited router buffer, that is no packet loss. Looking at the error of number of packets received, it is quite same as the sequential. But that is not case for STD. As the packet interarrivals are averaged and new packets are generated at constant interarrival times using this mean, a smoothing effect on the traffic is seen in the receiver federate. This results in serious deviation from the original flow characteristics so that in the simulations more than %20 error of STD occurred for Exponential case and nearly %10 error of STD for Pareto case, as seen in Table 4.3.

Deviation of flows‘ dynamic characteristics affects only STD error of the received packets at the receiver if the routers have unlimited buffer which is impractical for the realistic networks. But if one limits the router buffers, the importance of the burstiness of the flows increases. That is because bursty traffic can easily fill the router buffers in a short duration and the succeeding packets will be dropped. If a smoothing is applied on the traffic fewer packet drops will occur, affecting the simulation accuracy. This can clearly be seen in the case where bursty Pareto traffic is simulated over routers with limited buffer, as shown in Table 4.5. As the Pareto traffic has bursty nature and router buffers are limited, when a burst of packets arrive, a group of packets are dropped in the sequential simulation. But if inter-federate packets are simply represented by a mean, the resulting smoothened traffic will have smaller drop rate in the receiver federate showing the importance of correct flow characterization on the simulation results. Results show that ˜%10 error occurs for the number of packets simulated in sequential and federated simulations.

Table 4.2: Results for Modeling by a mean, Exponential traffic with unlimited router buffers

Exponential Distribution	Difference with Sequential Simulation
% # of Packets error	$0.44 < \%Error < 0.5$
% Variance error	$26.25 < \%Error < 31.89$

Table 4.3: Results for Modeling by a mean, Exponential traffic with limited router buffers ˜%20 packet drop

Exponential Distribution	Difference with Sequential Simulation
% # of Packets error	$3.73 < \%Error < 4.05$
% Variance error	$22.05 < \%Error < 26.57$

Table 4.4: Results for Modeling by a mean, Pareto traffic with unlimited router buffers

Pareto Distribution	Difference with Sequential Simulation
% # of Packets error	$0.17 < \%Error < 0.31$
% Variance error	$6.7 < \%Error < 9.91$

Table 4.5: Results for Modeling by a mean, Pareto traffic with limited router buffers ˜%30 packet drop

Pareto Distribution	Difference with Sequential Simulation
% # of Packets error	$9.58 < \%Error < 9.92$
% Variance error	$4.47 < \%Error < 6.9$

To see the need for better flow modeling, consider a new scenario where routers in the sender federate have unlimited memory and routers in the receiver federate have limited router memory. Traffic type was Pareto. Table 4.6 shows the error results obtained with federated simulation with flow modeling using flow means and sequential simulation. It is seen that in this case, the % error of the number of packets and % error of STD of packets are unacceptably high. This is because, if routers in the first federate were limited, packet drops in the first federate would smooth inter-federate packets, so fewer packet drops would occur in the second federate. But if the routers in first federate have unlimited buffer capacity, inter-federate packets would have the bursty nature of Pareto and more packet drops would occur in the second federate. So the overall error is larger for this special case, showing the importance of accurate flow modeling.

Last thing to mention before going the next section, is that STD errors of limited router buffer

Table 4.6: Results for Modeling by a mean, Pareto traffic with federate 1 unlimited federate2 limited router buffers

Pareto Distribution	Difference with Sequential Simulation
% # of Packets error	$18.65 < \%Error < 20.43$
% Variance error	$33.93 < \%Error < 45.92$

cases for both exponential and Pareto traffic are less than those of unlimited router buffer cases. This again results from the shaping of inter-federate packet traffics by packet drops in the sender federate having a smoothing effect. But if no drop occurs in the sender federate, inter-federate packets will have exponential or Pareto characteristics which differ significantly from constant packet generation.

4.3.2 Packets modeled according to a specific distribution

A better way of flow modeling can be estimating parameters of the flow if the probability distribution type of the flow is known (i.e. Exponential/Pareto). This method enables to characterize the packets more precisely than modeling by a mean. We will investigate the pros and cons of the approach for flows whose packets are generated according to Exponential and Pareto distributions.

Scenarios simulated in part 4.3.1 were repeated and compared with their sequential counterparts. Exponential inter-federate packets were modeled by calculating interarrival means of packets for each flow. Calculated mean is sent for every new 200 packets. At the receiver federate using this mean, Exponentially distributed packets similar to the original ones are generated. Both unlimited router buffer and limited router buffer cases were simulated. Results are presented in Table 4.7 and Table 4.8.

Table 4.7: Exponential modeling results for Exponential traffic with unlimited router buffers

Exponential Distribution	Difference with Sequential Simulation
% # of Packets error	$0.145 < \%Error < 0.25$
% Variance error	$9.6 < \%Error < 13.29$

Table 4.8: Exponential modeling results for exponential traffic with limited router buffers ˜%26 packet drop

Exponential Distribution	Difference with Sequential Simulation
% # of Packets error	$4.56 < \%Error < 4.9$
% Variance error	$17.55 < \%Error < 22.31$

From the Table 4.7 it is seen that if router buffers are not limited, Exponential modeling gives very close results to the sequential one. There exists an error on the STD. of packets that mainly originates from intermediate silences between two successive computations and transmissions of characteristics. Intermediate silences occur between two characteristics sent if the mean of the inter-federate packet is calculated a bit smaller than the actual value. This error will sum up for 200 packet generations and results in STD error.

Results shown in Table 4.8 show that if router buffers are limited and ˜%26 drop occurs, the packet interarrival characteristics at the boundary of the federate differ from exponential distribution due to packet drops. So the model accuracy reduces and simulation results differ from sequential simulation more than the error achieved for unlimited router memory case.

Next the packet generation distribution was changed to Pareto and approximation is tried on packets generated by using Pareto distribution with parameter of location $x_m = 60$ and parameter of shape $k = 1.5$.At the boundary of the federates, Pareto parameters of the flows are calculated for every 200 new inter-federate packets and at the receiving federate similar packets to the original ones are generated using this x_m and k parameters. Equations 4.2-4.3 shows algorithm for estimating x_m and k.

$$x_m(n) = min(x_m(n-1), sample[n]) \tag{4.2}$$

$$k = \frac{(n)}{\sum (\ln(sample[n]) - \ln(x_m))} \tag{4.3}$$

From the Table 4.9 it is seen that if router buffers are not limited Pareto modeling gives very close results to the sequential one for both number of packets generated and STD. of packet interarrivals. So flow modeling using Pareto estimation works accurately when router buffers are not limited.

Table 4.9: Pareto modeling results for Pareto traffic with unlimited router buffers

Pareto Distribution	Difference with Sequential Simulation
% # of Packets error	$0.18 < \%Error < 0.29$
% Variance error	$4.52 < \%Error < 7.07$

But if router buffers are limited and ˜%24.5 drop occurs, Table 4.10 shows that, the packet interarrival characteristics at the boundary of the federate differ much from the Pareto distribution due to packet drops, decreasing model efficiency. Both % Error of number of packets and STD of packet interarrivals are unacceptably high making Pareto approximation useless for such a scenario.

Table 4.10: Pareto modeling results for Pareto traffic with limited router buffers ˜%24.5 packet drop

Pareto Distribution	Difference with Sequential Simulation
% # of Packets error	$11.58 < \%Error < 13.5$
% Variance error	$36.96 < \%Error < 48.04$

From the results presented in 4.3.1 and 4.3.2, we can conclude that modeling inter-federate packets by a mean or trying to fit them to a specific distribution is not appropriate to be used as they can not give enough accuracy for some simulation scenarios. A better modeling approach that would give accurate results regardless of the scenario is required that flow modeling method can be applicable practically.

4.3.3 General Packet modeling

Given a packet stream for training purposes, hereafter called the training set, one can find parameters of the distribution from which samples are generated if the distribution is well-known like Exponential, Pareto etc. as discussed in 4.3.2. But because of random packet service times and packet drops due to insufficient node buffers, traffic that is generated at the source of the flow using a specific distribution differs much at the destination. Also different applications (Video on demand, VOIP, Telnet, FTP, surfing on the Internet, etc.) display different traffic characteristics that should be modeled and approximated individually. So a way of accurately fitting inter-federate packets to a general distribution for individual flow should be found so that flow modeling approach can be applicable for any type of traffic for any network configuration.

A popular method of fitting a set of data to a general probability density function (PDF) is the Gaussian Mixtures Algorithm (GMA)[31][34][37]. GMA represents a general PDF as a weighted sum of positive Gaussian-shaped functions called modes or kernels, each with a different mean, variance and weight and tries to find appropriate values for these variables that represent the training set. An illustration of how a PDF is represented by GMA is shown in Equation 4.4 and Figure 4.5.

$$f(x) = \sum_{i=1}^{n} a_i f_i(x) \; 0 \leq a_i \leq 1 \qquad (4.4)$$

where $a_1 + \cdots + a_n = 1$ and $f_i(x) = \frac{1}{\sigma_i \sqrt{2\pi}} exp(\frac{-(x-\mu_i)^2}{2\sigma_i^2})$
One can consider this approach as a classification problem where each sample in the training set is assumed to be generated from a kernel in the mixture. The problem is to find the correct number of the kernels which neither overestimates nor underestimates the training set and to estimate their parameters that best fit the training set.

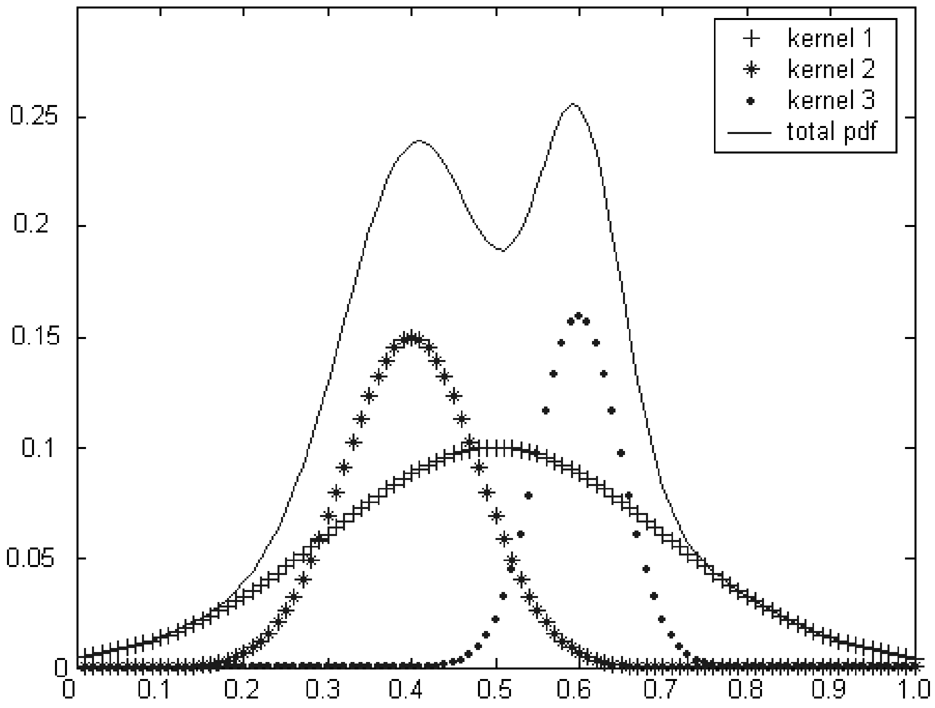


Figure 4.4: Illustration of GMA

GMA only presents the concept of the formulation of a general PDF in terms of Gaussian functions, so one should find a way to solve for GMA parameters. Below, two possible methods, namely, Expectation Maximization and Self Organizing Mixture Networks, for GMA parameter estimation will be presented and compared in terms of time and accuracy performance.

4.3.3.1 Expectation Maximization Method

The most commonly used method for finding GMA parameters from a training set is Expectation Maximization (EM) method [34][37]. As its name implies, EM is a two phase algorithm. The expectation step computes the average of the log likelihood function using current estimated (or initial) model parameters, while the maximization step maximizes this likelihood with respect to each learning parameter by updating them according to the new estimates. Iterations of an Expectation step followed by a Maximization step are made until a convergence is achieved. Figure 4.5 presents the EM algorithm.

---

1. Compute data weights for i=1,....,L:

$$\omega_{i,k} = \frac{\alpha_i N(z_k,\mu_i \sum_i)\gamma_k}{\sum_{i=1}^{L} \alpha_i N(z_k,\mu_i \sum_i)}$$

2. For i=1,...,L Let:

$$\alpha_i^{'} = \sum_{k=1}^{N} \omega_{i,k}$$

3. Update means. For i=1,...,L

$$\mu_i = \frac{1}{\alpha_i^{'}} \sum_{k=1}^{N} \omega_{i,k} z_k$$

4. Update covariances. For i=1,...,L

$$\sum_i = \frac{1}{\alpha_i^{'}} \sum_{k=1}^{N} \omega_{i,k}(z_k - \mu_i)(z_k - \mu_i)^{'}$$

5. Update mode weights: For i=1,...,L:

$$\alpha_i = \frac{\alpha_i^{'}}{\sum_{k=1}^{N} \gamma_k}$$

Figure 4.5: EM algorithm for GMA

EM algorithm was implemented and its time and accuracy performance was investigated over training sets of Exponential and Pareto numbers. Simulation results presented in Tables 4.11 and 4.12 show that that EM algorithm fits both distributions quite well.

Table 4.11: Accuracy and time results for GMA with EM modeling for Exponential Distribution

Exponential Distribution	GMA with EM
% Mean error	0,42
% Variance error	1,63

The main problem with EM algorithm was seen to be its slow convergence and required memory space especially when number of kernels or number of samples are large. Also

Table 4.12: Accuracy and time results for GMA with EM modeling for Pareto Distribution

Exponential Distribution	GMA with EM
% Mean error	1,4
% Variance error	1,97

there is no well defined procedure for finding number of kernels to be used and their initial parameters. Some proposed methods[34] insert a new kernel when needed or eliminate the unnecessary kernels dynamically, but they are more complex to implement. Furthermore, it would be unduly hard to coordinate exchanges between federates if the number of kernels are variable.

4.3.3.2 Self Organizing Mixture Networks Method

The problems related with EM method have motivated us to study new algorithms for GMA solution. Self-organizing mixture network (SOMN), which is a variant of classical self organizing maps, can be an alternative method for learning arbitrary density functions using GMA [31][32][35]. The SOMN structure is illustrated in Fig. 4.6. For a mixture of finite components, the network places nodes (kernels) which do not need to be exactly equal to the underlying components, if is not known a priori in the input space. The kernel parameters, e.g., mean vectors, and covariance matrices are the learning parameters. The output of a kernel is the conditional density of that component in the mixture. The upper layer, or the network output, sums the responses of these kernels weighted by the prior probabilities or mixing parameters.

At each time step, a sample point is randomly taken from a finite data set. Conditional probabilities of each kernel are calculated. A winner kernel, from which the sample seems to be generated from, is chosen according to its conditional probability output multiplied by its mixing parameter. Within a neighborhood of the winner, the kernel parameters are updated. The pseudo code for SOMN is given in Figure 4.7.

