NEPTSim: Simulating NEPTUNE Canada using OMNeT++

by

Burak Martonaltı
B.Sc., Near East University, 2009

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

NEPTSim: Simulating NEPTUNE Canada using OMNeT++

by

Burak Martonaltı
B.Sc., Near East University, 2009

Supervisory Committee

Dr. Yvonne Coady, Co-Supervisor
(Department of Computer Science)

Dr. Sudhakar Ganti, Co-Supervisor
(Department of Computer Science)

Supervisory Committee

Dr. Yvonne Coady, Co-Supervisor
(Department of Computer Science)

Dr. Sudhakar Ganti, Co-Supervisor
(Department of Computer Science)

ABSTRACT

North-East Pacific Undersea Network Experiments (NEPTUNE) is a multi-node cabled ocean observatory linked by 818 kilometers of powered fiber optic cable offshore from Vancouver Island across the northern Juan de Fuca tectonic plate. It includes a Data Management and Archive Station (DMAS) at the University of Victoria (UVic) and a shore station at Port Alberni, BC, Canada. The core of the network consists of 6 branching units, 6 node stations, 13 junction boxes and more than 130 instruments.

In this paper, we explore the costs and benefits of constructing a simulator for NEPTUNE using the OMNeT++ simulation platform—a C++ based discrete-event simulator. In this context, we present the design and implementation of a simple simulator that can work with a variety of configurations of instruments, where the instruments are connected to DMAS via junction boxes and branching units, and generate TCP and UDP traffic following certain patterns. The simulator is designed for supporting *what-if* scenario analysis, particularly with respect to system evaluation and discovery of limits associated with network traffic behaviors. Our study reveals that, although building the simulator in OMNeT++ has many advantages such as ease of tuning and calibration, capturing sufficient details regarding the working behavior of the actual NEPTUNE environment is still challenging. A survey of alternative tools, including NS-2/NS-3, OPNET, JiST/SWANS, J-Sim, SSFNet, and Qualnet reveals that these nuances would not be any less challenging within these simulation environments.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

My Dear Supervisors Dr. Yvonne Coady and Dr. Sudhakar Ganti for their endless mentoring, support, encouragement, and patience.

Dr. Yağız Onat Yazır for being everything: mentor, supervisor, teacher and *brother* no matter what kind of situation I am in.

Dr. Nadire Çavuş for supporting me during my undergraduate years, always mentoring me and lead me towards to success.

My Colleagues in (Mod)ularity Squad Labaratory for supporting me in the low moments.

In the NEPTSim Project Thanks to Martin Hofmann (NEPTUNE Canada Systems Manager) and Benoit Pirenne (NEPTUNE Canada Associate Director IT) for their assistance and advice on this project. .

DEDICATION

I dedicate this thesis to the best mother in the universe *Şengül Martonaltı*, my brother *Berkan Martonaltı* and the best father in the world *Hamdi Martonaltı*—may he rest in light.

Also I really appreciate my *girl*—the one—*Gözde Zenginli* who endlessly supports me by agreeing upon my all decisions that I have made and her unlimited patience, my close friends who share my all happiness and misery on the way of one of unforgettable events which makes them unforgettable in my life.

Addionally, I very appreciate my brothers *Barış Yavuz* and *Kerem Şeşen* as well as my Cypriot brothers *Kerem Yolak* and *Ulaş Şeherlioğlu* to support me and not to let me down in this formidable process.

My only purpose from the life is being a nice person.

&

My only expectation from the life is to recompense my all sacrifices that I have made to be a successful 'person', 'husband', and 'father'.

&

If I can achieve to be deemed worthy to take all these dignities then that means I am a SUCCESSFUL!!!

Beni bu *uzun*; *sarsıntılı* ama 'hızlı', *virajlı* fakat 'tutarlı 'trene benzettiğim hayatımda benimle birlikte yolculuk ettiğiniz için sizlere minnettarım...

Iyiki varsınız...

Evet baba duyuyorum seni...

En içten sesini...

Merak etme, ayaktayım, senin bıraktığın gibi,

Herzaman dimdik, basım önde,

Bir direk gibi destekliyorum onları,

Herzaman, ne pahasına olursa olsun,

CANIM AİLEME...

M A R T I

Chapter 1

Introduction

Since 2009, North-East Pacific Undersea Network Experiments (NEPTUNE) has been providing the scientists from various disciplines with a platform where comprehensive sets of data can be collected and experimented with. NEPTUNE has been used within the context of a number of research topics such as underwater volcanic processes, earthquakes and tsunamis, minerals, metals, hydrocarbons, ocean-atmosphere interactions, climate change, green house gas cycling in the ocean, marine ecosystems, long-term changes in ocean productivity, marine mammals, fish stocks, pollution and toxic blooms [23].

The underwater observatories accommodate high bandwidth connections to guide sensor networks remotely. The sensors in such networks usually provide data and imagery that can be used by multidisciplinary crews in order to observe and examine short and long-term events and progresses by querying a digital database [29]. Similarly, NEPTUNE is also a system that contains various sensory components such as bioflourings, CTDs, short period seismometers, pressure gauges, hydrophones and video cameras that are linked with each other either directly or via a sequence of routers [15]. The core of the system consists of more than 130 instruments that collect measurements and transmit them to a Data Management and Archive Station (DMAS) at the University of Victoria (UVIC) over TCP and UDP communication protocols. In addition there are 11 junction boxes, 6 node stations, 6 branching units, 1 shore station in Port Alberni, BC, Canada [29].

These components are connected on a backbone of thick and insulated cable called a *spur cable* that has two separated lines for transmitting the data in opposite directions. The instruments transmit the data either clockwise or counter clockwise within a 2 by 2 Gbps data flow. Since the primary purpose of the system is data collection

and analysis, the available instruments generate data of various size and nature. The collected data is then transmitted on the spur cable passing through all the routers that are on one of the best alternative path of the data flow. In a number of cases the data may not be received by DMAS properly due to a wide range of potential problems—such as packet drops, unit malfunctions, etc.—that may be encountered at the relay units. Furthermore, these problems may emerge in a way that may appear to be completely unpredictable, which in turn makes analysis and evaluation a crucial part of planning maintenance and future deployments.

