University of Alabama in Huntsville
LOUIS

Theses

UAH Electronic Theses and Dissertations

2011

Modeling CPU utilization and network traffic in a distributed interactive simulation system using discrete event simulation

Shreyas Shridhar Purohit

Follow this and additional works at: https://louis.uah.edu/uah-theses

Recommended Citation

Purohit, Shreyas Shridhar, "Modeling CPU utilization and network traffic in a distributed interactive simulation system using discrete event simulation" (2011). *Theses*. 583.
https://louis.uah.edu/uah-theses/583

This Thesis is brought to you for free and open access by the UAH Electronic Theses and Dissertations at LOUIS. It has been accepted for inclusion in Theses by an authorized administrator of LOUIS.

MODELING CPU UTILIZATION AND NETWORK TRAFFIC IN A DISTRIBUTED INTERACTIVE SIMULATION SYSTEM USING DISCRETE EVENT SIMULATION

by

SHREYAS SHRIDHAR PUROHIT

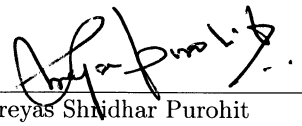
A THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science
in
The Department of Computer Science
to
The School of Graduate Studies
of
The University of Alabama in Huntsville

HUNTSVILLE, ALABAMA

2011

In presenting this thesis in partial fulfillment of the requirements for a master's degree from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department or the Dean of the School of Graduate Studies. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this thesis.



Shreyas Shridhar Purohit

7-MAR-2012

(date)

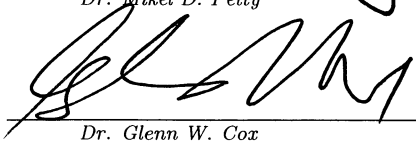
THESIS APPROVAL FORM

Submitted by Shreyas Shridhar Purohit in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and accepted on behalf of the Faculty of the School of Graduate Studies by the thesis committee.

We, the undersigned members of the Graduate Faculty of The University of Alabama in Huntsville, certify that we have advised and/or supervised the candidate of the work described in this thesis. We further certify that we have reviewed the thesis manuscript and approve it in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.


3-4-2011

Dr. Mikel D. Petty (Date) Committee Chair


3-7-2011

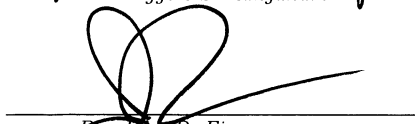
Dr. Glenn W. Cox (Date)


3/3/11

Dr. Letha H. Etzkorn (Date)


3-4-11

Dr. Heggere S. Ranganath (Date) Department Chair


3/7/11

Dr. John D. Fix (Date) College Dean


5/5/11

Dr. Rhonda K. Gaede (Date) Graduate Dean

ABSTRACT
School of Graduate Studies
The University of Alabama in Huntsville

Degree Master of Science College/Dept. Science/Computer Science

Name of Candidate Shreyas Shridhar Purohit

Title Modeling CPU Utilization And Network Traffic In a Distributed Interactive Simulation System Using Discrete Event Simulation

Distributed Interactive Simulation (DIS) is a IEEE standard 1278 protocol used to achieve distributed simulation. The load on the CPU and the network are important factors to consider in distributed simulation. In this work, a Discrete Event model is constructed to simulate a DIS System. The model predicts the load on CPU based on "tick" time variance. The model also outputs the bandwidth usage of network as number of nodes and number of entities on each node varies. The model is validated by comparing measured experimental values with a DIS software, VR-Forces, built by MÄK technologies. VR-Forces being a closed software, a surrogate was built to better validate the developed model. In the process, the battle of 73 Easting was simulated using VR-Forces to validate the model with a known valid scenario size.

Abstract Approval: Committee Chair Mikel D Petty
Dr. Mikel D. Petty

Department Chair Heggere S. Ranganath 3-4-2011
Dr. Heggere S. Ranganath

Graduate Dean Rhonda Kay Gaede 5-5-2011
Dr. Rhonda K. Gaede

ACKNOWLEDGMENTS

I would like take this page to thank numerous people without whom this thesis might not have been published.

Firstly, I would like to thank Dr. Mikel Petty, the thesis committee chair and my respected advisor, for his supervision, advice and guidance since the inception of my thesis. He provided valuable guidance through the thesis and motivated me to go past the edge. His innovative and thought provoking ideas have made me a better person at the very end of this work. I would also like to thank him in providing and helping me chose this very interesting thesis topic. This thesis work would not have been completed successfully if not for his motivation, guidance, advice, thoughts, and input.

I owe my deepest gratitude to Dr. Glenn Cox whose courses in Modeling and Simulation helped me tremendously to complete this work. The course projects developed under his guidance gave me an added advantage while pursuing this thesis work.

I would also like to thank Dr. Wes Colley for his valuable input on statistical testing which helped tremendously in this thesis work.

I would also like to thank my thesis committee members, Dr. Letha Etzkorn and Dr. Glenn Cox, for taking time out of their busy schedule and reviewing this work. I extend my thanks to Dr. Heggere Ranganath, chair of the Computer Science Department at UAH for the inspiration he provided throughout my graduate work. I would like to record my gratitude to him for giving me an opportunity in teaching an undergraduate course and making me realize my fondness towards teaching. I gratefully acknowledge him for providing the financial aid which helped me to stay focused through my work. I would like to add a special note of thanks to Ms. Betty Nelson,
Ms. Diane Cox and Ms. Maryann Bierer who have been like my mother, helping, motivating, and making me smile in this entire stay at Huntsville for pursuing my Graduate degree.

I offer my regards to the faculty and staff of the Department of Computer Science and Department of Modeling and Simulation at UAH for their support and for the facilities provided to me while working on my thesis.

I would like to thank my friends Thejaswi Raya, Sandeep Bettadapura, Ashly Alexander Mathew Manchadivilakathu, Deepak Kumar, Satyajeet Padhi, Karthik Venkiteswaran, Avinash Surya Prakash, Onkar Akolkar, Anish Bivalkar, Vineetha Bettaiah, Dharini Govindarajan, Yi Chen, Haritha Pulletikurti, Sneha Hirkannawar, Ayesha Bhatnagar, Spoorthi Myneni and many others for being ever supportive and keeping me cheerful all the time.

I would also like to thank my brother, Shravan Purohit, for being there whenever I needed him for the support.

Finally, this thesis would not have been possible without my parents' support and guidance. Their belief that I can achieve anything, even the impossible kept me going througout the course of the thesis and the graduate work. There aren't enough words to thank my parents for all the sacrifices and the trouble they have undertaken to make this thesis and my graduate work a grand success.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE PAGE

LIST OF TABLES

TABLE PAGE

Dedicated to my dear parents who have been ever supportive and motivative during the entire course of this thesis.

CHAPTER 1

BACKGROUND

This chapter provides the necessary background information required to understand the context and motivation of this research. Section 1.1 provides a background on distributed simulation explaining the need, advantages, and a brief history. Section 1.2 explains the Distributed Interactive Simulation interoperability protocol, the core domain of the research. Section 1.3 presents discrete event simulation which is used as a part of the solution in this research. Section 1.4 describes CPU and network modeling.

1.1 Distributed simulation

Distributed simulation is a technology which simulates a single virtual world using multiple nodes with a computer program running on each node. Multiple independent nodes are connected via a communication medium such as a computer network. The simulation could involve computer systems which are heterogeneous (hardware and software), could be spread across wide geographic area and administration domain with communication latency ranging from microsecond to seconds. Each node on the network is provided with a part of the entire simulation. All the nodes involved in the distributed simulation co-operate in simulating the modeled system correctly.

The simulations running on each node are typically sequential, i.e., single threaded. Each of these simulators on individual node controls a small part of the entire simulation [3]. The nodes pass messages to one another notifying events of interests that are consumed by every other node. A protocol guides the format and rules to send messages to other participating nodes [4]. The correctness of the simulation is dependent on the messages that are exchanged. The distributed

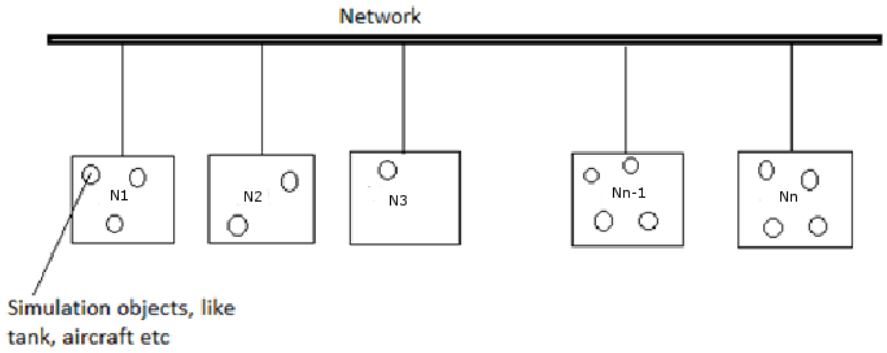


Figure 1.1: Distributed simulation organization

simulation must produce the same result as that of the simulation being entirely run on a single computer [3].

The nodes can be live nodes hosting real systems, virtual nodes hosting just the simulator, and constructive nodes which have Semi-automated forces [4]. There can be nodes on the network that are not directly related to the simulation under question such as data logging and GUI processing. The advantages include scalability, specialization of work performed on individual node, and distribution of node across wide geographic regions. Distributed simulation provides the required computational power that otherwise may not be present using a single computer system. Heterogeneity could be supported with respect to vendor or with respect to simulation components. With simulation being distributed over a wide geographic area, there is no need to fly in users to the same location. With many nodes taking part, different layers of security could be applied based on nodes.

The first distributed simulation protocol developed was called SIMulator NETworking (SIMNET) [5]. SIMNET was a distributed simulation program that was initiated in 1983 by Defense Advanced Research Projects Agency (DARPA) [3]. All the distributed simulation protocols present are based on SIMNET [5]. SIMNET was used for military training in a virtual environment. Vehicle simulators having real controls, with a computer to control network interface, and vehicle dynamics were connected to each other via a network. Each simulator also has a computer to display visual images. SIMNET was running on a LAN. The simulators would exchange messages which were of

many types. The messages format was defined in SIMNET standard. The standard also included algorithms for dead reckoning which is commonly used in distributed simulations. SIMNET decreased the cost of stand-alone simulations which were being used previously by the military [3]. SIMNET also helped to enable interaction of different simulation such as that of tanks, aircraft, and warships. SIMNET led to the beginning of standardization for implementing distributed simulation through Distributed Interactive Simulation (DIS). DIS is an IEEE standard that aid in implementation of a distributed simulation. Many concepts of SIMNET have been used in DIS. Many simulators running on different nodes provide a single virtual environment with each node being responsible for one or many simulation objects that interact with one another through communication medium such as LAN. Aggregate Level Simulation Protocol (ALSP) was developed in 1990's which allowed interaction of legacy military simulations over both LAN and WAN. In 1996, High level architecture (HLA) was developed which added better support for implementation of distributed simulations. Distributed simulation has been used in areas other than military. Distributed Interactive Virtual Environment (DIVE) was developed that provided a virtual world and user who can interact with each other. CORBA, Web services have also been used in implementing distributed simulations.

1.2 Distributed Interactive Simulation interoperability protocol

Distributed Interactive Simulation (DIS) is a protocol standard developed to aid simulation that is spread over several nodes such that the nodes collaborate to achieve a global system objective. DIS is a combined effort by the government and the industry to enable integration of simulations spread across a wide geographical area such as WAN or on LAN. The lessons learned from SIMNET were incorporated in DIS [5]. DIS was an open standard. The distributed interactive simulation represents a simulation exercise which has many simulation entities on many nodes interacting with one another [6]. Each node hosts one or many entities and is interconnected using a network. Each node has the capability to host an entity and the program logic, algorithms, data-structures, and resources to simulate the entity. The nodes exchange data and information using messages, termed as Protocol Data Units (PDUs) and communicate over network using the network protocol

to simulate an exercise. Data related to entities, activities, and events on each node is transmitted to all the participating nodes. The byte by byte format of the PDUs has been defined in [7]. The messages are broadcasted using TCP/IP or UDP/IP protocol [5]. The PDUs, rules for transmission of PDU, data to be populated in PDU, data format, data placement, and algorithms to be used are presented in DIS standards. Each node has a common understanding of the world co-ordinate which is standardized to WGS84 which aids in providing a representation of fixed world co-ordinate and hence a common spatial reference across all nodes. DIS standards allow models and simulations of different vendors with proprietary protocols to interact with one another [6]. DIS aids in developing virtual worlds for training individuals, testing equipment and research, and development. Though most DIS system have been developed as defense applications, it is not required to be so. One non-defense DIS application has been described in [8]. They used DIS for simulation of a baseball game. An air Traffic Control training system using DIS was developed by the MITRE Corporation [9]. Computer Gaming systems have also been developed using DIS [9].

Development of DIS started within a group of engineers and scientists at a research laboratory [5]. The progress of development was reported to bi-annual conference style workshops. The insights of the members were collected from these workshops to be included in the DIS standard. Workshops produced a DIS vision document which states the goal of DIS as "The primary mission of DIS is to define an infrastructure for linking simulations of various types at multiple locations to create realistic, complex, virtual 'worlds' for the simulation of highly interactive activities. This infrastructure brings together systems built for separate purposes, technologies from different eras, products from various vendors, and platforms from various services, and permits them to interoperate. DIS exercises are intended to support a mixture of virtual entities (human-in-the-loop simulators), live entities (operational platforms and test, and evaluation systems), and constructive entities (wargames and other automated simulations)" [10]. IEEE approved the standard and designated the number 1278 [5]. Simulation Interoperability Standards Organization (SISO) maintains and updates DIS standards in accordance to IEEE rules and procedures. IEEE 1278.1 and 1278.1a standards defines the DIS PDUs, their structure and usage, which is used in events such as motion

of an entity, a fire exchange between two entity, and detonation. It also defines algorithms to use such as dead reckoning algorithm, world co-ordinate system, architecture, and the rules of issuance of PDUs. IEEE 1278.2 standard provides the details related to the communication architecture. It provides the different classes of services as discussed in prior paragraphs that DIS PDUs could be used with. IEEE 1278.3 standard provides details related to exercise management and feedback. It also provides the significant details related to verification, validation, and the accreditation process. Session management for a DIS exercise has also been detailed in this set of standards. IEEE 1278.4 standard provides the details related to the verification, validation, and the accreditation process.

Data and control communications are supported. Data communication may include voice, video or images [11]. Different PDUs have been defined to represent and achieve different purposes. Broadcast, Multi-cast, and unicast may be used for communicating PDUs with distributed entities. Broadcast and Multi-cast are the most commonly used types of communications. These are best-effort services. UDP protocol is used to achieve best effort multi-cast or broadcast. Best effort signifies that communication medium defines the reliability and no effort is made by the application or the node for the PDU to be delivered across the destination. Reliable Uni-cast is also supported which provides guaranteed delivery and no duplication, but is seldom used. The entities that are being simulated are present on a node in LAN and can be added or removed from a simulation exercise. Multi-cast or broad-cast of PDUs on the LAN result in a lot of network traffic. The traffic is both outbound from a local entity and inbound from other remote entities. A system must be able to process simulation of one or more entities which it hosts and the incoming traffic from the remote entities. The load determines if the system is able to process it in an acceptable way without degrading the usability of the system by the users. The network throughput and hence the load on the network will determine the number of PDUs, entities, and nodes that can constitute a simulation exercise.

Three classes of services have been defined in [11] are

1. Class 1- Best Effort Multicast

2. Best Effort Unicast

3. Reliable Unicast

Broadcast is considered as the minimal form of best effort multicast. DIS uses the OSI reference model for the network. The reliable unicast is implemented using TCP/IP and best effort unicast is implemented using the UDP/IP protocol as defined by the communication profiles in [11].

The DIS protocol standard provides around 29 protocol data units which can be used by the participating entities. The PDUs work on top of the UDP/IP Ethernet frames and can use either broadcast or multicast mode of operation. The DIS PDUs are divided into the following categories as per [1] and [7]:

1. Entity Information/Interaction
2. Warfare
3. Logistics
4. Simulation Management
5. Distributed Emission Generation
6. Radio Communication
7. Entity Management
8. Minefield
9. Synthetic Environment
10. Simulation management with reliability
11. Live Entity Information/Interaction

The Entity Information/Interaction category consists of PDUs:

1. Entity State PDU (ESPDU): The Entity State PDU is the most frequently used DIS PDU. It provides information about the state of an entity such as a battle tank or an aircraft. The information that can be transmitted using the Entity state PDU includes velocity, location,

orientation, appearance, capabilities, entity type, and articulation parameter. The structure of the ESPDU is as shown in Figure 1.2. The first column provides the number of bits the field occupies. The second column provides the field name and its subparts. The field PDU Header is made up of 96 bits and contains the subparts such as protocol version, exercise id with number of bits as indicated in the figure. Similarly, placeholders for other information such as entity location, entity velocity, and the number of bits they occupy are specified. Entity state PDU has in total bits given by 1152 + 128 * number of articulated parts. The most frequently sent PDU in DIS exercise is the Entity State PDU [12].

2. Collision PDU: This PDU is used to communicate collisions between entities. The PDU contains data which notifies the issuer of this PDU, the colliding entity, type of collision, velocity at the time of collision, mass of the issuing entity, and location of the collision with respect to the colliding entity. The PDU has in total 480 bits.

3. Entity State Update PDU: The update of a state of the entity consisting of non-static information is provided using this PDU. The information it represents includes linear velocity, location, appearance, orientation, articulation parameters. This PDU has in total bits equal to 576 + 128 * Number of articulated parts.

The Warfare protocol family has the following PDUs:

1. Fire PDU (FPDU): The PDU represents the firing of a weapon. PDU transmits location, range, velocity, target entity ID, and firing entity ID. The PDU has in total 768 bits.

2. Detonation PDU (DPDU): The impact of munitions is provided by the Detonation PDU. A few of the attributes that it represents are firing entity, target entity, velocity of munition before impact, location in world co-ordinates, location in entities co-ordinates, result of the detonation, and articulation parameters. The PDU has in total bit equal to 832 + 128 * Number of articulated parts.

Bits	Entity State PDU Fields	
96	PDU Header	Protocol version- 8 bit enumeration Exercise ID- 8 bit unsigned integer PDU Type- 8 bit enumeration Protocol family- 8bit enumeration Timestamp- 32 bit unsigned integer Length- 16 bit unsigned integer Padding- 16bit unused
48	Entity ID	Site- 16bit unsigned integer Application- 16bit unsigned integer Entity- 16bit unsigned integer
8	Force ID	8 bit enumeration
8	No. Of articulation param	8 bit unsigned integer
64	Entity Type	Entity kind- 8 bit enum Domain- 8bit enum Country- 16bit enum Category- 8bit enum Subcategory- 8bit enum Specific- 8 bit enum Extra- 8bit enum
64	Alternative Entity Type	Entity kind- 8bit enum Domain- 8bit enum Country- 16bit enum Category- 8bit enum Subcategory- 8bit enum Specific- 8bit enum Extra- 8bit enum
96	Linear velocity	X component- 32 bit floating point Y component- 32 bit floating point Z component- 32 bit floating point
192	Location	X component- 64 bit floating point Y component- 64bit floating point Z component- 64 bit floating point
96	Orientation	Psi- 32 bit floating point Theta- 32 bit floating point Phi- 32 bit floating point
32	Appearence	32 bit enumeration
320	Dead reckoning parameter	Algorithm- 8bit enumeration Other parameter- 120bit reserved Linear acceleration- 3 x 32bit floating point Angular velocity- 3 x 32 bit floating point
96	Entity marking	Character set- 8bit 11 8-bit unsigned integer
32	Capability	32 boolean field
128n	Articulation Parameter	parameter type designator- 8 bit enum Change indicator- 8bit unsigned integer ID- Part attached to- 16bit unsigned integer Parameter Type- 32bit parameter record Parameter value- 64bit
	PDU size= 1152 + 128n bits Where n= Number of articulation parameter	

Figure 1.2: Entity state PDU format. Adopted from [1]

The Logistics protocol family includes the following PDUs:

1. Service Request PDU: The PDU is used to request logistics support and includes attributes describing the type of service requested, the number of supplies, and the supplies. The PDU has bits equal to 224 + 96 * Number of supply types.

2. Resupply offer PDU: The supplies that have been offered are transmitted using this PDU. The attributes include the supplying entity, the receiving entity, number of supplies, and the supply itself. The PDU has bits equal to 224 + 96 * Number of supply types.

3. Resupply Received PDU: The PDU represents a token to the receipt of the resupply and has 224 + 96 * Number of supply types bits.

4. Resupply Cancel PDU: The receiving or the supplying entity can cancel the previously issued resupply request using this PDU and has bits equal to 192.

5. Repair Complete PDU: The PDU is used by the repairing entity to notify that the repair request has been completed. The PDU has 224 bits.

6. Repair Response PDU: The PDU is an acknowledgment for repair completes PDU and has 224 bits.

The Simulation Management protocol family has PDUs that are used to manage the simulation exercise. Some PDUs of interest in this family are

1. Create Entity PDU: The PDU is used to communicate the creation of a new entity which has attributes identifying the receiving and originating entities. The PDU has 224 bits.

2. Remove Entity PDU: The PDU is used to inform the removal of an entity from the simulation exercise. The PDU has 224 bits.

3. Start/Resume PDU: The start/resume of an entity or exercise is denoted by the PDU. The attributes include simulation time, real world time, and the originating entity identification. The PDU has 352 bits.

4. Stop/Freeze PDU: The stop/freeze of an entity or the exercise is communicated using the PDU. The attribute includes a reason field that explains the reason for the entity to stop or freeze. The PDU has 320 bits.

5. Acknowledge PDU: The PDU is used to acknowledge the Create entity, remove entity, start/resume, stop/freeze PDUs, and has 256 bits.

There are many other simulation management PDUs such as Action request PDU, Action response PDU, Data query PDU, set data PDU, Data PDU, Event report PDU, and Comment PDU.

The Distributed Emission Regeneration protocol family includes the following PDU:

1. Electromagnetic Emission PDU: The PDU is used to transmit detail of Active electronic warfare emissions, and active EW countermeasures.

2. Designator PDU: The PDU is issued to designate operations.

3. UA PDU: The acoustic emissions are transmitted using UA PDU.

The Radio Communication protocol family PDU includes

1. Transmitter PDU: The information about radio transmitter is communicated by issuing a transmitter PDU.

2. Signal PDU: The PDU is used to transfer data. The data can be voice, audio, image or any other data. The PDU has 256 + data size + 0-31 padding bits.

3. Receiver PDU: The PDU determines the state of the receiver. The PDU has 288 bits.

The Entity Management protocol family PDUs includes

1. Aggregate state PDU: The PDU provides details about the aggregate entities and information about communication of these aggregates. The PDU provides information related to the formation of aggregate, type of aggregate, state of aggregate, dimensions, orientation, velocity, and the details of the entities in aggregate.

2. IsGroupOf PDU: The PDU transmits details of a group of entities.

There are many other PDUS in this family such as Transfer control request PDU, IsPartOF PDU.

The Minefield protocol family provides PDUs Minefield State PDU, Minefield Query PDU, Minefield Data PDU, and Minefield Response NACK PDU. The Synthetic Environment protocol family consists of Environmental Process PDU, Gridded Data PDU, Point Object State PDU, Linear Object State PDU, and Areal Object State PDU. The Simulation Management with reliability protocol family consists of Create Entity- R PDU, Remove Entity-R PDU, and so on similar to the normal simulation management PDUs.

Every DIS PDU consists of the PDU Header. The information in the PDU header is used by the receiving node to interpret the content that follows appropriately. It provides details such as DIS protocol version that is being used, the identification of the exercise the PDU belongs to (note that many exercises could be occurring on the same network simultaneously), PDU type, timestamp at which the PDU originated, the length of the PDU, and the family of protocol used. The enumerations for protocol version, PDU types, and protocol family have been defined in a separate document maintained by Simulation interoperability standards organization [13]. A value of 6 for protocol version identifies the IEEE 1278.1a DIS standard. A value of 1 for PDU type identifies the PDU to be ESPDU and a value of 1 for protocol family identifies the ESPDU to be belonging to category Entity Information/Interaction. The ESPDU fields such as force id have also been enumerated in [13].

Every change in position or orientation of an entity need not be transmitted as a PDU to all other nodes. Dead reckoning allows estimating the position and orientation of the entity reducing the number of PDUs required transmitting over the network. [1] provides a detailed explanation of the dead reckoning calculations and usage. If the difference between the estimated value and real value exceeds a certain threshold agreed upon, only then an update is sent to the network. [1] has specified that at a minimum the state of the entity must be updated to all the nodes every 5s.

1.3 Discrete event simulation

Discrete event simulation (DES) is a technique to model systems at discrete points in time. Systems are present in specific state at these points in time. The system is modeled by progressing simulation time in discrete steps. The state of the modeled system changes at points in time represented by simulation time. The model includes entities such as system resources, customers that use the system, and the events or activities in the system which resulted in the change of state in the system. These models are suitable when the system being modeled, changes state at discrete points in time.

Discrete event simulation models a system which has a collection of interacting entities such as people and machines [14]. A model is defined for the system which is a representation of the system, its entities (and their attributes), processes, events, activities, and delays. System has state variables which describe the state of the system at any instant in time. The objects such as customers, and servers are termed entities which are explicitly represented in the system. Generally, the objects such as customers are added to a list in the system which orders the entities in some predefined manner. An event is said to occur when certain activity changes the state of the system, such as arrival of a new customer. The event could be modeled in discrete event simulation by an event notice that provides details related to type of event, the time at which the event will occur (current or future), and any data that is needed for the execution of the event. The event notices are held in a list called the future event list (FEL) which is generally ordered by time of occurrence. The discrete event simulation consists of a clock that maintains simulation time and is not the same as current wall clock time. The lists or queues that hold the entities are ordered by some rule such as first in-first out or last in-first out depending on the system that is being modeled. Entity service time, entity interarrival time, and such measured period of time represents the activities in the system. The duration of the activities can be either specified deterministically or stochastically or by using a function dependent on variables in the system and entities. An event notice is scheduled by computing the duration it takes for the activity to complete from the time the activity begins. For example, if the current simulation time was 500 seconds and an activity called "customer service"

with time equal to 13 second begins, then an event notice of "customer service completion" is added to FEL with time 513 seconds to schedule the execution of that event. Contrasting activities in the discrete event simulation are the delay which are generally the unknown and are the desired measured outcome of the simulation. For example, the time the customer waits in a queue before he gets serviced by a server. This delay is dependent on the system state and factors present in the system. The activity completion is modeled in the system by placing an event notice in the FEL whereas completion of the delay is represented by putting the entity in a queue until the server is available. The state of the system, attributes of entity, FEL, simulation time, queues, activities, and delays are all dependent on time and, hence changes.

Entities enter the queue when the server is busy and thus it is unable to process the entity. A few of the measures of interests such as "wait time" of the entity in the queue determine the performance of the system and, hence generally an expected output of the simulation [3]. Along with wait time, factors depending on the reason the model is being developed could be evaluated such as maximum wait time, maximum number of entities in the queue at any point in time, time spent by the entity in the system, service utilization percentage, soon, and so forth. The answer for these questions will be useful to satisfy entities such as customers and will also aid the management of the system to improve performance without incurring any loss.