For SOMN, the number of nodes, however, needs not to be known a priori, but should be equal to or larger than the number of underlying components in the mixture to avoid the under-representation problem. That is, one can always use a large number of nodes to learn the mixture, and only the significant ones will remain. For an accurate estimation of an arbitrary

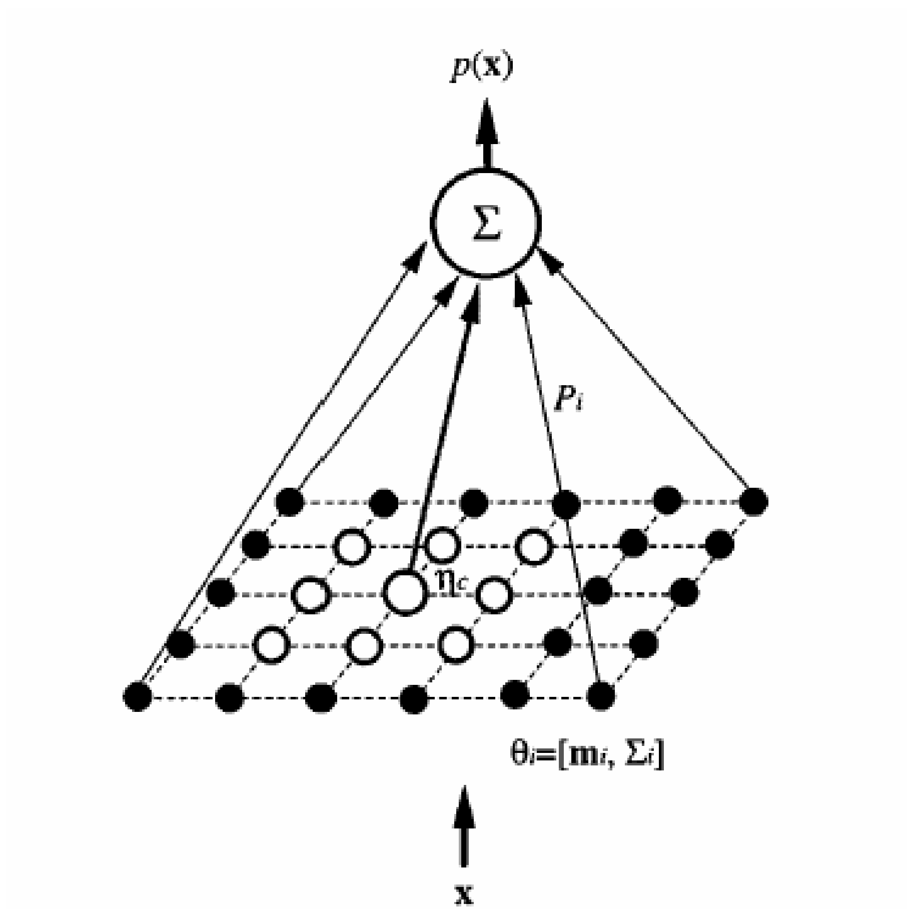


Figure 4.6: The structure of the self-organizing mixture network (SOMN).[31]

density, operation can be started with a large number of nodes. Only significant ones will have affect in the mixture while others will diminish having nearly zero mixing parameter.

SOMN algorithm with inverse learning rate methodology was implemented and its time and accuracy performance was investigated over training sets of Exponential and Pareto numbers. Simulation results presented in Tables 4.13 and 4.14 show that that SOMN algorithm fits both distributions quite well.

Table 4.13: Accuracy and time results for GMA with SOMN modeling for Exponential Distribution

Exponential Distribution	SOMN
% Mean error	1,19
% Variance error	0,62

```
1. Init the kernels randomly

2. While $\alpha_k > 0.001$ repeat steps 3-8.

3. Calculate learning rate using $\alpha_k = \frac{C\alpha_{initial}}{C+k}$

4. Choose a sample from dataset randomly

5. Calculate conditional probability of the sample for each kernel.

6. Find the best matching unit having the biggest conditional probability.

7. Update best matching unit and its two near neighbors according to the formulas given below

    $\Delta_{\mu_i} = P_i\alpha_k(sample - \mu_i)$
    $\Delta_{\sigma_i} = P_i\alpha_k((sample - \mu_i)^2 - \sigma_i)$
    $\mu_i = \mu_i + \Delta_{\mu_i}$
    $\sigma_i = \sigma_i + \Delta_{\sigma_i}$

8. Update kernel‘s probability using formula

$a_i = a_i + \alpha_k * (P_i - a_i)$
where $a_i$ stands for mixture weight $\mu_i$ stands for mean $\psi$ stands for variance of the kernel
```

Figure 4.7: Pseudo Code for SOMN.

Table 4.14: Accuracy and time results for GMA with SOMN modeling for Pareto Distribution

Pareto Distribution	SOMN
% Mean error	4,9
% Variance error	9,44

4.3.3.3 Comparison of SOMN and EM Algorithms

Although both algorithms can achieve similar final results, it is found that the SOMN algorithm has three advantages over the EM algorithm: Firstly, the stochastic-gradient-based SOMN algorithm converges much faster than the deterministic gradient-based EM (batch) algorithm. Second, it is observed that the initial conditions have a greater influence on the convergence of the EM algorithm. The SOMN, however, is more robust when random initial values are used. Third, the EM algorithm can be trapped in local optima as it is an exact gradient ascent/descent algorithm. The SOMN, being a stochastic gradient based algorithm, can escape shallow local minima. In SOMN, learning is limited to a small neighborhood of the winner only rather than to the entire network as the EM does - this is computationally advantageous especially when the number of kernels and/or size of training set are large. Tables 4.15-4.16 show the performance of 100 runs for both SOMN and EM where densities are Exponential with mean 200 and Pareto with $x_m = 60, k = 1.5$.It is seen that SOMN with inversely decreasing learning rate performs faster than EM in the order of 10 for estimating Exponential and in the order of 4 for estimating Pareto. Although accuracy results of SOMN with inversely decreasing learning rate for estimating Exponential distribution are quite acceptable, higher error rates achieved for estimating Pareto distribution give uncomfortable feeling with the method. That is if an error of ˜%5 for mean error and ˜%10 for variance error are incurred at the stage of modeling, summing up with other assumptions and inaccuracies, one will end up with larger final errors.

Table 4.15: Performance comparison of SOMN and EM for Exponential Distribution

Exponential Distribution	SOMN	EM
% Mean error	1,19	0,42
% Variance error	0,62	1,63
Time	2,8 sec	33,5 sec

Table 4.16: Performance comparison of SOMN and EM for Pareto Distribution

Exponential Distribution	SOMN	EM
% Mean error	4,9	1,4
% Variance error	9,44	1,97
Time	2,8 sec	12,6 sec

4.3.3.4 A Fast and Accurate Learning Algorithm Proposed for SOMN Training

Learning rate (α_k),has significant effect on the convergence of SOMN [32]. There are different learning parameter methods proposed in the literature for SOM training. The most widely used learning rate parameters are

$$\text{Exponentially decreasing learning rate:}\alpha_k = \alpha_{initial}\left(\frac{\alpha_{final}}{\alpha_{initial}}\right)^{\frac{k}{k_{max}}}$$

$$\text{Inversely decreasing learning rate:}\alpha_k = \frac{C\alpha_{initial}}{C+k}$$

$$\text{Linearly decreasing learning rate:}\alpha_k = \alpha_{initial}\frac{k_{max}-k}{k_{max}}$$

etc.

For the usual methods there is no well defined method for choosing $\alpha_{initial}$, α_{final} and number of iterations which effect convergence time and accuracy significantly. In the present study a novel learning rate and stop condition for SOMN training that significantly affects learning accuracy and speed is proposed. Unlike previous approaches, from the experience on trials to find a better learning rate methodology, it is seen that SOMN performs better if mean and variance of the kernels are trained with different learning rates. Two learning parameters α_{mean} for learning mean parameter of the kernels and $\alpha_{variance}$ for learning variance parameter of the kernels are defined. The proposed learning method, shown in Equation 4.5 and Equation 4.6, associates the shape for α_{mean} with the variance of the mixture mean error and $\alpha_{variance}$ with the variance of the mixture variance error. The idea was if the variance of mixture mean is high, this means kernels are far away from the training set, large values of α_{mean} is used for quickly bring the kernels to a better location. Conversely, if mixture mean is stabilized, i.e. smaller mean error variance, smaller α_{mean} should be applied to achieve fine tuning. Same approach is also used for $\alpha_{variance}$. Stop condition for training is reached when mean error and variance error is below a predefined threshold.

Calculation for α_{mean} and $\alpha_{variance}$ at step k.

$$\alpha_{mean}(k) = \begin{cases} 0.5 & \text{if } \frac{mean\_error\_variance}{k} > 0.5; \\ \frac{mean\_error\_variance}{k} & \text{otherwise .} \end{cases} \tag{4.5}$$

$$\alpha_{variance}(k) = \begin{cases} 0.01 & \text{if } \frac{variance\_error\_variance}{k} > 0.01; \\ \frac{variance\_error\_variance}{k} & \text{otherwise .} \end{cases} \tag{4.6}$$

At the start of the training, mean and variance of the set is computed. At each step, mean and variance of the trained Gaussian Mixture is computed according to the formulas given in Figure 4.8.

```
for(int s=0;s<no_modes;s++)
{
    mean=mean+mode[s].weight*mode[s].mean;
    variance_temp=variance_temp + mode[s].weight * ( mode[s].variance
    + mode[s].mean * mode[s].mean );
}
variance=variance_temp-mean*mean;
```

Figure 4.8: Computation of Mean and Variance for a Gaussian Mixture

For the investigation of accuracy and time performance of proposed learning rate algorithm and to compare them with the EM method, the experiments in part 4.3.4.3 were repeated. Results are presented in Tables 4.17-4.19 for Exponential, Pareto and Gaussian Distributions.

Table 4.17: Performance comparison of SOMN and EM for Exponential Distribution

Exponential Distribution	SOMN with proposed approach	EM
% Mean error	0,2	0,42
% Variance error	0,8	1,63
Time	4,1 sec	33,5 sec

Table 4.18: Performance comparison of SOMN and EM for Pareto Distribution

Pareto Distribution	SOMN with proposed approach	EM
% Mean error	2,5	1,4
% Variance error	1,9	1,97
Time	3,5 sec	12,6 sec

Table 4.19: Performance comparison of SOMN and EM for Gaussian Distribution

Gaussian Distribution	SOMN with proposed approach	EM
% Mean error	2,9	6,1
% Variance error	2,5	28,4
Time	24,2 sec	43,9 sec

With a quick look at the results, it is easily seen that for Exponential and Pareto distribution, proposed learning algorithm can converge to the training sets more accurately and much faster than EM. The Gaussian distribution is much harder as all the GMA will converge to one kernel from which the set is formed, other kernels will be diminished. So it takes longer time to converge for Gaussian Distribution than Exponential or Pareto Distribution. The method proposed here performs better than EM for Gaussian Distribution in both time and accuracy. EM even seems to produce unacceptably inaccurate results for Gaussian Distribution.

At this point, a fast and accurate method for modeling inter-federate flows has been obtained. The next part will present the accuracy and speedup performance of the proposed flow modeling approach implemented with SOMN, using the proposed learning rate methodology for a non-feedback based protocol.

4.3.4 Federated Simulation of Network Using GMA with SOMN for a Non-Feedback Based Protocol

To see the time and accuracy performance of the proposed SOMN training method in a federated network simulation, an 8 federate simulation was developed where inter-federate packets were modeled for each flow individually, independently, with GMA using the proposed learning mechanism. Simulations were done for the same scenario as in part 4.2 and speedup and accuracy results were investigated.

4.3.4.1 Speedup Results

Table 4.20: Simulation time (sec.) Sequential vs. Federated Simulation (flow modeling)

No of. Nodes	Sequential Simulation Time (sec.)	Federated Simulation Time (sec.) (Flow modeling)
80	39	52
200	201	76
400	831	134
600	2015	188
800	5174	264
2400	539500	2487

Results in Table 4.20 show that when the number of nodes is as small as 80, because of extra costs for communication and synchronization of federates, sequential simulation is faster. As network size increases beyond 80, flow modeling with GMA method performs better than sequential simulation. That is because for the sequential case, computational requirement increases more than linearly with network size, as expected. It is seen that in case of a 2400 node network, flow modeling with GMA approach achieves a speedup factor of 217 over sequential simulation which is larger than the number of CPUs used in the simulation. That is because as the event queue that was maintained by the sequential simulation has been divided among 8 federates, overall computation required to keep event list in time order is reduced significantly. Comparing speedup results with that of Federated simulation without flow modeling case, it is seen that flow modeling approach loosens the federates, reducing inter-federate communication, CPU's are better utilized that increases parallelism. Flow modeling approach achieves a speedup level of ˜160 over federated simulation without flow modeling when the no of simulated network nodes is 2400.

4.3.4.2 Accuracy Results Obtained for the Simulation of Flows with Different Traffic and Network Characteristics

In part 4.3.5.1 it is shown that federated simulation with flow modeling can achieve significant speedup results which make it a promising candidate to be applied in distributed simulation. Although in part 4.3.4.4, it was shown that SOMN with the proposed learning method converges fast and accurately for different packet distributions, it is also required to show that this approach works accurately in the context of federated simulation in a time efficient manner. Experiments were done for various network traffic configurations and accuracy results were compared with sequential simulation for the same configurations to show that the flow modeling approach can be used with any type of traffic for lossless as well as lossy networks. As mentioned before, Exponential traffic was chosen for its smooth nature whereas Pareto traffic was chosen for its bursty character. Also, performance of the approach for a mixed traffic like Hyper-exponential, which is a mixture of two exponentials, was investigated. A two-federate simulation, with each federate having 20 nodes was developed and the same network topology that was presented in part 4.2 was used. GMA pool size was 200 and number of GMA modes was 7. GMA parameters were calculated for every 50 new inter-federate packets and sent to the receiving federate. Simulated time interval was 900 sec. The scenarios and corresponding accuracy results are summarized in Tables 4.21 through 4.27:

Table 4.21: GMA modeling accuracy results for Exponential traffic with mean=200, unlimited router buffers

Exponential Distribution	Difference with Sequential Simulation
% Mean error	$0,17 < \%Error < 0,34$
% Variance error	$5,66 < \%Error < 9,41$

Table 4.21 shows that accuracy results for both % error of number of packets and % error of STD of packet interarrivals are quite acceptable. So proposed flow modeling approach was seen to successfully parameterize the inter-federate packets for Exponentially traffic with mean interarrival times of 200msec and intermediate routers having unlimited buffer space.

Table 4.22: GMA modeling results for Exponential traffic with mean=200 limited router buffers ˜%26 packet drop

Exponential Distribution	Difference with Sequential Simulation
% Mean error	$5.24 < \%Error < 7.66$
% Variance error	$4.08 < \%Error < 4.44$

From Table 4.22, it is seen that the accuracy results for both % error of number of packets and

% error of STD of packet interarrivals are quite acceptable for exponential traffic with limited router buffers where ˜%26 packet drop occurs. So GMA modeling successfully modeled a smooth traffic with packet losses.

Table 4.23: Pareto modeling results for Pareto traffic with $x_m = 60$ $k = 1.5$ **unlimited router buffers**

Pareto Distribution	Difference with Sequential Simulation
% Mean error	$0,12 < \%Error < 0,29$
% Variance error	$5,29 < \%Error < 6,68$

In Table 4.23, it is seen that accuracy results for both % error of number of packets and % error of STD of packet interarrivals are quite acceptable. So proposed flow modeling approach was seen to successfully parameterize the inter-federate packets for a bursty traffic like Pareto traffic with $x_m = 60$ and $k = 1.5$ and intermediate routers having unlimited buffer space.

Table 4.24: Pareto modeling results for Pareto traffic with $x_m = 60$ $k = 1.5$ **limited router buffers ˜%24,5 packet drop**

Pareto Distribution	Difference with Sequential Simulation
% Mean error	$5.0 < \%Error < 5.45$
% Variance error	$4.21 < \%Error < 6.79$

Table 4.24 shows that accuracy results for both % error of number of packets and % error of STD of packet interarrivals are quite acceptable for Pareto traffic with limited router buffer case as well, where ˜%24.5 packet drop occurs. So GMA modeling successfully modeled a bursty traffic with packet losses.

In the next experiment, first only the mean of the packets arriving at the boundary of federate 2 from federate 1 were transmitted to federate 2, as the model of the traffic exiting federate 1 and entering federate 2. Table 4.25 shows the discrepancy between the two-federate simulation with this flow approximation and the sequential classical simulation.