However, evaluating systems like NEPTUNE in terms of features and capabilities is extremely challenging. This is mainly due to the fact that observing the consequences of modification after maintenance and deployment is often the only method in order to evaluate the system's behavior. In addition, almost all maintenance and deployment operations impose substantial expenses and time due to the natural challenges of accessing and tuning an underwater network with components set up on the ocean floor at depths varying between 100 and 2700 meters [15]. Under these circumstances, building a simulator for pre-deployment and pre-maintenance analysis and evaluation appears to be a very appealing and cost-effective alternative to frequent physical tuning.

In this thesis, we address this need by introducing a simple NEPTUNE simulator implemented using the OMNeT++ simulation platform—a C++ based discrete-event simulator. The proposed simulator can work with a variety of configurations of instruments, junction boxes, node stations, branching units and traffic patterns. The main purpose of the simulator is to support *what-if* scenario analysis, particularly with respect to system evaluation and discovery of limits associated with network traffic behaviors. Our study reveals that, although building the simulator in OMNeT++ has many advantages—such as ease of tuning and calibration, capturing sufficient details regarding the working behavior of the actual NEPTUNE environment is still challenging. A survey of alternative simulation platforms reveals that these nuances would not be any less challenging within these simulation environments.

The rest of this thesis is organized as follows that chapter 2 outlines the reasons of why simulation environment is used for NEPTUNE Canada network system by providing detailed information about recent simulation environments as well as their comparison among them. Chapter 3 explains the implementation process of NEPTUNE Canada network including detailed explanations of all components of NEPTUNE Canada. Chapter 4 gives detail how NEPTUNE Canada network system

was carried to a simulated arena with its all calibrations and characteristics of each devices. Chapter 5 discusses the capabilities of the network system and how well NEPTSim reflects the calibrations of the real world system. Chapter 6 gives us a conclusion of the research and last chapter outlines that there are many things that we have not captured, but despite these missing parts, NEPTSim has value at the level of exploring high-level relationships associated with corresponding changes in the configuration of the system.

Chapter 2

Background and Problem Description

In this chapter, first we introduce the concept of simulation platforms with respect to motivation behind their emergence. In addition, some perspective about systems will be provided in terms of advantages of simulating them instead of using analytical processes, emulations or devising testbeds. Furthermore, the chosen simulation platform, OMNeT++, will be explained in detail. In addition, a set of other simulation platforms are explained briefly, and compared to OMNeT++ in terms of their ease of use, library support, etc. Section 2.1 introduces the features of the discrete event system simulation environments by providing the definitions of the terms that are related to the research. In this section, emulation and simulation models are discussed based on the practicability of NEPTUNE Canada network system in terms of the pros and cons of using both models. Afterwards, the following sections are related to the explanations of the most common simulators starting with the chosen simulation tool OMNeT++. Through the end of the chapter, OMNeT++ and the other simulation tools are compared to each other in terms of being chosen the most appropriate simulation tool to simulate NEPTUNE Canada network system.

2.1 Discrete Event System Simulation Environments

A *system* is a group of objects that are joined together in some regular interaction or interdependence towards the accomplishment of some purpose [38]. A system is affected by changes that occur outside. Such changes are said to occur in the *system*

environment. The *boundary* between a system and its environment depends on the purpose of the study. Therefore, by testing the boundaries—in other words *limits*—of the systems, we investigate the real-time systems to predict the consequences of future changes.

An *event* is an instantaneous occurrence that might change the state of the system. So in NEPTUNE Canada, each instrument produces an event during data generation. *Simulation* is the imitation of a real-world process or system over time which is used for the analysis and study of complex systems [4]. *Discrete system* is one for which the state variables change instantaneously at separated points in time.

Simulation requires the development of a simulation model and then conducting computer-based experiments with the model to describe, explain, and predict the behavior of the real system. A *model* is a simplification of the system that is represented for the purpose of studying that system. Different models of the same system may be required as the purpose of investigation change. Therefore, the model should be sufficiently detailed to permit valid conclusions to be drawn about the real systems.

2.1.1 Advantages and Disadvantages of Using a Simulation

Simulation is the imitation of a real world process or system over time that is used for the analysis and study of complex systems [4]. Simulations can be useful when the circumstances could meet the scientists needs. Thus, first of all, all the related conditions must be considered while deciding whether developing a simulation is appropriate. Simulation enables the study of, and experimentation with, the internal interactions of a complex system. The effects of changes in state variables or parameters on the model's behavior can be observed to be analyzed in conclusion. The knowledge gained during the design of a simulation model can be used to improve the system under investigation. Changing simulation inputs and observing the resulting outputs can produce valuable insights about which variables are most important and how variables interact. Simulations can be used to experiment with new designs or policies before implementation so as to prepare for what might happen; thus, simulations can be used to verify analytic solutions.

There are a number of advantages of developing simulations for real-time systems. For instance, simulations make sure that the effects of variations in system parameters can be explored without disturbing the real system. Therefore, new system designs can be tested without committing resources for their acquisition. Hypotheses about

how or why certain phenomena occur can be tested for feasibility. Furthermore, time can be compressed or expanded to allow for speed up or slow down of the phenomenon under investigation. Various levels of insight can be obtained about the interaction of variables and the importance of variables to the system performance. Bottle neck analysis—such as throughput—can be performed to discover where work process is being delayed excessively; it allows us to develop ideas in order to prevent problems.