The model that is developed need not be an exact replica of the system. Sufficient details are incorporated to answer the specific questions. Discrete event systems work on principle of Event scheduling/Time advance algorithm. The basis of this algorithm is the FEL [14]. The FEL consists of the event notices which are ordered according to event time. In each iteration, notice is removed from the top of the FEL and the associated event is executed. The event notices for the events could be added if the time at which the event is supposed to execute is known or by using a random draw from a statistical distribution to derive the time. Let's say that the FEL has event notices for events in the order of time-

$t < t_1 <= t_2 <= t_3 <= .. <= t_n$

Here, t is current simulation time. Once the event associated with t is executed, the simulation time is advanced to next event time in the ordered FEL. The event notice is retrieved from the FEL, and the associated event is executed. The execution of this event may result in generation of more events which would be inserted in FEL as event notices at appropriate positions. Once, this process of execution is complete, the next event notice is removed and the associated event is executed. The process continues. To start the process, at least one event must be scheduled at time zero by placing an event notice in FEL. This process is also termed bootstrapping. Events could be associated with activities such as start and completion of service, and corresponding notices placed on FEL. This technique could be used to simulate a server and collect data/statistics related to a server. The simulation ends when there are no events remaining in FEL to execute. Simulation can also end when a predetermined simulation time is exceeded.

1.4 CPU and Network modeling

This section describes the necessary background required for CPU and network modeling. Section 1.4.1 provides the modeling of a CPU and network in general, Section 1.4.2 considers the specifics for the DIS systems, and Section 1.4.3 provides a literature survey about CPU and network modeling.

1.4.1 CPU and Network modeling, in general

Every computer consists of a central processing unit (CPU) which is responsible for the execution of processes which are collection of instructions. A portion of the CPU time is provided to an application whose instructions are executed. The application blocks when input/output operations are encountered or the time slice associated with it expires. A discrete event simulation model can be developed for the CPU. The instructions associated with the application object that the CPU processes are customers. The CPU which processes these instructions is the server. The time slice is represented as service time which is associated with every object the CPU processes. Modern multi core CPU can be considered to be made up of multiple servers, who would process the application

object when they are available. The application object arrives at the CPU it needs to be processed. If the CPU is busy, then the application object will have to wait to get the portion of the CPU time it deserves.

The queuing models use Kandall's notation-

$A/B/c/N/K$,

Where A: interarrival distribution, B: Service time distribution, c: Number of servers, N: System capacity, and K: size of calling population [14].

For a CPU, it can be represented as M/M/1, where A=B=M: exponential interarrival time and service time distribution with a single server. The service times varies as per the type of instructions being executed and the inter arrival time varies based on the number of applications contending for the CPU. N and K are skipped to denote unlimited system queue capacity and infinite potential arrivals. The network modeling considers the type of network such as Ethernet and token passing, and the link speed specified in mega bits per second (Mbps). A detailed explanation on modeling Ethernet based network is provided in Section 4.2.

1.4.2 CPU and Network modeling, for DIS system

In a DIS system many nodes are interconnected through a network. The node hosts entities which are termed as local entities to that node. Generally, the entities are responsible for maintaining its state and logic. This is accomplished by computer instructions associated with the entity. The CPU time must be provided for execution of these instruction associated with an entity at a defined frequency. The term "tick" is used to define the process of giving CPU time to an entity to execute its instructions. The frequency at which the entity must be given the CPU so that it can execute its instructions is termed as tick rate. Every node has local entities that must be ticked at the frequency of tick time. The tick of an entity corresponds to allocation of CPU to execute the particular entity's code. Hence, the discrete event model for the DIS system must include service time for an individual entity. The entities arrive at the CPU at the interarrival time given by the tick time. If the CPU is busy, then the entity has to wait in a queue for its turn so that it can be

executed by the CPU. The service of the entity may result in the generation of network traffic as PDUs that must be transmitted to all other nodes on the network. The network traffic due to the remote node entities arrive and must be handled by executing some more code by the CPU. Hence, the entities, and incoming DIS PDUs become customers to the server, namely, the CPU. The entire system is synchronized using a common clock. Discrete event simulation can be used to model the system. The network is connected to the computer by a network interface card (NIC), and network modeling would be no different from normal but should ensure that the transmitted PDUs arrive at all nodes, so that they can interpret and respond to events appropriately. The details of network modeling for Ethernet are provided in Section 4.2.

1.4.3 Literature survey for CPU and Network modeling in DIS

The network traffic generated is dependent on the scenario that is being simulated. Simulation of a vehicle entity moving on a terrain will obviously generate less traffic than two battle tanks engaged in active conflict. This fact is explored in [9]. It measures traffic rate in bits/sec and PDUs/sec. The rate in bits/sec is determined by physical bandwidth of the medium and the rate at which interfaces can process data at the physical level. The maximum rate of PDUs/sec determines the number of entities that each node can track and update its display devices without any glitches. [15] considers two cases: a. All the PDUs except the Signal PDU which is made up of voice, audio, and other data and b. Signal PDUs. If 'α' is the fraction of entities involved in a conflict generating PDUs at a higher rate and if 'β' is the fraction of the entities producing lower rate of PDUs due to no involvement in any conflicts, then the rate in bits/s for the standard DIS traffic is given by

$$R_b^s = \alpha \left\{ \sum_{i=1}^{4} n_i r_{ij}^h p_{ij} \right\} + \beta \left\{ \sum_{i=1}^{4} n_i r_{ij}^l p_{ij} \right\}$$

Where, $n1$, $n2$, $n3$, $n4$ are the numbers of aircraft, ships, submarines, and weapons (missiles, torpedoes) respectively; r_{ij}^l, r_{ij}^h the low/high rates for entity i for PDU j and p_{ij} the size in bits of PDU j for entity i. Also, [15] provides rate in PDUs/sec and the traffic considering signal PDU respectively as

$$R_p^s = \alpha \left\{ \sum_{i=1}^{4} n_i r_{ij}^l \right\} + \beta \left\{ \sum_{i=1}^{4} n_i r_{ij}^l \right\}$$

$$R_p^{vd} = \alpha \left(n_5 V_l + n_6 D_l \right) + \beta \left(n_5 V_h + n_6 D_h \right),$$

where V_l, V_h, D_l, D_h are the rates in PDUs/s for low voice PDU rate, high voice PDU rate, low data link PDU rate, and high data link PDU rate.

A program, TRACMARS, was developed in Microsoft Visual Basic, for predicting traffic rates in bits/sec and PDUs/sec. The program had the provision to key in the number of different entities taking part in the simulation and configures the low and high rates of PDU emission. The paper, however, does not consider the possibility that there can be a third level of rate of PDU generation, between low and high. It also does not provide any experimental validations to show if the rates actually match the estimated rates of emission for any particular scenario.

ModSAF is used to experimentally measure the scalability of the DIS in [16]. [16] Claims that the standard model of DIS network traffic that predicts linear increase of network traffic with the linear increase in the number entities holds false. This is due to the fact that entities do not involve the same level of activities and hence do not produce the expected linear traffic. It used land warfare scenarios with Red/Blue assets on each side. The logged DIS traffic was analyzed using tools provided by MAK Technologies, UNIX utilities, and excel. The simulation involved US M1A2 and Soviet T72 tanks. The scenario used was located in the Standard Fort Knox terrain database. The tanks advance in 'hasty occupy' order. The two units engage when they see the opposite teams. The scenario was run till T72 tanks were destroyed by M1A2 tanks. Scenarios consisted of variation in number of tanks, starting from one M1A2 and one T72 to a platoon of M1A2 and T72. The scenario was run on LAN with Silicon Graphics incorporated Onyx systems. The analysis showed that the majority of the network traffic is by ESPDUs. An individual entity generates PDUs at different rates over its life time depending on its interaction. The network traffic increases linearly for silent, non-collision periods of activity, and increases in an unpredictable manner during intense engagement between entities. There also exists a burst in the network traffic in the initial period when the simulation is started. Our model simulates a particular action in the battle of 73 Easting which provides a more realistic simulation of entities.

Manned simulators and Semi-automated forces are used to characterize DIS traffic in [12]. The traffic pattern is determined to be uniform for ESPDUs which comprises over 90% of the DIS traffic. The analysis is performed using 1993 Interservice/Industry Training Systems and Education Conference (I/ITSEC) data where 44 organizations gathered for a period of two weeks to conduct DIS interoperability demonstrations. DIS data amounting to 2.8GB was logged and analyzed. The performance metrics used for CGF and manned simulators are issue rate of Entity state PDU per second. Stationary and constant velocity entities were compared. Manned and CGF entities were also compared with respect to traffic generated. Land, air, and surface entities were used for this exercise. The activity that was considered was a ground battle and air entities firing on the ground entities. The results showed that traffic from CGF entities are regular and predictable, and easier to generalize than manned simulators which are more dependent on fidelity. The paper focuses on categorization of DIS PDUs and presents the results that 90% PDUs are ESPDU. The use of DIS on a wide area network is considered and traffic characteristic for DIS exercises are analyzed in [17]. The draft Communication Architecture for DIS (CADIS) Guidance document is used to achieve this categorization. Entity states PDUs are considered the majority type of network traffic. The traffic increases at the beginning of the exercise reaching a peak and then decreasing. The emission PDUs also contribute to the impact on the network bandwidth. Network utilization was computed which determines the network capacity for the DIS exercises. The Ethernet utilization did not exceed 6% with PDUs generated at a rate of 60 PDU/s. A network flooding experiment produced 55% utilization with around 800 ESPDUs per second. The results were arrived at based on the collected experimental logged DIS traffic.

DIS traffic is categorized as self-similar and Poisson type by [18] using the unclassified network traffic data from the Synthetic Theater of War-Europe initiated by Advanced research projects agency (ARPA). The LAN traffic has a period of burstiness and smoothness whereas the WAN traffic has more burstiness on larger time scales. A high Hurst parameter is observed when compared with an asymptotically self-similar process. The STOW traffic is predicted to behave as a asymptotically modulated self-similar process. The analyzed data originated from a STOW-E

exercise conducted in 1994 November with sixteen sites in the USA, Germany, and England. Defense Simulation Internet (DSI) WAN with T1 links was used for the network. The application itself was hosted on LAN, and communicated using application gateways.

Instrumentation of application code with performance monitoring code has been used to monitor the DIS performance in [19]. The performance limits on CPUs and network are predicted using the monitored data. The monitoring method used is purely software and does not use any additional hardware. The PerfMETRICS tool is developed that provides instrumentation and monitors the DIS traffic and CPU performance. PerfMETRICS is used with Modular Semi-Automated Forces (ModSAF) to monitor performance. PerfMETRIC has also been used with STOW ACTD (Advanced concept technology demonstration). GUI is provided to display collected performance data. A cost is associated with instrumentation which reduces the capability of the number of entities that are simulated on a node. Performance with respect to entities - Kinematic Model, Outgoing and Incoming PDUs, Sensor Model, Weapons model, Behavior Model and Graphics, Simulation - Scheduler slack time, process time, Update performance, and Scheduler wait time, and System - Network Traffic, CPU Utilization, IPC Primitives and Memory utilization- is collected and displayed.

An experimental approach has been taken in [20] to analyze network requirements for a DIS exercise. [20] Used experiments involving land and air vehicles with radio communications in a scenario totaling to 1000 entities. The scenario had both manned and unmanned entities. The analysis assumed voice packets of 20 PDU per second with 8 bits 1 KHz sampling rate. The analysis showed that the average bandwidth of approximately 12MBits/sec with an average throughput of 3270 packets per second is required for the exercise. [20] also notes that with throughput of packets results in higher interrupt and processing requirements of individual nodes. [20] suggests mechanisms of multicast group addressing and dynamic multicast functions to solve the problem.

[21] Addresses an interesting issue of low and high resolution modeling. The high resolution modeling models at an individual entity level whereas the low resolution modeling considers the aggregation of lower level entities. The number of entities and hence the PDU rate is high for

the higher resolution than the aggregated models. [21] Identifies that in the case of high resolution modeling, simulation may slowdown or crash networks due to overloads are problems.

Core based trees and protocol independent multicasting protocols have been used with DIS. The performances of these protocols have been analyzed using the OPNET network simulation tool in [22]. A DIS multicast group may have large number of senders and receivers, and there may be 20000 of these multicast groups. With the large network traffic in DIS, analysis of CBT and PIM protocols have been based on DIS. This implies the importance of network analysis in an exercise based on DIS on networks.

Recognizing the importance of network in DIS, [23] develops techniques to reduce the required communication bandwidth and latency. Logged PDUs from a scenario are used for analysis and are not randomly generated. It is observed that the Entity state PDU affects the network traffic in a prime way with 47.3% of PDUs being them. The goal of the research was to estimate bandwidth requirements for wireless channels for communication between stations. The results show that at-least 256Kbps bandwidth is required in JEMPRS (a wireless flying LAN based on Joint Forces Commands Joint En route Mission Planning and Rehearsal system) network to sustain exercises.

[24] addresses the issue of latency when a DIS exercise is undertaken across large geographical areas over wide area networks (WAN). Utilities such as ping and traceroute are used to make the experimental measurements and analysis. The DIS exercise using WAN can be carried out in Australia with latencies of less than 100ms and the same is the case with countries that are nearby to Australia.

In summary, the DIS network and CPU performance of each node is a significant factor in DIS systems. Many attempts have been made to analyze and predict network traffic. All the work seen in this section uses either gathered data of a known simulation exercise or uses a synthetic scenario to relate DIS nodes and entities to network traffic. The relation is provided either in terms of bits/s or PDUs/sec. The network traffic as a function of nodes, entities, and DIS PDU type is observed. Our research work tries to create a discrete event model of the DIS system and predict network performance in terms of bits/s as number of entities and nodes increases. This eliminates any

requirement of observing, collecting and analyzing large amount of data from a known or synthetic DIS exercises.

Not much work has analyzed CPU performance of nodes in a DIS system. [19] is the only work that we came across while doing literature review that deals with analyzing CPU performance. They use instrumentation to observe and display various performance measures of a node by running DIS exercises. Our research work tries to create a discrete event model of the nodes in the DIS system and predicts the performance of the CPU in terms of *tick time*. As the number of local entities on a node increases, the tick time (maximum of the mean tick time values of every entity) is predicted. The respective DIS system can then decide the threshold tick time and hence the number of entities on that node which is acceptable for their exercises. This removes any need of instrumenting the application to measure CPU performance.

CHAPTER 2

RESEARCH STATEMENT

This chapter provides the necessary information that led to the beginning of this research. Section 2.1 provides the motivation and objectives that led to this work. Section 2.2 explicitly identifies the questions that this work tries to answer.

2.1 Motivation and objectives

As noted in Chapter 1, PDUs are used to communicate data and information among participating nodes in a DIS exercise. Certain PDUs are used more often and frequently than others [12]; these are the Entity state PDU which defines the current state of an entity in terms of position, orientation, velocity, etc., the Fire PDU which denotes an entity firing on another entity, and the Detonation PDU which denotes an entity in the simulation being hit by weapons fire from another entity. The ESPDU also has a defined heart beat time at which the ESPDU must be broadcasted to all the nodes irrespective of any changes in the state of the entity.

Often, DIS semi-automated forces (SAF) systems such as VR Forces use the concept of a tick [2]. The term "tick" is used to define the process of giving CPU time to an entity to execute its instructions. The frequency at which the entity must be given the CPU so that it can execute its instructions is termed tick rate. On a configured tick time interval, the application tick's, meaning, processes every entity present on that node. The lower the tick times, the higher the number of times the entity is processed. The tick is accompanied with an update to the entity state. Depending on the type of the application, every tick can also be accompanied with an ESPDU being sent over the network with the updated data or as in VR-Forces, a separate network tick is configured that

determines the sending of the ESPDU. As the number of entities increases, the load on the CPU to process the ticks adhering to the tick time increases. Also, the CPU must process ESPDUs received from the network for entities modeled on other nodes. As number of entities increases, the amount of DIS PDUs also increases, especially ESPDUs, increasing the load on the CPU to process the entities on that node and the incoming DIS PDUs. With the increase in the DIS PDUs, the network usage also increases.

As load on CPU increases, the tick rate decreases. A research goal was to develop a model that predicts the number of entities for a given tick rate at which node is overloaded so that the real tick time is no more in acceptable limits. As the tick time degrades and is greater than the configured tick time, the entity state is no longer updated at the desired frequency possibly reducing the quality/fidelity of the simulation. At that state, the node is overloaded due to the combined load of local entities modeled and incoming PDUs. A load is the sum of CPU load to simulate local entities and processing network traffic arriving over the network. A DIS system is said to be overloaded if the local entities cannot be updated at the rate of the tick time configured (plus defined threshold). We are also attempting to predict the number of nodes and the number of entities in each node that can be modeled without exceeding the network bandwidth of a LAN.

An estimate of the threshold for the number of entities for a node to be overloaded provides a mechanism to estimate number of nodes necessary for a DIS exercise with N entities. It could also be used as a tool by the developer when developing the application to maximize the number of entities that the application can support on a node without it being overloaded. It also provides an estimate of the network bandwidth requirement for exercises aiding in preparation for the simulation exercise.

2.2 Research questions

By creating this model, we are trying to answer the below questions-

1. Can a distributed simulation based on DIS be modeled at all using discrete event simulation?

2. If the answer to the question above is yes, then can a model detect or predict node or network overload in a DIS simulation?

3. Can a model recreate "closely" the DIS network traffic compared to a real DIS simulation?

4. How does DIS network traffic increase when scenario size increases, and can a model predict this?

Here scenario size is defined by the number of entities such as tanks or helicopters involved in the DIS simulation. Even though the processing requirements for each entity vary by application, and over time in an exercise, the scenario size metric is most often used and most widely understood in DIS [25].

We would also like to answer questions such as whether a given DIS system configuration with a given number of nodes and number of entities per node handles battles of size two or three times larger. This question has a direct relation to question 2 above.

We develop a model using discrete event simulation to simulate a DIS system consisting of many nodes spread over a LAN to answer these questions. The model simulates a distributed simulations system running the DIS protocol in a LAN.

CHAPTER 3

VR-FORCES

"VR-Forces" is a DIS application developed by MÄK Technologies used to validate our model predictions. Section 3.1 provides a brief overview about semi-automated forces, also called computer generated forces, of which type VR-Forces belongs to. Section 3.2 describes the VR-Forces tool that is used to develop a scenario for the research purpose. Section 3.3 presents the scenario that is simulated using VR-Forces.

3.1 Semi-automated forces systems

Virtual environments are often used during a modeling and simulation activity [4]. These environments may host entities such as tanks and aircrafts (friendly or hostile). The entities can be generated using computer programs and may be controlled in some manner by a human operator. The computer programs that generate these entities are called semi-automated forces systems and have behavior algorithms embedded in them that model the generated entity in a virtual environment as closely as required to a real world entity. SAF systems incorporate behavior generation algorithms so that the simulated entities behave and act as real entities. The simulation entities model physics and behavior at the required fidelity. They are capable of generating and controlling the entities dynamically under the control of a human operator. They could be used in scenarios involving virtual environments with large training simulators manning real personnel. Training simulators under complete control of human operators interact with the semi-automated forces system. The need for separate training simulators manned by opposing forces could be easily eliminated by using SAF to control the opposing forces on a computer system connected to the training simulators over

a network. This reduces the cost of the machine required and the manpower to execute a scenario to train military personnel. SAF systems have been programmed to use different tactical doctrine as required which would eliminate the training cost for the human operated opposing forces. They can also generate friendly forces in large quantities that could be useful in training personnels to work with a larger friendly force. Being able to generate and manage opposing or friendly forces in large numbers allows better control and management by a single or few SAF operators than real simulations with real people of the same size.

3.2 VR-Forces tool

VR-Forces is a semi-automated forces software package developed by MÄK Technologies. VR-Forces is used to simulate battles. It has two components: a graphical user interface and a simulation engine. The graphical user interface (GUI) is the visual component which is capable of displaying the on going battle that is being modeled and simulated. The simulation engine is the brain of VR-Forces which models physics and behavior of entities and is responsible for simulating the scenario defined using protocols such as DIS. The two components are independent of each other and run as separate processes. VR-Forces is compatible to work with the DIS protocol among many others. Many terrain database formats are supported.

Multiple simulation engines and GUIs can run simultaneously on the same or different nodes. With many GUIs running, multiple users have control over the simulation and have the capability to manage the same scenario simultaneously. Running multiple simulation engines aids in achieving the distributed simulation where each node can share the overall load of simulating many entities. The simulation engine is controlled using GUIs. GUI is the only way to interact with the simulation engine. The communication between GUI and the simulation engine takes place over the network through messages which are not DIS PDUs. The PDUs generated by the simulation engine are displayed by GUI. GUI displays various events such as motion of entities on GUI. [2] shows the interaction between simulation engine and GUI.

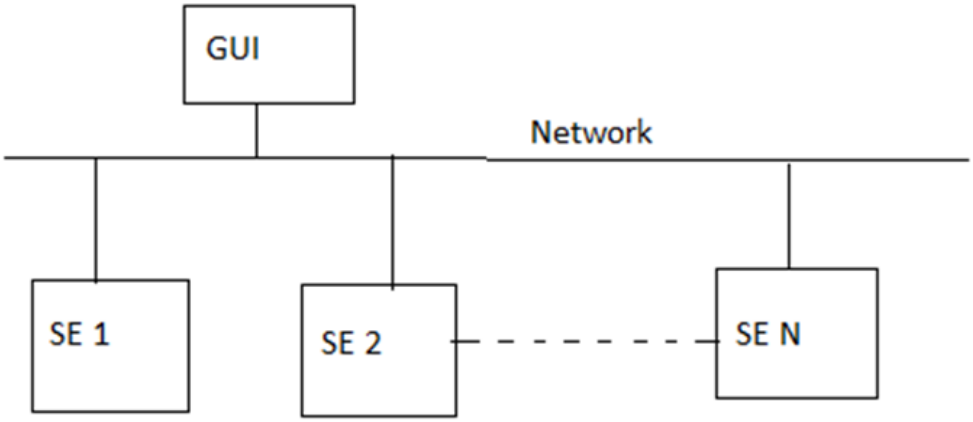


Figure 3.1: VR-Forces organization

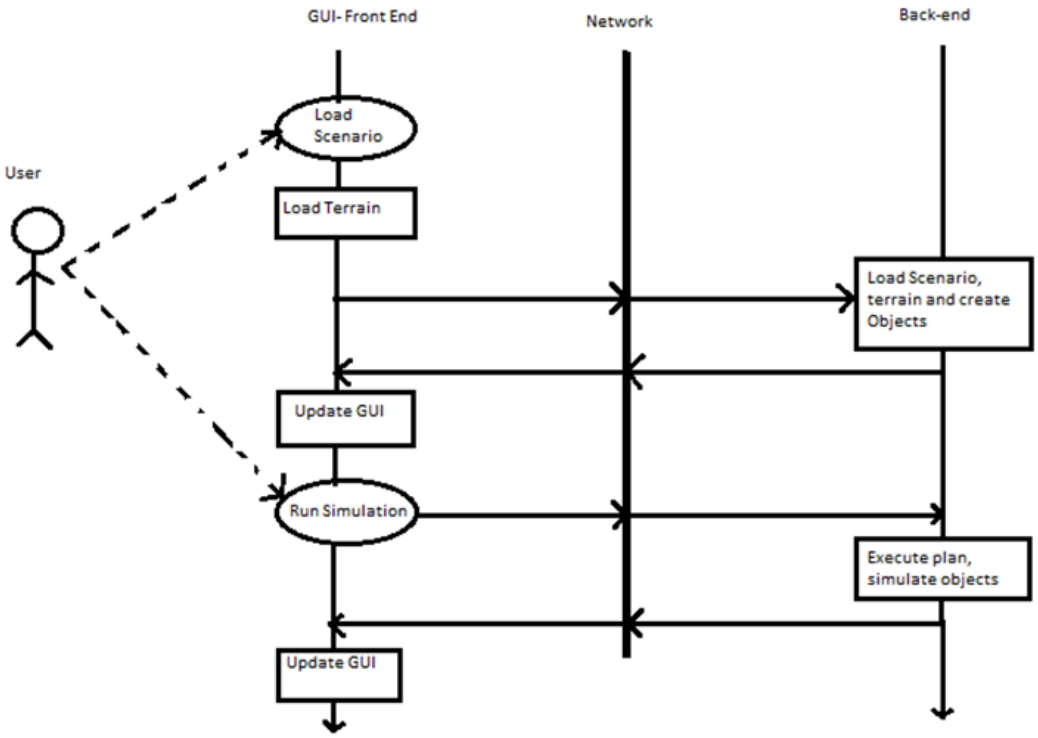


Figure 3.2: VR-Forces modules interaction. Adapted from [2]

The site id and application number fields are used to uniquely identify the GUI and simulation engines. The fields are also available in the PDUs to be populated with. Each GUI and simulation engine belongs to a session, which is provided by session identification. Multiple sessions can be in progress at the same time, but the GUI belonging to its session has control over the simulation engine belonging to that session. The different sessions do still belong to the same exercise. It is required that all the simulation engines are up and running when a session begins in the exercise. A new simulation engine cannot join in late to an already running session. Even if any of the simulation engines shutdown due to some error, the whole exercise must be restarted, as it cannot join in and catch up to the remaining simulation engines. A terrain database is associated with every scenario which has information about the geographic area in which entities will operate.

VR-Forces provides direct operator control of entities which are simulated. When a simulation is in process, right-clicking entities allow to assign tasks. The "Set" menu allows for assigning tasks to entities when the simulation is in process. The entity stops its current action and behaves as instructed by the selected command in the menu. The set menu provides instructions to entities such as fire, embark, move to location, change rules of engagement, change speed, and change formation. An entity can also be commanded to be destroyed or to restore full health and resources. The direct control is good for changing behavior or scenario at a runtime when the number of entities is small. VR-Forces is used to simulate a battle comprising 100 entities. A better approach to instruct entities was used by writing plans.

VR-Forces allows writing plans, which are short programs written in a special behavioral scripting language that entities will execute without human intervention. These plans could be instructions to move on a specific route based on certain conditions or it could be to fire at a hostile entity when it enters a specified area. VR-Forces supports the following constructs while writing a plan:

1. If/Else: A conditional expression that decides the block underneath to execute if its value is true. Once it is false, it continues executing the following instructions in the plan.

2. While: A block of statements is executed repeatedly while the condition remains true. Once the condition is false, VR-Forces continues executing the instructions after in the plan.

3. When: These are triggers. Triggers have conditions that when true at any point in the simulation, the instructions in its block will be executed. If and While will be executed when the control arrives at them in the plan, but "When" will execute irrespective of the point at which the plan is currently being executed.

VR-Forces supports many conditions that can be used with the above constructs, but the ones of our interest are

1. EntInArea: is the entity in the specified area?

2. EntLeftOfLine: is the entity to the left of the phase line?

3. EntDestroyed: is the entity destroyed?

The conditions can be combined using the logical operators AND, OR, and NOT. Every condition must be associated with an entity. This is achieved by providing the name of the entity in the condition or by using "self" which will apply the condition to the entity the plan is associated to.

A trigger statement can be placed at any place in the plan. VR-Forces executes the plan in a top down manner and once the "When" construct is arrived at, it registers the trigger. The trigger is valid and known to VR-Forces from the time it is registered. Trigger will be registered at the beginning of the plan. VR-Forces keeps checking the triggers periodically. Once the trigger is executed, it is not executed again. The trigger must be registered again for it to come into effect.

In our simulation using the VR-Forces, we also use the concept of a phase line. A phase line is made up of two points and a directed line joining the two. The line virtually extends indefinitely on either side. The direction of the phase line is the direction in which the line is drawn from the starting point to the ending point. Any entity can be decided to be left or right of this phase line relative to the direction of the phase line. This could be used in conditions when the entity must trigger some action when it crosses the phase line and hence it moves from either left of the phase line to the right or vice-versa.

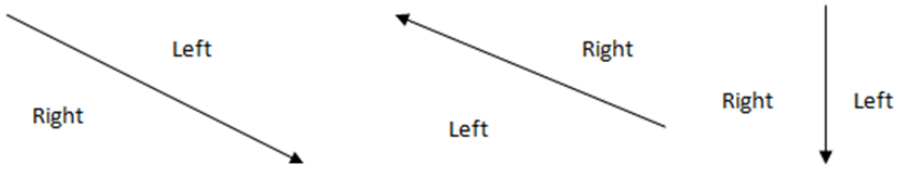


Figure 3.3: Phase lines

3.3 Modeling historical event with VR-Forces

The Battle of 73 Easting which was fought between USA and Iraqi forces is explained in this section. Section 3.3.1 explains the battle of 73 Easting and Section 3.3.2 provides the implementation details in VR-Forces. The battle was chosen as it provides a credible measure of the scenario size.