Table 4.25: Mean sent results for Pareto traffic with $x_m = 60$ $k = 1.5$**; federate 1 unlimited, federate2 limited router buffers.**

Pareto Distribution	Difference with Sequential Simulation
% Mean error	$1.56 < \%Error < 3.38$
% Variance error	$7.72 < \%Error < 14.8$

In Table 4.6, it is seen that modeling using only the mean gives unacceptably inaccurate results. The same experiment was repeated for proposed flow modeling with GMA. Accuracy results presented in Table 4.25, for both % error of number of packets and % error of STD of packet interarrivals are acceptable for this scenario. This case shows the importance of correct characterization of the inter-federate flows.

Table 4.26: GMA sent results for Hyper-exponential traffic with unlimited router buffers

Hyper-exponential Distribution	Difference with Sequential Simulation
% Mean error	$0.32 < \%Error < 0.52$
% Variance error	$3.04 < \%Error < 5.6$

Table 4.27: GMA sent results for Hyper-exponential traffic with limited router buffers

Hyper-exponential Distribution	Difference with Sequential Simulation
% Mean error	$0.53 < \%Error < 0.93$
% Variance error	$2.55 < \%Error < 4.08$

After the exponential and Pareto traffic experiments, simulation of hyper-exponential traffic was investigated. Accuracy results for both % Error of number of packets and % error of STD of packet interarrivals are quite acceptable. So GMA modeling successfully parameterized the inter-federate packets for a mixed traffic like hyper-exponential where intermediate routers have unlimited buffer space.

The next simulation was with hyper-exponential traffic and limited router buffers, causing packet losses. Accuracy results for both % error of number of packets and % error of STD of packet interarrivals are quite acceptable for hyper-exponential traffic with limited router buffers case where ˜%22 packet drop occurs. So GMA modeling was also successful with a mixed traffic with packet losses.

Results presented in Table 4.21-4.27 show that flow modeling with GMA approach results in acceptable accuracy for different kinds of lossless and lossy traffic.

Next, the effect of Pareto shape parameter, which controls the level of burst, on the accuracy of proposed method was investigated. Simulations were done for $x_m = 60$ and different k values changing between 1.2 to 2.0 with routers having unlimited buffer space. The value $k = 1.2$ was selected as the lower bound. That is because beyond this point, Pareto traffic becomes so bursty that it generates some large interarrival times for the packets that are even larger than simulation end time, so some of the flows would have no packets to be simulated. $k = 1.8$ was selected as the higher bound. This is because beyond these values of k, traffic becomes smoother, a case which has already been sufficiently explored.

Simulation results presented in Table 4.28 show that the proposed flow modeling approach works accurately for smooth traffics as in the case of $k = 1.8$ as well as for highly bursty traffic as in the case of $k = 1.2$.

Table 4.28: Error vs. Pareto Shape Parameter

k	% Error of # of Packets	% Error STD. Of Packet Interarrivals
1.2	$0.29 < \%Error < 0.568$	$4.35 < \%Error < 7.56$
1.3	$0.18 < \%Error < 0.31$	$5.5 < \%Error < 8.78$
1.4	$0.21 < \%Error < 0.32$	$5.79 < \%Error < 8.52$
1.5	$0.13 < \%Error < 0.23$	$6.1 < \%Error < 9.67$
1.6	$0.127 < \%Error < 0.22$	$4.82 < \%Error < 7.71$
1.7	$0.11 < \%Error < 0.18$	$5.53 < \%Error < 8.25$
1.8	$0.09 < \%Error < 0.16$	$6.15 < \%Error < 9.83$

Using the results presented in tables 4.20 to 4.28, the conclusion can be reached that, flow modeling with GMA solved with the proposed learning rate methodology provides accurate modeling for different kinds of flows under different network conditions.

4.3.4.3 Flow Modeling Pool Size

To investigate the effect of GMA pool size on the simulation accuracy and time, some simulations were done for Pareto traffic with $x_m = 60$ and $k = 1.5$ where routers have unlimited buffers. The effect of pool size on the accuracy can best be observed using a bursty traffic like Pareto. That is because bursty traffic will show different characteristics that change with the size of the window under view, where a smooth traffic would show similar characteristics for the windows having different size. For the simulations, GMA parameters are exchanged between federates for every new set of packet arrivals that fill up the selected pool size. A common lookahead value was chosen to be 10msec for all cases. Table 4.29 presents accuracy results and simulation times vs. pool size.

Table 4.29: Error vs. GMA Pool Size

Pool Size	Simulation Time	% Error of # of Packets	% Error STD. Of Packet Interarrivals
50	1925	$0.12 < \%Error < 0.18$	$2.87 < \%Error < 5.96$
100	1530	$0.18 < \%Error < 0.169$	$4.53 < \%Error < 7.11$
150	1402	$0.21 < \%Error < 0.264$	$4.73 < \%Error < 7.79$
200	1253	$0.13 < \%Error < 0.3$	$5.42 < \%Error < 9.65$

Table 4.29 shows that as the GMA pool size increases, % error of number of packets does not change much, so the convergence accuracy of GMA method does not change much either with pool size. But % error STD of packet interarrivals seems to get worse with increasing pool size. The reason for this behaviour is, when the GMA pool is large, the time between two consecutive characteristics being sent increases. So a small error on the estimation mean

of the GMA with respect to the original flow adds up for larger packets and results in a large error for the first packet of the newly generated window at the receiving federate side. This one large packet error increases the % error STD of packet interarrivals. But as GMA pool size is reduced, this error will end up with smaller value, so % error STD of packet interarrivals will improve.

The last comment is about the change of simulation time with the GMA pool size. Results presented in the second column of Table 4.29 show that smaller the pool size, larger the simulation time. Conversely larger the pool size, smaller the time required to finish the simulation. It is obvious that for small pool size, number of GMA computations held in the federates, the number of messaging and synchronization required between federates and RTI load increases, adding up to a larger simulation time.

4.3.5 Federated Simulation of Network Using GMA with SOMN for a TCP-like Feedback Based Protocol

While introducing the method of flow modeling, it was mentioned that modeling can only be applicable to the data packets whose content is not important or can be modeled. The packets whose content is important or can not be modeled, like packets that carry protocol control information, should be passed between federates as they are. For a UDP-like non-feedback based protocol, every packet can be modeled. For a TCP-like feedback based protocol, modeling can be done for data packets while ACK packets that carry protocol control information can be passed without modeling. In this section, validity of the proposed flow modeling approach and its accuracy and time performance will be investigated for a feedback based protocol. A TCP-like protocol, that implements flow and bandwidth control according to the ACK packets received, is simulated. This TCP-like protocol is summarized below:

1. At the sender, data packets are generated at constant inter-arrival times until Congestion Window Size (CWND) is reached. Then a timeout event is registered at $(last\_sent\_packet\_time + 2 * Round\_Trip\_Time(RTT))$.

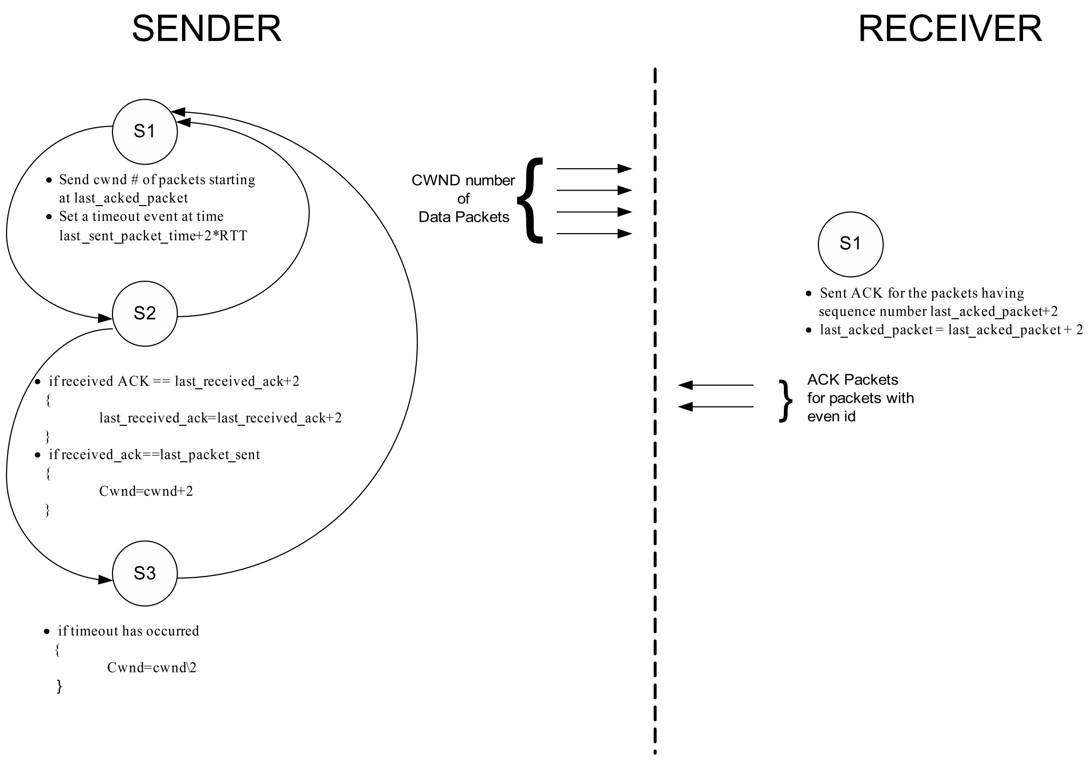
2. At the receiver side an acknowledgement (ACK) is sent for every two packets received.

3. At the sender, if an ACK is received for all the packets until the last_packet_sent, CWND is increased by two and a set of CWND new number of data packets is sent. CWND

can be increased up to 64.

4. But if a timeout has occurred, CWND is halved and a set of CWND new packets is sent packet_id starting from last_received_ack.

The state diagram for the sequential simulation is presented in Figure 4.9.



SENDER
RECEIVER
S1
• Send cwnd # of packets starting at last_acked_packet
• Set a timeout event at time last_sent_packet_time+2*RTT
S2
• if received ACK == last_received_ack+2 { last_received_ack=last_received_ack+2 }
• if received_ack==last_packet_sent { Cwnd=cwnd+2 }
S3
• if timeout has occurred { Cwnd=cwnd\2 }
CWND number of Data Packets
S1
• Sent ACK for the packets having sequence number last_acked_packet+2
• last_acked_packet = last_acked_packet + 2
ACK Packets for packets with even id

Figure 4.9: State Diagram of TCP-LIKE protocol for Sequential Simulation

Looking at the diagram for the sequential case, it is easy to implement such a protocol for sequential simulation. But as data packets and ACK packets carry packet_id information, which is required for flow control, some extra information exchange should be introduced for the correct operation of the protocol in a federated simulation.

For each flow, flow modeling is applied at the sender federate boundary to the CWND number of packets, which is variable in time. Then an interaction with the GMA parameters, the start_id of packets to be generated and CWND is exchanged between federates. At the receiver federate side a new set of CWND packets starting with start_id are generated using the GMA parameters received. Whenever a packet drop occurs at the first federate, its packet_id is conveyed to the second federate by an interaction so that no packets with that packet_id will be generated. ACK packets are generated for every two packets at the receiver and routed through the receiver federate boundary from where they are sent to the sender federate in terms of an interaction. When such an interaction is received in the sender federate, an ACK packet for the flow is generated and routed until it reaches the source of the flow it belongs to. Other protocol behavior is implemented identically as for sequential simulation.

The state diagram for the TCP-like protocol designed for federated simulation is given in Figure 4.10.

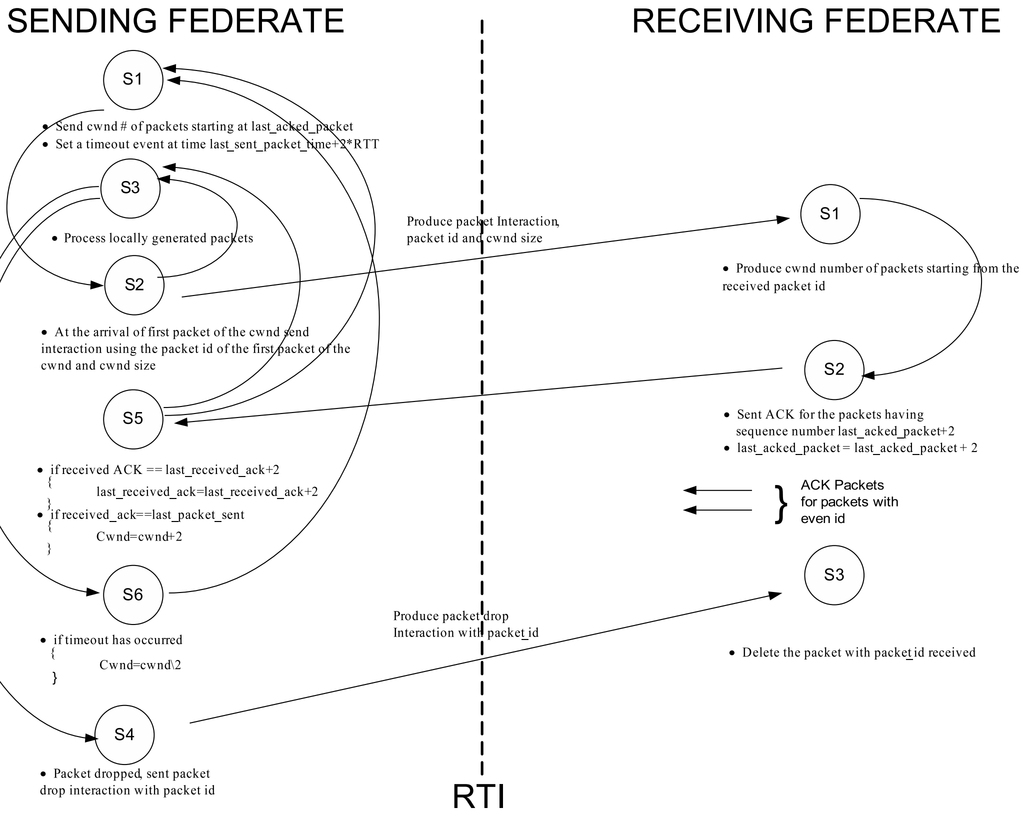

SENDING FEDERATE
RECEIVING FEDERATE
S1
• Send cwnd # of packets starting at last_acked_packet
• Set a timeout event at time last_sent_packet_time+2*RTT
S3
• Process locally generated packets
S2
• At the arrival of first packet of the cwnd send interaction using the packet id of the first packet of the cwnd and cwnd size
S5
• if received ACK == last_received_ack+2 { last_received_ack=last_received_ack+2 }
• if received_ack==last_packet_sent { Cwnd=cwnd+2 }
S6
• if timeout has occurred { Cwnd=cwnd\2 }
S4
• Packet dropped, sent packet drop interaction with packet id
Produce packet Interaction, packet id and cwnd size
Produce packet drop Interaction with packet_id
RTI
S1
• Produce cwnd number of packets starting from the received packet id
S2
• Sent ACK for the packets having sequence number last_acked_packet+2
• last_acked_packet = last_acked_packet + 2
ACK Packets for packets with even id
S3
• Delete the packet with packet_id received

Figure 4.10: State Diagram of TCP-LIKE protocol for Federated Simulation

Now to show that flow modeling with GMA works accurately for a feedback based protocol, federated simulation that deploys flow modeling and its sequential counterpart are implemented. Accuracy results are investigated for both a lossless network where intermediate routers have unlimited buffer space and for a network with limited buffers causing packet loss. Packet interarrival time was chosen as 20msec. That is, the source generates a packet every 20msec until CWND is reached. Simulations were also done for both constant service time (st=10 msec) and random service times (st=rand () %10) in the routers.