However, sometimes developing a simulation could be time consuming. Particularly in the cases where a problem can be solved analytically or by common sense. Furthermore, it could be less expensive to perform direct experiments. Cost of modeling and simulation may exceed savings. Also, the system could be very complex to be defined, there could be lack of necessary data, or sufficient resources and time may not available to simulate a system. And if resources or time are not available to simulate a system. Under these circumstances, developing a simulation is inappropriate.

In addition, simulating a system may have disadvantages depending on the circumstances. For example, modeling and analysis of a simulation can be time consuming and expensive because model building requires special training. Additionally, simulation results can be difficult to interpret owing to the fact that most simulation outputs are random variables—based on random inputs. Random variable is a series of numbers that are changed based on specific algorithm in order to enrich the variety of numbers that are used in the statistical analysis. Thus, it can be hard to distinguish whether an observation is the result of system interactions or of randomness.

Then a major problem that needs to be solved during building simulations is to find proper ways of compensating for the disadvantages that may emerge from using a simulated setting. Many simulation packages have output analysis capabilities such as OMNeT++, NS-2, NS-3 or OPNET. Simulation has become faster due to advances in hardware. In the next two sections, popular simulation platforms and frameworks. We will also explain how their embedded packages make implementing simulations for real world systems easier.

Even though a simulation model and an emulation model are implemented for the similar purposes, there are notable differences in the utilization and the performance. Simulation models are used to test and develop different solutions to be analyzed in order to obtain stronger predictions to reach at the best alternative solutions. During the simulation run, parameters and components must be defined beforehand; thus, simulation does not aim to test hardware metrics of the system that is desired to be simulated [19].

Emulations may not be designed to test the operation of the control system, as they often run only in real time while testing such real-time systems like NEPTUNE owing to their sizes and expensiveness. Therefore, it is advisable to use emulation models for experimental use due to their unfeasible nature. Simulation model give the user interdependency to think of several scenarios by providing flexible environment and cost-efficient operations. Thus, the user can explore various ideas and have several experiments without any concern for expenses and limits. The purpose of the emulation is to reflect the system that will be implemented in a more precise manner. Therefore, an emulation typically has faster run times than a simulation. By doing so, the model carries out a limited series of verification processes to guarantee the reaction and the performance of the control systems.

Simulation models provide an unlimited and commutative time span during the operations. This allows a simulation model to be built and implemented much faster than an emulation model because its speed of execution is much faster than emulation's. Thus, simulation models allow time savings which is a precious feature for the researchers. On the other hand, because simulations are not executed in real time, the results may have errors; thus, the response of emulation models may be more reliable than these of simulation [19].

Two or more model runs will always execute in exactly the same way and produce precisely the same results if no parameters are changed between runs. Any impression of randomness in a simulation model is due to the use of pseudo—random numbers to generate certain events such as break—downs, cycle times and so on. Repeatability is necessary in order to recreate and understand events during the model run, as well as to debug the model as it is built. All events that influence the model execution are contained within the model and are therefore repeatable [19].

We cannot claim that it is impossible to emulate NEPTUNE but it will be more expensive, much more time consuming and limited than using a simulation model due to the comparisons that were discussed above. Therefore, simulating NEPTUNE Canada rather than emulating it may be one of the most suitable research in order to investigate NEPTUNE Canada network behavior.

2.2 OMNeT++ and Other Simulation Platforms

Simulation platforms are designed to support simulation of various computational environments such as network systems, distributed systems, parallel computing, etc.

Since, a number of real-world scenarios involve communication between components, most of the available simulation platforms typically provide built-in support for a majority of today's conventional network protocols such as WLAN, Wi-Max, UDP and TCP.

The modern simulation platforms are both fast and inexpensive compared to maintaining test beds of actual hardware components. Most of the network simulation tools are built around the idea of representing time as a set of discrete events. These events must be discrete in order to be captured and played back and forth when they are needed. In this section, we provide an introductory explanation of the OMNeT++ discrete event simulation platform. Additionally, other simulation platforms will also be outlined as alternatives to OMNeT++.

2.2.1 OMNeT++ [25]

OMNeT++ was designed to support simulation of network systems, distributed and parallel systems. It primarily relies on building and presenting hierarchical structures by utilizing reusable components. As with its contemporaries, OMNeT++ is a discrete-event simulator. The platform is mainly implemented using the C++ programming language. Since OMNeT++ uses GCC tool chain or the Microsoft Visual C++ compiler, it can be executed in three most common platforms: Windows, OS X, Linux.

OMNeT++ can be operated freely for non-profit use, and provides a powerful and consistent open-source library that can easily be used by research-oriented institutions. It is primarily used in research fields such as communication networks, multiprocessors, distributed systems and parallel computing. Different model structures can be implemented in OMNeT++ even though these models are built outside of OMNeT++ platform. Furthermore, specific models and frameworks such as INET [8], MANET [32] and Mobility Frameworks [9] can be used to extend the functionality provided by OMNeT++ platform.

OMNeT++ was designed to simplify the simulation of network systems. It provides visualization and customization with its integrated development environment (IDE). OMNeT++ IDE contains multiple source code editors related to different activities. Its IDE gathers multiple structures in a single compound and compiles them into a single unit. Furthermore, OMNeT++ IDE provides tool sets to modify network components without coding.

OMNeT++ model contains different modules that can communicate with each other. These modules can be assembled to create complex the networks of computational units [43]. The functional modules are called *simple modules* that can be bundled into *compound modules*. In this context, a network component can either be represented as a simple module or as a compound module that contains multiple simple modules. Having no distinction among module types allows the user to transparently split a functionality among several simple modules within a compound module [39].

The model behind OMNeT++ itself is a combination of simple modules and compound modules that are active in message passing in between various linked modules. In terms of message transmission, each component can be configured independently based on the desired message routes. Thus, there is no limited hierarchy level in between data transmitters. Data could be circulated several times as long as it is delivered to the receiver.