3.3.1 The battle of 73 Easting

The Battle of 73 Easting was fought between US allied forces and Iraqi forces in 1991; the battle is described in [26]. The name originates from the fact that the battle was fought at 73 Easting, an imaginary north-south line which is 73 kilometers east of an origin point in the Iraqi desert.

The Troops: The 2nd and 3rd Squadrons of the 2nd Armored Cavalry Regiment (ACR) would be involved in the main battle. The squadrons had G Troop (Ghost), E Troop (Eagle), F Troop (Fox), H Troop, and I Troop (Iron) (In the parlance of U.S Army cavalry units, a "squadron" is battalion and a "troop" is a company). G Troop held the left flank of the regiment, E Troop was on right of G Troop. F Troop followed G Troop and H Troop followed E Troop. I Troop was on E Troop's right. G, E, and I troops each consisted of 140 men in four platoons. Six M3A2 Bradley Fighting Vehicles were part of 1st and 3rd Scout platoons and four M1A1 Abrams "heavy" tanks were part of 2nd and 4th Tank Platoons. The Iraqi opposition consisted of the 18th Mechanized Brigade, and the 3rd Tawalkana division of the Republican Guard Corps.

The overall battle was fought in many separate small actions. Action 2 and Action 3 are described as they are relevant with the modeled and simulated scenario. Action 2 and 3 occurred on

February 26, 1991. Action 2 started with the E Troop discovering 39 Iraqi T72s, 14 BMPs and 40 other armored vehicles. E Troop destroyed 8 T72s within few seconds. The I Troop present on the right side of E Troop encountered more T72s and destroyed 8 more. There was no damage caused during Action 2 for the United States Allied forces. Action 2 resulted in loss of 16 T72s to the Iraqi army. Action 3 started with E Troop advancing and encountering 4 T72s and 4 BMPs ahead. E Troop destroyed these Iraqi forces and moved ahead. Parallely, I Troop advanced ahead. E Troop encountered 17 T 72s and destroyed them all. I Troop encountered another formation of Iraqi forces and successfully destroyed 2 T72s. A M3A2 from the 1st platoon encountered an electronic malfunction and moved back from the 73 Easting. The Killer Troop assumed that the M3A2 was an Iraqi tank and fired a TOW missile using its M3A2 destroying the friendly retreating vehicle. Three soldiers were wounded. Action 3 resulted in the loss of 1 M3A2 BFV, 3 wounded soldiers for the United States army, and 24 T72s, 4 BMPs for the Iraqi army.

There are few differences between the modeled scenarios and the real battle of 73 Easting. The VR-Forces provides M1A2 and BMP-2 which was used for simulating Action 2 and 3. M1A1 and M1A2 are different versions of the M1 Abrams Main battle tanks made by General Dynamics Land Systems Division of USA. The first M1A1 was produced in 1985, and the first M1A2 was produced in 1986. M1A2 is upgraded version of M1A1 consisting of all the features of M1A1 and addition of "commander's independent thermal viewer, an independent commander's weapon station with second generation thermal imager; commander's display for digital color terrain maps; second generation thermal imaging gunner's sight with increased range; driver's integrated display and thermal management system, position navigation equipment, and a digital data bus and radio interface unit providing a common picture among M1A2s on the battlefield" [27]. As given in [28], the Commander's Independant Thermal Viewer in front of the loader's hatch, which is a tall, armored housing for the thermal gunsight, the Commander's Weapon Station (CWS), which has improved vision devices, a differently-shaped commander's hatch, and a simplified mount for the .50 cal MG are the important differences to be modeled. None of these new features affects the net result of our intended analysis and hence M1A2 suffices for our simulation.

BMP-1 and 2 are infantry fighting vehicles with the major difference as provided in [29] and [30] being that the BMP-2 has a larger two man turret with 30 mm auto cannon whereas the BMP-1 has a single man turret with a 73mm high velocity gun. We observe that none of the Iraqi forces, specifically BMPs, as described in Actions 2 and 3 fire, causing any causality to the United State forces, making the difference insignificant for the purpose of simulating Actions 2 and 3.

3.3.2 VR-Forces implementation of the battle of 73 Easting

The above battle was simulated using VR-Forces to collect data which can be used to validate against our model of the DIS System. This section will concentrate on detailing the implementation of the battle using VR-Forces.

To simulate the two actions described above, we will need three phase lines, which when crossed by the US army, will fire on the Iraqi tanks. The figures show the US tanks M1A2 and M3A2 in blue, Iraqi army with T72 and BMP-2 in Red.

Figure 3.4 displays the location where the battle of 73 Easting was simulated in the virtual world.

Figure 3.5 displays the initial arrangement and planning of the battle. The red lines denote the routes that were written for the tanks to move. The vertical blue lines are the phase lines which when crossed by the US tanks, action will take place.

Figure 3.6 shows Action 1, when the E and I troops engage in battle with T72s on crossing phase line 1.

Action 3 begins when the E-Troop crosses phase line 2 and attacks the 4 T72 and 4 BMPs of the Iraqi army. The route 30 is the planned route for the M3A2 with the electronic malfunction that will head back and will be shot by friendly M3A2 tank assuming it is a hostile T72.

To implement this simulation, plans were written and assigned to entities. Most of the plans take the form shown in Figure 3.7. When a tank present in the E-Troop represented by "M3A2 6" crosses the phase line, M3A2 26 will Fire on T72 2 until the Iraqi tank T72 2 is destroyed. Similarly,

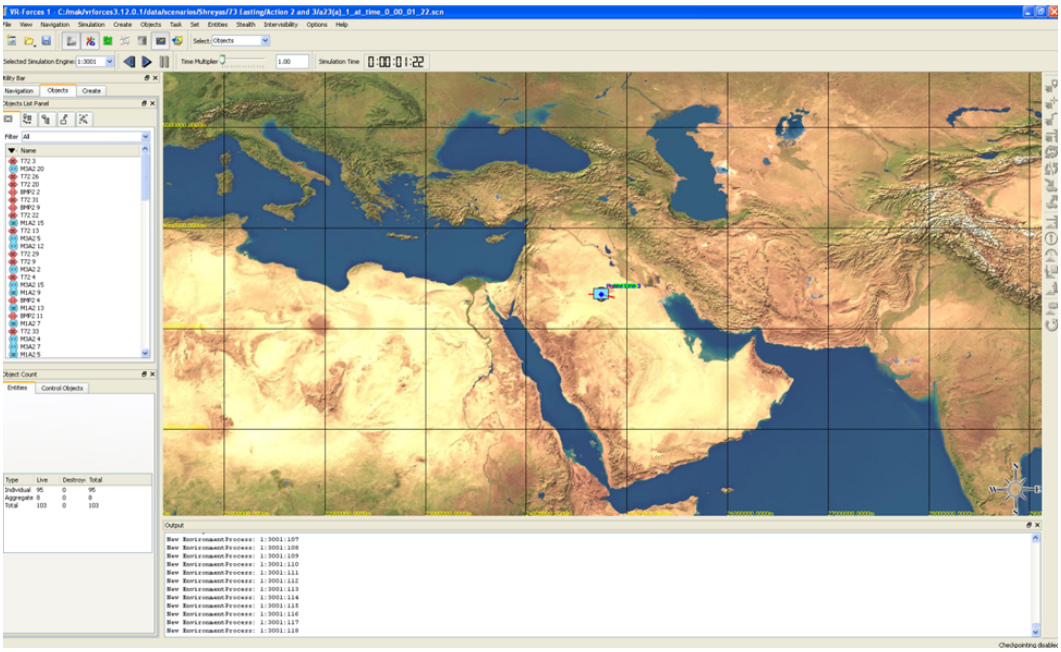


Figure 3.4: VR-Forces screenshot of the virtual world where battle of 73 Eastings is simulated

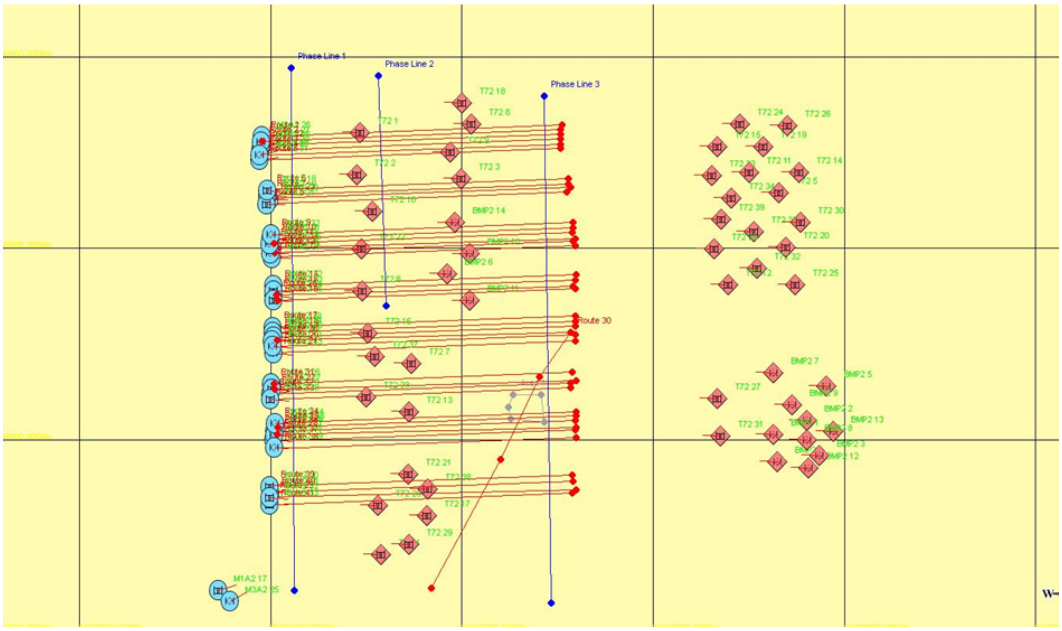


Figure 3.5: Battle of 73 Easting's, initial state. (Screenshot from VR-Force)

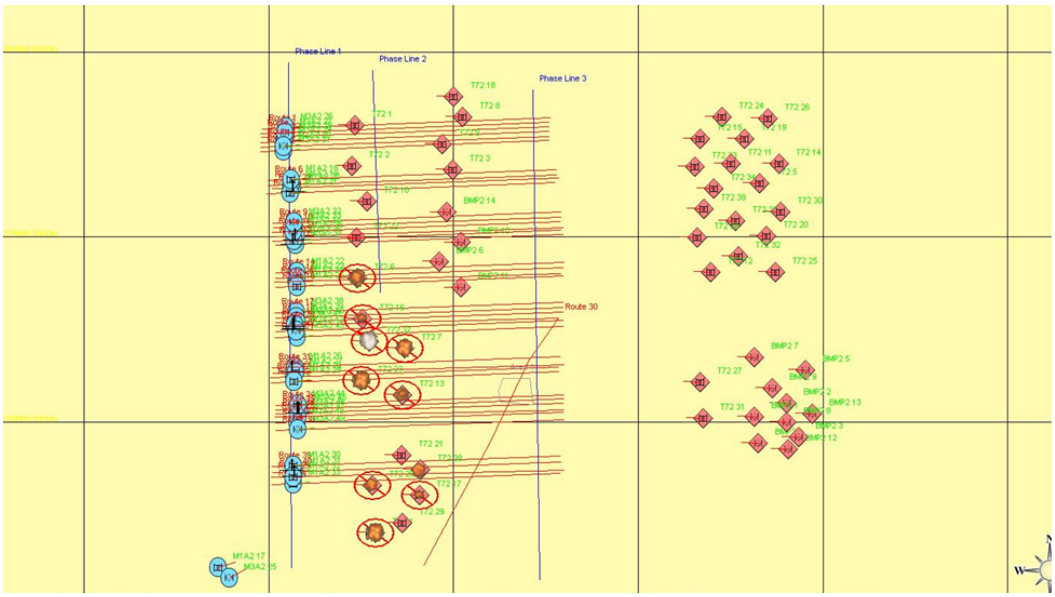


Figure 3.6: Battle of 73 Easting's, action 1. (Screenshot from VR-Force)

```
When (Entity-Left-Of-Line: SELF,, Entity:"" Line:"Phase Line 1") do
  While (NOT(Entity-Destroyed ,, Entity: "T72 2")) do
    Set Data Request; Object: M3A2 26 Request: Fire Now:"T72 2"
  endwhile
  While (NOT(Entity-Destroyed ,, Entity: "T72 1")) do
    Set Data Request; Object: M3A2 27 Request: Fire Now:"T72 1"
  endwhile
endwhen
```

Figure 3.7: General plan format in VR-Forces (Screenshot from VR-Force)

M3A2 27 will attack and destroy T72 1. These constructs are triggers. Every platoon also has instructions to move along its specified route.

The plan corresponding to the I-Troop which has a malfunction to one of its tanks that gets destroyed is shown in Figure 3.8. All the plans take similar forms and can be found in Appendix A.

```
When (Entity-Left-Of-Line: SELF,, Entity:"" Line:"Phase Line 3") do
  While (NOT(Entity-Destroyed ,, Entity: "T72 27")) do
    Set Data Request; Object: M3A2 38 Request: Fire Now:"T72 27"
  endwhile
  While (NOT(Entity-Destroyed ,, Entity: "T72 31")) do
    Set Data Request; Object: M3A2 39 Request: Fire Now:"T72 31"
  endwhile
  Task Object M3A2 38 Task: Move-Along Route: "Route 30"
endwhen
```

Figure 3.8: Plan for the I-Troop entity having electronic malfunction in VR-Forces (Screenshot from VR-Force)

CHAPTER 4

DISTRIBUTED INTERACTIVE SIMULATION MODEL

This chapter details the discrete event model of the DIS system developed. Section 4.1 provides the requirements which were considered for developing this model. Section 4.2 describes the architecture and design decisions taken for the model. Section 4.3 speaks about implementation and testing strategies employed. Section 4.4 lists the metrics and data which were collected by running the model.

4.1 Requirements

The following requirements were noted for the development of the model.

1. System requirements

   (a) Multiple nodes with a copy of DIS application installed, would be present. This application need not be the sole application running in the system.

   (b) Every node has many simulated entities associated with it. These entities are called local entities.

   (c) Every node contains a CPU for processing an entity or a PDU received over the network.

   (d) The processing of the entity generates PDUs that must be sent over the Ethernet using a given probability distribution.

   (e) The processing time for an entity is given. The processing time could be a discrete constant value or a range of uniform distribution.

2. Tick time requirements

   (a) Nodes process each entity at a defined tick for a certain duration (entity service time).

   (b) One tick processes exactly one entity. Each entity is associated with its tick event.

   (c) For N entities on a node, N Ticks are necessary to allow each entity state to be updated.

   (d) The Tick event is responsible for processing an entity on a node.

   (e) The Tick event schedules the next Tick event for the same entity.

3. Network requirements

   (a) All nodes connected by an Ethernet LAN of defined bandwidth.

   (b) Each node could transmit to the Ethernet through a Network Interface Card (NIC).

   (c) Each node could receive from the Ethernet through a NIC.

   (d) Nodes do not communicate directly, rather communication is via the Ethernet.

   (e) PDUs are broadcasted to each node connected on the Ethernet. DIS can work in multicast mode too, with each DIS application running on the system subscribed to the same multicast address. Irrespective of broadcast or multicast, the PDUs must be delivered to all the nodes.

   (f) Nodes process each PDU received from the Ethernet for a certain duration (PDU service time or PDU processing time).

   (g) NIC is responsible communicating with the Ethernet.

   (h) The Ethernet is responsible for transmitting a copy of the PDU to all the nodes connected to it.

4.2 Architecture and design

The architecture of the model is developed based on the identified requirements. Identified functionalities are distributed among many modules. The list of modules which satisfies the overall goal of the model includes

1. Node
2. DIS System
3. Ethernet
4. NIC
    (a) Send
    (b) Receive
    (c) Events
5. CPU
    (a) Events
6. Event processing engine
7. Future event list manager

Each node is made up of a module responsible for handling the NIC and CPU. Each node must also have its own event processing engine which is the core of discrete event simulation and is responsible for running the simulation. The FEL is associated with the event processing engine. The Event processing engine runs the simulation on a node by executing the events which are present on the FEL associated with that node. Figure 4.1 shows the architecture of a node in the DIS model.

The DIS system is a module consisting of many such node modules. The nodes are also associated with the Ethernet, the sole means of communication mechanism between the nodes. The Engine module present in the DIS system module controls the execution of the event processing engine on each node. This engine is also responsible for updating the overall simulation time. Figure 4.2 shows the architecture of the entire DIS system model.

The Ethernet uses Half-Duplex instead of Full-Duplex as Full-Duplex restricts the number of nodes to be exactly 2 on the network.

The NIC on each node transmits and receives from the Ethernet. The NIC has two queues associated with it. A send queue containing PDUs to be transmitted of the Ethernet to all the nodes

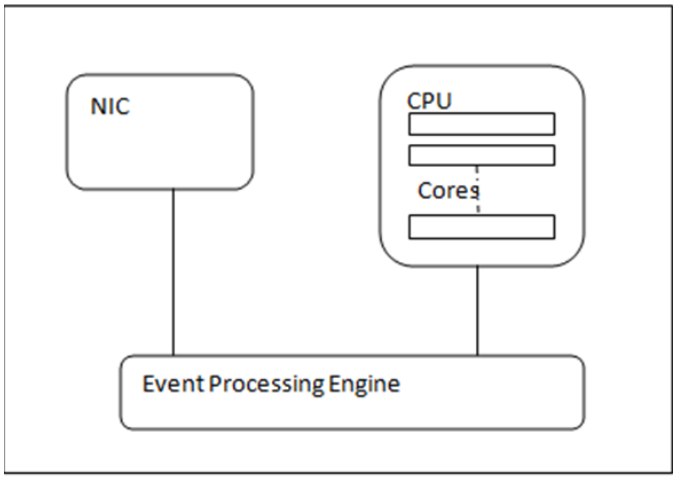


Figure 4.1: Architecture of a node in DIS model

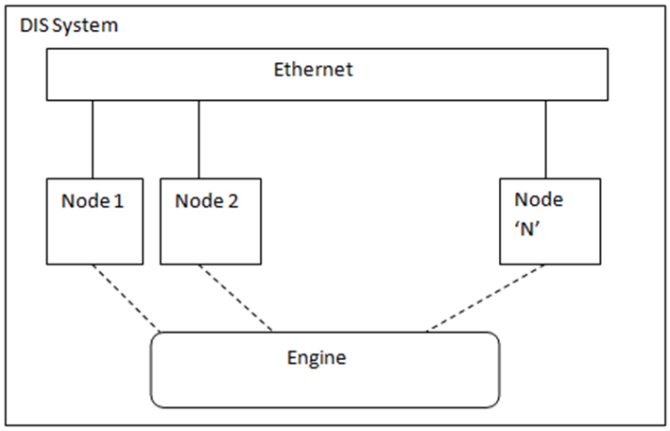


Figure 4.2: Architecture of the model of DIS system

and a receive queue containing all the PDUs that have been transmitted over the Ethernet. The CPU must read from the receive queue to process the PDUs and must put PDUs in the send queue to allow NIC to transmit over the Ethernet. The CPU does not interfere or use its CPU cycles for allowing the NIC to transmit/receive PDU from the Ethernet. This is the case when direct memory access is used.

The PDUs behave in two ways. The PDUs when transmitted over the network act as NetworkPDUs (can be defined using an interface in Java programming language) which denote the

medium access control (MAC) frame. The PDUs received by the NIC must be processed by the CPU, in which case PDUs are converted to PDU entities to be processed by the CPU. The NIC is a server which manages sending and receiving of data over the Ethernet. DIS PDU in the form of PDUEntity (can be defined using a class in Java programming language) and entities are the customers in the system which are processed by the servers.

If the Ethernet is gigabit, then the minimum MAC frame size is 4096 bits. Data packets smaller than this size are padded to 4096 bits. For other Ethernet, 10 or 100 Mbps, 512 bits is the minimum MAC frame size. The maximum MAC frame size is 1518 bytes (12144 bits) and is never exceeded in our situation.

The header details per data packet to be transmitted are

UDP Header = 2 bytes = 12 bits

IPv4 Header size = minimum size: 20 Bytes or 160 bits, maximum size: 60 Bytes or 480 bits

MAC header = 18 Bytes = 144 bits

Total Max Header overload = 636 bits

Total Min Header overload = 316 bits

The CPU deposits each PDU that needs to be broadcast in the send queue of the NIC. The NIC begins transmission immediately if the Ethernet is not in use (NIC Send model). In other words, the PDU is copied to the Ethernet. The Ethernet copies the PDU to all the NICs connected to Ethernet. Each NIC listens and receives any data that arrives on the Ethernet. The time T to transmit X bits on a NIC running at D bits/sec is

D/X seconds or D/X * 1000 Milli seconds or D/X * 1000000 Micro seconds.

If the Ethernet is in use when the NIC tries to send the PDU, then PDU is added to the queue and a re-transmission attempt is scheduled at a calculated time. The retransmission is dependent on *Slot Time*. One slot time S is the time required to transmit the minimum MAC frame on the Ethernet. If M bits is the minimum size of the MAC frame, then the slot time is given by M/X seconds. Retransmission is scheduled by using the *Truncated Binary exponential backoff*

algorithm [14]. Each node generates a random number R in the given range (0 to 2^j) on the kth iteration for an attempt to retransmission. The value of j varies from 0 to 10 in steps of 1 for $0 <= k < 10$. For $k >= 10, j = 10$. The maximum number of iterations is 16, hence the maximum value of k is 16. The attempt to retransmit the frame is scheduled at $R * S$ seconds from current time (RT). If the node cannot transmit the frame in 16 attempts, then the frame is dropped and the process of transmission attempt for any more packets in the buffer repeats after $2 * S$ time units.

The *Jamming signal* sent on detecting a collision is ignored along with any propagation delay associated with the frame. It is assumed that the frame is available to all the nodes connected to the Ethernet instantaneously. But, no node can transmit on the Ethernet until the other node completes transmission time given by T above.

Once the node is able to transmit to the Ethernet the frame is available in all the NIC of all the nodes. An event is scheduled to process the received DIS PDU by the CPU at the current simulation time (NIC receive model).

The events associated with the NIC are

1. EndOfNICTransmissionEvent

   (a) Created on: Once a NIC detects that the Ethernet is not in use, it sends the data to the Ethernet, flags Ethernet as being used, and schedules a EndOfNICTransmissionEvent at a time $t = T +$ current simulation time.

   (b) Action on execution: Flags the Ethernet as not in use and schedules a transmission event if there are frames pending in the send queue.

2. NICTransmitAttemptEvent

   (a) Created on: If the node cannot transmit the frame to the Ethernet as the Ethernet is flagged busy by some other node, then a NICTransmitAttemptEvent is created and scheduled to occur at 'RT' using the exponential backoff algorithm.

   (b) Action on execution: The node tries to transmit the frame on the top of the send queue to the Ethernet.

3. NetworkTickEvent

   (a) Created on: Using a configured Network tick time per local entity on the given node, the every NetworkTickEvent schedules the next NetworkTickEvent for the same entity at a time equal to sum of current simulation time and network tick time. Network tick time is the rate at which the PDU is transmitted to the network by every entity.

   (b) Action on execution: The next network tick event is scheduled for the same entity associated with the current network tick event at a time given by the sum of current simulation time and configured network tick time. The PDUs associated with the entity associated with the current network tick event are also broadcasted.

4. ProcessDISPDUEvent

   (a) Created on: The Ethernet copies the frame to all the NICs that are connected and hence listening for it. Once the frame is copied to the receive queue on a NIC, a ProcessDISPDUEvent is scheduled to execute at the current simulation time.

   (b) Action on execution: The DIS PDU is extracted from the MAC frame, service time for the PDU of the given type is generated, and a ProcessingTask (interface) is added to the CPU to process the PDU.

The Ethernet maintains the state of busy or available depending on if a node is transmitting a frame on it. It sends the data to the NIC of each node to be copied on their respective receive queue.

The model of the CPU considers that a CPU may contain P processing units. Once a task is added to be processed by the CPU, if any of the P units are idle, then the task is associated with that unit for processing. The CPU is said to be busy if all the P units are busy, in which case the task is moved to a process queue. The CPUs processing units execute the task from the queue when they complete executing the current task. If the process queue is empty, then the unit declares itself idle and waits for more tasks to arrive which will be assigned to it.

When is task is associated with one of the P processing unit, a ProcessingCompleteEvent is scheduled at a time defined by current simulation time plus the task service time. Every task arriving

at the CPU has the service time required calculated and present in it. CPU is the server containing P processing units where each unit can serve the customer, PDUs or entities simultaneously.

The events associated with the CPU are

1. ProcessingCompleteEvent

   (a) Created on: A task is assigned to one of the units in the CPU to process at a time given by the sum of current simulation time and task service time.

   (b) Action on execution: The task is removed from the CPU, and a new task if available in the process queue is assigned to newly idle CPU. Also, the processing of the task is simulated. If the task processed was a DIS PDU and if the PDU was a Detonation PDU, then the number of entities present on the node associated with the Detonation PDU is decreased by 1, and all the TickEvent and NetworkTickEvent for that entity are removed from the FEL.

2. TickEvent

   (a) Created on: Using the configured tick time per entity on given node, every TickEvent schedules the next TickEvent for the same entity at a time equal to sum of current simulation time and tick time.

   (b) Action on execution: The entity associated with the TickEvent is sent to the CPU for processing. If there are any entities remaining on node, then the next TickEvent is scheduled in the FEL.

A node module is made up of a NIC, CPU, and EventProcessingEngine. NIC and CPU were discussed above. The node also holds the count of number of entities present at any given time. This count decreases in time when Detonation PDUs are received by the node.

The Event processing engine (EPE) is the component responsible for executing events present in the system. The EPE runs if there exist any events to be executed at the current simulation time. EPE is the interface to all the remaining components which require scheduling events of some type. The EPE is also responsible for identifying all the TickEvents that must be

removed for an entity when the Detonation PDU is processed. The EPE uses the services of the FutureEventListManager.

The FutureEventListManager (FELM) is the module that maintains the FEL, and is the interface to access FEL. It is responsible for adding an event notice to the FEL and keeping FEL in order sorted by scheduled notice execution time. It also provides the functionality to determine the next least available time when an event notice must be processed.

The entire DIS system is modeled as a collection of individual nodes. The DISSystem module is responsible for advancing the simulation time to the next available minimum from all the nodes so that no events are missed in any node. The DISSystem goes through the nodes in a round robin fashion indefinitely allowing each node to execute at the current simulation time. The DISSystem continues to execute until the simulation is stopped or no entities exist in any node. A probability distribution is associated with detonation logic. This sometimes results in detonation of all the entities present on any given node.

The system is initialized by scheduling a TickEvent at every node for all the entities as given by the configuration evenly distributed over the application's unit time (second or millisecond or microsecond) *bootstrapping* the process. After initialization, each TickEvent generates subsequent TickEvents for the ticked entity.

The inputs required for the model include entity service time (for every node), ESPDU, FPDU, and DPDU service time (for every node), tick time (for every node), total number of nodes, number of entities on each node, number of processor on each node, Ethernet link speed, and probability of ESPDU, FPDU and DPDU generation.

The following packages are identified:

1. com.ssb.core - Holds the classes which form the core part of the DIS model.

2. com.ssb.event- Holds the classes representing the various events described above.

3. com.ssb.exception- Holds classes responsible for exceptions.