Unlike the case with non-feedback based protocol, where it is possible to compare sequential and federated simulation with flow modeling in a flow by flow fashion, in the case of a feedback based protocol, flow by flow comparison is impossible. That is because, for a

TCP-like feedback based protocol, when the network gets congested and a packet is lost, the flow control mechanism will halve the CWND of the flow to which the dropped packet belongs to. Then the network becomes less congested and not as frequent consecutive drops will occur. But at the end of the simulation, the flow that halved its CWND will have fewer packets received at the receiver. So any small difference on the generation time of packets and/or service times changes the order of packet drops and the throughput of the flows are altered. This is the case even for the repeated runs of sequential simulation with different random number generation seeds. Although throughputs of individual flows are altered, the overall network throughput will be the same when averaged over flows. So accuracy results are compared in terms of the throughput of the network and the standard deviation (STD) of packet interarrivals at the receivers. Throughput of the network is calculated per flow, by the sum of number of packets carried for the simulated flows divided by the number of flows and the STD is calculated in the same way.

Accuracy results achieved for per flow throughput and STD of packet interarrivals for sequential simulation and 2 federate simulation for 40 flows are presented below. Simulated time duration was 900sec. Simulations are repeated for different scenarios.

Table 4.30: GMA modeling results for TCP-like protocol, constant service time, unlimited router buffers

	Number of Packets / flow	Packet Interarrival STD.
Sequential	3637,6	1462,675
Federated with flow modeling	3643,5	1465,625
% Error between sequential and federated with flow modeling	0,16	0,2

Table 4.31: GMA modeling results for TCP-like protocol, constant service time, limited router buffers

	Number of Packets / flow	Packet Interarrival STD.
Sequential	2659,55	2337,925
Federated with flow modeling	2677,75	2305,7
% Error between sequential and federated with flow modeling	0,68	-1,38

Table 4.32: GMA modeling results for TCP-like protocol, random service time, unlimited router buffers

	Number of Packets / flow	Packet Interarrival STD.
Sequential	6577,6	796,15
Federated with flow modeling	6607,55	803,6
% Error between sequential and federated with flow modeling	-0,45	0,93

Table 4.33: GMA modeling results for TCP-like protocol, random service time, limited router buffers

	Number of Packets / flow	Packet Interarrival STD.
Sequential	4285,25	1951,075
Federated with flow modeling	4239,6	1957,925
% Error between sequential and federated with flow modeling	-1,065	0,35

Results presented in Tables 4.30 through 4.33 show that the proposed flow modeling approach successfully models inter-federate packets so that results compared with sequential counterpart are acceptably close. Hence, it has been shown that the flow modeling approach can safely be used for a feedback based protocol where data packets are modeled with GMA and protocol packets are sent as they are. The errors achieved for the simulation of a feedback based protocol using flow modeling are less than that of the non-feedback based protocol. That is because in feedback based protocol more control information is sent between federates, this makes the packets generated at the receiver federate more controllable resulting in smaller errors.

To investigate the speedup performance of the proposed method, a series of simulations were done for different numbers of flows for a sequential case and federated case with flow modeling using 8 federates. Lookahead value has to chosen as small as 2msec, otherwise protocol violation occurs. That is, protocol state machine relies on in order delivery of data packets. If lookahead is chosen to a larger value, two packets may possibly have the same timestamp. Then, RTI may change the order of packets, so out of order delivery of the packets to the receiving federate occurs. Although none of the packets are dropped, a timeout event will be raised and a serious error both in the number of packets and STD of interarrival packets will occur. The time required to finish the simulations are noted. Results are given in Table 4.34:

Table 4.34: Simulation time (sec.) of Sequential and Federated Simulation with flow modeling for TCP-like Protocol vs. # of Flows

Number of flows	Time required to simulate for Federated Simulation with flow modeling (sec.)	Time required to simulate for Sequential Simulation (sec.)
160	717	1002
400	1423	34460
600	2255	95345
1000	5133	357800

Table 4.34 shows that the flow modeling approach performs better than sequential simulation even for a small number of flows like 160. More impressive speedup results are achieved for larger values. It is seen that in the case of 1000 flows, flow modeling with GMA approach achieves a speedup factor of 40 over sequential simulation which is larger than the number of CPUs used in the simulation. As the event queue was divided into 8, overall computation required to keep event list in time order is reduced significantly, that was the same case with the simulation of non-feedback based traffic.

Next the performance difference of federated simulation with flow modeling in comparison to that of federated simulation without flow modeling is investigated. Time results are presented in Table 4.35.

Table 4.35: Simulation time (sec.) Federated simulation without flow modeling vs. Federated Simulation with flow modeling for TCP-like Protocol

Number of flows	Time required to simulate for Federated Simulation with flow modeling (sec.)	Time required to simulate for Federated Simulation without flow modeling (sec.)
160	717	1172
400	1423	2161
600	2255	3325
1000	5133	7882

Surprisingly, for the feedback based TCP-like protocol, flow modeling approach seems unable to achieve significant speedups over federated simulation without flow modeling. As flow modeling is applied only to the data packets, the intensity and number of ACK packets communicated through RTI in terms of interaction makes federates tightly coupled again. In this case, the lookahead value had to be chosen as small as 2msec again, to achieve acceptable simulation accuracy.

But if flow modeling is applied to both data and ACK packets, the frequency of information exchange is significantly reduced. Hence, a larger lookahead value (i.e. 10msec) could be chosen for this case. Also, we should note that the timing error between information exchanges does not cause protocol error as in the former cases, federated simulation without flow modeling or flow modeling applied to data packets only. That is because as packets and ACKs are regenerated at the correct order at the receiver federate, timing errors can not cause protocol errors, may only affect the number of packets or STD of flows. So for the case of flow modeling applied to both data and ACK packets, a larger lookahead can be comfortably chosen.

Table 4.36: GMA modeling for both data and ACK packets, results for TCP-like protocol, constant service time, unlimited router buffers

	Number of Packets / flow	Packet Interarrival STD.
Sequential	3637,6	1462,67
Federated with flow modeling	3585,5	1477,95
% Error between sequential and federated with flow modeling applied to both data and ACK packets	1,43	-1,044

Table 4.37: Simulation time (sec.) Federated simulation with flow modeling data packet only vs. Federated Simulation with flow modeling data and ACK packets for TCP-like Protocol

Number of flows	Time required to simulate for Federated Simulation with flow modeling, ACK modeled also (sec.)	Time required to simulate for Federated Simulation without flow modeling (sec.)
160	286	1172
400	371	2161
600	684	3325
1000	1020	7882

While the accuracy obtained is close to that of the case with flow modeling applied exclusively to data packets, time results for the federated simulation without flow modeling and with flow modeling applied to both data and ACK packets are shown in Table 4.37.

Results in Table 4.37 show that in this case better speedups are achieved comparing with the case of flow modeling with only data packets. An approximate speedup of 8 has been achieved for the simulation of 1000 flows over simulation with flow modeling applied to the data packets only. It seems that better speedup is achieved with increasing numbers of flows. This again shows the importance of loosening inter-federate coupling by flow modeling on the time performance.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Discussion and Conclusions

For the distributed packet level simulation of a large scale communication network, it is possible that different parts of a flow path will be simulated in different LPs. So communication and synchronization is required for each packet that finishes its part in a LP and will be simulated along its remaining route in an other LP. It is well known that frequent information exchange and tight synchronization of LPs decrease parallelism, resulting in wasted computational power.

In this study a novel approach for distributed simulation of a network, based on packet level flow modeling that reduces synchronization requirements between LP‘s was proposed. The idea was, instead of tracing individual packets across federate boundaries, modeling the flows and conveying flow characteristics only. The Gaussian Mixtures Algorithm (GMA) is used for flow modeling. Collected characteristics are sent across federate boundaries for a predefined number of packets. At the receiving LP side, packets that constitute a flow with the same characteristics as the one in the sending LP are generated. GMA models a probability density function in terms of sums of Gaussian functions. So a time efficient and accurate way should be found for finding the correct parameters of GMA. Expectation Maximization (EM) and Self Organizing Mixture Network (SOMN) approaches for solving GMA were implemented and compared. Although EM and SOMN were successful in capturing traffic characteristics, SOMN is observed to have three advantages over EM: First, SOMN is more robust to initial points, second, it has computational advantages, and lastly, SOMN converges to global optima due to its stochastic behaviour. The simulation experiments on solving GMA with SOMN for different distributions showed that the accuracy achieved and the time required for SOMN depend on how the GMA is trained. In the literature some methods were proposed on the learning parameter with the number of steps as stop condition, but their performance changes with the distribution of the samples. So a novel learning method and a stop condition is proposed for SOMN training that models a variety of distributions accurately and in a time efficient manner. The idea was to get a feedback from the variance of error of mean and variance of the training set with the GMA parameters estimated. It was observed that applying two different learning rate parameters for mean and variance of the Gaussian kernels in the GMA lead to faster and more accurate convergence. If the variance of the error is large, larger learning rates are used to quickly stabilize the GMA kernels. Conversely, when the variance of the error is small, this means kernels are near the correct point so a fine tuning is realized by applying small learning rates. Stop condition for training was proposed to be the point when the mean and variance error between the training set and GMA estimated drops under some value. This method was shown to work faster and more accurately than EM and SOMN with traditional learning rate methods.

In short the major contribution of this study is packet flow modeling for inter-federate packets. A lesser contribution is enhancing SOMN with a novel learning rate adaptation mechanism.

Although the flow modeling idea can be used for any kind of parallel simulation, federated architecture was chosen to investigate the achievements. To make a comparison, a packet based discrete event sequential simulation and its federated counterpart was developed. First for a UDP-like non-feedback based protocol, a set of simulations were done for packets generated from different distributions including Exponential, Hyper-exponential and Pareto on the network having unlimited and limited buffer cases. Exponential traffic was chosen for its smooth nature while Pareto was chosen for its burstiness controllable using the shape parameter k. The Hyper-exponential traffic was chosen to see the performance on a mixed traffic. Simulation results have shown that flow modeling leads to acceptable accuracy for all the cases, including highly bursty Pareto traffic. Next, Pareto traffic was simulated for different values of the shape parameter and behavior of the flow modeling approach was investigated for different burst characteristics. Results show that the proposed approach works well for a wide range of characteristics from highly bursty traffic to smooth traffic. Also the dependence of model accuracy on the number of packets registered as training set was investigated. Results show that for smaller training sets, although accuracy results for % error of number of packets does not change much, the % error of STD of interarrival of packets gets better at the cost of increased simulation time.

To see the time performance of the proposed approach for a non-feedback protocol, three simulations, namely, sequential simulation, 8 federate simulation without flow modeling and 8 federate simulation with flow modeling were developed. Simulations were repeated for various numbers of nodes and the simulation durations were recorded. Results show that for the sequential simulation, computational requirement increases more than linearly with network size, as expected. Federated simulation without flow modeling is seen to be slower than sequential simulation for small networks because of the extra cost of RTI communication and synchronization. When the number of nodes increases to around 400, it performs better than sequential showing the advantage of parallelism. It was expected that for increasing number of nodes, federated simulation without flow modeling would consistently perform better than sequential. However, there is a region in which the sequential version performs better again. In this region the cost of communication and synchronization dominates the advantage of dividing event queue between federates so federated simulation without flow modeling looses its advantage over sequential simulation. After ˜900 nodes, federated simulation without flow modeling performs better than sequential simulation again. But speed up results were not so bright. For the simulation of 2400 nodes, a speed up factor of only 1.35 is achieved using 8 CPUs. This shows clearly the need for flow modeling to loosen the coupling between federates.

For the proposed approach, federated simulation with flow modeling, the simulations were repeated. Results show that when the number of nodes is very small, because of extra costs for communication and synchronization of federates, sequential simulation is faster. As the network size increases beyond 80 nodes, the proposed flow modeling method performs better than sequential simulation. It is seen that in case of a 2400 node network, the proposed approach achieves a speedup factor of 217 over sequential simulation which is larger than the number of CPUs used in the simulation. Larger speedup values would be achieved for the simulation of larger number of nodes. It is seen that the flow modeling approach highly reduces the communication and synchronization requirements between federates so that better utilization of CPUs are achieved. Simulation results also verified that federated simulation with flow modeling has two main speedup advantages over sequential simulation. First, federated approach uses more than one CPU, second, as the network is divided into sub-networks and is assigned to the federates, smaller event lists are managed in each federate. So overall computational power used for maintaining event list comparing to the sequential simulation is reduced.

Next, applicability of flow modeling to a TCP-like feedback based protocol is investigated. The idea was that flow modeling can be used to model the data packets whereas packets that carry critical information like protocol information can be communicated as they are. So, for the simulation of TCP-like feedback based protocol, data packets were modeled for each flow using GMA, and ACK packets are conveyed individually between federates. Accuracy results show that federated simulation with flow modeling applied exclusively to data packets performs quite accurately for all the cases where network has limited or unlimited router buffers and packets are served with constant or random service times at the routers. To compare the speedup performance of the approach two comparisons were made. Firstly, simulation results were compared with sequential simulation and it was seen that a speed up factor of ˜70 was achieved over sequential simulation. Secondly, federated simulation with flow modeling applied exclusively to data packets was compared with the federated simulation without flow modeling and it was seen that the speed up factor was only ˜1.5, which was much less than expected. This was due to the intensity of ACK packets communicated between federates. As the next step, flows of ACK packets were also modeled with GMA. In this case, communication between federates was reduced significantly and larger lookahead values could be used without corrupting the correct operation of the protocol. Time results show that a speed up factor of ˜8 was achieved over federated simulation with flow modeling applied to data packets only, and the speed up factor in comparison to sequential simulation was ˜358 for a network size of 1000 nodes.

The proposed approach, flow modeling with GMA, is a straightforward method which can easily be applied to any available simulator as it does not interfere with the simulator kernel. One only has to implement a proxy node which stores and models the inter-federate packets at the sender federate side and another proxy node at the receiving federate that generates packets according to these computed GMA parameters. Extra storage is only required for keeping GMA pool size of packet interarrivals, which is usually represented by a data type long int. No extra space is required for keeping multiple copies of former states as in the case of optimistic approaches. For example for a federate that simulates for 100 flows having GMA pool size of 50, will require only $100 * 50 * 4 = 20kBs$ of memory.

The proposed approach has two main advantages over the iterative method proposed in GENESIS: First of all, GENESIS requires major changes in the kernel of the simulator. GENESIS requires that simulators work iteratively over predefined time intervals in a time driven manner. Simulators that are designed to work on a single CPU, are usually implemented as a single run, event driven simulations. So modifications that will make them work with GENESIS seem to require lots of time and effort or may even be impossible to achieve because of the simulator structure. The proposed approach does not require iteration mechanism, can also be applied to both event driven or time driven simulators without any modifications in the simulation kernel.

As the second advantage, proposed approach achieves better speedup results over sequential simulation for both non-feedback based traffic and feedback based traffic. Although it is not fair to compare the speedup results between approaches as simulation implementations are different, iterative structure of GENESIS requires 2 to 3 iterations for UDP traffic and 5 to 10 iterations for TCP traffic, leading to the expectation that the proposed approach would achieve a better speedup performance than GENESIS.

GENESIS's performance may strictly depend on the chosen simulation step interval. No investigation was done for the effect of this parameter on speedup and accuracy. For our case, as we have continuously running models and the times between information exchanges are high, once a precise modeling is achieved for the flows, timing errors do not bring much error on accuracy.

Last comment is about the accuracy results. GENESIS has not reported any results on the STD of the buffer utilizations, which is an important accuracy metric that should be investigated. Simulation experiences presented in parts 4.3.2-4.3.3 show that accurate modeling of inter-federate packets has great importance on STD error. As GENESIS represents flow characteristics simply with the mean of inter-federate packets and drop probability, STD error achieved with the GENESIS approach will be large. For the proposed flow modeling approach, inter-federate packet characteristics are modeled with GMA which was shown to characterize different flows accurately.

5.2 Future Work

Although the flow modeling approach was shown to work for both feedback-based and non-feedback based protocols on networks with static routes, its performance can be investigated on networks where the routes are dynamic.

In this study, simulated flows were partitioned to two federates only. Both accuracy and speed up performance of the approach can be investigated on networks where flows are partitioned over more than two federates.