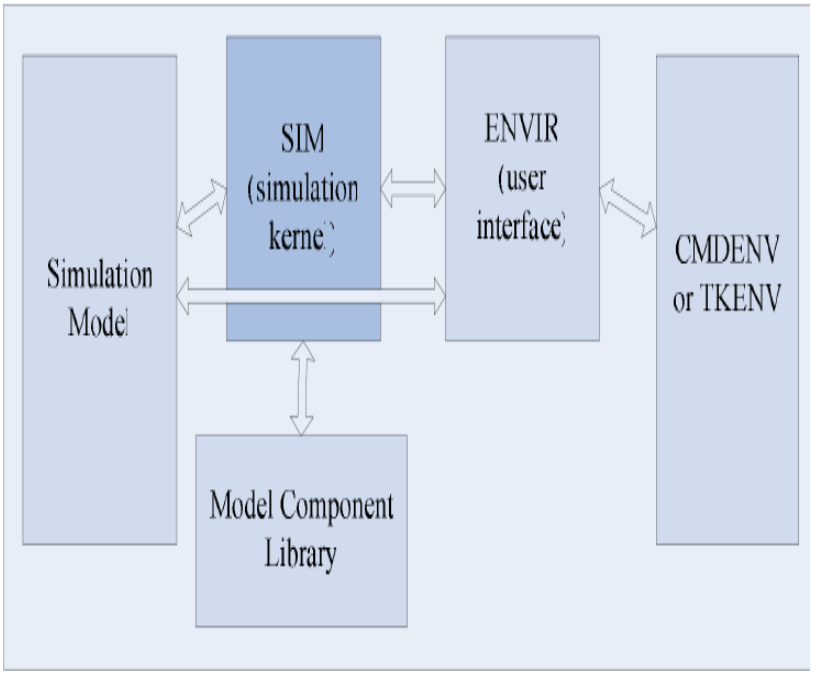

Figure 2.1: Logical Architecture of OMNeT++ [43]

The Model Component Library consists of the code of compiled simple and compound modules. Figure 2.1 illustrates the logical architecture of OMNeT++. As the application of OMNeT++ became more widespread, more reusable models, network protocols and communication models were added into the OMNeT++ model

library. Users can customize the full environment in which the simulation runs, and even embed an OMNeT++ simulation into a larger application which is shown in Figure 2.2.

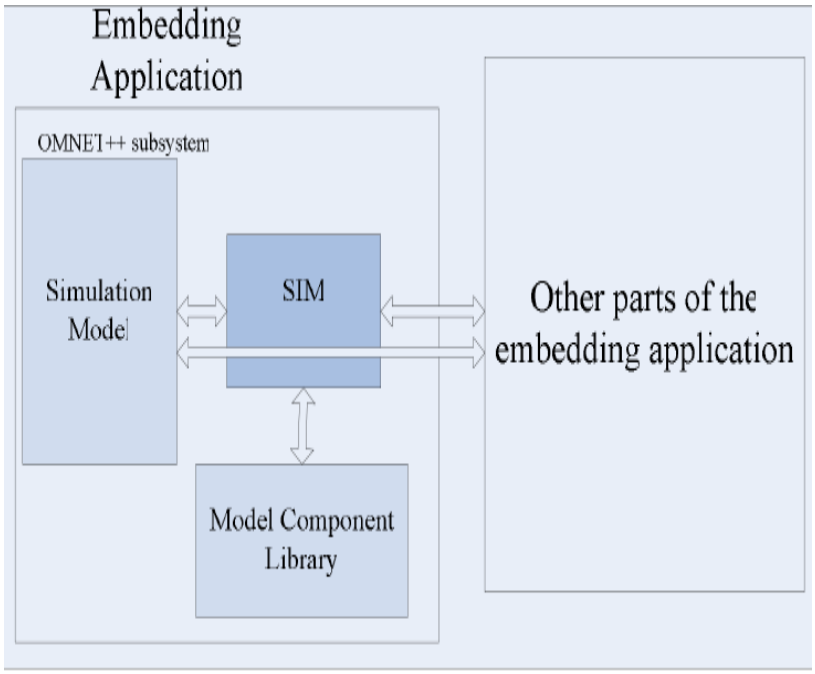

Figure 2.2: Embedding OMNeT++ [43]

Modules communicate with *messages* that may contain arbitrary data. Message(s) are generally sent by simple modules via *gates*. Messages are sent in and out through gates that can be linked to other modules with *connections*. Connections are built in between two modules' gates within a single level of module hierarchy. It is possible to connect a gate of a simple module and a gate of a compound module. Furthermore, connections can be assigned characteristics such as delay or data rate. *Channel* is a term that have pre-assigned connections in order to be reused in several places in a single network [39].

Modules can have parameters which can be used to further define the topology of the system that is being simulated. These parameters may be of different data types, valued expressions or can reference other parameters. Using parameters within modules are often used to generate different characteristics for different simulation scenarios.

OMNeT++ uses a network description language (NED) to define network topologies in terms of connections, modules, gates and parameters. The network definitions

are outlined using NED files. The information in a NED file is represented as classes, subclasses, connections, gate types, modules and submodules in terms of compound or simple modules, and imported libraries. One of the most significant properties of the NED language is inheritance—in comparison to other simulation tools. This allows developers to declare and define any component in a NED file—such as modules or channels—as subclasses of already existing components. For instance, a TCP-Client can be derived to an FTPClient by setting specific parameters. Furthermore, any standard-host submodule can be implemented as using either TCP or UDP port communications as well as using different queue models and network configurators without creating it more than once.

OMNeT++ also provides a visual perspective within the its IDE which contains a graphical editor using a NED file as its native format. The graphical editor is a built-in feature where network topologies can be defined either by directly coding them or by interacting with the graphical interface. Topologies and contents of modules can be implemented via simple drag-and-drop actions using the topology screen and the tools panels. Furthermore, some common network topologies such as ring, grid, star are already embedded into the panel, and can be created in a simple fashion using the graphical editor. Also, the modules can be decorated with other modules using the graphical editor.