4. com.ssb.model- Holds model classes such as Entity and Processing Task

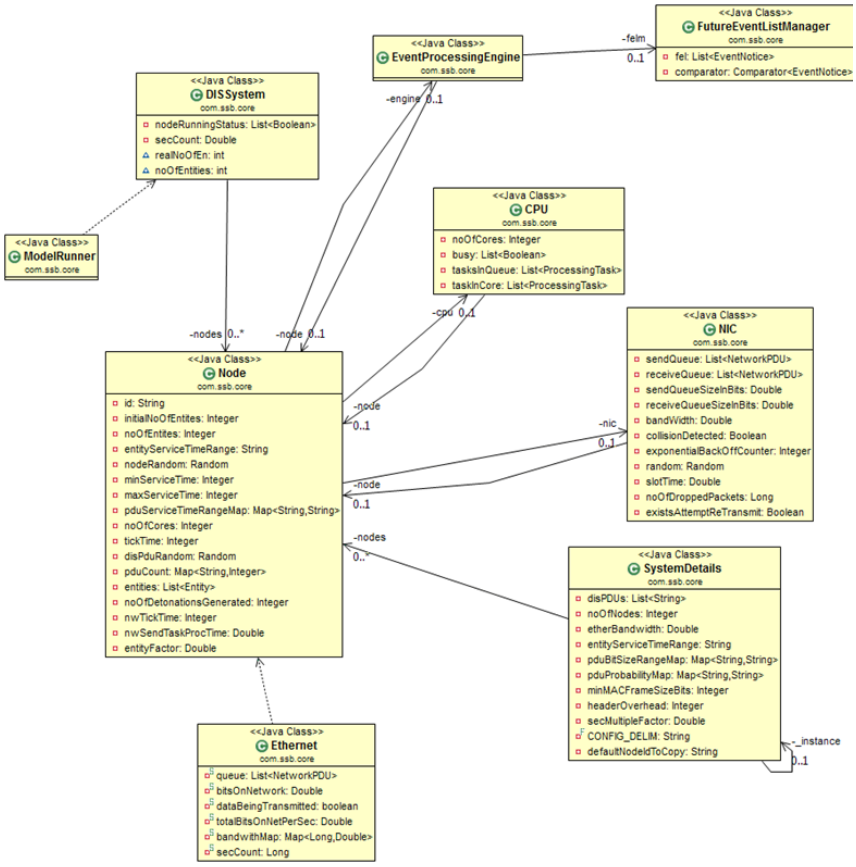


Figure 4.3: Class diagram of core package

5. com.ssb.pdu- Holds classes representing the PDUs.

6. com.ssb.util- Holds classes which provide utility functions such as logging and configuration manager.

Classes are identified and class diagrams are drawn for classes in each of the packages. Figure 4.3 shows the UM: class diagram for the classes in the core package, Figure 4.4 shows the UML class diagram for the event package, Figure 4.5 shows the UML class diagram for the model package, Figure 4.6 shows the UML class diagram for PDU package, and Figure 4.7 shows the UML class diagram for the exception package.

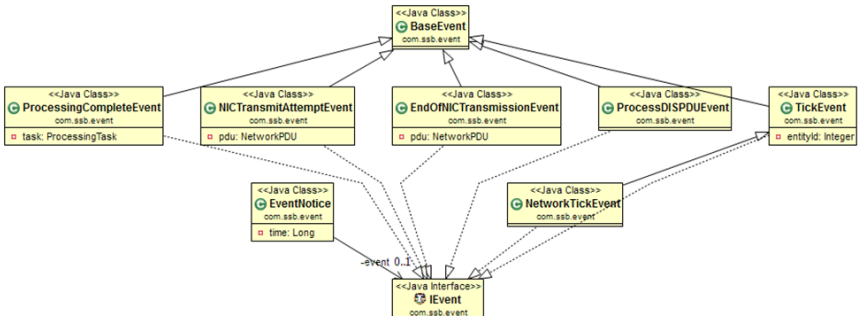


Figure 4.4: Class diagram of event package

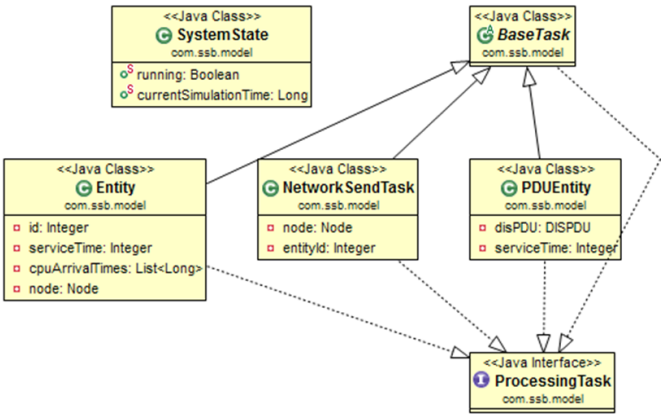


Figure 4.5: Class diagram of model package

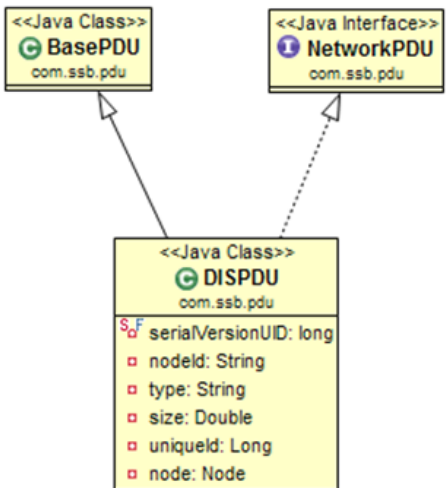


Figure 4.6: Class diagram of pdu package

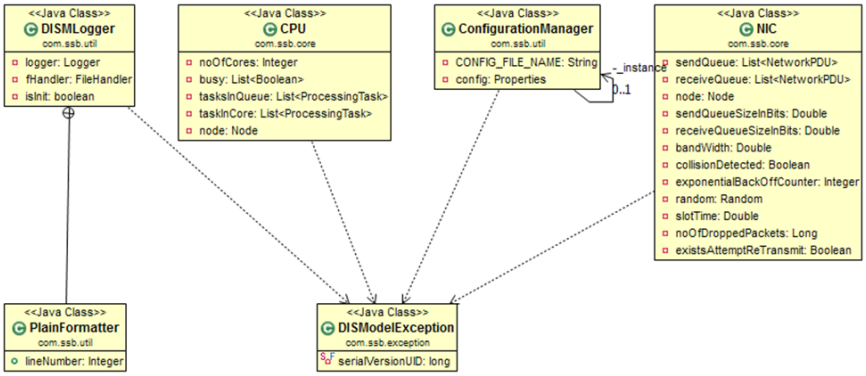


Figure 4.7: Class diagram of exception package

4.3 Implementation and testing

The model is implemented in the Java programming language as per the design. Required operations are added to the developed classes to achieve the desired functionality. Selected examples of the implementation code are provided in Appendix B. Approximately 1900 lines of code were developed.

Unit testing was done to confirm that the model was behaving as desired. Boundary values were tested (when there is exactly one entity and exactly one node). Simulation was run for a large period of time to see if any exceptions are generated.

Major parts of the testing included tracing the log files generated that are described in the next section to determine that the system was behaving as expected. The developed system being a discrete event simulation, a manual run of events was done and verified with the trace that was generated. A simulation was run for a small period of time and the logs were traced to determine whether the system was functioning as desired. As described in next section, every event, CPU state, NIC state, and Ethernet state are logged along with the current simulation time aiding the trace process. Multiple runs of the testing were done with 1 core, 2 core, 1 node, 2 node, and 3 node permutation and combination. Numbers of generated PDUs are logged, and the probability was calculated and verified with the configured data. The configured PDU sizes are observed in the logs along with the service times to verify that the model was behaving as configured.

Service time of entity was reduced to zero to test if the simulation was behaving as desired under such circumstances. Similarly, the PDU size was reduced to zero in another run. The PDU processing time was also reduced to zero to test if PDU processing is behaving as expected.

The model was debugged as suggested in [14] to verify that it was behaving as anticipated. The model outputs a variety of statistics which was observed for reasonableness to see if anything wrong has happened during execution of the model. The model outputs the initialized system from the configuration file which was analyzed to determine if the initial state of the system was as expected.

4.4 Metrics and data collection

As part of running the simulation, system metrics used to study the performance of the DIS system and the run-time data used to verify and validate the model are collected. The collected system metrics are given below.

1. Number of PDUs of each type generated for each node. The output is logged to a file in comma separated values (CSV) format, pdustat.csv. The format of the CSV is Node ID, PDU Type, Number of PDUs.

2. Receive queue size in KB in NIC every sec. The output is logged to nicStat.csv. The format of the CSV is Node ID, Current Simulation Time, Queue size in KB.

3. Number of dropped packets due to collision on a NIC. The output is logged to nicStat.csv at the end of the file. The format is Node ID, Number of Dripped packet.

4. Ethernet bandwidth with per second window size. The output is logged to ethernetbandwidth.csv. The format being Second, Bandwidth in bps for that second window.

5. Average Ethernet bandwidth for the whole run. The output is logged to ethernetbandwidth.csv at the end of the file.

6. Frequency per sec, each entity was ticked. The output is logged to entityStat.csv, the format being Second, Frequency of tick. Each entity log list is provided a heading indicating the Node id and the entity id.

7. Number of times the configured tick time was exceeded between two successive processing times for each entity. The output is logged to entityStat.csv with heading *Number of times the tick time was exceeded.*

8. Mean interarrival times of the Ticks for individual entities. Here Ticks refer to the TickEvent and not the NetworkTickEvent. The output is logged to entityStat.csv with heading *Mean Interarrival Time.*

9. Number of entities that remained not destroyed after the simulation run. The output is logged to entites.csv, with the format of the CSV being, Node ID, number of surviving entities.

10. The maximum of the mean interarrival time of the ticks for every entity. Here the ticks refer to the TickEvent and not the NetworkTickEvent. The output is logged to entityStat.csv.

The runtime data which was collected are listed below.

1. A log of PDUs which was generated and processed. The output is logged to pdu.csv. The format of CSV is Node ID, Current Simulation Log Time, G:PDU is Generated/P: PDU is processed, PDU Details(Generated Node ID; PDU Type; Size in Bits; Unique ID).

2. A log of tick events is output to TickEvent.csv. The format of the CSV is Node ID, Simulation Time, A: Arrival/E: Event executed, TickEvent[Entity ID], Time of arrival when the third field is A.

3. A log of NIC transmission events is output to NIC.csv. The format of the CSV is Node ID, Simulation Time, A: Arrival/E: Event executed, NICEvent[DIS PDU details], Time of arrival when the third field is A.

4. A log of Ethernet states through time is output to Ethernet.csv. The format of the CSV being Current Simulation Time, is Ethernet Busy, the data being transmitted (PDU), size in bits.

5. A log of CPU queue size through time. The output is logged to cpuQueue.csv with the format of CSV being Node ID, Current simulation time, Queue size(Number of tasks pending in queue waiting for CPU).

6. A log of event changes in CPU is output to cpu.csv. The format being Node Id, Current simulation time, A:Arrival of task/D:Removal of task from CPU, is CPU busy, queue size, task, index of free core if CPU is not busy/index of core from which task is removed.

CHAPTER 5

VR-FORCES SURROGATE MODEL

The VR-Forces tool developed by MÄK Technologies is proprietary. Because of that, many limitations on its use exist that led to the development of a surrogate which was a stripped down version of the real VR-Forces. This chapter describes the Surrogate model. Section 5.1 presents the necessity and the requirements considered while developing this tool. Section 5.2 describes the architecture and design used while developing the tool. Section 5.3 explains the implementation and testing strategy employed. Section 5.4 speaks about the metrics and the data that is collected.

5.1 Requirements

The VR-Forces tool does not provide access to find out the service time for PDUs such as ESPDU, FPDU, and DPDU. We can estimate the service time of the entities by monitoring network traffic and finding the difference between two successive PDU arrivals when the VR-Forces is loaded. But there were other issues with VR-Forces that were not so easily overcome. The VR-Forces logger starts to crash (a bug reported) when collecting data for more than a minute at low values of tick time (200ms). The play button disables at random and never enables again to run the simulation, forcing recreating the entire scenario from the beginning. Entities start to disappear at random, possibly due to overload. Because of the issues, VR-Forces cannot be used effectively for analysis once the total entity count in a scenario exceeds a certain size, approximately 300. Hence, we built a surrogate for VR-Forces that can be used to validate our model more closely. The Surrogate is a custom reduced-functionality version of the VR-Forces.

The Surrogate is similar to VR-Forces in two aspects. Firstly, both VR-Forces and Surrogate use DIS and generate DIS PDUs. Our research is concerned with only ESPDU, FPDU, and DPDUs. Secondly, both are used to model battles, although the capabilities of VR-Forces are much more extensive than that of the Surrogate. The objective of our research is not to model VR-Forces specifically but to model DIS application systems. The VR-Forces Surrogate is a true DIS based application and is similar to VR-Forces and other SAF systems, especially with respect to DIS network traffic and CPU utilization. It is used in this research.

To keep the scope of the Surrogate within check, only the required capabilities given below were included in the Surrogate.

1. Application must allow creating routes.
2. It must support entities which are tanks.
3. Friendly and opposing entities must be supported.
4. Each tank must be associated with a route along which it will travel.
5. Each entity may have a different velocity.
6. The tank is capable of firing and is configured by providing the range in kilometers.
7. The orientation of the entity could be ignored.
8. The application must be based on DIS protocol and must use ESPDU, FPDU, and DPDU.
9. The entity fires on hostile entities based on the range and FPDU is used to communicate the fire.
10. The destruction of a target entity is decided by the target; the probability of destruction is assumed to be 50% for simplicity.
11. The tick time must be configurable.
12. Every tick of an entity must also result in ESPDU sent on a network.

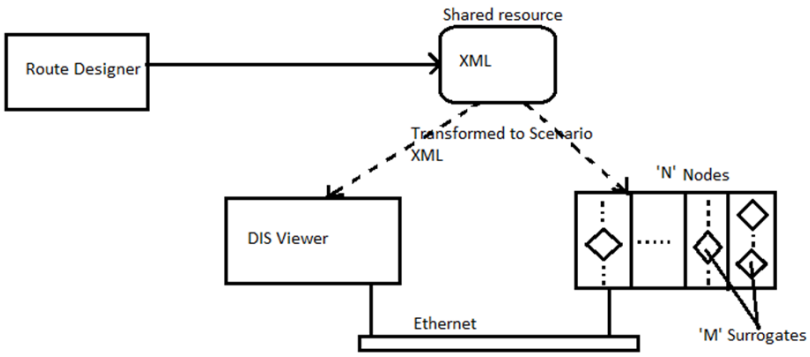


Figure 5.1: VR-Forces Surrogate architecture

13. Multiple Surrogates must be able to run simultaneously on the same machine, and multiple machines could be active on a LAN.

14. Ethernet broadcast/multicast must be used.

5.2 Architecture and design

The Surrogate is made up of 3 components which are independent of each other and can run on any machine.

1. Route designer: This component is used to design routes and generate XML which will be provided as input to other components.

2. DIS Viewer: This component takes scenario XML as input and listens to PDUs on the network. Scenario XML describes the routes and entities in the scenario. The viewer components show the motion of friendly and opposing forces graphically along with fires and detonations which take place using FPDU and DPDU.

3. DIS App: The DIS based simulation engine takes the scenario XML as input and ticks the locally configured entities producing and publishing PDUs to the network.

Figure 5.1 displays the relation between different modules. A scenario consists of entities such as tanks, their properties, and routes on which they travel. This detail is represented by using

an XML file. We call this as a scenario repository. A very simple scenario XML file content is given below:

```
<ScenarioRepository applicationId="3000" siteId="1">
    <routeInfos>
      <routeInfo>
        <pathWays>
          <pathWay>
            <p1 x="0.0" y="0.0"/>
            <p2 x="0.0" y="1.0"/>
          </pathWay>
          <pathWay>
            <p1 x="0.0" y="1.0"/>
            <p2 x="1.0" y="3.0"/>
          </pathWay>
        </pathWays>
        <routeId>0</routeId>
      </routeInfo>
      <routeInfo>
        <pathWays>
          <pathWay>
            <p1 x="1.0" y="0.0"/>
            <p2 x="0.0" y="1.0"/>
          </pathWay>
          <pathWay>
            <p1 x="0.0" y="1.0"/>
            <p2 x="1.0" y="2.0"/>
          </pathWay>
```

```
        </pathWays>
        <routeId>1</routeId>
      </routeInfo>
    </routeInfos>
    <entityInfos>
      <entityInfo entityId="1"
       applicationId="3000"
       forceId="1"
       routeId="0"
       siteId="1"
       tickTime="0.4"
       velocity="25.0"
       fireRange="3000.0"
       firePerSec="1"/>
      <entityInfo entityId="2"
       applicationId="3000"
       forceId="2"
       routeId="1"
       siteId="1"
       tickTime="0.4"
       velocity="25.0"
       fireRange="4.3"
       firePerSec="1"/>
    </entityInfos>
  </ScenarioRepository>
```

The root element *ScenarioRepository* has two fields, *siteid* and *application id*, that must be configured to denote the node on which the DISApp will run and the application id on that node. Multiple

DISApp can run on the same node (may be on virtual machines). The routes are described using pathways, with each pathway consisting of two points representing a line segment in the route. The *routeInfos* element can be generated using the route designer. Every route is also given an *Id*. The element *entityInfos* consists of the *entityInfo* which describes different entities which will be a part of the simulation. Each entity is provided with *entity id*, a *site id*, and *application id* describing its deployed location with respect to node and application. The *force id* of 1 represents the blue team and *force id* of 2 represents the red team. The *routeid* attribute provides the route that the entity must follow and is one of the route ids from the route info in the XML file. The attribute *tickTime* provides the expected tick duration of the entity. The entity must update its state and perform its designated action at a rate given by this element. It is provided in seconds. The velocity in kilometer/second is provided for every entity as an attribute. The *fireRange* attribute defines the range of the entity in kilometers. The *firesPerSec* attribute defines the number of fires per second by the entity.

The *scenario_repo.xml* file must be identical between the DIS Apps and DIS Viewer. If they are not, then the behavior is not predictable.

When each entity is ticked, it loops through its list of entities in the repository and uses the latest available state to determine its distance to a hostile entity. A probability distribution is used to determine whether or not the entity fires at each hostile entity. A step function given in Figure 5.2 is used with the probability to fire increases as the range to the entity is closer. If the range is between 75-100%, then a probability of 0.25 is used to fire. If the range is between 50-75%, then a probability of 0.50 is used to fire. If the range is between 25-50%, then a probability of 0.75 is used to fire. If the range is between 0-25%, then a probability of 1 is used to fire. An entity receiving the FPDU with itself as a target will determine whether or not it has been destroyed, using a constant probability of 0.50. If it is destroyed, then a DPDU is broadcast. If two entities fire on each other, then the entity that fired first is given higher precedence to destroy the other entity.

P(Detonation) = 0.5

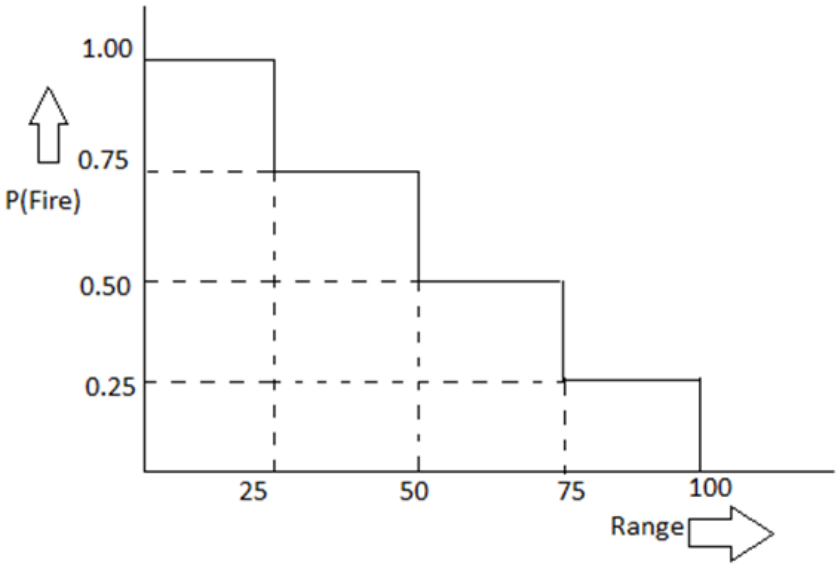


Figure 5.2: Probability distribution for fire in VR-Forces Surrogate

5.3 Implementation and testing

The Surrogate was implemented using Java and JavaFX programming languages. The various module's implementation is described.

1. Route designer: This tool is used to design the routes for the entities. A scenario is configured in this Surrogate as an XML and hence this tool can be used to generate the routes part of the XML. The description of the XML is given in previous section. This has been implemented in JavaFX and Java. Lines are drawn and route id is automatically attached to each route when the "Next Route" button is clicked. The directions of the routes are shown using the arrowhead at each line segment generated. The Generate XML button generates XML for the routes on the workarea and copies it to clipboard. It also dumps a "latest.xml" file to the disk for usage. The screenshot of the Route Designer tool with 2 routes is in Figure 5.3.

2. DIS Viewer: A tool developed using JavaFX and Java which shows the battlefield with all the configured routes. The routes are provided with directed arrows to aid the direction. When the DISApp component is run, the DIS Viewer listens to the PDUs and displays the current

Figure 5.3: Snapshot of route designer

position of the entities involved in the battle. It understands two types of entities: Red and Blue. The entities firing at each other are shown by a red line segment between the two engaging entities. The entity that is destroyed is displayed with a superimposed *X*. A log of the firing entity and the destroyed entity is also displayed. The simulation time retrieved from the timestamp field of the header is also displayed. When the mouse pointer is held or moved over the entity, the screen displays the Entity ID made up of site id, application id, and entity id which are retrieved from the ESPDU. A snapshot of the viewer is given in Figure 5.4. The DIS Viewer needs the scenario_repo.xml file containing scenario initial details such as route and entities to initialize.

3. DIS App: This is a Java backend that is responsible for simulating any entity deployed on it. It is responsible for updating the entity positions, firing between entities, destroying entities, and sending respective PDUs. The logic of the scenario is encapsulated in this component. The component initializes using the scenario_repo.xml that describes the routes, entities in the scenario being simulated.

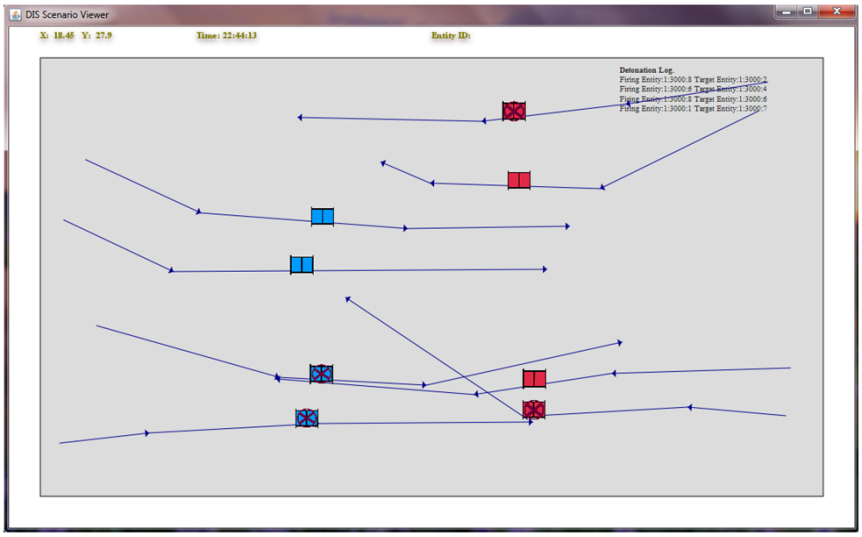


Figure 5.4: DIS Scenario viewer

5.4 Metrics and data collection

The above DISApp code is instrumented by adding Java code to collect statistics. The DISApp is exported to a Java executable archive, jar, and run to collect statistics.

The following statistics are collected.

1. Entity service time: The time taken to execute the instructions of an entity is monitored to determine the time taken by its tick function which is tried to invoke at every "tickTime" as configured in the scenario repository. The mean of this over all entities and all trials is used to configure the model. The entityServiceTime.log file consists of the list of service time for every entity.

2. PDU Service time: Service time of ESPDU, FPDU, DPDU by DIS App is monitored, and the mean of all PDUs is published to the log. These values are used to configure the DIS Model. The pduServiceTime.log file provides the individual mean service time for processing ESPDU, FPDU, and DPDU.

3. Entity tick time: Every entity running in the DIS App is monitored to log the time at which it is ticked. The interarrivals, which is the difference between successive ticks, are calculated

from this log and the mean is computed for each entity. The maximums of these means are used as the entity tick time when the DIS App was loaded with that many number of entities. The maximum of means is used because in DIS it is usually not acceptable for even one of the entity to be ticked more slowly than desired as that entity could be critical in the battle that is being simulated. The entityMaxOfMeanTickTime.log file provides the maximum of the mean tick time which is collected on running the DIS App.

A graph of the number of entities versus the tick time is plotted which is compared with the one generated by the model.

CHAPTER 6

VALIDATION AND EXPERIMENTATION

This chapter describes the experiments which were carried to validate the model against VR-Forces and VR-Forces Surrogate. Section 6.1 describes the experiments and validation for the CPU model developed. Section 6.2 explains the experiments and validation for the network model developed. Section 6.3 presents the lessons learned due to the experiment and the result. The lessons learned are not the major research findings, which are presented in Chapter 7, but they do provide guidance for future research work of this type.

6.1 CPU utilization validation and experimentation

Experiments were conducted to validate the developed model for CPU utilization with the actual behavior of the DIS application. The results are given in graphical form and discussed below; the tabulated values are present in the appendix C. The tick time is used to measure CPU performance. As the load on the node increases, the tick time increases and deviates from its original expected and configured value. The load on a node increases due to increase in number of local entities and DIS network traffic.

CPU experiment 1, DIS application,: VR-Forces Surrogate

This experiment was run with tick time set to 200ms on a single node. The scenario consisted of equal number of friendly and opposing entities who interact after a certain period of time. The experiment was run 23 times with the runs differing by number of local entities: 1, 2, 4, 40, 80, 100, 120, 140, 160, 200, 240, 245, 282, 290, 295, 300, 320, 350, 400, 450, 500, 550, and 600. Each run of the simulation totally amounted to 100sec. Data was collected to obtain the needed configuration

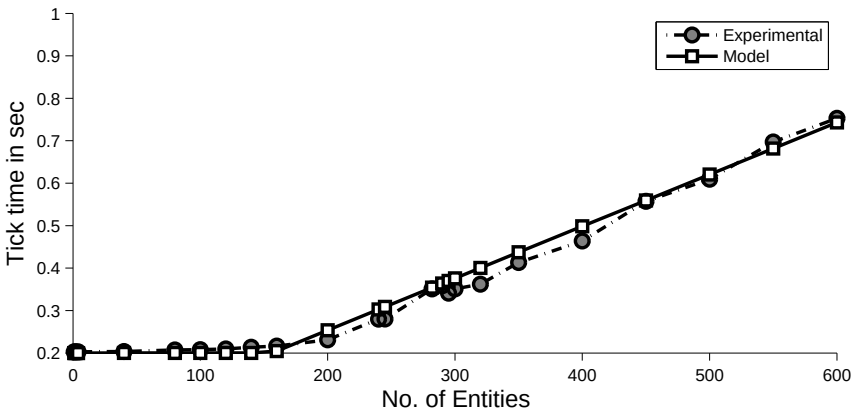


Figure 6.1: Experiment 1 CPU utilization XY graph

data for the DIS model developed. The collected data included mean service times for an entity, ESPDU, FPDU, and DPDU, number of ESPDU, FPDU, DPDU, and hence the probabilities of the same. The maximum of the mean tick time was also collected for each run so as to validate the experimental results with the model. A graph of maximum of mean tick times on the y-axis and number of entities on x-axis was plotted for the experimental results and as predicted by the model to compare the two. The XY graph is shown in Figure 6.1. A linear regression analysis is done by plotting a graph with expected values on the x-axis and model predicted values on the y-axis. R^2, the co-efficient of determination is calculated. R^2 values are between 0 and 1; high R^2 values (closer to 1) imply that the model is a good fit to the data. Figure 6.2 shows the scatter plot and the regression line drawn using polyfit and polyval functions in Matlab. The calculated value for experiment 1 $R^2 = 0.9896$ is quite high and implies a good fit. Clustering of the scatter points along the regression line indicates accurate estimates.