It was observed that the accuracy and time performance of the EM algorithm was strictly dependent on the initial values. But as EM is a gradient based algorithm, if a good starting point is given, it will reach the optima quickly and with a small error where it would take longer for SOMN due to its stochastic nature. A hybrid algorithm that uses both SOMN and EM can be proposed. At the first phase, initial values for EM can be found by SOMN quickly then EM can be used to fine tune the accuracy.

A dynamic lookahead estimation method that monitors inter-federate communication intervals and change lookahead value during the simulation execution can be developed. Effect of this approach on accuracy and time performance of the simulations can be investigated.

Recent usage of communications networks have demonstrated trends towards peer-to-peer and multicast traffic. Simulation of large scale networks with such traffic characteristics would possibly benefit to a great extent from the proposed flow modeling approach, but this also remains to be extensively studied on concrete examples.

An alternative way of decreasing the communication required between federates can be achieved by dividing the network topology between federates in a optimized manner, so that most of the communicating node pairs will be simulated in the same federate. In this way biggest part of the traffic will be simulated inside the federate boundary, so less inter-federate packets should be communicated between federates. This method will reduce the communication and synchronization requirements so that better parallelism will be achieved. The problems that can be investigated in this area is to find an optimal way of partitioning the network at the start of the simulation and find a way of keeping it optimized when communicating pairs changes in time during the simulation.

REFERENCES

[1] Fujimoto R.M., Perumalla K., Park A., Wu H., Ammar M.H., Riley G.F., “Large-scale network simulation: how big? how fast?”, 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems, 12-15 Oct. 2003, pp. 116 - 123

[2] Bhatt S., Fujimoto R.M., Ogielski A., Perumalla K. S., “Parallel Simulation Techniques for Large Scale Networks,” IEEE Communications Magazine, vol. 36, 2002, pp. 42-47

[3] Nicol D. M., Liljenstam M., Liu J., December., “Advanced concepts in large-scale network simulation”, Proceedings of the 2005 Winter Simulation Conference, 2005, pp. 153-166

[4] Floyd, S., and V. Paxson., “Difficulties in simulating the Internet”, IEEE/ACM Transactions on Networking Volume 9, Issue 4, 2001, p. 392-403

[5] Liu Y., Szymanski B.K. and Saifee A. , “Genesis: A Scalable Distributed System for Large-scale Parallel Network Simulation”, Computer Networks Journal, vol. 50, no. 12, August 2006, p. 2028-2053

[6] Fujimoto R.M., “Parallel and Distributed Simulation Systems”, Wiley Interscience, 2000

[7] “The Network Simulator - ns-2.”, http://www.isi.edu/nsnam/ns/, Last accessed on 14 March, 2010.

[8] “PDNS - Parallel/Distributed NS.”, http://www.cc.gatech.edu/computing/compass/pdns/, Last accessed on 14 March, 2010.

[9] Nutaro J., “Parallel Discrete Event Simulation with Application to Continuous Systems”, PhD. thesis University of Arizona, 2003.

[10] Wikipedia, “High level architecture (simulation)”, http://en.wikipedia.org/wiki/High_Level_Architecture_(simulation), Last accessed on 14 March, 2010

[11] Riley G.F. , Mostafa H. A., Fujimoto R.M., Park A. , Perumalla K.S. , Xu D. : "A federated approach to distributed network simulation", ACM Trans. Model. Comput. Simul. 14(2), pp. 116-148 (2004)

[12] Fujimoto, R.M., "Parallel and distributed simulation systems"; Simulation Conference, 2001. Proceedings of the Winter, Volume: 1, 9-12 Dec. 2001, pp. 147 - 157

[13] Fujimoto R.M, "Time Management in the High Level Architecture", SIMULATION, Vol. 71, No. 6, 1998, p. 388-400

[14] Perumalla K. S., "Parallel and Distributed Simulation: Traditional Techniques and Recent Advances", Proceedings of the 2006 Winter Simulation Conference, 2006, pp. 84-95

[15] Ferenci S. L., Perumalla K. S., Fujimoto R.M., "An Approach for Federating Parallel Simulators", Proceedings of the 14th Workshop on Parallel and Distributed Simulation, 2000, pp. 63-70

[16] Perumalla K. S., A. Park, Fujimoto R.M., Riley G. F., "Scalable RTI-Based Parallel Simulation of Networks", Proceedings of the Seventeenth Workshop on Parallel and Distributed Simulation (PADS'03), 2003, pp. 97

[17] Riley G. F., Ammar M. H., Fujimoto R.M., A. Park, Perumalla K. S., Xu D.,"A Federated Approach to Distributed Network Simulation": ACM Transactions on Modeling and Computer Simulation, Vol. 14, No. 2, April 2004, pp. 116-148

[18] Perumalla K. S., "Parallel and Distributed Simulation Systems and the High Level Architecture", Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC) 2005

[19] IEEE Std 1516.1-2000, "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) Federate Interface Specification"

[20] Defense Modeling and Simulation Office, "HLA Technical Specifications - Rules, Version 1.3", 1998, http://hla.dmso.mil/tech/rules.html, Last accessed on 21 September, 2006

[21] Department of Defense Defense Modeling and Simulation Office (DMSO), "RTI 1.3-Next Generation Programmer's Guide Version 4"

[22] Liu B. , Figueiredo D. R. , Guo Y., Kurose J. , Towsley D., "A Study of Networks Simulation Efficiency: Fluid Simulation vs. Packet-level Simulation", IEEE INFOCOM,

2001, pp. 1244-1953

[23] Ahn J. S., Danzing P. B., “Packet Network Simulation: Speedup and Accuracy Versus Timing Granularity”, IEEE ACM Transactions On Networking, vol. 4, no. 5, October 1996, p. 743-757.

[24] Liu J., “Packet-Level Integration Of Fluid TCP Models In Real-Time Network Simulation”, Proceedings of the 2006 Winter Simulation Conference, 2006, p. 2162-2169.

[25] Shanbhag D. N, Rao C. R., “Handbook of Statistics 19, Stochastic Processes: Theory and Methods”, North Holland, 2000

[26] Samson Lee, John Leaney, Tim O’Neill, Mark Hunter, “Performance benchmark of a parallel and distributed network simulator”, Proceedings of the Workshop on Principles of Advanced and Distributed Simulation, 2005, p. 101-108

[27] Nicol D. M., and G. Yan., “Simulation of network traffic at coarse time-scales.”, Proceedings of the 19th Workshop on Parallel and Distributed Simulation, 2005 , p. 141-150.

[28] R. Brown, “Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem,” Communications of the ACM, vol. 31, p. 1220-1227, 1988.

[29] Paul M. Baggenstoss, “Statistical Modeling Using Gaussian Mixtures and HMMs with MATLAB”, http://www.nuwc.navy.mil/npt/Csf/htmldoc/pdf/pdf.html, Last accessed on 18 November, 2007.

[30] Fischer M.J., Bevilacqua Masi D.M, Gross D., Shortle J., Brill P.H., “Using quantile estimates in simulating Internet queues with Paretoservice times”;Simulation Conference, 2001. Proceedings of the Winter, Volume: 1, 9-12 Dec. 2001, Pages: 477-485

[31] Yin H., Allinson N.M. , “Self-Organizing Mixture Networks for Probability Density Estimation”, IEEE Transactions on Neural Networks, vol. 12, no. 2, March 2001, p. 405-411

[32] Germen E., Bilgen S., “StatSOM - Statistical Self Organizing Map”, International Conference on NeuroComputing, NC’98, Vienna, 1998.

[33] “CERTI an efficient and Open Source HLA Runtime Infrastructure (RTI)”, http://www.nongnu.org/certi/, Last accessed on 14 March, 2010.

[34] Tomasi C., “Estimating Gaussian Mixture Densities with EM - A Tutorial”, http://www.cs.duke.edu/courses/spring04/cps196.1/handouts/EM/tomasiEM.pdf, Last accessed on 14 March, 2010.

[35] Verbeek J.J., Vlassis N., Kröse B.J.A., “Self-Organizing Mixture Models”, Elsevier, Neurocomputing 63, 2005, pp. 99-123

[36] Germen E.,“STATISTICAL SELF-ORGANIZING MAP”, PhD. Thesis METU,1999

[37] “Chapter 4 - Statistical Modeling”, www.npt.nuwc.navy.mil/Csf/papers/pdf.pdf, Last accessed on 22 May, 2008.

[38] Bilgen, S., Confidence of Simulation Results, unpublished lecture note, METU, 1986

[39] D. Anick, D. Mitra, and M.M. Sondhi, “Stochastic theory of a datahandling system with multiple sources,” The Bell System Technical Journal, vol. 61, no. 8, Oct. 1982, pp. 1871-1894

[40] B. Liu, Y. Guo, J. F. Kurose, D. F. Towsley, and W. Gong, “Fluid simulation of large scale networks: Issues and tradeoffs,” in International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas, Nevada, USA, 1999, pp. 2136-2142.

[41] Dahmann J. S.,Fujimoto R. M.,Weatherly R. M.,“The Department Of Defense High Level Architecture”, Proceedings of the 1997 Winter Simulation Conference, p. 142-148.

[42] T. Demirci, S. Bilgen, “Federated Simulation of Network Performance Using Packet Flow Modeling”, Summer Simulation Multiconference 2009 (SCSC’2009), July 2009.

APPENDIX A

SIMULATION RESULTS

Table A.1: Non-Feedback Based Traffic, Flow Modeling with mean, Results for Exponential traffic with unlimited router buffers

Federated		Sequential			
# of Packets	STD. of Packets	# of Packets	STD. of Packets	% Packet Error	% STD Error
16470	137	16382	203	0,54	32,51
16489	135	16404	200	0,52	32,50
16477	157	16398	199	0,48	21,11
16471	162	16403	202	0,41	19,80
16642	153	16529	198	0,68	22,73
16842	150	16767	196	0,45	23,47
16492	99	16441	198	0,31	50,00
16567	153	16495	201	0,44	23,88
16702	167	16585	196	0,71	14,80
16857	115	16794	195	0,38	41,03
16596	135	16505	199	0,55	32,16
16529	143	16474	200	0,33	28,50
16804	140	16712	196	0,55	28,57
16483	150	16391	202	0,56	25,74
16674	132	16600	198	0,45	33,33
16612	124	16552	198	0,36	37,37
16447	97	16381	201	0,40	51,74
16728	161	16643	198	0,51	18,69
16270	139	16208	201	0,38	30,85
16425	175	16337	199	0,54	12,06
16673	137	16606	200	0,40	31,50
16692	132	16604	197	0,53	32,99
16454	138	16413	201	0,25	31,34
16578	137	16498	199	0,48	31,16
16690	139	16601	198	0,54	29,80
16555	127	16452	203	0,63	37,44
16446	173	16343	202	0,63	14,36
16460	146	16377	198	0,51	26,26
16691	154	16612	198	0,48	22,22
16603	155	16520	199	0,50	22,11
16512	140	16465	201	0,29	30,35
16680	107	16594	199	0,52	46,23
16535	146	16480	199	0,33	26,63
16286	142	16220	203	0,41	30,05
16370	126	16312	205	0,36	38,54
16592	141	16527	195	0,39	27,69
16458	165	16358	202	0,61	18,32
16436	135	16358	201	0,48	32,84
16527	129	16462	200	0,39	35,50
16513	165	16434	198	0,48	16,67

Table A.2: Non-Feedback Based Traffic, Flow Modeling with mean, Results for Exponential traffic with limited router buffers ˜%20 packet drop

Federated		Sequential			
# of Packets	STD. of Packets	# of Packets	STD. of Packets	% Packet Error	% STD Error
13369	194	12781	254	4,60	23,62
13262	183	12607	255	5,20	28,24
13147	204	12731	247	3,27	17,41
13265	170	12738	250	4,14	32,00
13240	195	12747	247	3,87	21,05
13459	168	13001	242	3,52	30,58
13179	164	12718	247	3,62	33,60
13248	199	12747	249	3,93	20,08
13363	198	12825	244	4,19	18,85
13408	197	12929	244	3,70	19,26
13239	179	12699	246	4,25	27,24
13194	224	12662	247	4,20	9,31
13422	178	12983	243	3,38	26,75
13206	189	12648	254	4,41	25,59
13395	198	12980	244	3,20	18,85
13295	174	12814	244	3,75	28,69
13174	135	12695	250	3,77	46,00
13383	194	12988	243	3,04	20,16
13099	198	12603	252	3,94	21,43
13099	196	12672	249	3,37	21,29
13193	175	12836	247	2,78	29,15
13344	219	12763	249	4,55	12,05
13135	212	12668	253	3,69	16,21
13181	197	12691	250	3,86	21,20
13367	180	12887	246	3,72	26,83
13160	195	12668	253	3,88	22,92
13155	188	12643	251	4,05	25,10
13132	209	12680	249	3,56	16,06
13286	214	12879	246	3,16	13,01
13379	151	12854	245	4,08	38,37
13234	180	12817	249	3,25	27,71
13434	160	12858	246	4,48	34,96
13268	172	12826	249	3,45	30,92
13159	207	12563	252	4,74	17,86
13162	188	12650	252	4,05	25,40
13336	178	12832	242	3,93	26,45
13178	184	12577	252	4,78	26,98
13255	173	12717	249	4,23	30,52
13245	199	12783	249	3,61	20,08
13246	196	12683	247	4,44	20,65

Table A.3: Non-Feedback Based Traffic, Flow Modeling with mean, Results for Pareto traffic with unlimited router buffers

Federated		Sequential			
# of Packets	STD. of Packets	# of Packets	STD. of Packets	% Packet Error	% STD Error
18688	672	18653	648	0,19	3,70
18130	1071	18135	929	0,03	15,29
16537	2505	16426	2424	0,68	3,34
18620	648	18621	663	0,01	2,26
19124	518	19063	469	0,32	10,45
19271	359	19282	412	0,06	12,86
19426	405	19379	410	0,24	1,22
18765	603	18711	552	0,29	9,24
18609	624	18633	643	0,13	2,95
14789	4582	14791	4434	0,01	3,34
18824	611	18831	553	0,04	10,49
19657	319	19561	389	0,49	17,99
18949	650	18901	576	0,25	12,85
19555	418	19422	422	0,68	0,95
17644	1642	17591	1547	0,30	6,14
19039	599	19054	624	0,08	4,01
18624	612	18510	668	0,62	8,38
18524	624	18596	599	0,39	4,17
18834	622	18846	577	0,06	7,80
18553	835	18529	795	0,13	5,03
18582	442	18598	430	0,09	2,79
19355	1120	19357	1028	0,01	8,95
18936	562	18906	511	0,16	9,98
19260	613	19253	544	0,04	12,68
19074	1076	19064	946	0,05	13,74
18489	501	18514	537	0,14	6,70
18325	637	18177	649	0,81	1,85
16660	511	16626	548	0,20	6,75
19436	408	19384	474	0,27	13,92
18568	693	18468	727	0,54	4,68
17594	1090	17593	989	0,01	10,21
19058	779	18968	798	0,47	2,38
17776	1295	17783	1243	0,04	4,18
19001	612	18932	549	0,36	11,48
19319	382	19228	372	0,47	2,69
19777	305	19747	347	0,15	12,10
19656	298	19564	348	0,47	14,37
19175	367	19168	424	0,04	13,44
19270	365	19256	440	0,07	17,05
19728	323	19686	403	0,21	19,85

Table A.4: Non-Feedback Based Traffic, Flow Modeling with mean, Results for Pareto traffic with limited router buffers ˜%30 packet drop