OMNeT++ provides two user interfaces: TKENV and CMDENV. TKENV is OMNeT++'s GUI. It provides three methods: automatic animation, module output windows and object inspectors. Automatic animation in OMNeT++ is capable of animating the flow of messages on network charts. Module output windows make it possible to open separate windows for the output of individual modules or module groups. Object inspector, which can be used to display the state or contents of an object in most appropriate way, is a GUI window associated with a simulation object. Additionally the objects can be modified manually. It is by default possible to inspect every simulation object. There is no need to write additional code in the simple modules to make use of inspectors. These three methods made OMNeT++ have easy traceability and debuggability [43]. CMDENV is a pure command-line interface that is useful for batched simulation runs.

OMNeT++ has a separate file type called an initialization file that is used to further define scenario specifications. Ultimately, the scenario specifications that are given in the initialization file define how each run will be undertaken by the platform. A single initialization file can operate on more than one network descriptions.

2.2.2 Other Simulation Platforms

There are various simulation platforms that have been widely used by both the industry and academia. Some of these platforms can be listed as NS, OPNET, JiST/SWANS, J-Sim, SSFNet and Qualsim.

2.2.3 NS Platforms

Network Simulator (NS) is a name used for a series of simulation platforms—such as ns-1, ns-2 and ns-3. It is one of the most widely used network simulation platforms. The NS variants are built using programming languages such as C++ and Python. It also use the network animator (Nam) for visualization of simulation scenarios.

NS-2 [24]

NS-2 is an open source software that was started to be developed since 1989 [24]. The architecture of NS-2 relies on the Open Systems Interconnection (OSI) Reference Model. This model provides the ways in which a simulated packet is passed through the network layer, link layer, MAC layer and physical layer respectively. NS-2 supports Local Area Network(LAN), Wireless LAN(WLAN) and satellite networks. Various queueing techniques, algorithms and stochastic fair queueing such as First-In First-Out(FIFO) are supported by NS-2. Thus, NS-2 provides multicast, unicast, dynamic and static routing methods. On the transport layer, NS-2 supports popular protocols such as TCP, UDP and Real-Time Transport Protocol(RTP).

NS-2 is composed of C++ code that is used for modeling the behaviour of the networks and the network topologies can be controled and specified by oTcl scripts. Tcl needs to be written as a script at the beginning of the implementation; thus, no C++ code needs be written and compiled. Only when the user wants to add a new protocol(s) or model(s), C++ code needs to be written and compiled. oTcl-based design is chosen in order to prevent unnecessary recompilations during the simulation set up. However, this choice is questionable when the programmers try to conduct scalable network simulations. [40]

NS-2 uses three different tools to provide the users more effective analysis and user interfaces. *Nam* allows the user to see a simulation while it is running. The simulation components such as network topologies(except wireless traffic), packets flows and queues can be shown in Nam with their parameters. Nam is a part of NS-2 bundle

that provides the ability to manipulate running simulations by adjusting the time and speed of the simulation run [34]. *Trace Graph* is a useful tool to provide statistical analysis of the simulations. The performance tests such as throughputs, round-trip times or packet drop rates can be graphically displayed by trace graph. Trace graph is not a part of NS-2 simulator. *INSpect* is another visualization tool—besides Nam—that can support wireless networks. The main purpose of the development of INSpect is that Nam cannot support wireless traffic flows as well as INSpect which has more visual effects and features than Nam does.

NS-3 [21]

Like NS-2, NS-3 is a C++ code based simulator but it can be implemented in pure C++. In other words, oTcl is no longer used in NS-3; thus, abandoning the problems which were introduced by the combination of C++ and oTcl in ns-2. Furthermore, Python language can be optionally used to implement network simulations in NS-3. The removal of oTcl/C++ duality clearly reflects on the speed of the simulation execution.

Making a decision at expense of compatibility, NS-3 integrates architectural concepts and code from GTNetS [40], a simulator with good scalability characteristics. Besides performance improvements, the feature set of the simulator is also about to be extended. For example, NS-3 is slated to support the integration of real implementations code by providing standard APIs, such as Berkeley sockets or POSIX threads, which are transparently mapped to the simulation [40].

2.2.4 OPNET [17]

OPNET Modeler is a C based simulation platform that contains a number of network protocols and provides a comprehensive development environment supporting the modeling of communication networks and distributed systems. Some of these protocols can be listed as IPv6, QoS, Ethernet, MPLS, MIPv6, WiMAX, OSPFv3, IPv4, etc. Both behavior and performance of a model can be analyzed by performing discrete event simulations. It is free for educational and research and development use in many academic institutions and universities [39]. OPNET is not an open source project; thus, it has no source code editor while simulating kernel operations. OPNET and OMNeT++ are based on similar architectures in terms of their hierarchical models. However, OPNET has some limitations; for example, node levels cannot be

nested indiscriminately. It supports user interface debugging (graphical debugger and editor as well as animated outputs). OPNET uses fixed topologies by means of using parameters only allowed in its graphical editor [17].

Graphical user interface of OPNET Modeler supports the configuration of the scenarios and the development of network models. The work dynamics of OPNET differentiates the hierarchical levels in order to tune the simulation features. In the network level the network topology is created to be under investigation. Behaviors of the nodes are implemented in the node level by controlling the flow of the data among different functional elements inside the nodes. The protocols are represented by finite state machines(FSMs) and are created with states and transitions between states in the process level [11].

OPNET provides abundant network models such as, ATM, x.25, WiMAX, WLAN and Ethernet etc. and also have equipment for different vendors such as CISCO, 3COM and Sun etc. to allow researchers to either modify existing models or develop new communication models of their own [11]. OPNET provides the opportunity to user to obtain customized statistical data and detailed network performance analysis.