As can be seen from the graphs, the model predicts accurately the tick time for entities as the number of entities (load) on a node increases.

CPU experiment 2, DIS application: VR-Forces Surrogate

With the same set of data, configuration, and node, the experiment was repeated, but the send and receive PDU functionality was disabled. The logic to determine if the other entity in range is hostile or friendly was executed but no action was taken if a hostile entity was found. With no

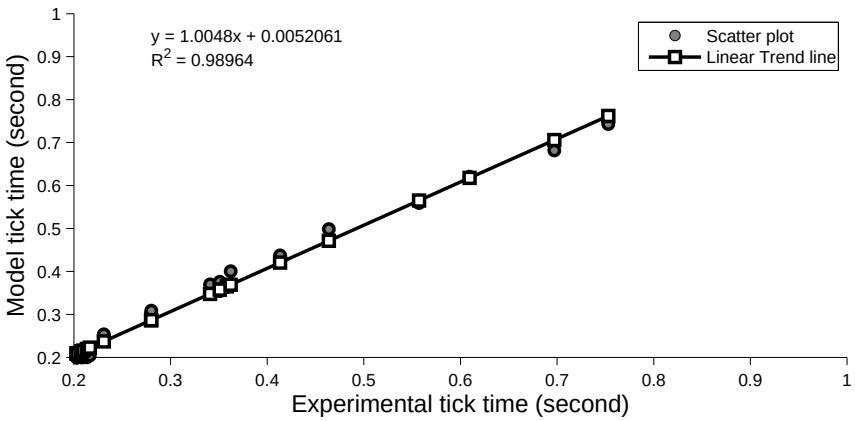


Figure 6.2: Experiment 1 linear regression analysis

FPDU and DPDU, each entity existed independent of the other entities. Because the experiment was conducted on a single node, all the friendly and opposing entities were local to that node and the entity states such as position were being shared with no network traffic being generated. The goal was to determine if the model can predict the real DIS application tick times for entities more closely and accurately by removing any network related dependency.

The XY graph is shown in Figure 6.3. A linear regression analysis is done by plotting a graph with expected values on the x axis and model predicted values on the y axis. Figure 6.2 shows the scatter plot and the regression line drawn using polyfit and polyval functions in Matlab. $R^2 = 0.9988$ is quite high and implies a good fit by the model. Clustering of the scatter points along the regression line indicates accurate estimates.

The graph clearly indicates the fact that by removing the probabilities of FPDU and DPDU and decreasing the network traffic, the tick times predicted by the model are more closer to the tick times observed in the VR-Forces Surrogate. The variance in slope displayed by the line graph for the model and experimental tick times also matches.

CPU experiment 3, DIS application: VR-Forces Surrogate

Experiment 1 was repeated on 2 other computers in the Center for Modeling, Simulation and Analysis (CMSA) lab with the same configurations and graphs were plotted. The number of nodes in the DIS system is 1 in both the runs. The XY graph for node 1 is shown in Figure 6.5.

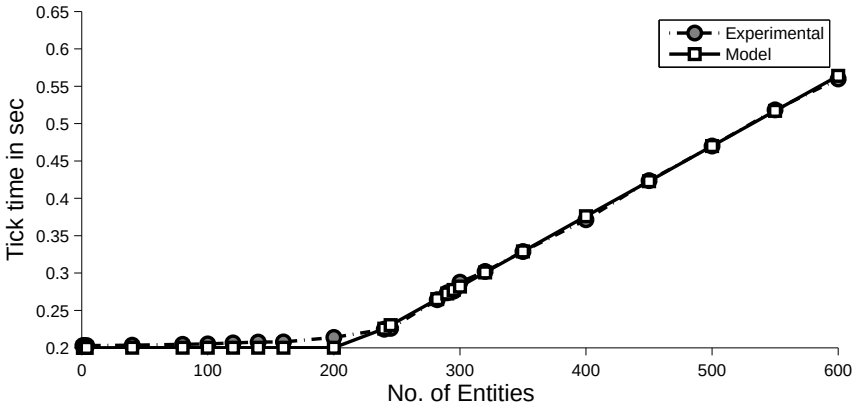


Figure 6.3: Experiment 2 CPU utilization XY graph

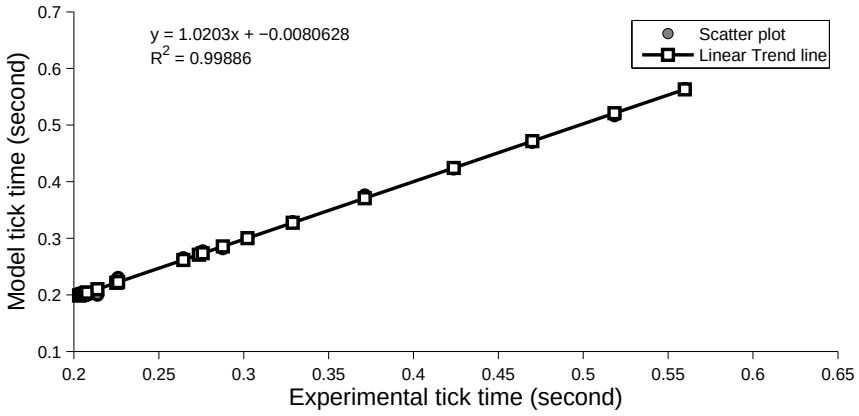


Figure 6.4: Experiment 2 linear regression analysis

The XY graph for node 2 is shown in Figure 6.7. Figure 6.6 and Figure 6.8 show the plot of the regression analysis for nodes 1 and 2 respectively. The R^2 values are displayed in the graph (0.9965 and 0.9995) and are quite high representing a good fit by the model.

As can be seen from the above graphs for both the nodes, the tick times as predicted by the model are accurate and match that of the experimental values.

CPU experiment 4, DIS application: VR-Forces Surrogate

DIS being a distributed application, the model must be validated with distributed run of the simulation. A 2 node simulation was run with tick time configured to be 200ms. The scenario consisted of an equal number of friendly and opposing entities who interact after a certain period

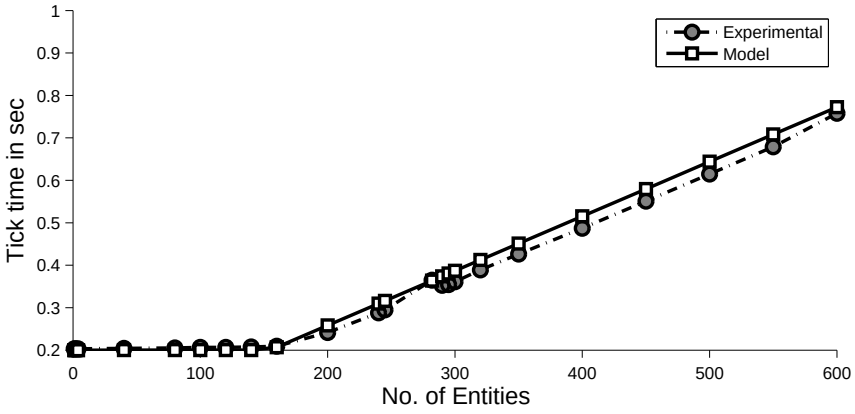


Figure 6.5: Experiment 3 CPU utilization XY graph for Node 1

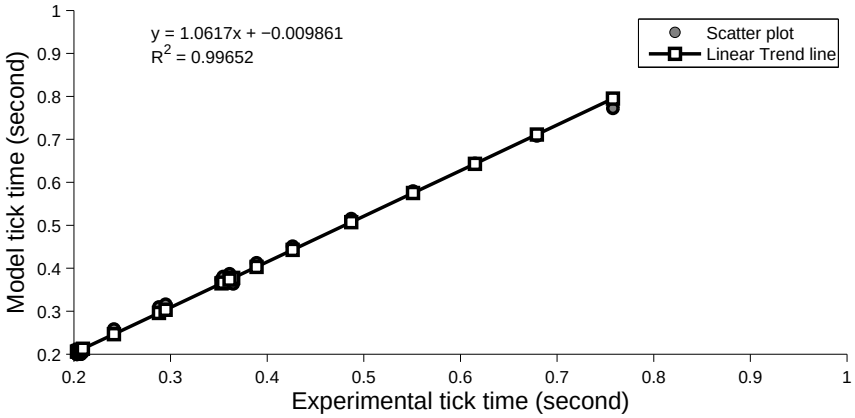


Figure 6.6: Experiment 3 linear regression analysis for Node 1

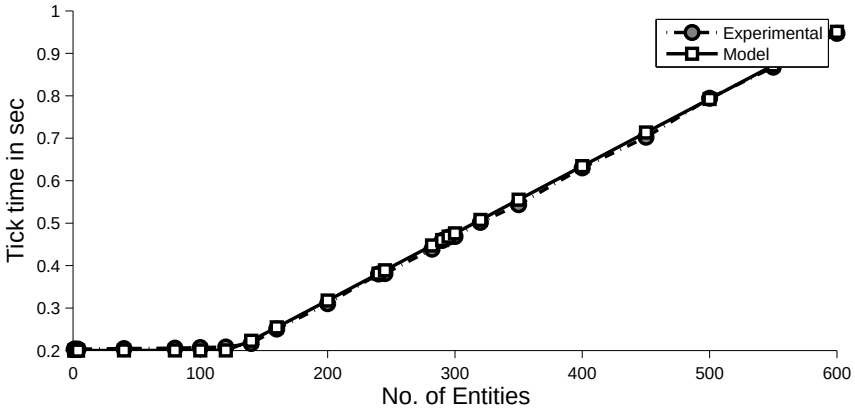


Figure 6.7: Experiment 3 CPU utilization XY graph for Node 2

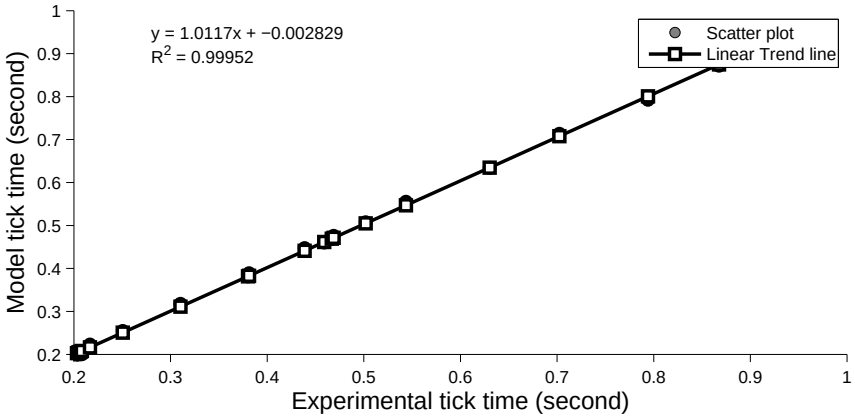


Figure 6.8: Experiment 3 linear regression analysis for Node 2

of time. An experiment was run each time with the total number of entities per node to be 1, 2, 4, 40, 80, 100, 120, 140, 160, 200, 240, 245, 282, 290, 295, 300, 320, 350, 400, 450, 500, 550, and 600. Each run of the simulation totally amounted to 100sec. Node 1 simulated friendly entities and node 2 simulated an equal number of opposing entities. Data was collected to obtain the needed configuration data for the DIS model developed. The collected data included mean service times for an entity, ESPDU, FPDU, and DPDU, number of ESPDU, FPDU, DPDU, and hence the probabilities of the same. The maximum of the mean tick times was also collected for each run so as to validate the experimental results with the model. A graph of the maximum of mean tick times on the y-axis and number of entities on x-axis was plotted for the experimental results and as predicted by the model to compare the two. The XY graph of both the nodes comparing model and the experimental tick times is shown in Figure 6.9 and Figure 6.11. Figure 6.10 and Figure 6.12 show the plot of the regression analysis for nodes 1 and 2 respectively. The R^2 values are displayed in the graph (0.9951 and 0.9966) and are quite high representing a good fit by the model.

As can be seen from the graphs, the discrete event model of the DIS system can predict accurately the tick times as the number of entities (load) on the system increases.

CPU experiment 5, DIS application: VR-Forces Surrogate

The experiment was repeated with the same configuration, but the total number of nodes in the distributed system increased to 4. The graphs are shown below for individual nodes. The

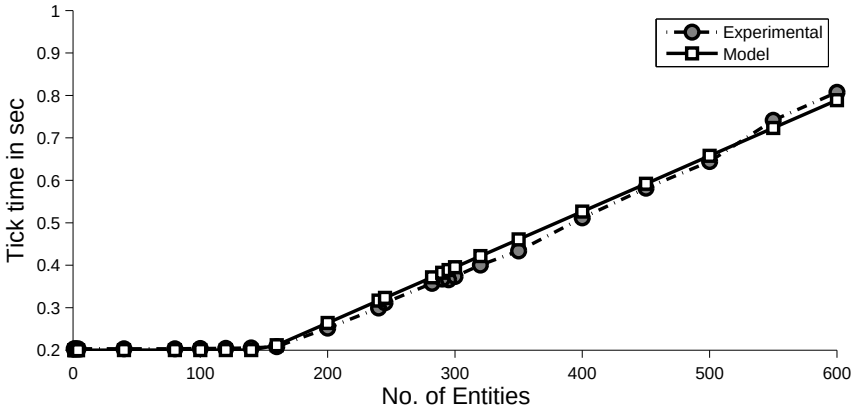


Figure 6.9: Experiment 4 CPU utilization XY graph for Node 1

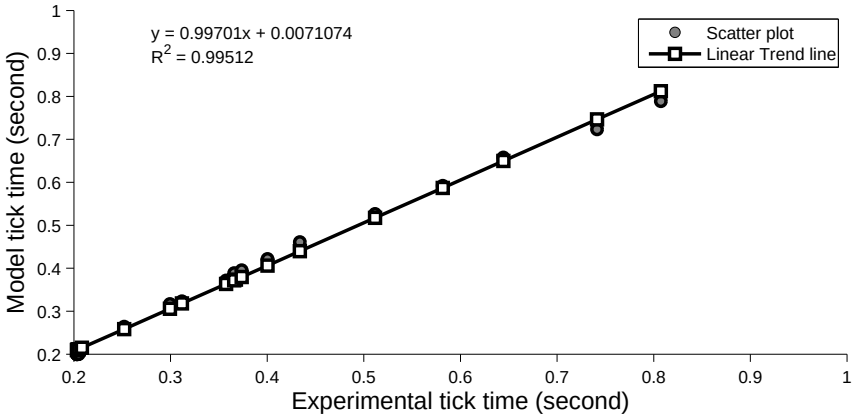


Figure 6.10: Experiment 4 linear regression analysis for Node 1

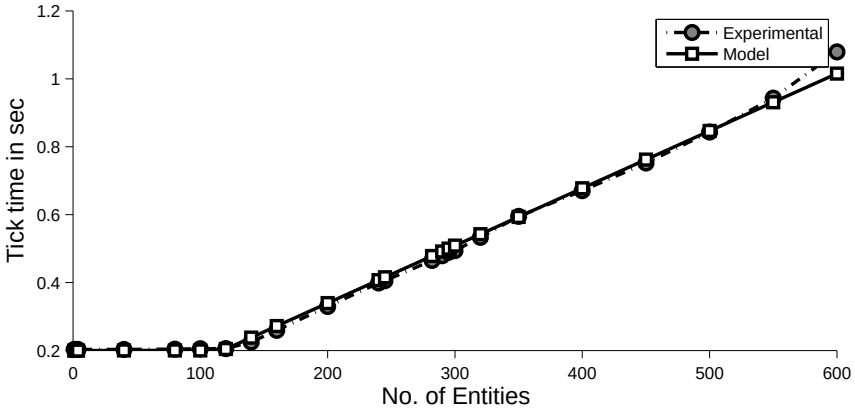


Figure 6.11: Experiment 4 CPU utilization XY graph for Node 2

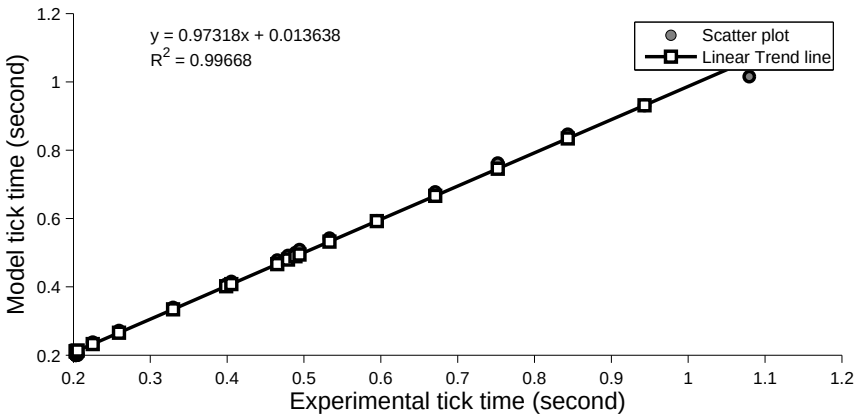


Figure 6.12: Experiment 4 linear regression analysis for Node 2

results, while satisfactory, are not so good in comparison with the previous results. It was observed that when the experiment was conducted the nodes were being used by other applications such as antivirus and remote desktop clients resulting in the spikes shown in the graphs around 100 entities. Though it is unclear exactly which application caused the discrepancy between the model and the DIS based VR-Forces Surrogate, it is clear from the graph that the node had increased CPU utilization when the number of entities per node was in range if 4 to 120. This experiment also serves as a test case that demonstrates the deviation between tick times predicted by the model and real experimental tick times, when the nodes running the experiment have CPU usage more or less than expected (due to any reason). The XY plot and regression plot for node 1 are shown in Figure 6.13 and Figure 6.14 respectively.The XY and regression plots for node 2 are shown in Figure 6.15 and Figure 6.16 respectively.The XY and regression plots for node 3 are shown in Figure 6.17 and Figure 6.18 respectively.The XY and regression plots for node 4 are shown in Figure 6.19 and Figure 6.20 respectively. The R^2 values approximated to two decimal places are 0.74, 0.88, 0.96, and 0.81 for nodes 1, 2, 3, and 4 respectively. The values are not as good as those for earlier experiments but still satisfactory.

CPU experiment 6, DIS application: VR-Forces Surrogate

The experiment on 4 nodes was repeated and the results were tabulated. The XY plot and regression plot for node 1 are shown in Figure 6.21 and Figure 6.22 respectively. The XY and

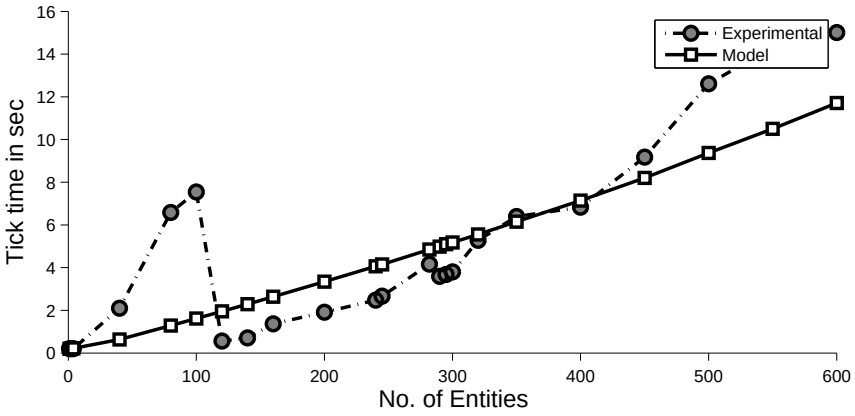


Figure 6.13: Experiment 5 CPU utilization XY graph for Node 1

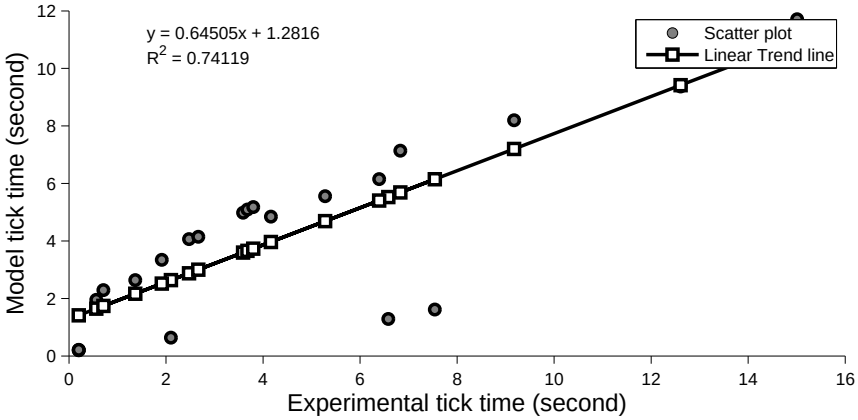


Figure 6.14: Experiment 5 linear regression analysis for Node 1

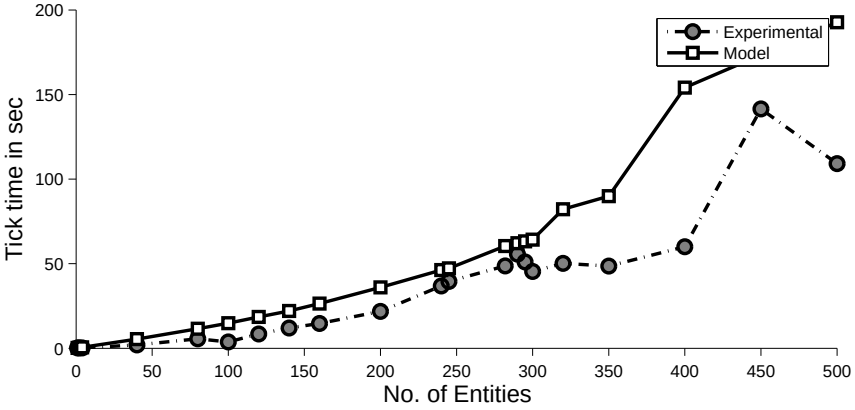


Figure 6.15: Experiment 5 CPU utilization XY graph for Node 2

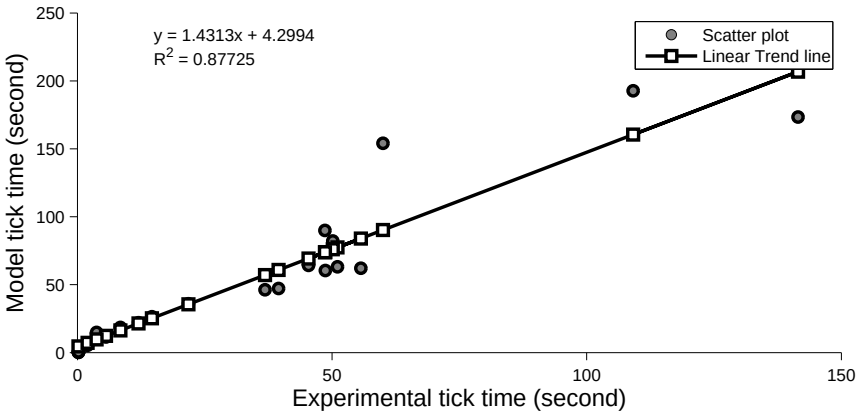


Figure 6.16: Experiment 5 linear regression analysis for Node 2

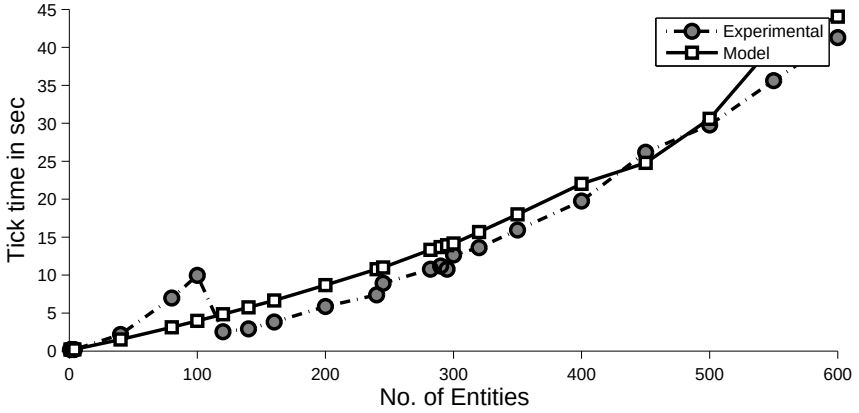


Figure 6.17: Experiment 5 CPU utilization XY graph for Node 3

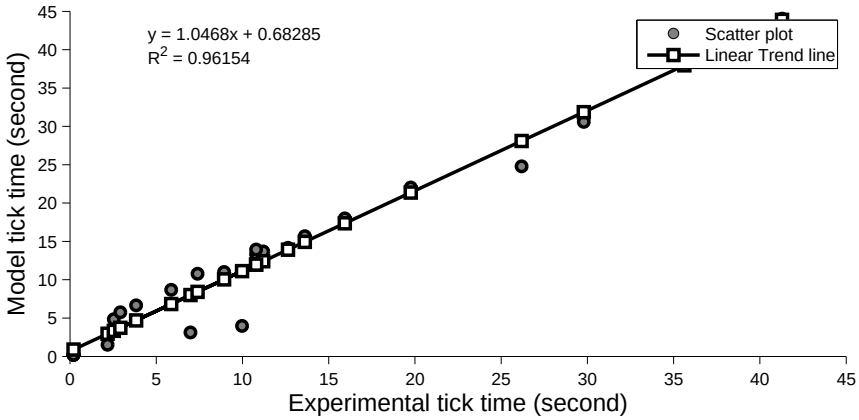


Figure 6.18: Experiment 5 linear regression analysis for Node 3

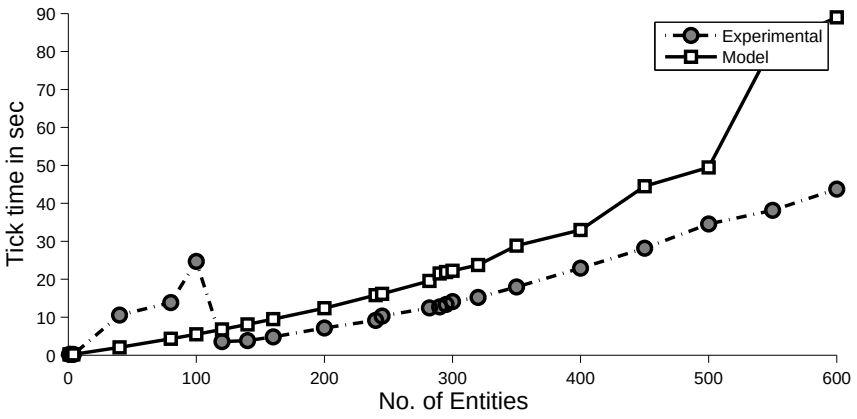


Figure 6.19: Experiment 5 CPU utilization XY graph for Node 4

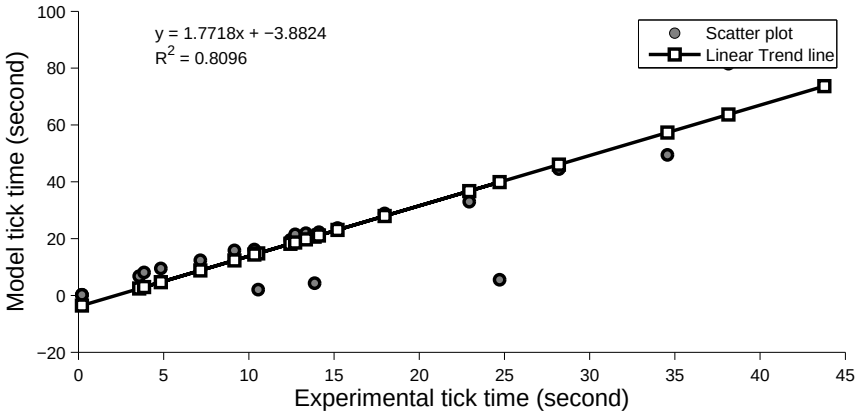


Figure 6.20: Experiment 5 linear regression analysis for Node 4

regression plots for node 2 are shown in Figure 6.23 and Figure 6.24 respectively. The XY and regression plots for node 3 are shown in Figure 6.25 and Figure 6.26 respectively. The XY and regression plots for node 4 are shown in Figure 6.27 and Figure 6.28 respectively. The R^2 values approximated to two decimal places are 0.90, 0.95, 0.97, and 0.95 for nodes 1, 2, 3, and 4 respectively. The values are quite high signifying a good fit by the model.