Federated		Sequential			
# of Packets	STD. of Packets	# of Packets	STD. of Packets	% Packet Error	% STD Error
12973	820	14153	791	8,34	3,67
12564	1278	13795	1211	8,92	5,53
12395	1592	13607	1507	8,91	5,64
12765	746	14166	808	9,89	7,67
13102	648	14589	680	10,19	4,71
13198	466	14622	505	9,74	7,72
13316	466	14737	563	9,64	17,23
12764	798	14172	762	9,94	4,72
12472	853	13846	886	9,92	3,72
12793	589	14271	659	10,36	10,62
12618	788	13975	796	9,71	1,01
13098	466	14707	509	10,94	8,45
12632	863	14129	808	10,60	6,81
13116	490	14525	510	9,70	3,92
11716	1994	12991	1898	9,81	5,06
12782	773	14165	753	9,76	2,66
12522	778	13831	797	9,46	2,38
12498	748	13785	808	9,34	7,43
12867	783	14199	780	9,38	0,38
12633	1046	13940	1009	9,38	3,67
12402	614	13803	616	10,15	0,32
13030	1229	14434	1247	9,73	1,44
12698	713	13999	684	9,29	4,24
12960	795	14393	727	9,96	9,35
12790	1283	14341	1249	10,82	2,72
12500	619	13798	681	9,41	9,10
12528	784	13959	820	10,25	4,39
12662	681	13948	734	9,22	7,22
13423	538	14755	527	9,03	2,09
12632	839	14106	836	10,45	0,36
11951	1377	13363	1219	10,57	12,96
13129	912	14492	924	9,41	1,30
12131	1664	13551	1548	10,48	7,49
12896	843	14361	785	10,20	7,39
13205	434	14645	498	9,83	12,85
13450	403	14846	462	9,40	12,77
13350	460	14713	485	9,26	5,15
12945	524	14298	568	9,46	7,75
13043	584	14414	592	9,51	1,35
13344	474	14767	494	9,64	4,05

Table A.5: Non-Feedback Based Traffic, Flow Modeling with mean, Results for Pareto traffic with federate 1 has unlimited, federate2 has limited router buffers

Federated		Sequential			
# of Packets	STD. of Packets	# of Packets	STD. of Packets	% Packet Error	% STD Error
8248	77	6984	123	18,10	37,40
8323	112	7089	119	17,41	5,88
8277	83	7058	121	17,27	31,40
8331	88	7081	121	17,65	27,27
8420	89	7191	116	17,09	23,28
8276	110	7121	117	16,22	5,98
8288	82	6950	119	19,25	31,09
8337	69	7119	120	17,11	42,50
8375	76	7107	121	17,84	37,19
8404	73	7138	119	17,74	38,66
8304	88	7001	122	18,61	27,87
8289	93	7027	120	17,96	22,50
8486	62	7170	117	18,35	47,01
8242	81	6949	124	18,61	34,68
8341	83	7106	122	17,38	31,97
8289	89	7125	118	16,34	24,58
8250	82	7030	121	17,35	32,23
8474	101	7149	122	18,53	17,21
8311	91	7142	118	16,37	22,88
8271	95	7108	118	16,36	19,49
907	589	751	1121	20,77	47,46
1005	515	874	982	14,99	47,56
953	1145	780	1098	22,18	4,28
992	297	823	1081	20,53	72,53
989	300	798	1121	23,93	73,24
952	587	754	1205	26,26	51,29
985	318	822	1037	19,83	69,33
979	656	812	1023	20,57	35,87
1017	424	845	1060	20,36	60,00
969	455	786	1207	23,28	62,30
947	363	799	1111	18,52	67,33
972	360	795	1138	22,26	68,37
983	365	823	1082	19,44	66,27
978	666	808	1077	21,04	38,16
940	637	740	1230	27,03	48,21
988	398	808	1135	22,28	64,93
954	1497	805	1082	18,51	38,35
943	866	757	1240	24,57	30,16
1020	336	854	1034	19,44	67,50
1008	834	812	1081	24,14	22,85

Table A.6: Non-Feedback Based Traffic, Flow Modeling with Exponential modeling, Results for Exponential traffic with unlimited router buffers

Federated		Sequential			
# of Packets	STD. of Packets	# of Packets	STD. of Packets	% Packet Error	% STD Error
16400	222	16382	203	0,11	9,36
16400	235	16404	200	0,02	17,50
16365	240	16398	199	0,20	20,60
16339	215	16403	202	0,39	6,44
16455	210	16529	198	0,45	6,06
16764	237	16767	196	0,02	20,92
16423	235	16441	198	0,11	18,69
16564	207	16495	201	0,42	2,99
16627	226	16585	196	0,25	15,31
16700	211	16794	195	0,56	8,21
16534	226	16505	199	0,18	13,57
16473	251	16474	200	0,01	25,50
16699	213	16712	196	0,08	8,67
16380	203	16391	202	0,07	0,50
16592	242	16600	198	0,05	22,22
16449	209	16552	198	0,62	5,56
16360	210	16381	201	0,13	4,48
16650	233	16643	198	0,04	17,68
16187	212	16208	201	0,13	5,47
16318	222	16337	199	0,12	11,56
16614	215	16606	200	0,05	7,50
16612	212	16604	197	0,05	7,61
16413	237	16413	201	0,00	17,91
16539	209	16498	199	0,25	5,03
16583	220	16601	198	0,11	11,11
16414	222	16452	203	0,23	9,36
16315	217	16343	202	0,17	7,43
16348	221	16377	198	0,18	11,62
16626	211	16612	198	0,08	6,57
16514	231	16520	199	0,04	16,08
16375	221	16465	201	0,55	9,95
16635	203	16594	199	0,25	2,01
16430	219	16480	199	0,30	10,05
16201	235	16220	203	0,12	15,76
16300	221	16312	205	0,07	7,80
16585	222	16527	195	0,35	13,85
16326	233	16358	202	0,20	15,35
16281	235	16358	201	0,47	16,92
16393	222	16462	200	0,42	11,00
16443	225	16434	198	0,05	13,64

Table A.7: Non-Feedback Based Traffic, Flow Modeling with Exponential modeling, Results for Exponential traffic with limited router buffers ˜%26 packet drop

Federated		Sequential			
# of Packets	STD. of Packets	# of Packets	STD. of Packets	% Packet Error	% STD Error
12054	308	12781	254	5,69	21,26
12159	312	12607	255	3,55	22,35
12161	291	12731	247	4,48	17,81
12068	313	12738	250	5,26	25,20
12315	301	12747	247	3,39	21,86
12340	307	13001	242	5,08	26,86
12050	332	12718	247	5,25	34,41
12121	318	12747	249	4,91	27,71
12178	312	12825	244	5,04	27,87
12412	302	12929	244	4,00	23,77
12111	307	12699	246	4,63	24,80
12048	346	12662	247	4,85	40,08
12380	280	12983	243	4,64	15,23
12036	305	12648	254	4,84	20,08
12293	306	12980	244	5,29	25,41
12185	340	12814	244	4,91	39,34
12094	293	12695	250	4,73	17,20
12217	305	12988	243	5,94	25,51
11928	313	12603	252	5,36	24,21
12132	290	12672	249	4,26	16,47
12240	276	12836	247	4,64	11,74
12173	289	12763	249	4,62	16,06
12031	286	12668	253	5,03	13,04
12035	289	12691	250	5,17	15,60
12288	303	12887	246	4,65	23,17
12101	300	12668	253	4,48	18,58
12089	283	12643	251	4,38	12,75
12112	283	12680	249	4,48	13,65
12255	272	12879	246	4,85	10,57
12314	270	12854	245	4,20	10,20
12170	274	12817	249	5,05	10,04
12218	287	12858	246	4,98	16,67
12271	276	12826	249	4,33	10,84
12041	274	12563	252	4,16	8,73
11944	293	12650	252	5,58	16,27
12230	302	12832	242	4,69	24,79
12097	273	12577	252	3,82	8,33
12172	293	12717	249	4,29	17,67
12186	303	12783	249	4,67	21,69
12035	295	12683	247	5,11	19,43

Table A.8: Non-Feedback Based Traffic, Flow Modeling with Pareto modeling, Results for Pareto traffic with unlimited router buffers

Federated		Sequential			
# of Packets	STD. of Packets	# of Packets	STD. of Packets	% Packet Error	% STD Error
18640	647	18590	685	0,27	5,55
18059	1025	18106	1054	0,26	2,75
17950	1301	17882	1312	0,38	0,84
18561	641	18550	700	0,06	8,43
19037	524	19054	590	0,09	11,19
19192	392	19204	434	0,06	9,68
19292	452	19336	483	0,23	6,42
18669	583	18709	657	0,21	11,26
18520	689	18468	763	0,28	9,70
18942	506	18987	565	0,24	10,44
18659	634	18714	679	0,29	6,63
19552	393	19515	432	0,19	9,03
18790	656	18882	695	0,49	5,61
19414	428	19425	433	0,06	1,15
17522	1624	17475	1633	0,27	0,55
18965	643	18971	645	0,03	0,31
18594	660	18559	682	0,19	3,23
18441	616	18356	694	0,46	11,24
18788	635	18777	673	0,06	5,65
18464	850	18456	873	0,04	2,63
18559	468	18478	522	0,44	10,34
19222	1015	19281	1076	0,31	5,67
18895	567	18881	582	0,07	2,58
19190	599	19208	622	0,09	3,70
18988	1073	18974	1084	0,07	1,01
18367	537	18370	584	0,02	8,05
18250	667	18308	711	0,32	6,19
18496	620	18431	629	0,35	1,43
19270	403	19380	453	0,57	11,04
18432	603	18501	725	0,37	16,83
17476	1061	17584	1061	0,61	0,00
18932	780	18963	805	0,16	3,11
17697	1336	17724	1351	0,15	1,11
18789	636	18931	676	0,75	5,92
19180	420	19226	425	0,24	1,18
19716	357	19677	393	0,20	9,16
19534	406	19553	412	0,10	1,46
19047	434	19076	485	0,15	10,52
19162	482	19149	507	0,07	4,93
19586	396	19629	418	0,22	5,26

Table A.9: Non-Feedback Based Traffic, Flow Modeling with Pareto modeling, Results for Pareto traffic with limited router buffers ˜%24.5 packet drop

Federated		Sequential			
# of Packets	STD. of Packets	# of Packets	STD. of Packets	% Packet Error	% STD Error
12695	419	14153	791	10,30	47,03
12587	397	13795	1211	8,76	67,22
12234	424	13607	1507	10,09	71,86
12864	385	14166	808	9,19	52,35
13009	378	14589	680	10,83	44,41
12494	411	14622	505	14,55	18,61
12340	411	14737	563	16,27	27,00
12224	443	14172	762	13,75	41,86
12428	401	13846	886	10,24	54,74
12204	432	14271	659	14,48	34,45
12319	410	13975	796	11,85	48,49
12043	428	14707	509	18,11	15,91
12340	423	14129	808	12,66	47,65
12601	418	14525	510	13,25	18,04
12526	410	12991	1898	3,58	78,40
12523	405	14165	753	11,59	46,22
12179	418	13831	797	11,94	47,55
12400	419	13785	808	10,05	48,14
12249	441	14199	780	13,73	43,46
11898	446	13940	1009	14,65	55,80
12626	432	13803	616	8,53	29,87
12513	402	14434	1247	13,31	67,76
12504	414	13999	684	10,68	39,47
12106	427	14393	727	15,89	41,27
12217	529	14341	1249	14,81	57,65
12352	397	13798	681	10,48	41,70
12377	404	13959	820	11,33	50,73
12500	424	13948	734	10,38	42,23
12607	409	14755	527	14,56	22,39
12200	437	14106	836	13,51	47,73
12401	428	13363	1219	7,20	64,89
12585	418	14492	924	13,16	54,76
12366	429	13551	1548	8,74	72,29
12393	412	14361	785	13,70	47,52
12089	430	14645	498	17,45	13,65
12384	424	14846	462	16,58	8,23
12376	405	14713	485	15,88	16,49
12260	423	14298	568	14,25	25,53
12445	405	14414	592	13,66	31,59
12177	419	14767	494	17,54	15,18

Table A.10: Non-Feedback Based Traffic, Flow Modeling with GMA, Results for Exponential traffic with mean=200 unlimited router buffers

Federated		Sequential			
# of Packets	STD. of Packets	# of Packets	STD. of Packets	% Packet Error	% STD Error
16330	202	16382	203	0,32	0,49
16220	212	16404	200	1,12	6,00
16343	215	16398	199	0,34	8,04
16335	207	16403	202	0,41	2,48
16475	201	16529	198	0,33	1,52
16766	200	16767	196	0,01	2,04
16385	209	16441	198	0,34	5,56
16493	199	16495	201	0,01	1,00
16576	208	16585	196	0,05	6,12
16624	192	16794	195	1,01	1,54
16455	199	16505	199	0,30	0,00
16312	197	16474	200	0,98	1,50
16641	195	16712	196	0,42	0,51
16385	217	16391	202	0,04	7,43
16637	199	16600	198	0,22	0,51
16579	210	16552	198	0,16	6,06
16306	208	16381	201	0,46	3,48
16694	195	16643	198	0,31	1,52
16230	224	16208	201	0,14	11,44
16336	223	16337	199	0,01	12,06
16635	230	16606	200	0,17	15,00
16566	214	16604	197	0,23	8,63
16397	222	16413	201	0,10	10,45
16495	222	16498	199	0,02	11,56
16609	209	16601	198	0,05	5,56
16432	209	16452	203	0,12	2,96
16317	225	16343	202	0,16	11,39
16379	243	16377	198	0,01	22,73
16653	238	16612	198	0,25	20,20
16443	227	16520	199	0,47	14,07
16324	208	16465	201	0,86	3,48
16606	220	16594	199	0,07	10,55
16462	239	16480	199	0,11	20,10
16208	233	16220	203	0,07	14,78
16305	235	16312	205	0,04	14,63
16507	196	16527	195	0,12	0,51
16349	218	16358	202	0,06	7,92
16380	213	16358	201	0,13	5,97
16438	222	16462	200	0,15	11,00
16425	219	16434	198	0,05	10,61

Table A.11: Non-Feedback Based Traffic, Flow Modeling with GMA, Results for Exponential traffic with mean=200 limited router buffers ˜%26 packet drop

Federated		Sequential			
# of Packets	STD. of Packets	# of Packets	STD. of Packets	% Packet Error	% STD Error
12152	279	12781	254	4,92	9,84
12140	266	12607	255	3,70	4,31
12232	265	12731	247	3,92	7,29
12201	263	12738	250	4,22	5,20
12270	253	12747	247	3,74	2,43
12377	277	13001	242	4,80	14,46
12171	272	12718	247	4,30	10,12
12079	261	12747	249	5,24	4,82
12358	263	12825	244	3,64	7,79
12406	267	12929	244	4,05	9,43
12159	270	12699	246	4,25	9,76
12080	266	12662	247	4,60	7,69
12345	262	12983	243	4,91	7,82
12163	260	12648	254	3,83	2,36
12353	256	12980	244	4,83	4,92
12196	249	12814	244	4,82	2,05
12157	257	12695	250	4,24	2,80
12479	253	12988	243	3,92	4,12
12110	257	12603	252	3,91	1,98
12213	255	12672	249	3,62	2,41
12434	255	12836	247	3,13	3,24
12223	266	12763	249	4,23	6,83
12073	283	12668	253	4,70	11,86
12213	276	12691	250	3,77	10,40
12217	251	12887	246	5,20	2,03
12149	282	12668	253	4,10	11,46
11907	268	12643	251	5,82	6,77
12100	273	12680	249	4,57	9,64
12328	256	12879	246	4,28	4,07
12298	254	12854	245	4,33	3,67
12383	272	12817	249	3,39	9,24
12321	251	12858	246	4,18	2,03
12251	260	12826	249	4,48	4,42
12048	270	12563	252	4,10	7,14
12126	301	12650	252	4,14	19,44
12349	250	12832	242	3,76	3,31
12153	260	12577	252	3,37	3,17
12068	257	12717	249	5,10	3,21
12232	271	12783	249	4,31	8,84
12184	261	12683	247	3,93	5,67

Table A.12: Non-Feedback Based Traffic, Flow Modeling with GMA, Results for Pareto traffic with $x_m = 60$,$k = 1.5$ unlimited router buffers