To conclude, OPNET is widely used technology which is equipped with all features to design, model and simulate systematically all types of networks such as home, corporate and wide area network. OPNET with its unique approach can provide objective and reliable quantitative basis for network planning and design and it could shorten network construction period, improve the exactitude of decision making on network building and reduce the risk of network construction investment [35]. However, it is quite expensive; thus, the project that will be simulated should be chosen carefully otherwise the expenses may exceed the real-world systems.

2.2.5 JiST [12]

JiST is a simulation kernel which uses the Java Virtual Machine to run the applications. SWANS is a scalable wireless network simulator built on top of JiST platform, which is organized as independent software components that can be used in wireless network or sensor network configurations [12]. JiST has a different approach than the other simulation platforms. The components of the network are formed as entities and an entity is implemented based on the method invocations with the other entities. The simulation core is notified independently during the entities run in a certain simulation time. Therefore, in a certain simulation time, the interaction in

between entities can be carried out during the code execution. These interactions between entities correspond to synchronization points and facilitate the parallel execution of code at different entities [40]. Owing to the use of Java Virtual Machine, a custom dynamic Java class loader is utilized by JiST because Java class loaders can rewrites the application's byte codes dynamically. JiST is no longer developed by its author Rimon Barr; however, Ulm University released a couple developments and enhancements on JiST.

2.2.6 J-Sim [22]

J-Sim is a simulation environment implemented in Java. J-Sim has its own graphical editor regardless. It uses Java to create and implement classes into simulation environment using Tcl. The graphical editor's native format is XML. XML files are directly loaded into the simulator. The topology format may be the same as NED format files in OMNeT++ [39]. Since it uses JAVA development tools, the implementations in J-Sim may be considerably fast on model development and debugging operations. Nevertheless, the performance of the simulation is notably weaker than C++-based environment [39]. And it is impossible to reprocess existing real-life protocol implementations written in C as simulation models [37]. J-Sim provides INET package containing IPv4, TCP, MPLS and other protocols.

J-Sim simulation uses queues and processes as building blocks. Working process of J-Sim is the execution of a simulation must be implemented step-by-step. For instance, if the processes are active, the other elements need to be passive. Therefore, in a single process, only one operation can have a chance to run. So, life of a process, described as a sequence of actions. It is mentioned above that J-Sim is a *Simula*-like simulation environment written in Java; thus, it has several principals that are inherited from the Simula language. Simulation process is controlled by process state manipulation to be routed; hence, the simulation time shared by all processes within a simulation world [22]. The message passing in between nodes of the simulation including blocking and non-blocking send and receive operations are supported based on whether the communication either symmetric, asymmetric or indirect. Random number generators can be defined by user-seed; thus, the diversity of experiments can be guaranteed. It has two GUI modes: batch and interactive.

J-Sim can be downloaded from the website [22] and implemented in all popular operating systems including Windows, Linux, OS X, and Solaris with installed Java

runtime environment.

2.2.7 SSFNet [26]

SSFNet is an acronym for Scalable Simulation Framework. SSFNet uses an API for public-domain standard events based on discrete-event simulation models implemented in JAVA and C++. SSFNet uses Domain Modeling Language(DML) which is a text-based format and has its own syntax. DML is defined as a NED file format similar to OMNeT++, and is used to define and configure network topologies. DML files contain both network topology and configuration data in it [16]. SSFNet is based on four different development packages: DaSSF [31] which is implemented using C++, and Renesys Raceway [13] and JSSF using JAVA implementation tools.

It has been designed for the expansion of network including topology, protocols, traffic, and etc, and is able to support simulation for the large-scale network like Internet. However it is not easy for general users to perform network simulation using SSFNet because the SSFNet does not provide users with any supplementary tools for designing of network elements and topology, and analyzing of simulation results. The network modeling and analysis process must be done manually by users themselves. This circumstance makes it difficult to perform reliable network simulation [44].

The network simulation modeling process and simulation analysis method vary according to the simulation application for specific network state. The SSFNet-based network simulation application processes the variety of network objects provided by SSF. However, although the network simulation modeling varies according to the network simulation application, the DML is used for every SSF network model [26].

SSFNet provides GUI and a special interface to obtain various statistical analysis. When the simulation is completed with respect to the network simulation model, the simulation results are carried out to the statistics processor through the simulation core interface to be calculated various statistics according to a statistics calculation rule predefined in the system logic set, based on the simulation results transferred from the SSFNet. The various calculated statistics are provided to the user with the GUI module using a variety of methods in order to support a simulation analysis. If the simulation result through the simulation analysis satisfies the user's request, the network simulation model is stored for future reuse.

2.2.8 QualNet—formerly GloMoSim [27]

QualNet is developed using the Parsec parallel simulation language mainly for use in wireless networks. It especially used in military. QualNet is developed in University of California, Los Angeles (UCLA) Parallel Computing Laboratory on top of GloMoSim (Global Mobile System Simulation) [27]. Simulation models are implemented using *entities* which contain time series, where these time series can be configured by using a language similar to the C programming language.

The architecture of QualNet consists of the simulation kernel as basic layer, model libraries as second layer and QualNet Developer GUI as top layer. QualNet uses eXtensible Markup Language(XML) files to be stored all basic information such as node locations or parameters. There is no needed to edit this file by hand owing to the fact that its GUI is notably sophisticated. QualNet supports several application layer protocols such as VoIP, File Transfer Protocol(FTP) or Constant Bit Rate(CBR) [34]. QualNet has also 'animator' tool to visualize the simulation run. Furthermore, the animator provides several outputs to be analyzed as results if they are activated. Like the other animators of the simulations, simulation run can be manipulated during the run process.

In contrast to freely available simulators, the graphical environment in QualNet is more comprehensive than the others because the creation and visualization of the network scenarios and the analysis of the simulation results are implemented in one single GUI. Entity configuration source codes are implemented for each network node as in OMNet .ini files. Because Parsec is used as runtime, kernel is only operated to support event scheduling and parallel simulation services. Use of Parsec with GloMoSim and QualNet for modeling simulations have been resulted in excellent parallel performance in wireless network simulations by providing a very efficient parallel simulation infrastructure [39].