As seen from the graphs for experiment 6, the model predicts the increase in tick time very closely when compared to the real application behavior.

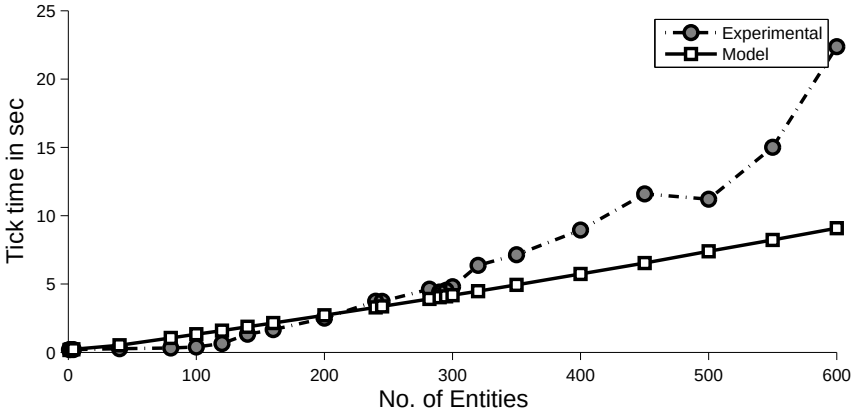


Figure 6.21: Experiment 6 CPU utilization XY graph for Node 1

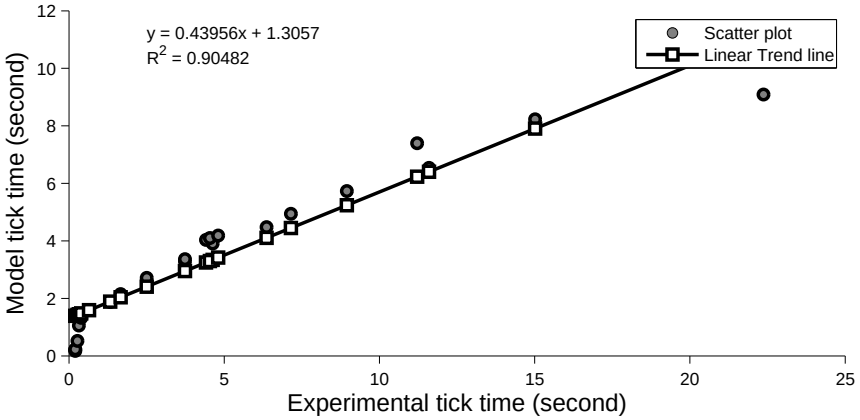


Figure 6.22: Experiment 6 linear regression analysis for Node 1

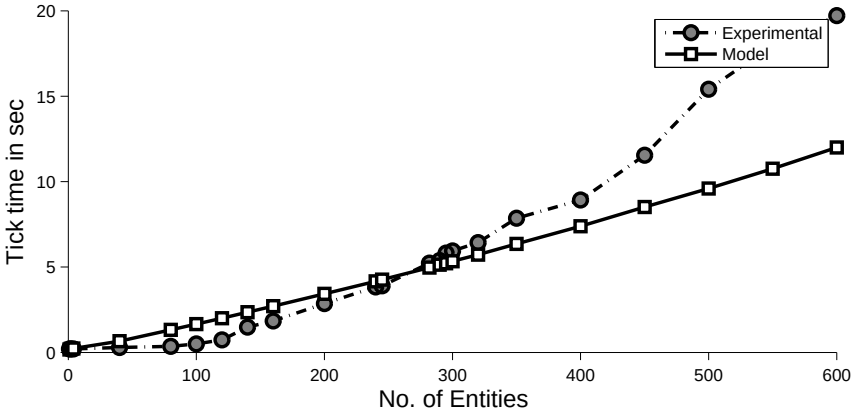


Figure 6.23: Experiment 6 CPU utilization XY graph for Node 2

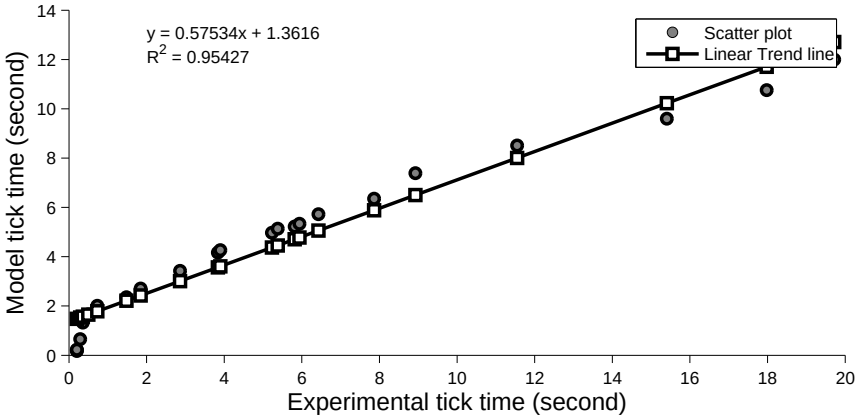


Figure 6.24: Experiment 6 linear regression analysis for Node 2

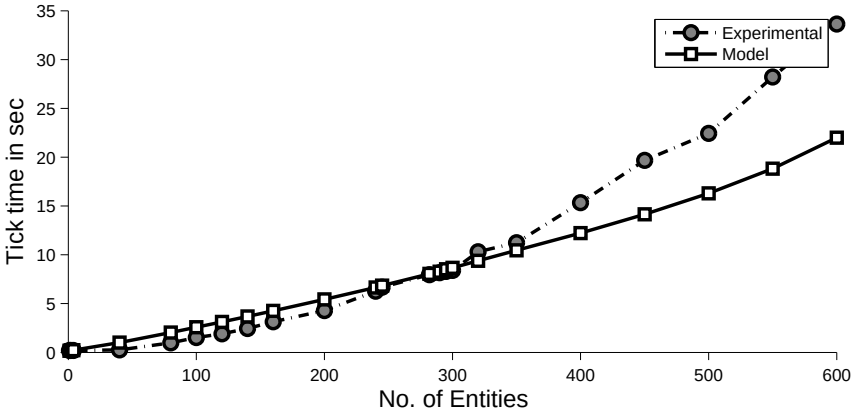


Figure 6.25: Experiment 6 CPU utilization XY graph for Node 3

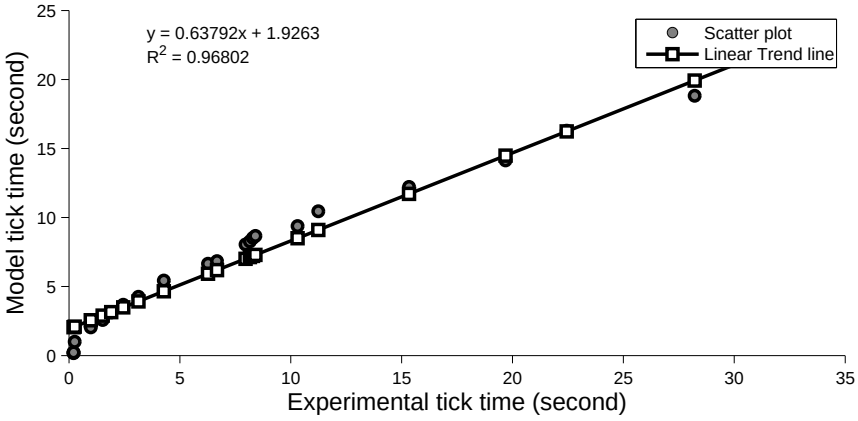


Figure 6.26: Experiment 6 linear regression analysis for Node 3

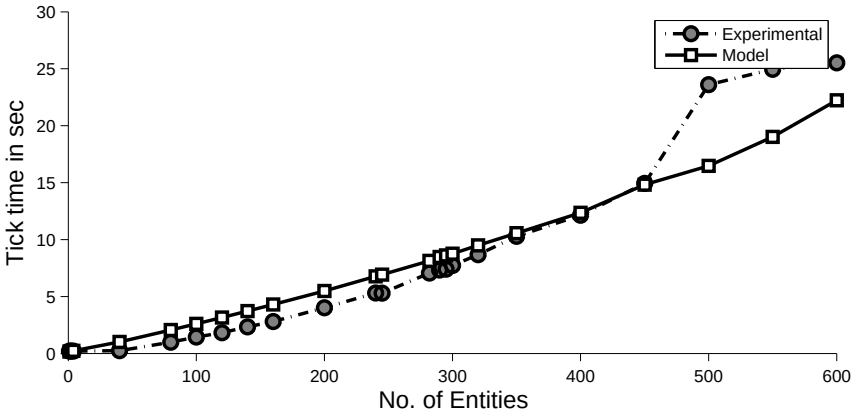


Figure 6.27: Experiment 6 CPU utilization XY graph for Node 4

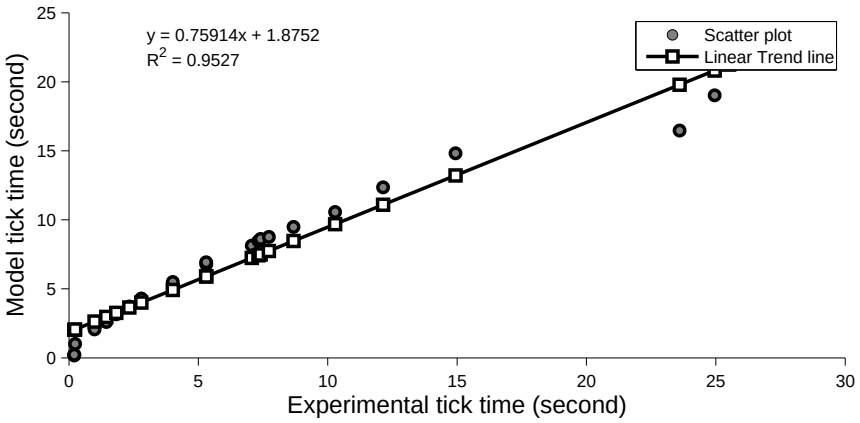


Figure 6.28: Experiment 6 linear regression analysis for Node 4

CPU experiment 7, DIS application: VR-Forces

The model is used to validate the predictions of the tick times with the actual VR-Forces application, i.e., not the Surrogate. The battle of 73 Easting which was simulated included 95 entities. The model could predict the tick time to be 200ms for the number of entities in the simulated battle. To test the model with larger numbers of entities on VR-Forces, additional experiments were run. The experiment was run with only friendly entities, with the number of entities set to 1, 2, 4, 40, 80, 160, 240, 245, 280, 290, 300, 310, and 320. Exactly one node was used in the experiment. The data was collected using the data logger provided by MÄK Technologies and was exported to a

database. The mean of maximum tick time for individual run was recorded for plotting the line and XY graph. Specifically, the tick time and the network tick time were configured to be 200ms and 210ms respectively. The simulation was run and limited to 60 sec, as the amount of data that was being generated was very large and the data logger software became unstable. The interarrival time between two ESPDU for the same entity was considered to be the tick times as the simulation was run on the same node and the effect of network would be negligible. The mean of interarrival times for each entity was calculated and the maximum of the means was selected as the tick time shown in the graph below. It was observed that the tick time was constant at 200ms for the first 240 entities irrespective of the total number of entities that were being simulated. The entities at the end of the list, i.e., entity numbered 300 when the total numbers of entities were 300, had the worst tick times. This was observed throughout the experiment and led to the simplification to determine the maximum of mean tick times by looking backwards with entity numbered highest to lowest.

The recorded data was not sufficient to determine all the parameters required to run the DIS model developed and approximations were done to determine the ESPDU service time and entity service time. The mean of the difference between successive ESPDU timestamps was calculated to determine the mean entity service time for runs having node loaded with entities ($>$ 300 entities). The means of many runs were used to determine a range of entity service times. The logic used to determine the entity service time was that each tick an ESPDU must be generated. When the node is loaded with entities, the difference between ESPDU timestamp for any two entities provides the approximate time needed to service the entity. There was no way to determine the ESPDU service time- the time needed to service an arrival of ESPDU from the network. With no access to source code and no help from the MÄK Technologies, the ESPDU service time was approximated to average of the entity service time/3 and entity service time/2. Experiments were conducted by increasing and decreasing the ESPDU service time.

The model was configured with these parameters:

1. tickTime=0.21s

2. netTickTime=0.21s

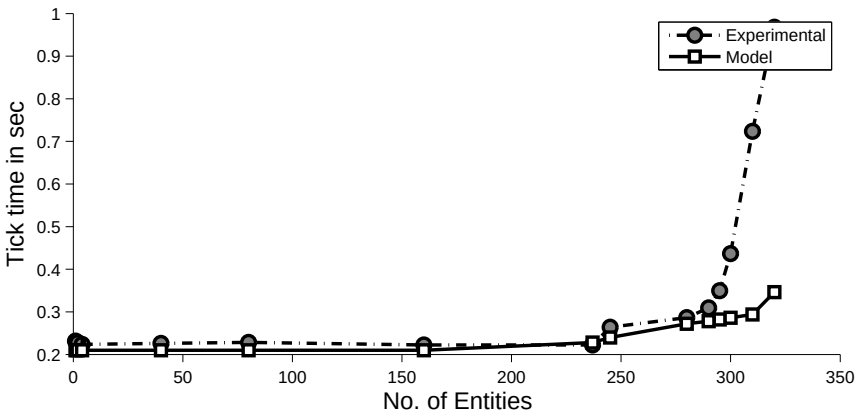


Figure 6.29: Experiment 7 CPU utilization XY graph

3. ESPDU service time = 300 micro sec

4. Entity Service Time = 775-790 micro sec

VR-Forces was configured with these parameters:

1. tickTime = 0.2s (200ms)

2. netTickTime = 0.21s (210ms)

3. Collection Duration = 1 min

A XY graph is plotted, which is shown in Figure 6.29. The regression plot is shown in Figure 6.30 with R^2 value to be 0.74.

As seen from the graphs above, the model could predict fairly accurately the tick time till 290 entities, where the tick time is 278ms as per the model, and experimentally it was found to be 309ms. The spike in the graph at around 320 entities could not be attributed in the DIS model. The R^2 value of 0.74 which is not as high as for other experiments could be attributed to these inaccurate values at entities greater than 300. With no accurate data available for service times of PDU and entities, the generated graph from the model is expected to deviate. An increase in the entity service time will shift the DIS model curve to the left, and increase in the ESPDU service time raises the curve upward as shown in the below graphs. The spike cannot be accounted for by the model, and

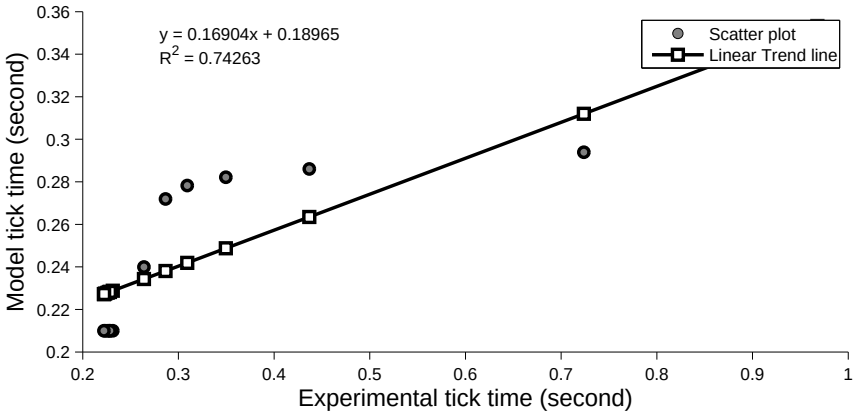


Figure 6.30: Experiment 7 linear regression analysis

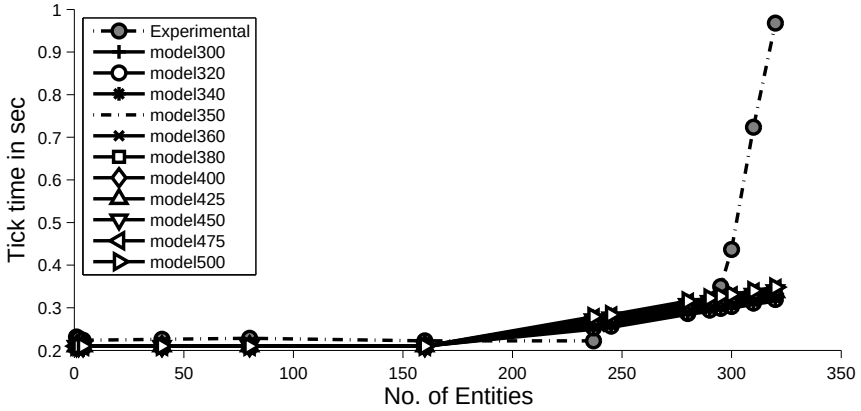


Figure 6.31: XY graph for validation with varying ESPDU service time

is estimated to be dependent on the architecture and optimizations that are incorporated in the DIS application. The graph when the configuration used for ESPDU service time is 320, 340, 350, 360, 380, 400, 425, and 500 is given in Figure 6.31.

6.2 Network traffic validation and experimentation

Network experiment 1, DIS application: VR-Forces Surrogate

The DIS application, VR-Forces Surrogate is run with sending FPDU turned off. As per [12], the majority, 90%, of the network traffic is due to ESPDU. FPDU and DPDU being probabilistic in the model, removing them will increase the accuracy of the bandwidth estimate. FPDU and DPDU

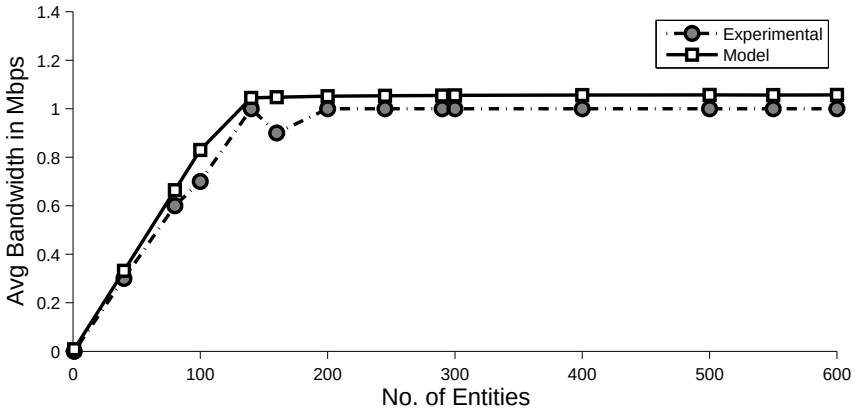


Figure 6.32: Network experiment 1 XY graph

are smaller in size than ESPDU, and hence the network utilization is not affected if it is considered to be replaced by another ESPDU. The experiment was with the number of entities varying as 1, 40, 80, 100, 140, 160, 200, 245, 290, 300, 400, 500, 550, and 600. Exactly one node is used in the experiment. The bandwidth is monitored using a tool called Wireshark. Wireshark is a software available freely to monitor data on Ethernet LAN. Data is also collected to determine the ESPDU and entity service times so as to enable the DIS model developed to be run and collect predicted network traffic. The graph displays the number of entities on the x-axis and the bandwidth in Mbps on the y-axis. Both model and experimental bandwidth have been plotted in Figure 6.32. Figure 6.33 shows the regression plot for the experiment.

As seen in the graphs, the predicted estimates by the model are fairly close to the real DIS application. The R^2 value of 0.99 is quite high signifying a good fit. It must be noted that the network simulation in the model omits two details: propagation delay of the Ethernet frames between nodes and the "jamming signal" which is sent if a collision is detected. Hence, the models prediction of the network traffic would provide an upper limit of the DIS traffic rather than an exact prediction of the network utilization.

Network experiment 2, DIS application: VR-Forces Surrogate

Experiment 1 was repeated with 2 nodes in the DIS application for entities per node to be 1, 40, 80, 100, 140, 160, 200, 245, 290, 300, 400, 500, 550, and 600. The experiment was repeated twice

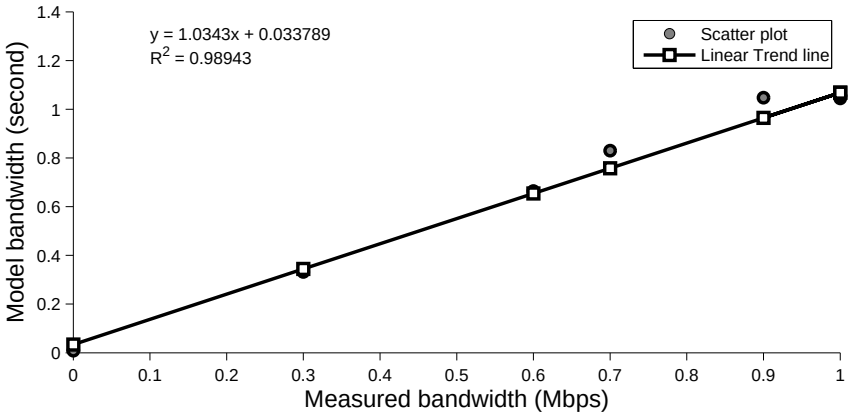


Figure 6.33: Network experiment 1 linear regression analysis

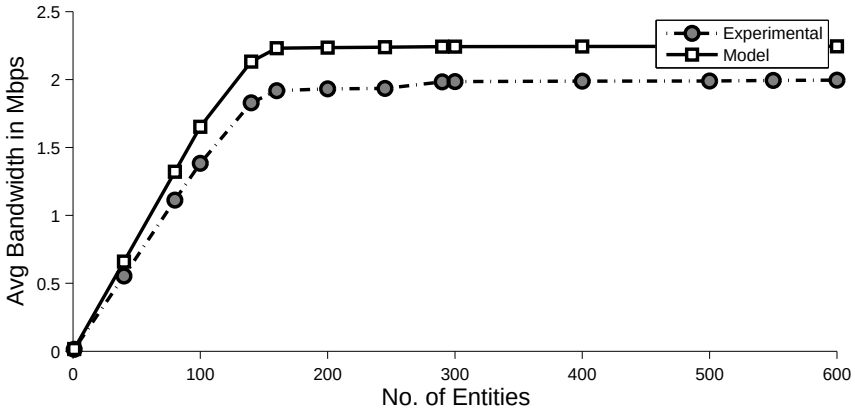


Figure 6.34: Network experiment 2a XY graph

(2a and 2b) to see if there was any significant variance in the collected data. The XY graphs are shown in Figure 6.34 and Figure 6.36. The regression plots are shown in Figure 6.35 and Figure 6.37 for experiments 2a and 2b.

The model predicted the upper bound of 2.2Mbps, whereas the experimental results showed about 2Mbps. The R^2 values are 0.997 and 0.998 for experiments 2a and 2b respectively validating a very good fit by the model. *Network experiment 3, DIS application: VR-Forces Surrogate*

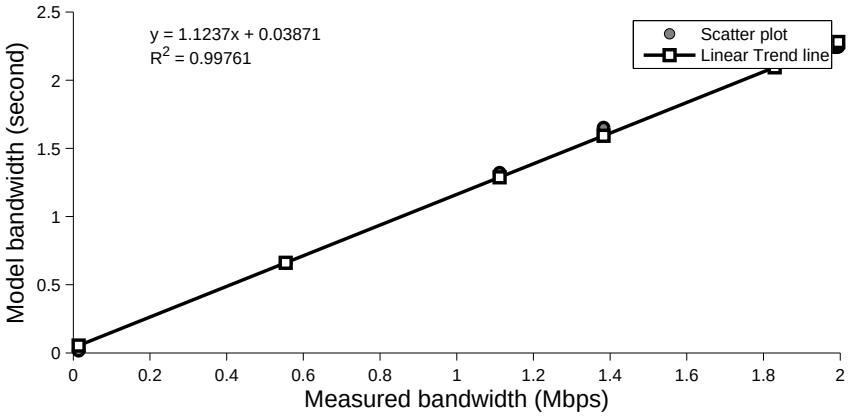


Figure 6.35: Network experiment 2a linear regression analysis

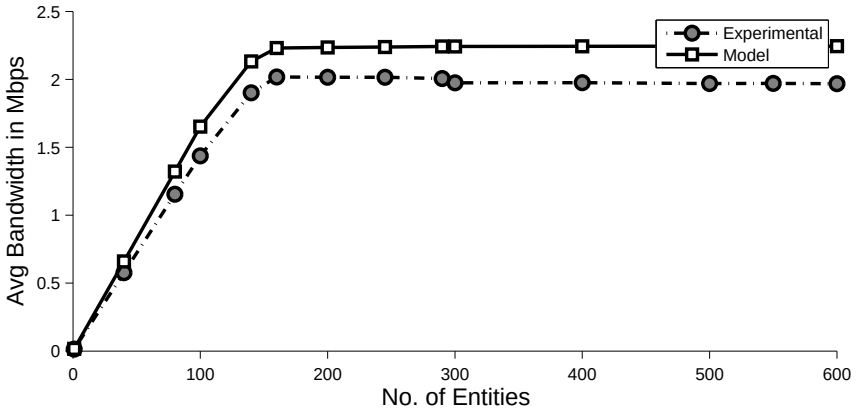


Figure 6.36: Network experiment 2b XY graph

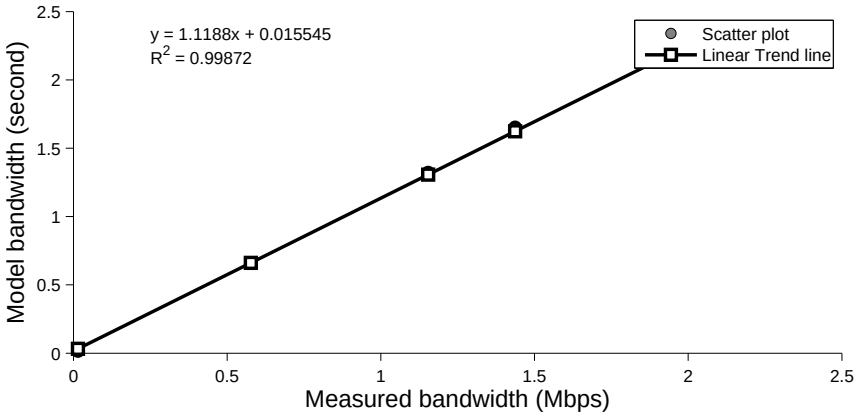


Figure 6.37: Network experiment 2b linear regression analysis

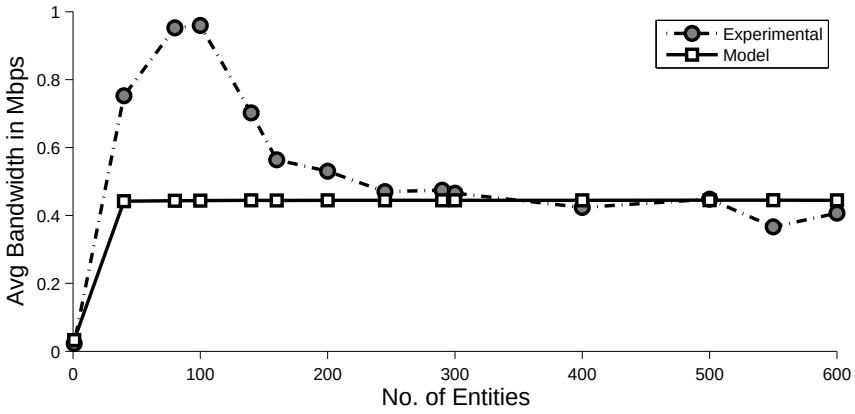


Figure 6.38: Network experiment 3 XY graph

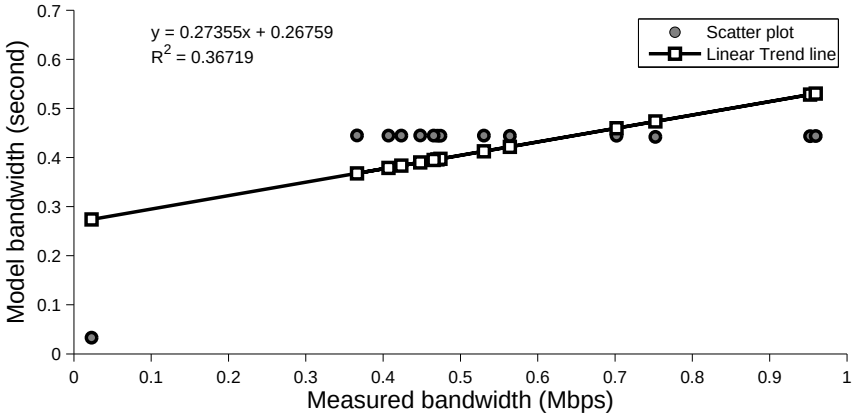


Figure 6.39: Network experiment 3 linear regression analysis

Experiment 1 was repeated with 4 nodes in the DIS application for entities per node to be 1, 40, 80, 100, 140, 160, 200, 245, 290, 300, 400, 500, 550, and 600. The XY graph is shown in Figure 6.38. Figure 6.39 displays the regression plot for the experiment.