Federated		Sequential			
# of Packets	STD. of Packets	# of Packets	STD. of Packets	% Packet Error	% STD Error
18562	647	18590	685	0,15	5,55
18046	979	18106	1054	0,33	7,12
16376	2438	16459	2476	0,50	1,53
18523	630	18559	700	0,19	10,00
19065	492	19050	590	0,08	16,61
19322	409	19194	434	0,67	5,76
19341	401	19335	483	0,03	16,98
18675	556	18698	657	0,12	15,37
18545	613	18477	763	0,37	19,66
14534	4464	14766	4488	1,57	0,53
18714	503	18712	679	0,01	25,92
19588	389	19520	432	0,35	9,95
18897	584	18885	695	0,06	15,97
19507	423	19432	432	0,39	2,08
17553	1512	17491	1632	0,35	7,35
18924	559	18966	645	0,22	13,33
18509	530	18551	682	0,23	22,29
18456	615	18362	694	0,51	11,38
18773	551	18781	673	0,04	18,13
18496	805	18449	873	0,25	7,79
18527	569	18485	522	0,23	9,00
19300	1081	19284	1076	0,08	0,46
18925	623	18877	582	0,25	7,04
19200	618	19209	622	0,05	0,64
18962	1116	18966	1084	0,02	2,95
18413	640	18376	584	0,20	9,59
18281	743	18294	711	0,07	4,50
16600	652	16615	653	0,09	0,15
19382	506	19376	453	0,03	11,70
18500	760	18503	725	0,02	4,83
17549	1152	17562	1061	0,07	8,58
18959	800	18962	805	0,02	0,62
17715	1363	17730	1351	0,08	0,89
18945	670	18926	676	0,10	0,89
19238	466	19227	425	0,06	9,65
19684	416	19689	393	0,03	5,85
19557	441	19555	412	0,01	7,04
19075	525	19091	485	0,08	8,25
19183	483	19157	464	0,14	4,09
19643	446	19628	418	0,08	6,70

Table A.13: Non-Feedback Based Traffic, Flow Modeling with GMA, Results for Pareto traffic with $x_m = 60$,$k = 1.5$ limited router buffers ˜%24.5 packet drop

Federated		Sequential			
# of Packets	STD. of Packets	# of Packets	STD. of Packets	% Packet Error	% STD Error
13549	858	14153	791	4,27	8,47
13031	1104	13795	1211	5,54	8,84
13002	1374	13607	1507	4,45	8,83
13323	746	14166	808	5,95	7,67
13778	661	14589	680	5,56	2,79
13957	479	14622	505	4,55	5,15
13854	529	14737	563	5,99	6,04
13349	747	14172	762	5,81	1,97
13154	820	13846	886	5,00	7,45
13370	634	14271	659	6,31	3,79
13194	776	13975	796	5,59	2,51
13857	481	14707	509	5,78	5,50
13328	696	14129	808	5,67	13,86
13762	507	14525	510	5,25	0,59
12410	1925	12991	1898	4,47	1,42
13427	717	14165	753	5,21	4,78
13175	819	13831	797	4,74	2,76
13144	798	13785	808	4,65	1,24
13256	724	14199	780	6,64	7,18
13336	1011	13940	1009	4,33	0,20
13097	557	13803	616	5,11	9,58
13848	1223	14434	1247	4,06	1,92
13224	635	13999	684	5,54	7,16
13586	633	14393	727	5,61	12,93
13407	1267	14341	1249	6,51	1,44
13220	663	13798	681	4,19	2,64
13012	852	13959	820	6,78	3,90
13209	624	13948	734	5,30	14,99
13933	535	14755	527	5,57	1,52
13354	850	14106	836	5,33	1,67
12639	1227	13363	1219	5,42	0,66
13785	848	14492	924	4,88	8,23
12740	1526	13551	1548	5,98	1,42
13672	724	14361	785	4,80	7,77
13941	570	14645	498	4,81	14,46
14101	455	14846	462	5,02	1,52
14094	536	14713	485	4,21	10,52
13754	586	14298	568	3,80	3,17
13707	571	14414	592	4,90	3,55
13968	445	14767	494	5,41	9,92

Table A.14: Non-Feedback Based Traffic, Flow Modeling with GMA, Results for Pareto traffic with $x_m = 60$,$k = 1.5$ federate 1 unlimited federate2 limited router buffers

Federated		Sequential			
# of Packets	STD. of Packets	# of Packets	STD. of Packets	% Packet Error	% STD Error
7021	119	6984	123	0,53	3,25
7136	123	7089	119	0,66	3,36
7051	125	7058	121	0,10	3,31
7068	126	7081	121	0,18	4,13
7250	113	7191	116	0,82	2,59
7084	122	7121	117	0,52	4,27
7036	131	6950	119	1,24	10,08
7104	128	7119	120	0,21	6,67
6922	133	7107	121	2,60	9,92
7190	121	7138	119	0,73	1,68
7079	136	7001	122	1,11	11,48
7073	126	7027	120	0,65	5,00
7191	122	7170	117	0,29	4,27
6993	126	6949	124	0,63	1,61
7096	126	7106	122	0,14	3,28
7046	118	7125	118	1,11	0,00
7047	118	7030	121	0,24	2,48
7046	121	7149	122	1,44	0,82
7057	120	7142	118	1,19	1,69
7044	125	7108	118	0,90	5,93
747	1365	751	1121	0,53	21,77
775	1361	874	982	11,33	38,59
752	1483	780	1098	3,59	35,06
761	1550	823	1081	7,53	43,39
799	1166	798	1121	0,13	4,01
715	1569	754	1205	5,17	30,21
759	1168	822	1037	7,66	12,63
741	1277	812	1023	8,74	24,83
821	1073	845	1060	2,84	1,23
735	1148	786	1207	6,49	4,89
783	1306	799	1111	2,00	17,55
789	1283	795	1138	0,75	12,74
814	1152	823	1082	1,09	6,47
793	1191	808	1077	1,86	10,58
753	1441	740	1230	1,76	17,15
738	1487	808	1135	8,66	31,01
752	1295	805	1082	6,58	19,69
762	1253	757	1240	0,66	1,05
819	1156	854	1034	4,10	11,80
829	1295	812	1081	2,09	19,80

Table A.15: Non-Feedback Based Traffic, Flow Modeling with GMA, Results for Hyper-exponential traffic with unlimited router buffers

Federated		Sequential			
# of Packets	STD. of Packets	# of Packets	STD. of Packets	% Packet Error	% STD Error
8076	494	8075	502	0,01	1,59
8315	498	8277	488	0,46	2,05
8220	511	8226	488	0,07	4,71
8265	565	8248	491	0,21	15,07
8465	485	8453	471	0,14	2,97
8225	479	8251	478	0,32	0,21
7980	540	7996	494	0,20	9,31
8353	485	8315	478	0,46	1,46
8339	486	8408	478	0,82	1,67
8334	464	8313	487	0,25	4,72
8299	484	8289	484	0,12	0,00
8225	529	8249	486	0,29	8,85
8388	501	8456	479	0,80	4,59
8310	502	8328	490	0,22	2,45
8229	503	8280	492	0,62	2,24
8218	479	8287	483	0,83	0,83
8018	487	8045	500	0,34	2,60
8305	554	8294	483	0,13	14,70
8365	508	8414	480	0,58	5,83
8041	480	8139	489	1,20	1,84
8158	507	8157	499	0,01	1,60
8379	483	8404	470	0,30	2,77
8191	519	8148	498	0,53	4,22
8162	497	8260	480	1,19	3,54
8313	521	8286	483	0,33	7,87
8000	510	8050	505	0,62	0,99
8241	497	8290	482	0,59	3,11
8298	541	8297	488	0,01	10,86
8346	473	8334	472	0,14	0,21
8113	494	8108	491	0,06	0,61
8354	502	8354	489	0,00	2,66
8192	496	8236	494	0,53	0,40
8296	502	8353	477	0,68	5,24
8363	482	8418	473	0,65	1,90
8132	536	8091	496	0,51	8,06
8168	489	8254	479	1,04	2,09
8330	510	8378	489	0,57	4,29
8270	492	8273	491	0,04	0,20
8300	565	8333	497	0,40	13,68
8167	540	8222	488	0,67	10,66

Table A.16: Non-Feedback Based Traffic, Flow Modeling with GMA, Results for Hyper-exponential traffic with limited router buffers

Federated		Sequential			
# of Packets	STD. of Packets	# of Packets	STD. of Packets	% Packet Error	% STD Error
8000	528	7981	505	0,24	4,55
8209	484	8205	491	0,05	1,43
8130	532	8132	491	0,02	8,35
8068	519	8160	494	1,13	5,06
8225	487	8344	474	1,43	2,74
8140	521	8153	481	0,16	8,32
7870	499	7878	497	0,10	0,40
8113	471	8221	481	1,31	2,08
8323	488	8301	481	0,27	1,46
8135	507	8217	490	1,00	3,47
8137	499	8173	487	0,44	2,46
8181	492	8156	490	0,31	0,41
8049	487	8352	483	3,63	0,83
8141	484	8239	494	1,19	2,02
8040	484	8181	495	1,72	2,22
8219	502	8187	486	0,39	3,29
7994	548	7931	505	0,79	8,51
8101	472	8194	486	1,13	2,88
8283	494	8316	483	0,40	2,28
7964	493	8057	491	1,15	0,41
7985	511	8052	502	0,83	1,79
8250	495	8296	474	0,55	4,43
8092	516	8036	503	0,70	2,58
8129	493	8162	483	0,40	2,07
8266	485	8198	485	0,83	0,00
7889	518	7967	507	0,98	2,17
8075	488	8191	485	1,42	0,62
8176	508	8193	492	0,21	3,25
8295	490	8258	475	0,45	3,16
7980	527	7988	495	0,10	6,46
8183	499	8256	492	0,88	1,42
8024	529	8122	499	1,21	6,01
8252	483	8260	481	0,10	0,42
8303	505	8331	477	0,34	5,87
7946	530	8001	499	0,69	6,21
8059	506	8140	484	1,00	4,55
8255	494	8311	493	0,67	0,20
8198	526	8185	496	0,16	6,05
8223	521	8243	501	0,24	3,99
8068	530	8124	490	0,69	8,16

Table A.17: Feedback Based Traffic, Flow Modeling with GMA, Results for constant service time, unlimited router buffers

Federated		Sequential	
# of Packets	STD. of Packets	# of	STD. of
3616	1452	3620	1424
3680	1478	3640	1491
3654	1463	3616	1477
3616	1464	3572	1499
3680	1432	3608	1432
3616	1457	3616	1492
3680	1363	3680	1499
3616	1458	3680	1439
3680	1427	3680	1413
3616	1521	3616	1487
3680	1482	3616	1505
3616	1464	3616	1475
3616	1445	3616	1512
3680	1473	3616	1435
3622	1472	3616	1502
3680	1455	3680	1435
3616	1489	3622	1520
3618	1509	3680	1421
3616	1513	3680	1432
3616	1505	3680	1420
3616	1443	3616	1477
3608	1482	3616	1463
3680	1464	3616	1529
3632	1505	3616	1452
3680	1457	3620	1466
3616	1480	3680	1399
3680	1449	3680	1464
3616	1487	3616	1452
3672	1414	3616	1514
3616	1482	3616	1452
3680	1442	3616	1493
3680	1473	3616	1471
3680	1409	3648	1414
3616	1531	3680	1419
3616	1510	3680	1453
3646	1442	3678	1412
3616	1451	3632	1504
3680	1427	3680	1384
3616	1465	3616	1504
3680	1490	3616	1475

Table A.18: Feedback Based Traffic, Flow Modeling with GMA, Results for constant service time, limited router buffers

Federated		Sequential	
# of Packets	STD. of Packets	# of	STD. of
2500	2644	2644	2414
2174	3386	3386	1975
2628	3030	3030	2122
2824	2712	2712	2224
3360	3482	3482	1997
2624	2512	2512	2296
3136	2094	2094	2657
2480	1986	1986	2679
2510	3254	3254	2042
2632	3056	3056	2107
2966	3414	3414	1981
2196	2492	2492	2358
3324	2620	2620	2293
2406	2740	2740	2227
2580	2980	2980	2132
2178	3080	3080	2097
2496	2294	2294	2528
2358	1894	1894	2811
2692	2224	2224	2609
2124	2554	2554	2364
1992	2308	2308	2500
3208	3144	3144	2074
2912	2302	2302	2443
2558	1716	1716	2949
2558	2142	2142	2604
2958	2882	2882	2165
2742	3386	3386	2009
2584	1680	1680	2887
2402	2980	2980	2199
2288	2644	2644	2291
2878	1936	1936	2707
2170	2522	2522	2357
2538	2500	2500	2440
4574	2420	2420	2450
2148	3720	3720	1914
2654	1820	1820	2854
3228	3510	3510	1980
3004	3238	3238	2036
2888	2228	2228	2521
2638	2856	2856	2224

Table A.19: Feedback Based Traffic, Flow Modeling with GMA, Results for random service time, unlimited router buffers

Federated		Sequential	
# of Packets	STD. of Packets	# of	STD. of
6624	803	6612	825
6656	817	6624	758
6620	773	6624	766
6598	794	6688	773
6560	801	6560	829
6598	802	6624	801
6560	803	6560	811
6504	842	6560	824
6560	774	6626	775
6624	819	6624	803
6604	785	6624	819
6520	785	6624	758
6560	814	6624	777
6624	815	6624	826
6624	758	6598	779
6560	810	6624	809
6560	833	6560	780
6576	812	6574	782
6576	798	6560	806
6550	826	6624	759
6560	813	6624	780
6526	842	6624	790
6578	773	6624	800
6560	819	6624	788
6598	786	6678	792
6560	849	6560	823
6496	810	6560	827
6560	813	6624	809
6584	789	6560	813
6560	831	6688	774
6624	748	6624	776
6650	819	6624	795
6560	771	6560	813
6550	845	6624	779
6560	795	6560	812
6584	776	6620	773
6560	846	6560	797
6560	796	6624	811
6592	785	6578	807
6624	774	6624	827

Table A.20: Feedback Based Traffic, Flow Modeling with GMA, Results for random service time, unlimited router buffers

Federated		Sequential	
# of Packets	STD. of Packets	# of	STD. of
4374	1893	4444	1871
4766	1815	4338	1902
4842	1851	4568	1894
5024	1775	3402	2181
3782	2065	4008	2027
4996	1783	4626	1873
3500	2134	4402	1863
4130	1943	3418	2211
3536	2135	4944	1774
4396	1885	4310	1942
4322	1896	4578	1858
3592	2120	5022	1763
4726	1831	4706	1869
4396	1871	4694	1796
4308	1897	5066	1773
3452	2198	4194	1957
4532	1861	2756	2501
4364	1899	3062	2283
4358	1924	4532	1894
3466	2151	5144	1746
5570	1647	4368	1931
3818	2015	3422	2214
3538	2139	4574	1850
4754	1797	3498	2177
4490	1875	4068	1953
2864	2411	5404	1695
4840	1795	4130	1973
4408	1926	4062	2005
4910	1803	4220	1908
4098	1957	4594	1839
5454	1689	4852	1791
4490	1867	4332	1954
4268	1945	5172	1742
3130	2334	3616	2088
4122	1999	5148	1747
3712	2084	3284	2228
3010	2357	3742	2124
4298	1926	3670	2096
3702	2085	5140	1703
5246	1739	3900	2047