2.3 Summary

In this chapter, some of the most popular network simulators were introduced with their characteristics. In the NEPTSim simulator, OMNeT++ Simulation Platform is used and explained with detailed information based on its working dynamics and features. OMNeT++ is one of the best alternative simulators to be implemented

Simulator	Possible Disadvantages
NS-2/3	Limits in TCP/IP Model Excessively unitary Impossible to create a graphical editor Library provides less function Use a lot memories to simulate Very slow in debugging Either a thread/coroutine-based programming model or FSMs built upon a message-receiving function
OPNET	Expensive Use fixed topology C based Either a thread/coroutine-based programming model or FSMs built upon a message-receiving function
JiST	No longer developed
J-Sim	Process must be run step-by-step
SSFNet	Does not provide users with any supplementary tools for designing of network elements and topology, and analyzing of simulation results Network modeling and analysis process must be done manually Difficult to perform reliable network simulation
QualNet	Parsec must be used with

Table 2.1: Defects of the Discussed Simulators to Compared To OMNeT++ in terms of the Efficiency of Implementing NEPTSim

for NEPTUNE Canada Cabled Ocean Observatory network owing to several reasons. Table 2.1 displays that OMNeT++ has various advantages to be used in this Master Thesis among the other simulators. To sum up the basic reasons why OMNeT++ has been chosen to be used for this research is: Initially, OMNeT++ can be downloaded from the website [25]; thus, it is a free tool for the academic studies. It is an open source project that can be improved by using C++ code base. Furthermore, although it is a free simulator, it is supported by several packages including INET, MANET, Ad-Hoc [1], Mobility Framework, Castalia [36] and Wireless Sensor Networks [36]. Therefore, it is not only simulator, it is a complete discrete event based simulation framework. All packages can be built-in with a quick installations if they are needed and these packages are working in OMNeT++ platform without any problems.

The most significant characteristics of OMNEt++ is that it is ease-of-use. Using the other simulators needs a strong programming language and networking background as a knowledge base. Also, the platforms themselves very hard to be learned

how to use by following their tutorials if the user does not have a graduate degree in computing or engineering science. Because their user interfaces are not user friendly unlike the user interface of OMNeT++, it is easy to capture the events during the simulation run in OMNeT++. A lot of them use their own script languages; thus, the user need to learn another programming language. According to undergraduate and master level graduate students, learning OMNeT++ is very easy owing to the fact that it needs only basic C and C++ programming language skills. The dynamic tutorial Tic-Tac-Toe was developed for the new learners of OMNeT++ that comes built-in with OMNeT++ Simulator.

Even though the well-known network simulators were introduced in the previous sections, except OPNET and NS-2/3 simulator, the others are not used frequently in academic researches because according to reviewed papers that are about network simulators for the literature review of this thesis, the other simulators neither good enough to implement NEPTUNE Canada network system owing to their performance nor they do not support related network models of network. Therefore, we most likely try to compare OMNeT++ with NS platforms and OPNET Modeler. OPNET has lots of protocol models, including TCP/IP, ATM, Ethernet, etc., but it is so expensive to get the license. NS-2 also has a large number of protocol models, but mostly centered around TCP/IP, that limit NS-2 in wide scalable network simulation. OMNeT++ currently has TCP/IP, SCSI and FDDI models. Along with the fast increase of users, the model library also rapidly consummates and it can satisfy the large-scale sensor network simulation the demand. OPNET is so expensive as well as it is known that there is an academic use version of OPNET but it is fairly limited to create a simulation for massive systems like NEPTUNE and NS-2's protocol model is excessively unitary, so OMNeT++ has the big advantage in the model library and the available model aspects [43].

Use of oTcl define network topology makes NS-2 impossible to create a graphical editor without using Nam; thus, it causes a big problem for the new users. A notably difference between OMNeT++ and OPNET is that OPNET uses fixed topology; however, OMNeT++ provides customize parameterized topologies with its NED language and graphical editor. In OMNeT++, simulations can be debugged and traced in three ways: object inspectors, module output windows and automatic animation; however, unlikely other simulators do not have all in one. For instance, although OPNET has a strong command-line simulation debugger, it does not have a graphical runtime environment. NS environments are a bit slow based on the simulation run

time in contrast to OMNeT++ owing to the use of memory [43].

Based on the facts mentioned above, simulation software in implementing NEPTUNE Canada network simulation, the performance of NS platforms are not very good, OPNET has good performance (like OMNeT++), but it is too expensive. So, OMNeT++ is the better than other simulators in implementing NEPTSim.

In this chapter it shows that OMNeT++ is an excellent simulation software based on its functions for the requirements of NEPTUNE Canada network system. Compare with other simulators, it reflects that OMNeT++ have better performance than the others in terms of capturing the network traffic of NEPTUNE. Its excellent performance, animating graphical user interface, convenience topology describing language and easy operation obtains more and more user's favor.

In the next chapter, the simulation environment for NEPTUNE Canada will be introduced by explaining that how NEPTUNE components can be implemented in a simulated arena.

Chapter 3

Simulation Environment for Neptune CANADA

3.1 Introduction

The simulation environment has the most significant affect on the network simulation because the network is composed of the all network components which have different features and parameters to be implemented. In this chapter, the implementation process of NEPTUNE Canada will be introduced with details. Initially, it will explain that how the network behaves during the data transmission process. Then each component will be introduced with their quantities and features. These components are classified based on INET functionality such as INET routers could be more than one different instruments. Thus, it will be explained the characteristics of the instruments in the network traffic. Furthermore, Tables A.1 to A.3(in the Appendix section) shows us that all the message frequencies and message sizes of the instruments.