The initial increase in traffic between 50 to 100 entities in the experimental data cannot be explained. The model approximately predicts the bandwidth usage as the number of entities per node is higher than 300. The R^2 value of 0.37 is bad and is due the initial increase in traffic between 50 and 100 entities which the model did not predict.

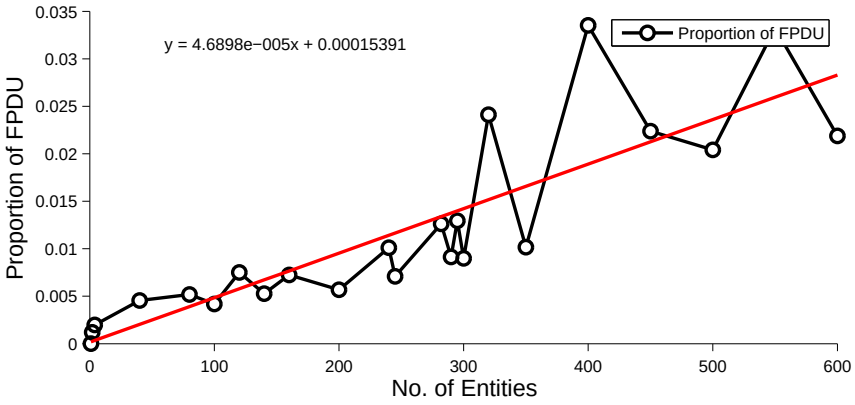


Figure 6.40: Proportion of Fire PDU as number of entities increases

6.3 Lessons learned

Many experiments were conducted, many simulation runs were executed, and many different kinds of analysis were done in the process of arriving at the results displayed above. Several key observations and lessons learned regarding the experiments are given below.

1. As the number of entities increases, the proportion of the FPDU and DPDU generated with respect to total PDUs increases. The experimental scenarios consisted of equal number of friendly and opposing forces which interacted for a certain period of time. The graphs denoting the FPDU and DPDU proportions are given in Figure 6.40 and Figure 6.41 for one of the runs. The equation of the curve fitted to the graphs as generated by Matlab is shown in Figure 6.40 and Figure 6.41.

2. The processing time for ESPDUs, FPDUs, and DPDUs are not constant, but vary with the number of entities. However, the processing times are so small (in micro seconds in case of the VR-Forces Surrogate) that variation has very little effect on the overall prediction by the DIS model.

3. As the number of nodes increases, the collisions on the Ethernet also increase based on the network tick time and may result in a decrease in network traffic when compared to a lesser number of nodes.

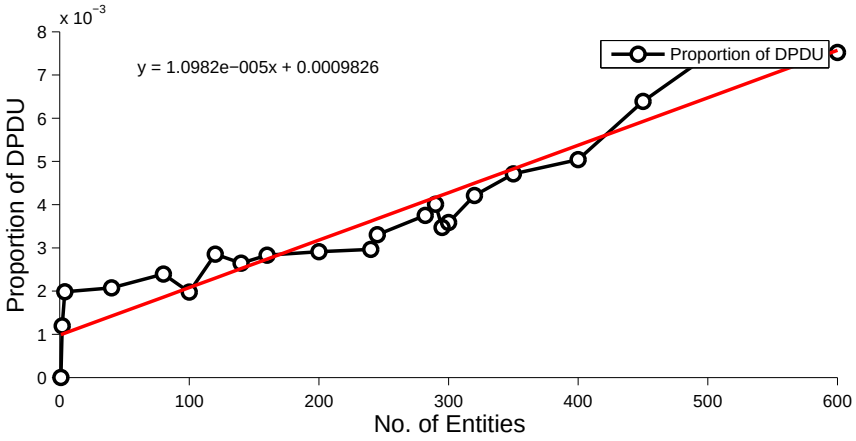


Figure 6.41: Proportion of Detonation PDU as number of entities increases

4. The firing range of the entities in question such as tanks does not affect the overall tick time. The frequency of FPDU will increase, but the processing times for FPDU and DPDU are so small that significant changes in the overall result will only result from a very large increase in the interaction between entities due to the increase in the range. It also depends on the number of entities that have been destroyed, as that decreases the number of FPDUs and DPDUs generated.

5. As the number of entities that have been destroyed increases, the total load on the system decreases. The remaining entities start to tick at the expected rate and the network traffic decreases. The data collected as time passes by results in inaccurate results. Experiments must be run with a initial quiescent period when forces do not interact followed by a period of interaction, and they must be stopped soon after so as to not give much time for the generated data to smoothen out.

6. The internal architecture of the DIS application plays a major role in determining the overall performance and the way the system degrades when the number of entities increases. A small change in the VR-Forces Surrogate to optimize the performance resulted in tick time degradation that could not be predicted by the model. For the most precise predictions, the

model may have to be changed to reflect any special architectural features or optimizations in the DIS application with respect to increasing performance and network optimizations.

7. Experiments must be run standalone and not within an integrated development environment (IDE). The results observed in those two situations were significantly different. The ease of use and debugging in an IDE does not scale to the desired accuracy and results which are obtained when an experiment is run in a standalone Java virtual machine on a command line.

8. Instrumentation in Java can be achieved using Aspects. These were used initially to tap into the DIS application VR-Forces Surrogate to log the necessary data such as entity service time, and ESPDU processing time. It was observed that exceptions were encountered within the Aspects code making it necessary to dig deep inside Aspects itself, resulting in an unnecessary waste of time. Finally, a conventional way was used by adding Java code in between to collect statistics. Unless one is absolutely sure about using Aspects/instrumentation and has sufficient time to debug the Aspects itself if errors are encountered, the conventional method is the best when source code is available.

9. The number of threads and thread scheduling used is a part of the DIS application architecture and will determine the performance and degradation of the application. The model does not consider this.

10. Experiments can be automated by doing a small amount of batch scripting which is really useful to save time and effort. Logging in the form of CSV also helps as it can be imported easily into analysis tools and graphs can be plotted easily. Automation could not be achieved when the speeds of nodes were significantly different as different nodes ended the simulation at different instants in time and a message passing mechanism would be needed to synchronize the execution between experimental runs. Due to time and scope limitations, this was not implemented; instead, runs were executed manually when automation of successive runs were not possible.

CHAPTER 7

CONCLUSION

This chapter describes the overall research findings. Section 7.1 provides the answers to the research questions which were stated earlier and Section 7.2 presents the future work that can be carried ahead.

7.1 Results and research findings

This research was intended to produce answers to several questions. The questions posed and the answers uncovered are given below.

1. Can a distributed simulation based on DIS be modeled at all using discrete event simulation?

   Yes, discrete event simulation of a distributed simulation based on the DIS protocol is possible. The comparisons of experiment and model data during validation clearly show that the model developed predicts CPU utilization and network traffic in the DIS based distributed simulation quite well in most cases. It must also be noted that the model developed is very tightly coupled with the architecture of the DIS Application. Any performance optimizations and architecture changes in the application must be incorporated in the model. Because VR-Forces is a proprietary application, sufficient data were not available to understand its architecture and modify the model to better model the degenerative part of the curve when the number of entities increases beyond 290 in the graph of number of entities versus tick time.

2. Can a model detect or predict node or network overload in a DIS simulation?

   Yes, the model can be used to predict the number of entities that can be modeled in a DIS application on a node without exceeding the required tick times of the entity. The graph of the

number of entities versus the tick time in various experiments shows that the model is able to predict the threshold point of number of entities without exceeding the configured tick time. The results show that network usage saturates and becomes constant as the number of entities on a certain number of nodes increases. The model can be used to predict this bandwidth requirement and hence estimate the network overload.

3. Can a model recreate "closely" the DIS network traffic compared to real DIS simulation?

   Yes, it is possible to recreate the network traffic closely by using probability distribution for various PDUs.

4. How does DIS network traffic increase when scenario size increases, and can a model predict this?

   The bandwidth graphs generated by model clearly provided an upper bound on the variation of the network traffic as scenario size increases. The model produced an upper bound, rather than a precise prediction because the model developed omitted certain network details. Improvements could be made to get more accuracy.

5. Is it possible to answer questions about whether a DIS system configuration with a given number of nodes and number of entities per node is able to handle battles of size two to three times the size of the Battle of 73 Easting?

   Yes, as the model predicted and was validated experimentally, 240 entities could be supported on the experimental node. With around 100 entities involved in the battle, at a maximum, a battle twice the size could be simulated on the experimental node. The answer to this question is a direct relation to the answer to the question *2* which was posed above.

7.2 Future work

The model developed does not predict the CPU degradation of VR-Forces accurately as number of entities on the node increases. This probably depends on the architecture of the DIS application. The model could be modified to support various architectures which are present in

the industry and could be made a configurable option. This allows for selecting the architecture of the DIS application before running the model and hence providing a more accurate prediction. The network model used does not consider propagation delay and "jamming signal" which can be incorporated in the Ethernet model present in the developed DIS model. The network model must be improved for better accuracy in results of bandwidth usage and CPU utilization. More experiments must be conducted with a larger set of nodes to determine if the model behaves accurately as the number of nodes increases.

APPENDICES

APPENDIX A

VR-FORCES SCENARIO PLANS

All the plans provided below are snapshots taken from VR-Forces tool.

```
M3A2 26
  When (Resource(M791-AP-25mm) < 10) do
    Set Resource(weapon|M791-AP-25mm)=1500
  endwhen
  When (Entity-Left-Of-Line: SELF,, Entity:"" Line:"Phase Line 1") do
    While (NOT(Entity-Destroyed ,, Entity: "T72 2")) do
      Set Data Request; Object: M3A2 26 Request: Fire Now:"T72 2"
    endwhile
    While (NOT(Entity-Destroyed ,, Entity: "T72 1")) do
      Set Data Request; Object: M3A2 27 Request: Fire Now:"T72 1"
    endwhile
  endwhen
  When (Entity-Left-Of-Line: SELF,, Entity:"" Line:"Phase Line 2") do
    While (NOT(Entity-Destroyed ,, Entity: "T72 18")) do
      Set Data Request; Object: M3A2 28 Request: Fire Now:"T72 18"
    endwhile
    While (NOT(Entity-Destroyed ,, Entity: "T72 8")) do
      Set Data Request; Object: M3A2 29 Request: Fire Now:"T72 8"
    endwhile
  endwhen
  When (Entity-Left-Of-Line: SELF,, Entity:"" Line:"Phase Line 3") do
    While (NOT(Entity-Destroyed ,, Entity: "T72 15")) do
      Set Data Request; Object: M3A2 30 Request: Fire Now:"T72 15"
    endwhile
    While (NOT(Entity-Destroyed ,, Entity: "T72 33")) do
      Set Data Request; Object: M3A2 31 Request: Fire Now:"T72 33"
    endwhile
    While (NOT(Entity-Destroyed ,, Entity: "T72 11")) do
      Set Data Request; Object: M3A2 26 Request: Fire Now:"T72 11"
    endwhile
    While (NOT(Entity-Destroyed ,, Entity: "T72 35")) do
      Set Data Request; Object: M3A2 27 Request: Fire Now:"T72 35"
    endwhile
    While (NOT(Entity-Destroyed ,, Entity: "T72 39")) do
      Set Data Request; Object: M3A2 28 Request: Fire Now:"T72 39"
    endwhile
    While (NOT(Entity-Destroyed ,, Entity: "T72 12")) do
      Set Data Request; Object: M3A2 29 Request: Fire Now:"T72 12"
    endwhile
  endwhen
  Move-Along Route: "Route 1"
```

```
M1A2 18
  When (Resource(M829A1-AP-120mm) < 2) do
    Set Resource(weapon|M829A1-AP-120mm)=20
  endwhen
  When (Resource(M830-HEAT-120mm) < 2) do
    Set Resource(weapon|M830-HEAT-120mm)=20
  endwhen
  When (Resource(M2-12.7mm) < 20) do
    Set Resource(weapon-2|M2-12.7mm)=500
  endwhen
  When (Entity-Left-Of-Line: SELF,, Entity:"" Line:"Phase Line 2") do
    While (NOT(Entity-Destroyed ,, Entity: "T72 9")) do
      Set Data Request; Object: M1A2 18 Request: Fire Now:"T72 9"
    endwhile
    While (NOT(Entity-Destroyed ,, Entity: "T72 3")) do
      Set Data Request; Object: M1A2 19 Request: Fire Now:"T72 3"
    endwhile
  endwhen
  When (Entity-Left-Of-Line: SELF,, Entity:"" Line:"Phase Line 1") do
    While (NOT(Entity-Destroyed ,, Entity: "T72 10")) do
      Set Data Request; Object: M1A2 20 Request: Fire Now:"T72 10"
    endwhile
    While (NOT(Entity-Destroyed ,, Entity: "T72 22")) do
      Set Data Request; Object: M1A2 21 Request: Fire Now:"T72 22"
    endwhile
  endwhen
  Move-Along Route: "Route 6"
```

Figure A.1: VR-Forces Plans - 1

Figure A.2: VR-Forces Plans - 2

APPENDIX B

CPU AND NETWORK MODEL CODE EXCERPTS

All the code presented are snapshots from the IDE eclipse.

```
public CPU(Node node,Integer noOfCore){
    this.node = node;
    this.noOfCores = noOfCore;
    initCPU();

}
/**
 * @param noOfCore
 *
 * Initializes the state of the CPU.
 *
 */
private void initCPU() {
    busy = new ArrayList<Boolean>();
    for(int i = 0; i < noOfCores; i++){
        busy.add(false);
    }
    taskInCore = new ArrayList<ProcessingTask>(noOfCores);
    tasksInQueue = new ArrayList<ProcessingTask>();
}
/**
 *
 * @param task ProcessingTask
 *
 * API to add task to the CPU for processing. If the CPU is busy, then
 * the task is added to a queue for later processing. The task will be
 * assigned to any available core. Tasks do not have priority.
 */
public void addTask(ProcessingTask task){
    if(isBusy()){
        DISMLogManager.logFine("cpu", node.getId() + "," + SystemState.currentSimulationTime + ",A,true," + tasksInQueue.size() + "," + task);
        tasksInQueue.add(task);
        DISMLogManager.logFine("cpuqsize", node.getId() + "," + SystemState.currentSimulationTime + "," + getQueueSize());
    }else{
        int indexOfFreeCore = busy.indexOf(false);
        if(taskInCore.size() <= indexOfFreeCore){
            taskInCore.add(task);
        }else{
            taskInCore.set(indexOfFreeCore, task);
        }
        busy.set(indexOfFreeCore, true);
        task.processingStarted();

        DISMLogManager.logFine("cpu", node.getId() + "," + SystemState.currentSimulationTime + ",A,false," + tasksInQueue.size() + "," + task + ","

        ProcessingCompleteEvent pce = new ProcessingCompleteEvent(task);
        EventNotice pcen = new EventNotice(pce, SystemState.currentSimulationTime + task.serviceTime());
        node.getEngine().addEventNotice(pcen);

        if(task instanceof PDUEntity){
            DISMLogManager.logFine("pdu", node.getId() + "," + SystemState.currentSimulationTime + ",P," + ((PDUEntity)task).getDisPDU());
        }
    }
}
/**
 *
 * @param task ProcessingTask
 * @throws DISModelException If the API is called for a task which has not been associated to a core.
 *
 * API to remove a task from the core. ProcessingTask.same()
 * method is invoked to determine if argument task is same
 * as the one in the core.
 *
 */
public void removeTask(ProcessingTask task){
    int indexToRemove = -1;
    for(int i = 0; i < noOfCores; i++){
        if(busy.get(i).equals(true)){
            ProcessingTask coreTask = taskInCore.get(i);
            if(task.same(coreTask)){
                indexToRemove = i;
                break;
            }
        }
    }
    if(indexToRemove != -1){
        taskInCore.set(indexToRemove, null);
        busy.set(indexToRemove, false);

        DISMLogManager.logFine("cpuqsize", node.getId() + "," + SystemState.currentSimulationTime + "," + getQueueSize());
        DISMLogManager.logFine("cpu", node.getId() + "," + SystemState.currentSimulationTime + ",D,"+ isBusy() + "," + tasksInQueue.size() + "," + t

    }else{
        throw new DISModelException("Remove task called for a task not in any of the cores");
    }
}
```

Figure B.1: Few important code excerpts of class CPU

```
public NIC(Node node, Double bandWidth){
    this.node = node;
    this.bandWidth = bandWidth;
    sendQueue = new ArrayList<NetworkPDU>();
    receiveQueue = new ArrayList<NetworkPDU>();
    sendQueueSizeInBits = 0.0;
    receiveQueueSizeInBits = 0.0;
    slotTime = (Integer.valueOf(ConfigurationManager.getInstance().getConfigValue(DISModelConstants.CONFIG_KEY_MIN_MAC_FRAME_SIZE_BITS)) / bandWidth)
    noOfDroppedPackets = 0L;
}
public void send(NetworkPDU pdu){
    if(sendQueue.isEmpty()){
        if(!sendToEthernet(pdu)){
            //If Ethernet cant be used then add PDU to queue
            sendQueueSizeInBits += pdu.sizeInBits();
            sendQueue.add(pdu);
        }
    }else{
        sendQueueSizeInBits += pdu.sizeInBits();
        sendQueue.add(pdu);
    }
}
private Boolean sendToEthernet(NetworkPDU pdu) {
    Boolean success = false;
    if(!Ethernet.isDataBeingTransmitted()){
        //Reset NIC state to address reseting the exponential backoff counter
        collisionDetected = false;
        exponentialBackOffCounter = 0;

        Double noOfNetworkPduBits = 0.0;
        noOfNetworkPduBits += pdu.sizeInBits();
        noOfNetworkPduBits += SystemDetails.getInstance().getHeaderOverhead();

        //Min MAC Frame size is x bits given in the configuration
        if(noOfNetworkPduBits < SystemDetails.getInstance().getMinMACFrameSizeBits()){
            noOfNetworkPduBits = Double.valueOf(SystemDetails.getInstance().getMinMACFrameSizeBits());
        }

        Ethernet.add(pdu, noOfNetworkPduBits);
        Integer timeToTransmit = calculateTimeToTransmit(noOfNetworkPduBits);

        //Create event to end Transmission on Ethernet from this node
        EndOfNICTransmissionEvent event = new EndOfNICTransmissionEvent(pdu);
        EventNotice nicNotice = new EventNotice(event, SystemState.currentSimulationTime + timeToTransmit);
        node.getEngine().addEventNotice(nicNotice);
        success = true;
    }else if(existsAttemptReTransmit == false){
        //Schedule a retransmission attempt
        collisionDetected = true;
        //Calculate backoff range, with max counter value to be 10
        Integer backoffSlotTimes = random.nextInt(new Double(Math.pow(2, (exponentialBackOffCounter > 10 ? 10 : exponentialBackOffCounter))).intVal
        exponentialBackOffCounter++;
        scheduleReTransmitAfter2SlotTimes();
        success = true; //Dropping of packet is a successful state making retransmission un-necessary
        noOfDroppedPackets++;
        return success;
    }

    //Schedule a retransmission attempt at slot time * backoff
    Integer backOffPeriod = new Double(backoffSlotTimes * slotTime).intValue();

    //Create event to attempt Re-Transmission on Ethernet from this node
    NICTransmitAttemptEvent event = new NICTransmitAttemptEvent(pdu);
    EventNotice nicNotice = new EventNotice(event, SystemState.currentSimulationTime + backOffPeriod);
    node.getEngine().addEventNotice(nicNotice);
    existsAttemptReTransmit = true;

    success = false;
}
return success;

public void attemptToReTransmit(){
    if(sendQueue.size() > 0){
        if(sendToEthernet(sendQueue.get(0))== true){
            sendQueue.remove(0);
        }
    }
}

public void scheduleReTransmitAfter2SlotTimes(){
    if(sendQueue.size() > 0 && existsAttemptReTransmit == false){
        //Create event to attempt Re-Transmission on Ethernet from this node
        NICTransmitAttemptEvent event = new NICTransmitAttemptEvent(sendQueue.get(0));
        EventNotice nicNotice = new EventNotice(event, new Double(SystemState.currentSimulationTime + (2 * slotTime)).longValue());
        node.getEngine().addEventNotice(nicNotice);
        existsAttemptReTransmit = true;
    }
}

private Integer calculateTimeToTransmit(Double noOfNetworkPduBits) {
    Integer timeToTransmit = 0;
    Double transmissionTime = noOfNetworkPduBits/bandWidth;
    timeToTransmit = new Double(SystemDetails.getInstance().getSecMultipleFactor() * transmissionTime).intValue();
    return timeToTransmit;
}

public void addToReceiveBuffer(NetworkPDU pdu, Double sizeInBits){
    receiveQueue.add(pdu);
    pdu.updateSizeInBits(sizeInBits);
    receiveQueueSizeInBits += pdu.sizeInBits();
    //Schedule a process event to attend to the received PDU
    ProcessDISPDUEvent event = new ProcessDISPDUEvent();
    EventNotice disNotice = new EventNotice(event, SystemState.currentSimulationTime);
    node.getEngine().addEventNotice(disNotice);
}

public NetworkPDU readFromReceiveBuffer(){
    NetworkPDU npdu = receiveQueue.remove(0);
    receiveQueueSizeInBits -= npdu.sizeInBits();
    return npdu;
}

public void clearReTransmitAttemptFlag(){
    existsAttemptReTransmit = false;
}
```

Figure B.2: Few important code excerpts of class NIC

```
public static void add(NetworkPDU pdu, Double sizeInBits){
    bitsOnNetwork = sizeInBits;
    queue.add(pdu);
    dataBeingTransmitted = true;
    List<Node> nodes = SystemDetails.getInstance().getNodes();
    for(int i=0; i<nodes.size(); i++){
        NetworkPDU clonePdu = pdu.clone();
        Node otherNode = nodes.get(i);
        clonePdu.setNode(otherNode);
        otherNode.getNic().addToReceiveBuffer(clonePdu,sizeInBits);
    }
    DISMLogManager.logFine("ethernet", SystemState.currentSimulationTime + "," + dataBeingTransmitted + "," + pdu + "," + sizeInBits);

    if(SystemState.currentSimulationTime < secCount * SystemDetails.getInstance().getSecMultipleFactor()){
        totalBitsOnNetPerSec += sizeInBits;
    }else{
        bandwithMap.put(secCount, totalBitsOnNetPerSec);
        totalBitsOnNetPerSec = 0.0;
        secCount++;
        totalBitsOnNetPerSec += sizeInBits;
    }
}
```

Figure B.3: Few important code excerpts of class Ethernet

```
public class Node {
    private String id;
    private Integer initialNoOfEntites;
    private Integer noOfEntites;
    private EventProcessingEngine engine;
    private String entityServiceTimeRange;
    private Random nodeRandom;
    private Integer minServiceTime;
    private Integer maxServiceTime;
    private Map<String, String> pduServiceTimeRangeMap;
    private Integer noOfCores;
    private CPU cpu;
    private Integer tickTime;
    private NIC nic;;
    private Random disPduRandom;
    private Map<String, Integer> pduCount;
    private List<Entity> entities;
    private Integer noOfDetonationsGenerated;
    private Integer nwTickTime;
    private Double nwSendTaskProcTime;
    private Double entityFactor = 1.0;

    public Boolean run(){
        return engine.run();
    }
```

Figure B.4: Few important code excerpts of class Node

APPENDIX C

DATA SHEETS

The data used in the experiments are presented here using snapshots from Microsoft excel. Not all the experiments are presented.

				range=5				
Entities		ESPDUst	FPDUst	FPDUst	EPDUs No	FPDUs No	DPDUS Nc	P(FPDU)	P(DPDU)
1		35123.02			492				
2		28617.19	3752728	169623	846	3	1	0.003529	0.001176
4		31735.27	5947196	26560	1510	4	3	0.002637	0.001978
40		16593.43	7539621	10572.63	14356	75	35	0.005185	0.002419
80		11019.94	3291025	7354.954	29083	131	65	0.004474	0.00222
100		9296.972	8085902	8051.667	35601	202	87	0.005628	0.002424
120		8730.422	3998258	11078.48	43223	194	98	0.004458	0.002252
140		7692.47	4673936	7813.733	47372	285	131	0.005964	0.002741
160		6077.01	6112456	5370.266	55246	453	143	0.008112	0.002561
200		5886.974	5566319	5673.851	66143	482	168	0.007216	0.002515
240		5615.021	4378520	7330.081	64979	486	210	0.0074	0.003198
245		6744.818	4747457	5725.922	67643	737	206	0.010746	0.003004
282		7249.804	4711956	5920.716	63698	551	232	0.008545	0.003598
290		6745.446	4230347	12622.97	62597	651	254	0.010252	0.004
295		7218.906	4027286	5269.685	65439	907	273	0.013615	0.004098
300		6007.119	3717023	6282.846	63533	590	272	0.009162	0.004224
320		6651.981	3861210	4418.795	69656	1331	278	0.018677	0.003901
350		7096.83	3866191	6586.155	69456	819	283	0.011607	0.004011
400		6312.54	3360451	4175.795	71238	1599	341	0.021851	0.00466
450		7517.194	2831211	6411.823	70431	976	351	0.013601	0.004891
500		6944.744	3206712	4602.135	70762	1702	422	0.023352	0.00579
550		7200.145	3095477	7149.357	62112	1338	510	0.020919	0.007974
600		6973.539	3160288	4840.161	69698	2384	508	0.032842	0.006998

range=5	only update, no receive or send		receive or send, est=937	
Experimental Tick Time			Our Model Tick Ti	Our Model, range=5
0.202895457	0.202958907		0.2	0.2
0.202991215	0.20295833		0.2	0.2
0.202928492	0.202958953		0.2	0.2
0.203637366	0.2034604		0.200007535	0.20002
0.207198038	0.204641813		0.200007535	0.20002
0.207998902	0.205237626		0.200007535	0.200018
0.209716287	0.206502214		0.200007471	0.200017
0.21357822	0.207684516		0.200007343	0.200048
0.216437909	0.207754288		0.200007535	0.20468
0.230950699	0.213835269		0.200007535	0.253683
0.279837344	0.2249218		0.225600925	0.302536
0.280068246	0.225984111		0.230300747	0.308727
0.351085928	0.264373326		0.26508	0.353943
0.358318054	0.273591982		0.2726	0.363686
0.340877135	0.275862267		0.2773	0.369821
0.350944065	0.28770695		0.282	0.375928
0.362275562	0.302186918		0.3008	0.400548
0.413305964	0.328830402		0.329	0.437245
0.463862714	0.371277495		0.376	0.498435
0.557196153	0.423731289		0.423	0.559304
0.609537612	0.470010597		0.47	0.620513
0.697121162	0.518405955		0.517	0.68168
0.753036777	0.55976932		0.564	0.742986