APPENDIX B

EXPECTATION MAXIMIZATION AND SELF ORGINIZING MIXTURE NETWORK CODES

B.1 Expectation Maximization Code

```
void calculate(void)
{
    Q_pre=-100000;
    int iteration_count=0;
    while( 1 )
    {
        iteration_count++;
        for(int r=0;r<no_of_modes;r++)
        {
            w[r]=lqr_eval(samples,mode[r].mean,mode[r].variance);
            w[r]=w[r]+log(mode[r].weight);
        }

        for(int k=0;k<sample_count;k++)
        {
            double max=w[0][k];
            for(int c=1;c<no_of_modes;c++)
            {
                if(w[c][k]>max)
                {
                    max=w[c][k];
                }
            }
            mw[k]=max;
        }

      nor=0.0;
      for(int n=0;n<no_of_modes;n++)
      {
         w[n]=w[n]-mw;
         w[n]=exp(w[n]);
         nor=nor+w[n];
      }

      temp_arr_4=(log(nor)+mw);
      Q=temp_arr_4.sum();

      if( (Q-Q_pre) < 0.0000001 )
      {
         break;
         average_iteration=average_iteration+iteration_count;
      }

        double weight_sum=0;
        for(int p=0;p<no_of_modes;p++)
        {
            w[p]=w[p]/nor;
            a[p]=w[p].sum();

            temp_arr_4=samples*w[p];
            mode[p].mean=temp_arr_4.sum()/a[p];

            temp_arr_4=(samples-mode[p].mean)*sqrt(w[p]/a[p]);
            mode[p].variance=sqrt((temp_arr_4*temp_arr_4).sum());
            mode[p].variance=-sqrt(mode[p].variance*mode[p].variance+1);

            weight_sum=weight_sum+a[p];
        }

        for(int d=0;d<no_of_modes;d++)
```

```
        {
            mode[d].weight=a[d]/weight_sum;
        }

        Q_pre=Q;
    }

void init(void)
{

    for(int j=0;j<7;j++)
    {
        mode[j].weight=0.0;
        mode[j].mean=0.0;
        mode[j].variance=0.0;
    }

    for(int i=0;i<no_of_modes;i++)
    {
        w[i].resize(ARRAY_SIZE);
        mode[i].weight=1.0/(double)no_of_modes;
        mode[i].mean=samples[rand()%sample_count];
        mode[i].variance=(samples.max()-samples.min())/4.0;

    }
}

DB_VARRAY lqr_eval(DB_VARRAY &array,double mean,double variance)
{
    double logdet=2.0*log(fabs(variance));

    DB_VARRAY tmpidx=array-mean;

    tmpidx=(tmpidx/variance);

    DB_VARRAY result;
    tmpidx=tmpidx*tmpidx;

    result=-0.5*tmpidx-0.5*log(6.2832)-0.5*logdet;

    return result;
}
```

B.2 Self Orginizing Mixture Network Code

```
void calculate(void)
{
    long int index=0;
    double sample=0;

    double l_rate=0;
    double l_rate2=0;
    double l_rate2_variance=0;
    double p_i_x[10];

    double P_x[10];

    double wm_mean=0;
    double wm_variance=0;
    double wm_sum_of_squares=0;
    double wm_sums=0;

    double wm_mean2=0;
    double wm_variance2=0;
    double wm_sum_of_squares2=0;
    double wm_sums2=0;

    double wm_variance2_old=0;
    double wm_variance2_diff=0;

    double a2_old=0;
    double l_rate_mean=0;
    double l_rate_variance=0;
    double l_rate_sum_of_squares=0;
    double l_rate_sums=0;
    double l_rate_variance_old=-10;

    int counter=0;

    for(long int y=0;y<100000;y++)
    {

        index=rand()%200;
        sample=sample_arr[index];

        for(int t=0;t<no_modes;t++)
        {
            p_i_x[t]=(1.0/( sqrt(2.0*3.14*mode[t][2]) )) *
            exp( -0.5* (sample-mode[t][1])*(sample-mode[t][1])/mode[t][2]);

            if(p_i_x[t]<-1000)
            {
                p_i_x[t]=0.0005;
            }
        }

        double p_x=0;
        for(int w=0;w<no_modes;w++)
        {
            p_x=p_x+p_i_x[w]*mode[w][0];
        }

        if(p_x==0)
        {
            p_x=1;
        }

        for(int r=0;r<no_modes;r++)
```

```
{
    P_x[r]=p_i_x[r]*mode[r][0]/p_x;
}

double max=-1.0;
double min=1000;
int max_index=-1;
int min_index=-1;
for(int e=0;e<no_modes;e++)
{
    if(P_x[e]>max)
    {
        max=P_x[e];
        max_index=e;
    }

    if(P_x[e]<min)
    {
        min=P_x[e];
        min_index=e;
    }
}

if(y==0)
{
    wm_mean=P_x[min_index];
}
else
{
    wm_mean=(1.0/double(y+1))*fabs(sample-mode[min_index][1])
            +((double)(y)/(double)(y+1))*wm_mean;
}

double a=0;
double b=0;
```

```
for(int s=0;s<no_modes;s++)
{
    if(mode[s][0]>0.05)
    {
        a=a+mode[s][0]*mode[s][1];
        b=b+mode[s][0]*(mode[s][2]+mode[s][1]*mode[s][1]);
    }
}

b=b-a*a;

double mean_error=(a-mean_set)/mean_set*100;
double variance_error=(sqrt(b)-variance_set)/variance_set*100;

a=fabs(mean_set-a);
b=fabs(variance_set-sqrt(b));

wm_sum_of_squares=wm_sum_of_squares+a*a;
wm_sums=wm_sums+a;

double temp=( 1.0 / (double)(y+1) ) *
              ( wm_sum_of_squares - ( ( 1.0 / (double)(y+1) )
                                        * wm_sums * wm_sums ) ) ;
wm_variance=sqrt(temp);

wm_sum_of_squares2=wm_sum_of_squares2+b*b;
wm_sums2=wm_sums2+b;

temp=( 1.0 / (double)(y+1) ) *
       ( wm_sum_of_squares2 - ( ( 1.0 / (double)(y+1) )
                                 * wm_sums2 * wm_sums2 ) ) ;
wm_variance2=sqrt(temp);

if((wm_variance/double(y+1))>(0.5) || wm_variance==0.0 )
{
    l_rate2=(0.5);
}
```

```
else
{
    l_rate2=wm_variance/(double)(y+1);
}

if((wm_variance2/double(y+1))>(0.01) || wm_variance2==0.0 )
{
    l_rate2_variance=(0.01);
}
else
{
    l_rate2_variance=wm_variance2/(double)(y+1);
}

if((l_rate2<0.01)&& (fabs(mean_error)<0.5) && (fabs(variance_error)<1.0) )
{
    break;
}

if(max_index==0)
{
    delta_m=(l_rate2)*P_x[0]*(sample-mode[0][1]);
    delta_var=(l_rate2_variance)*P_x[0]*((sample-mode[0][1])*
                                      (sample-mode[0][1])-mode[0][2]);
    mode[0][1]=mode[0][1]+delta_m;
    mode[0][2]=mode[0][2]+delta_var;

    delta_m=(l_rate2)*P_x[1]*(sample-mode[1][1]);
    delta_var=(l_rate2_variance)*P_x[1]*((sample-mode[1][1])*
                                      (sample-mode[1][1])-mode[1][2]);
    mode[1][1]=mode[1][1]+delta_m;
    mode[1][2]=mode[1][2]+delta_var;

}
else if(max_index==(no_modes-1))
{
    delta_m=(l_rate2)*P_x[no_modes-2]*(sample-mode[no_modes-2][1]);
    delta_var=(l_rate2_variance)*P_x[no_modes-2]*
              ((sample-mode[no_modes-2][1])*(sample-mode[no_modes-2][1])
                -mode[no_modes-2][2]);
    mode[no_modes-2][1]=mode[no_modes-2][1]+delta_m;
    mode[no_modes-2][2]=mode[no_modes-2][2]+delta_var;

    delta_m=(l_rate2)*P_x[no_modes-1]*(sample-mode[no_modes-1][1]);
    delta_var=(l_rate2_variance)*P_x[no_modes-1]*
              ((sample-mode[no_modes-1][1])*(sample-mode[no_modes-1][1])
                -mode[no_modes-1][2]);
    mode[no_modes-1][1]=mode[no_modes-1][1]+delta_m;
    mode[no_modes-1][2]=mode[no_modes-1][2]+delta_var;
}
else
{
    delta_m=(l_rate2)*P_x[max_index-1]*(sample-mode[max_index-1][1]);
    delta_var=(l_rate2_variance)*P_x[max_index-1]
               *((sample-mode[max_index-1][1])*(sample-mode[max_index-1][1])
                -mode[max_index-1][2]);
    mode[max_index-1][1]=mode[max_index-1][1]+delta_m;
    mode[max_index-1][2]=mode[max_index-1][2]+delta_var;

    delta_m=(l_rate2)*P_x[max_index]*(sample-mode[max_index][1]);
    delta_var=(l_rate2_variance)*P_x[max_index]
              *((sample-mode[max_index][1])*(sample-mode[max_index][1])
                 -mode[max_index][2]);
    mode[max_index][1]=mode[max_index][1]+delta_m;
    mode[max_index][2]=mode[max_index][2]+delta_var;

    delta_m=(l_rate2)*P_x[max_index+1]*(sample-mode[max_index+1][1]);
    delta_var=(l_rate2_variance)*P_x[max_index+1]
             *((sample-mode[max_index+1][1])*(sample-mode[max_index+1][1])
                -mode[max_index+1][2]);
    mode[max_index+1][1]=mode[max_index+1][1]+delta_m;
    mode[max_index+1][2]=mode[max_index+1][2]+delta_var;
}
```

```
        double p_sum=0;
        for (int u=0;u<no_modes;u++)
        {
            mode[u][0]=mode[u][0]+(l_rate2)*(P_x[u]-mode[u][0]);
            p_sum=p_sum+mode[u][0];
        }
    }
}


void init(void)
{

    double sum_of_squares=0;
    double sums=0;
    double sample=0;
    long int max=-10;
    long int min=10000000;

    for(int u=0;u<200;u++)
    {
        sample=sample_arr[u];
        sum_of_squares=sum_of_squares+sample*sample;
        sums=sums+sample;

        mean_set=(1.0/double(u+1))*sample+((double)(u)/(double)(u+1))*mean_set;
        variance_set=sqrt( ( 1.0 / (double)(u+1) ) *
                           ( sum_of_squares - ( ( 1.0 / (double)(u+1) ) *
                                                 sums * sums ) ) );

        if(sample<min)
        {
            min=sample;
        }

        if(sample>max)
```

```
            {
                max=sample;
            }
        }

        mode[0][0]=0;
        mode[1][0]=0;
        mode[2][0]=0;
        mode[3][0]=0;
        mode[4][0]=0;
        mode[5][0]=0;
        mode[6][0]=0;

        for(int e=0;e<no_modes;e++)
        {
            mode[e][0]=1.0/((double)no_modes);
            mode[e][1]=min+(max-min)/no_modes*(e+1);
        }

        for(int a=0;a<no_modes;a++)
        {
            mode[a][2]=variance_set*variance_set*10.0;
        }
    }
```

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Demirci, Turan

Nationality: Turkish (TC)

Date and Place of Birth: 28 September 1978, Ankara

Marital Status: Married

Phone: +90 312 2101310

Fax: +90 312 2101315

email: turan.demirci@uzay.tubitak.gov.tr

EDUCATION

Degree	**Institution**	**Year of Graduation**
MS	METU Electrical and Electronics Eng.	2002
BS	METU Electrical and Electronics Eng.	2000
High School	Atatürk Anadolu High School, Ankara	1996

WORK EXPERIENCE

Year	**Place**	**Enrollment**
2006-Present	TUBITAK UZAY	Chief Senior Researcher
2002-2006	TUBITAK UZAY	Senior Researcher
2000-2002	METU EE Dept.	Teaching Assistant

PUBLICATIONS

1. O. Salor, S. Buhan, O. Unsar, B. Boyrazoglu, E. Altintas, T. Atalık, B. Haliloglu, T. Inan, A. Kalaycıoglu, A. Terciyanlı, A. Açık, T. Demirci, E. Ozdemirci, I. Çadırcı, M. Ermis, "Mobile Monitoring System to Take Nationwide PQ Measurements on Electricity Transmission Systems", Elsevier Measurement Journal, Vol.42, pp. 501-515, 2009.

2. D. Kuçuk, T. Inan, O. Salor, T. Demirci, Y. Akkaya, S. Buhan, B. Boyrazoglu, O. Unsar, E. Altıntas, B. Haliloglu, I. Cadircı, M. Ermis, "An extensible database architecture for nationwide power quality monitoring", International Journal of Electrical Power and Energy Systems, accepted for publication, Nov. 2009.

3. T. Demirci, S. Bilgen, "Federated Simulation of Network Performance Using Packet Flow Modeling", Summer Simulation Multiconference 2009 (SCSC'2009), July 2009.

4. O. Salor, D. Kuçuk, M. Guder, T. Demirci, Y. Akkaya, I. Cadırcı, M. Ermis, "Turkiye Elektrik Iletim Sisteminde Harmonik Bozulma ve Kırpısma Parametrelerinin Olusturulan Guç Kalitesi Veritabanı Yapısıyla Degerlendirilmesi (Assessment of Harmonic Distortions and Flicker in the Turkish Electricity Transmission System Based on the Developed Database Architecture)", 3. EMO Enerji Verimliligi ve Kalitesi Sempozyumu, EVK 2009, Kocaeli.

5. T. Demirci, A. Kalaycıoglu, O. Salor, S. Pakhuylu, T. Inan, D. Kuçuk, M. Guder, T. Can, Y. Akkaya, S. Bilgen, I. Cadirci, M. Ermis, "Türkiye Elektrik İletim Sistemi için Yurt Çapında Güç Kalitesi İzleme Ağı ve Veri Degerlendirme Merkezi: Güncel Gelişmeler", IEEE 16. Sinyal İşleme, İletişim ve Uygulamaları Kurultayı, Didim, Türkiye, 20-22 Nisan 2008

6. E. Ozdemirci, Y. Akkaya, B. Boyrazoglu, S. Buhan, A. Terciyanli, O. Unsar, E. Altintas, B. Haliloglu, A. Acik, T. Atalik, O. Salor, T. Demirci, I. Cadirci, M .Ermis, "Mobile Monitoring System to Take PQ Snapshots of Turkish Electricity Transmission System", Instrumentation and Measurement Technology Conference Proceedings, 2007. IMTC 2007,Page(s): 1 - 6

7. T. Demirci, A. Kalaycioglu, O. Salor, S. Pakhuylu, M. Dagli, T. Kara, H. S. Aksuyek, C. Topcu, B. Polat, S. Bilgen, S. Umut, I. Cadirci, M. Ermis,"National PQ Monitoring Network for Turkish Electricity Transmission System", Instrumentation and Measurement Technology Conference Proceedings, 2007. IMTC 2007,Page(s): 1 - 6

8. H. F. Bilgin, M. Ermis, K. N. Kose, A. Cetin, I. Cadirci, A. Acik, T. Demirci, A. Terciyanli, C. Kocak, M. Yorukoglu,"Reactive-Power Compensation of Coal Mining Excavators by Using a New-Generation STATCOM", Industry Applications, IEEE Transactions on Volume: 43 , Issue: 1, 2007,Page(s): 97 - 110

9. A. Cetin, H. F. Bilgin, A. Acik, T. Demirci, K. N. Kose, A. Terciyanli, B. Gultekin, N. Aksoy, B. Mutluer, I. Cadirci, M. Ermis, K. Ongan, N. Akinci, “Reactive Power Compensation of Coal Conveyor Belt Drives by Using D-STATCOMs ”, Industry Applications Conference, 2007, Page(s): 1731 - 1740

10. A. Terciyanli, A. Acık, A. Cetin, T. Demirci, T. Kılınç, “Thyristor Switched Reactor - Active Power Filter Combination for Power Quality Solution of a Light Rail Transportation System”, PCIM, Nuremberg, Germany, 21-24.05.2007

11. A. Acik, A. Cetin, M. Ermis, A. Terciyanli, T. Demirci, I. Cadirci, C. Ermis, A. Aydinay, F. Yilmaz, and E. Ozkaya, “A Fully Static Solution to Power Quality Problem of Light Rail Transportation System in Bursa, Turkey”, ACEMP’04 Conference Proc., 26-28 Mayis 2004, Istanbul, pp. 446-452.

12. T. Demirci, S. Bilgen, “A Performance Study on Real-Time IP Multicasting”; Eighth IEEE International Symposium on Computers and Communication 2003. (ISCC 2003), 2003,vol. 1,Page(s): 441 - 446