Figure C.1: Experiment 1 and Experiment 2 Section 6.1

Entities		ESPDUst	FPDUst	DPDUst	EPDUs No	FPDUs No	DPDUS Nc	P(FPDU)	P(DPDU)
1		11294.75			492				
2		8554.605	1351534	8631	846	2	1	0.002356	0.001178
4		7798.05	1516927	5683.5	1510	3	2	0.00198	0.00132
40		2397.743	885207.3	2862.564	14356	66	39	0.004564	0.002697
80		2129.403	689183	2770.984	29083	139	63	0.004746	0.002151
100		2171.772	756856.4	3926.247	35601	160	81	0.004464	0.00226
120		1968.841	754731.6	2810.283	43223	214	106	0.004915	0.002434
140		1978.597	770524.8	3214.317	47372	238	120	0.004986	0.002514
160		1902.29	754228.3	2799.328	55246	257	128	0.00462	0.002301
200		2023.336	713996	3873.04	66143	386	177	0.005787	0.002653
240		1864.081	760286.1	2691.911	64979	415	203	0.006327	0.003095
245		2033.216	884101.2	2937.039	67643	406	230	0.005946	0.003369
282		1996.158	734132	2861.297	63698	571	279	0.008846	0.004322
290		1902.435	740559.9	2989.575	62597	590	280	0.009296	0.004412
295		1943.342	767727	2949.949	65439	558	276	0.00842	0.004165
300		1969.832	816294.8	2981.109	63533	561	293	0.008713	0.004551
320		1984.433	720433.9	3054.627	69656	618	292	0.008758	0.004138
350		1874.681	794989.5	3107.751	69456	604	313	0.008583	0.004448
400		1822.258	739356	2900.18	71238	655	311	0.009072	0.004307
450		2011.395	803716.7	2870.489	70431	886	421	0.01235	0.005869
500		2346.934	852951.7	3085.656	70762	806	433	0.011194	0.006014
550		2383.276	1097544	3049.291	62112	886	460	0.013962	0.007249
600		2177.353	848762	3331.838	69698	1119	562	0.015677	0.007873

Experimental Tick Time				Our Model, range=5	
0.20318838				0.2	
0.203166544				0.2	
0.203168746				0.2	
0.204231643				0.200021	
0.205177742				0.200022	
0.206510426				0.200021	
0.206602323				0.20002	
0.207455643				0.200019	
0.209390224				0.20697	
0.241477487				0.258371	
0.288023239				0.309775	
0.2949006				0.316204	
0.364818126				0.363737	
0.352773077				0.374024	
0.354524027				0.38046	
0.361150392				0.38689	
0.389014199				0.412595	
0.426423672				0.451144	
0.486952007				0.515369	
0.550965996				0.579625	
0.614942668				0.643871	
0.679117879				0.708132	
0.757850619				0.772415	

Figure C.2: Experiment 3 Section 6.1

Entities/node	ESPDUst	FPDUst	DPDUst	EPDUs No	EPDU/node	FPDUs No	FPDU/noc	DPDUS Nc	DPDU/noc	P(FPDU)	P(DPDU)
1	17491.9	1785349	10109	836	418	2	1	1	0.5	0.002384	0.001192
2	12267.99	49379.4	11999.5	1667	833.5	5	2.5	2	1	0.002987	0.001195
4	8739.969	239214	7443.4	3168	1584	9	4.5	5	2.5	0.002828	0.001571
40	2314.817	195235.2	3582.059	31498	15749	104	52	51	25.5	0.003286	0.001611
80	2187.838	668527.4	3371.162	62598	31299	221	110.5	105	52.5	0.003512	0.001669
100	2156.049	358640.5	3390.754	70450	35225	392	196	183	91.5	0.005519	0.002577
120	2082.464	519648.9	3785.192	89774	44887	364	182	182	91	0.00403	0.002015
140	2236.45	294793.3	3699.627	100602	50301	433	216.5	209	104.5	0.004277	0.002064
160	2029.338	474174.8	3413.3	107425	53712.5	526	263	247	123.5	0.004861	0.002283
200	2044.12	556429.7	3375.226	111950	55975	622	311	301	150.5	0.005511	0.002667
240	2130.917	354798	3318.176	102365	51182.5	954	477	460	230	0.009193	0.004432
245	2090.083	266650.9	3887.781	113868	56934	812	406	374	187	0.007058	0.003251
282	2221.902	314079.3	3415.294	106728	53364	1112	556	496	248	0.010264	0.004578
290	2280.172	340876.6	3212.406	98408	49204	1245	622.5	559	279.5	0.012424	0.005578
295	2186.249	458376.7	3306.547	109913	54956.5	1029	514.5	466	233	0.009236	0.004183
300	2226.271	434341.4	3363.805	106956	53478	1111	555.5	508	254	0.010233	0.004679
320	2241.862	243274.8	3532.45	120257	60128.5	1039	519.5	471	235.5	0.008533	0.003868
350	2285.337	312320.6	3329.81	110239	55119.5	1268	634	588	294	0.011312	0.005246
400	2393.785	318730.6	3522.508	109302	54651	1444	722	683	341.5	0.012959	0.006129
450	2154.294	222015.3	3190.794	119724	59862	1294	647	608	304	0.010639	0.004999
500	2484.772	350721.4	3748.306	106558	53279	1868	934	889	444.5	0.017088	0.008132
550	2403.008	259653.1	8053.093	118125	59062.5	1778	889	808	404	0.014729	0.006694
600	2583.964	345351.5	3883.302	109598	54799	2339	1169.5	1067	533.5	0.020698	0.009442

		Node 2	range=5				
ESPDUst	FPDUst	DPDUst	EPDUs No	FPDUs No	DPDUS Nc	P(FPDU)	P(DPDU)
17129.83	95430.5	20079	837	2	1	0.002381	0.00119
12436.83	1102305	5686	1657	5	2	0.003005	0.001202
9394.72	1036140	6021	3148	9	5	0.002846	0.001581
2182.944	768939.1	2919.922	31295	104	51	0.003307	0.001622
2010.135	221066.1	3222.733	62581	221	105	0.003513	0.001669
2055.37	512034.4	2779.814	69750	392	183	0.005574	0.002602
2045.708	333299.6	3415.038	89512	364	182	0.004042	0.002021
1919.303	682730	3508.775	99228	433	209	0.004336	0.002093
2122.294	329732.7	3177.154	107329	526	247	0.004866	0.002285
1900.43	328085	3423.93	111940	622	301	0.005511	0.002667
2059.698	493338.2	3143.709	101957	954	460	0.009229	0.00445
1919.416	582968.4	3165	113761	812	374	0.007064	0.003254
2076.788	498211.5	3146.063	107603	1112	496	0.010182	0.004542
2036.528	445155.1	3319.71	101199	1245	559	0.012087	0.005427
2034.154	341233.1	3471.811	113976	1029	466	0.008911	0.004036
2074.128	369817.4	3348.795	110565	1111	508	0.009903	0.004528
1939.832	594769.6	3242.333	119083	1039	471	0.008616	0.003906
2118.894	562780.4	3109.337	111013	1268	588	0.011234	0.00521
2081.415	560795.3	3052.29	109651	1444	683	0.012918	0.00611
2111.011	697807.3	3235.493	115017	1294	608	0.011067	0.0052
2160.013	530454	3246.479	110640	1868	889	0.016473	0.00784
2076.692	599084.6	3229.99	117823	1778	808	0.014766	0.00671
2115.105	481653.1	3603.881	109370	2339	1067	0.02074	0.009461

range=5			
Experimental Tick Time - n1	Experimental Tick Time - n2	Our Model, node 1,	Our model, node 2, range=5
0.203166819	0.203118194	0.2	0.2
0.203153843	0.203117815	0.2	0.2
0.203168061	0.203118148	0.2	0.2
0.203169731	0.203155074	0.200024409	0.200022
0.203352157	0.204019077	0.20002142	0.200026
0.203933021	0.205452485	0.200021776	0.200028
0.204311144	0.205498842	0.200022386	0.204408
0.205071917	0.225133098	0.200021751	0.23818
0.208333201	0.259245478	0.211544117	0.271897
0.252012652	0.329584496	0.264015168	0.339741
0.299375651	0.398679596	0.316653712	0.407436
0.311840254	0.4055493	0.323182213	0.415796
0.357461021	0.465166156	0.371660391	0.478241
0.367606546	0.479253788	0.382238492	0.491856
0.365819096	0.488988429	0.388755906	0.50028
0.373677625	0.494179396	0.39535422	0.508708
0.400340719	0.533097379	0.421561521	0.542472
0.433838348	0.594867733	0.460973319	0.593253
0.511701927	0.670734886	0.526576433	0.677631
0.581658571	0.752260167	0.592089727	0.762005
0.6443875	0.843290759	0.657816291	0.846635
0.741440681	0.943110877	0.72341883	0.931072
0.807408922	1.079418231	0.788977444	1.015407

Figure C.3: Experiment 4 Section 6.1

			Node 1	range=5							Node 2	range=5				
Entities/node	ESPDUst	FPDUst	DPDUst	EPDUs No	FPDUs No	DPDUS Nc	P(FPDU)	P(DPDU)	ESPDUst	FPDUst	DPDUst	EPDUs No	FPDUs No	DPDUS Nc	P(FPDU)	P(DPDU)
1	19774.32	688878	12105	1632	4	2	0.002442	0.001221	17870.26	64391.25	14483.5	1646	4	2	0.002421	0.001211
2	12507.88	218423.4	8098.833	3033	19	6	0.006213	0.001962	12843.43	333189.7	7955.667	3033	19	6	0.006213	0.001962
4	5443.155	221545.4	8533.692	5883	33	13	0.005566	0.002193	5588.675	276725.5	5298.769	5878	33	13	0.005571	0.002194
40	27440.88	205648.9	4039.745	43775	353	157	0.007971	0.003545	31191.41	210644.6	3707.439	43560	353	157	0.00801	0.003563
80	24971.61	208806.7	3318.044	48971	638	296	0.012784	0.005931	27994.53	240460.5	3493.223	48711	638	296	0.012851	0.005962
100	29175.73	227239.2	3333.313	48650	739	355	0.014856	0.007137	28161.85	261890	3433.282	48437	739	355	0.01492	0.007167
120	44113.6	178860.4	3352.406	38216	989	461	0.024933	0.011622	45064.03	229094.1	3419.809	37810	989	461	0.025191	0.011742
140	64217.95	160716.1	3320.503	28265	1280	537	0.04255	0.017851	70242.69	205857.1	3971.369	27846	1280	537	0.043151	0.018103
160	74112.03	206892.3	3347.556	28533	1129	550	0.037369	0.018205	81093.83	303026.9	3343.52	28085	1129	550	0.037932	0.018479
200	97476.32	157982.8	3770.629	29871	1697	644	0.052682	0.019993	98597.75	239530.9	3400.539	29321	1697	644	0.053597	0.02034
240	182713.9	181349.5	3369.368	23748	2120	914	0.079158	0.034127	274414.5	219516.4	3361.764	23370	2120	914	0.080291	0.034616
245	190491.4	206652.5	3852.707	22210	1960	798	0.0785	0.031961	277793.3	237597.4	3665.959	21832	1960	798	0.079707	0.032452
282	202981.4	169155.6	3402.035	24319	2537	1011	0.09104	0.036279	330138.6	225635.6	3309.283	23977	2537	1011	0.092171	0.03673
290	163273.6	206048.2	3207.868	23333	2311	1027	0.086648	0.038506	182108.1	248629.4	3461.629	22806	2311	1027	0.088395	0.039282
295	160224.2	91870.8	4100.497	24903	4410	936	0.14579	0.030943	174922.8	123926.4	3687.966	24305	4557	936	0.15293	0.031412
300	145246.3	81025.5	4507.876	28625	4397	981	0.129312	0.02885	158584.2	133741.4	4009.424	28102	4397	981	0.131332	0.029301
320	224601.2	185179.3	3661.18	25879	2601	1077	0.087999	0.036438	316293.4	214541.9	4195.928	25753	2601	1077	0.088376	0.036594
350	240218.6	156670.8	4438.14	25910	2949	1150	0.098271	0.038322	403652.5	247727.1	3871.959	25595	2949	1150	0.099313	0.038728
400	352944.8	189681.6	3706.73	19968	3199	1393	0.130252	0.056718	433107.1	251920.5	3849.293	19621	3199	1393	0.132119	0.057531
450	335389.3	50537.07	4655.004	24017	6954	1075	0.217001	0.033546	336114.1	131255.5	3835.604	23016	6797	1066	0.220117	0.034522
500	606501.7	211687.3	3897.368	18085	3741	1519	0.160248	0.065067	521029.7	187419.6	4740.573	18168	3942	1544	0.166653	0.065274
550	657024	170220.8	4692.212	20683	4562	1342	0.171588	0.050476	583196.6	246477.5	4494.837	20175	4653	1378	0.177555	0.052583
600	646335.8	65578.06	4617.98	24221	7697	1052	0.233455	0.031908	470449.2	151112	4130.315	23586	7571	1047	0.235095	0.032511
			Node 3	Range=5								Node 4	Range=5			
Entities/node	ESPDUst	FPDUst	DPDUst	EPDUs No	FPDUs No	DPDUS Nc	P(FPDU)	P(DPDU)	ESPDUst	FPDUst	DPDUst	EPDUs No	FPDUs No	DPDUS Nc	P(FPDU)	P(DPDU)
1	32358.1	1276017	32511.5	1639	4	2	0.002432	0.001216	33228.19	46557.75	30268.5	1600	4	2	0.002491	0.001245
2	25473.35	558726.8	17454.67	2989	19	6	0.006304	0.001991	23784.19	63051.21	20929.83	2946	19	6	0.006395	0.00202
4	10733.96	469371.8	11918.46	5773	33	13	0.005671	0.002234	11252.8	526705.5	14838.62	5620	33	13	0.005824	0.002294
40	13540.91	569760.2	11144.4	43196	353	157	0.008077	0.003592	13679.04	614052.2	10227.57	42710	353	157	0.008168	0.003633
80	11932.02	635824.3	11009.06	48406	638	296	0.012931	0.005999	11721.41	507902.3	11537.13	48257	638	296	0.01297	0.006017
100	13400.38	396383.8	10895.14	48322	739	355	0.014955	0.007184	12850.74	798081.7	11026.35	47882	739	355	0.015089	0.007248
120	23551.99	610286.1	10221.94	37105	989	461	0.025652	0.011957	15062.73	605183.6	10340.58	36847	989	461	0.025824	0.012037
140	18642.99	521510	10182.37	27082	1280	537	0.044292	0.018582	19194.79	665077.4	11211.15	26580	1280	537	0.045075	0.01891
160	17540.69	692845.6	9751.878	27817	1129	550	0.038276	0.018647	19321.13	368688.4	10635.31	28029	1129	550	0.038003	0.018514
200	16107.72	231655.4	9819.638	29671	1697	644	0.053011	0.020117	18188.69	738624.8	11180.89	29083	1697	644	0.054003	0.020494
240	24717.04	583780.8	10728.52	23515	2120	914	0.079852	0.034427	21801.57	538530	10019.55	23644	2120	914	0.079466	0.03426
245	21722.92	644921.6	10336.12	21954	1945	790	0.07878	0.031998	23105.46	202004.1	11813.98	22434	1960	798	0.077802	0.031677
282	23434.15	333105	9929.632	24412	2537	1011	0.090737	0.036159	22390.22	693725.9	11286.27	23941	2536	1011	0.092258	0.03678
290	25844.88	335507.6	10544.41	22971	2309	1027	0.087771	0.039039	24096.06	757118.5	11713.69	22432	2311	1027	0.089678	0.039853
295	24960.16	96768.26	10900.31	23896	5141	912	0.171658	0.030452	22835.21	365993.4	11505.06	23295	5138	910	0.175101	0.031013
300	19224.69	172478.9	11198.7	28195	4395	981	0.130917	0.029222	21239.17	465930	13377.38	27572	4394	981	0.133366	0.029775
320	21398.6	626394.1	11461.15	25832	2601	1077	0.08814	0.036496	21685.37	413629.2	12048.08	26164	2601	1076	0.087162	0.036058
350	19772.4	271754.4	10260.53	26276	2949	1150	0.097086	0.03786	20564.04	776809.7	11405.72	25832	2949	1150	0.098527	0.038422
400	21065.58	306729	9430	20132	3199	1392	0.129394	0.056304	28876.43	769276.4	12000.7	19638	3196	1391	0.13193	0.05742
450	26652.54	101420	11167	23573	7737	1120	0.238575	0.034536	24418.14	288139.5	11367.94	23423	7803	1126	0.241191	0.034805
500	45427.42	674886	10200.64	17375	3936	1543	0.172224	0.067516	30454.24	679980.4	10549.22	17803	3942	1544	0.169264	0.066297
550	36813.6	292836.6	10064.69	20644	5827	1801	0.206105	0.063703	41657.16	1140561	10132.77	20171	5873	1792	0.210986	0.064377
600	31051.69	139148	12835.48	23431	8858	1180	0.264663	0.035257	28011.75	205726.7	20070.59	23362	9190	1182	0.272425	0.035039

range=5

Experimental Tick Time - n1	Experimental Tick Time - n2	Our Model, node 1,	Our model, node 2, range=5
0.203473641	0.203375056	0.171428714	0.171429
0.203168361	0.20303585	0.213043522	0.213044
0.203168565	0.203120585	0.228947421	0.228947
0.272072591	0.290955707	0.523662232	0.65566
0.321332117	0.356088492	1.055659053	1.323041
0.395770335	0.496679239	1.324172373	1.661962
0.64630174	0.731786084	1.595746242	2.006624
1.327614381	1.481638092	1.869933566	2.354591
1.661876595	1.844785437	2.147497087	2.709795
2.502677891	2.862191835	2.712931278	3.427602
3.738544246	3.835576627	3.287794167	4.166076
3.737723987	3.899864101	3.363311724	4.266835
4.626363802	5.227798436	3.91058888	4.970165
4.41358412	5.37956419	4.032568083	5.133571
4.539579401	5.817474666	4.101950167	5.221937
4.806825927	5.938772645	4.185220043	5.338229
6.366573024	6.429563051	4.479857364	5.726902
7.146241801	7.861239299	4.93974995	6.350606
8.951823163	8.927926735	5.734453118	7.387587
11.59822908	11.54825907	6.5405148	8.510747
11.21268709	15.40580056	7.397534846	9.600695
15.01311019	17.97862806	8.226648	10.75509
22.36938448	19.71938535	9.090063273	11.99835

Experimental Tick Time - n3	Experimental Tick Time - n4	Our Model Node 3	Our Model Node 4
0.203346651	0.203412873	0.171428714	0.171429
0.203164919	0.203176718	0.213043522	0.213044
0.203212065	0.203867213	0.228947421	0.228947
0.256272885	0.239875157	0.99962993	1.011018
0.981597662	0.989269182	2.037685571	2.061587
1.521184134	1.442176615	2.5753105	2.603391
1.905562524	1.826027582	3.118744813	3.158576
2.450467817	2.331471575	3.677555259	3.727212
3.134620761	2.800408524	4.252714739	4.298901
4.280060441	4.014136248	5.433332778	5.492296
6.269588632	5.306030854	6.650015667	6.778266
6.677981676	5.302885123	6.845301857	6.919563
7.962437659	7.064449115	8.0467535	8.133992
8.1616313	7.321224401	8.270997667	8.468395
8.296684554	7.404841278	8.522142455	8.614509
8.404961835	7.726518507	8.666696545	8.760623
10.30983449	8.680562963	9.3863461	9.488064
11.24367794	10.28674367	10.45613578	10.56942
15.33419363	12.14000488	12.22164738	12.35404
19.68277439	14.94053067	14.141861	14.82238
22.44034578	23.60004695	16.294427	16.47084
28.21503453	24.95533235	18.8188442	19.02257
33.63742695	25.5184243	21.993795	22.23187

Figure C.4: Experiment 6 Section 6.1

VR Forces Experiment

Number of Entities	Inter-arrival Time(Mean)(sec)
1	0.231460727
2	0.227966192
4	0.22359131
40	0.226081765
80	0.228417303
160	0.222239881
237	0.222293764
245	0.264071962
280	0.286483996
290	0.309288566
295	0.34955857
300	0.436843619
310	0.723769117
320	0.967849571

Used Configuration
tickTime = 0.2s (200ms)
netTickTime = 0.21s (210ms)
Collection Duration = 1 min

DIS Model

No. Of Entities	CPU Inter-arrival Time(mean)(s
1	0.21
2	0.21
4	0.21
40	0.21
80	0.21
160	0.21
237	0.2281
245	0.239958
280	0.271964
290	0.278224
295	0.282142
300	0.28605
310	0.293889
320	0.346406
340	0.368052

Figure C.5: Experiment 7 Section 6.1

APPENDIX D

ACRONYMS AND ABBREVIATIONS

Table D.1: Acronyms and abbreviations - I

ABBREVIATION	MEANING
CPU	Central Processing Unit
DARPA	Defense Advanced Research Projects Agency
DES	Discrete Event Simulation
DIS	Distributed Interactive Simulation
EPE	Event processing Engine
FEL	Future Event List
FELM	Future event list manager
HLA	High Level Architecture
I/ITSEC	Interservice/Industry Training Systems and Education Conference
LAN	Local Area network
Mbps	Mega Bits Per Second
ModSAF	Modular Semi-Automated Forces
NIC	Network Interface Card
PDU	Protocol Data Unit

Table D.2: Acronyms and abbreviations - II

ABBREVIATION	MEANING
SISO	Simulation Interoperability Standards Organization
VM	Virtual Machine
WAN	Wide Area network
GUI	Graphical user interface
ACR	Armored Cavalry Regiment

REFERENCES

[1] Ieee standard for distributed interactive simulation - application protocols. *IEEE Std 1278.1-1995*, page i, 1996.

[2] VT MAK. *VR-Forces developer guide*. VT MAK, 2009.

[3] John Sokolowski. *Modeling and Simulation Fundamentals*. Wiley, New York, 2010.

[4] M.D Petty. Behavior generation in semi-automated forces. In D. Schmorrow D. Nicholson and J. Cohn, editors, *The Psi Handbook of Virtual Environments for Training and Education*, pages 189–204. Praeger Security International, Westport, 2009.

[5] M.D. Petty. Current and historical dod m&s standards. Unpublished document, 2010.

[6] R.C. Hofer and M.L. Loper. Dis today [distributed interactive simulation]. *Proceedings of the IEEE*, 83(8):1124 –1137, August 1995.

[7] Ieee standard for distributed interactive simulation - application protocols. *IEEE Std 1278.1a-1998*, page i, 1998.

[8] C. S. Bradley, Sal Barbosa, Greg Reed, and Shreyas Purohit. A baseball simulation using distributed interactive simulation. Huntsville Simulation Conference, Huntsville, AL, 2010.

[9] D. Fullford. Distributed interactive simulation: it's past, present, and future. In *Simulation Conference, 1996. Proceedings. Winter*, pages 179 –185, 1996.

[10] DIS Steering Committee. The dis vision, a map to the future of distributed simulation, version 1, institute for simulation and training. Orlando, Florida, USA, IST-SP-94-01, May 1994.

[11] Ieee standard for distributed interactive simulation - communication services and profiles. *IEEE Std 1278.2-1995*, page i, 1996.

[12] S.E. Cheung and M.L. Loper. Traffic characterization of manned-simulators and computer generated forces in dis exercises. In *AI, Simulation, and Planning in High Autonomy Systems, 1994. 'Distributed Interactive Simulation Environments'. Proceedings of the Fifth Annual Conference on*, pages 70 –76, December 1994.

[13] SISO. Enumeration and bit encoded values for use with dis. `http://www.sisostds.org/DigitalLibrary.aspx?Command=Core_Download&EntryId=30794`, 2006. Online; Accessed Aug 2010.

[14] Jerry Banks. *Discrete-Event System Simulation*. Pearson Prentice Hall, Upper Saddle River, 2005.

[15] P. Ryan and J. Morton. Network traffic prediction for distributed interactive simulation exercises. In *In: Proceedings of the Second International SimTecT Conference*, 1997.

[16] Peter Ryan, Lucien Zalcman, and John Fulton. Scalability of dis traffic using modsaf. In *In: Proceedings of the Fourth International SimTecT Conference*, pages 139–144, 1999.

[17] J.M. Pullen and D.C. Wood. Networking technology and dis. *Proceedings of the IEEE*, 83(8):1156 –1167, August 1995.

[18] G.D. Nguyen and S.G. Batsell. Statistical characteristics of dis traffic. In *Military Communications Conference, 1996. MILCOM '96, Conference Proceedings, IEEE*, volume 2, pages 637 –641 vol.2, October 1996.

[19] D.B. Cavitt, C.M. Overstreet, and K.J. Maly. A performance monitoring application for distributed interactive simulations (dis). In *Simulation Conference, 1997., Proceedings of the 1997 Winter*, pages 421 –428, December 1997.

[20] T.L. Gehl. Mapping distributed interactive simulation network requirements onto broadband networks and services. In *Communications, 1994. ICC '94, SUPERCOMM/ICC '94, Conference Record, 'Serving Humanity Through Communications.' IEEE International Conference on*, pages 154 –159 vol.1, May 1994.

[21] J. M. Kloeber and J. A. Jackson. Issues in using a dis facility in analysis. In *Simulation Conference Proceedings, 1995. Winter*, pages 1229 –1236, December 1995.

[22] T. Billhartz, J.B. Cain, E. Farrey-Goudreau, D. Fieg, and S.G. Batsell. Performance and resource cost comparisons for the cbt and pim multicast routing protocols. *Selected Areas in Communications, IEEE Journal on*, 15(3):304 –315, April 1997.

[23] Juan J. Vargas, Ronald F. DeMara, Michael Georgiopoulos, Avelino J. Gonzalez, and Henry Marshall. PDU Bundling and Replication for Reduction of Distributed Simulation Communication Traffic. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 1(3):171–183, 2004.

[24] Lucien Zalcman and Peter Ryan. Wide area network latencies for a dis/hla exercise. 2009.

[25] R. W. Franceschini, M.D Petty, S. A. Schricker, D. J. Franceschini, and McCulley G. Measuring and improving cgf performance. *Proceedings of the Eighth Conference on Computer Generated Forces and Behavioral Representation*, pages 9–15, 1999.

[26] Matthew Menarchek. The battle of 73 easting. Unpublished document, University of Alabama in Huntsville, 2010.

[27] Fabio Prado. M1a1/2 abrams. `http://www.fprado.com/armorsite/abrams.htm`, 2010. Online; Accessed Aug 2010.

[28] Dough Chaltry. M1 abrams comparison. `http://www.onthewaymodels.com/articles/abramscomp.htm`, 2010. Online; Accessed Aug 2010.

[29] Wikipedia. Bmp-2. `http://en.wikipedia.org/wiki/BMP-2`, 2010. Online; Accessed Aug 2010].

[30] Bmp-1 and bmp-2 lower hulls. `http://modelshipwrights.kitmaker.net/modules.php?op=modload&name=SquawkBox&file=index&req=viewtopic&topic_id=153738&page=1`, 2010. Online; Accessed Aug 2010.