3.2 Implementation

NEPTUNE Canada simulator (NEPTSim) was built based on the current NEPTUNE Canada Ocean Cabled Observatory network topology by using OMNeT++ simulation platform. All submodules, connections, configurators and protocols were included in the simulator considering actual NEPTUNE topology that was provided by NEP-

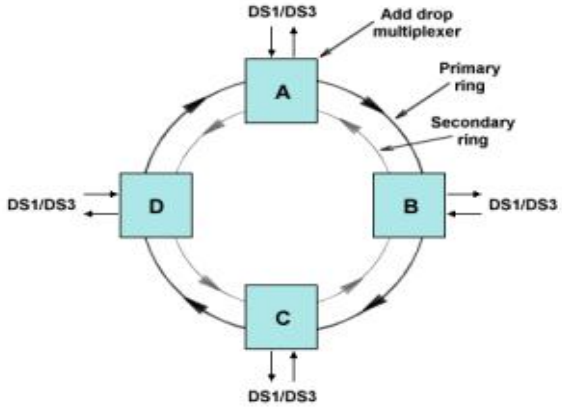

Figure 3.1: Bi-Directional Line Switched Ring of SONET [28]

TUNE Canada Systems Staff[1].

With respect to the network topology, SONET type networking was used in the implementation of the NEPTUNE's network. SONET is an acronym of Synchronous Optical Networking that is mostly used to connect two points using in between 155 Mbps and 2.5 Gbps data rates in process of the data transmission [3]. NEPTUNE network spans 200.000 km^2 area with its 818 km long fiber-optic cables. So building high-bandwidth data streams like NEPTUNE, SONET may be one of the most effective network topology method among the others for NEPTUNE network system. Compared to Ethernet cabling, SONET fiber runs much farther 100 meters in terms of the efficiency of transmitting data . SONET multiplexes channels having bandwidth as low as 64 Kpbs together, where data frames that are sent at fixed intervals [3]. The data channel can be used *bi-directionally.*

Figure 3.2: Linear Automatic Protection Switching System of SONET [28]

One of the most significant characteristics of SONET is that SONET supports a ring topology. Figure 3.1 illustrates the ring shape that allows the data to be

[1]NEPTUNE Canada Information Technology/Systems Staff: Martin Hoffman as *systems manager* and *mission-critical system administrators*: Austin Henry, Shane Kerschtien and Nic Scott

transmitted in either clockwise or counter-clockwise direction in between each nodes. Having bi-directional data flow provides reliable transmission. SONET uses one side/direction of the ring topology as *the working ring* or *the primary ring* and the other side stands for *the protection ring* or *the secondary ring*. SONET automatically detects failures in a fraction of time with its data transfer control. As a result, if the working ring fails, the protection ring handles the network traffic [3]. Figure 3.2 outlines the Linear Automatic Protection Switching System used for this purpose. Therefore, SONET can be described as a *self-healing* network. Figure 3.3 a problematic scenario where the fiber between two nodes breaks, yet, the SONET still functions. This functionality was recently used on the NEPTUNE network when a fault in a branching unit caused one node of the network to be taken offline and traffic from the other nodes was routed clockwise around the loop to bypass the fault [6].

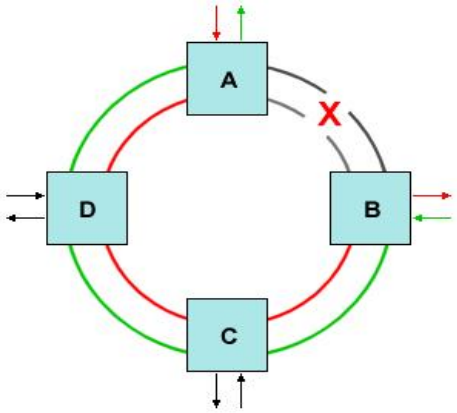

Figure 3.3: A Fibre Break Between Node A and Node B. Then The Traffic can still flow [28]

NEPTSim contains 2 different types of network units on the backhaul: standard hosts and routers and these unit types are derived form the INET package; thus, they are INET nodes. The reason why INET package was used is that INET nodes are already built-in with their messages and modules which have the same properties as NEPTUNE nodes in place today.

3.2.1 Implementation of the NEPTUNE Nodes

The Submodules are configured with respect to INET elements on OMNeT++ network description environment and the initialization platform. Initially, we start importing specific integrated INET nodes including inet.networkayer.autorouting.FlatNetworkConfigur

inet.nodes.inet.Router, inet.nodes.inet.StandardHost and ned.DatarateChannel to create related INET modules. In this section, INET nodes are introduced according to their types and roles in the NEPTUNE backhaul.

The NEPTUNE Canada simulator contains 96 instruments including 38 different types of nodes, 11 junction boxes, 6 node stations —one of them is empty in Middle Valley, 6 branching units, 1 shore station in Port Alberni and 1 UVIC DMAS; in total there are 121 devices that are linked on the NEPTUNE network. Figure 3.4 provides a view of the NEPTUNE network architecture. In the rest of this section we describe the submodules of the NEPTSim.

3.2.2 Data Channels and Connections

In NEPTUNE Canada network traffic, there are 4 different type of network channels that are defined in the network description file based on the data provided. The first channel provides 10 Gbps data rate between UVIC DMAS and shore station in Port Alberni. This channel has the largest volume in the system, and is located in between the edge of the network of the system and the main receiver—UVIC DMAS.

The shore station is linked to the first branching unit as followed with 5 other branching units that are also linked to other branching units. This solid connection—*spur cable*—forms a ring-shaped SONET network that has two cables in order to control the network traffic in two directions. A 2 by 2 Gbps data channel is implemented on the spur cable. Furthermore, each branching unit is connected to a regional node station. And the connection between node stations and branching units provide 1 Gbps data rate using a fiber-optic cable.

6 Node stations split the backhaul into 6 separated regions and the network behavior in between each regions' devices is identical. Similar to the connections between node stations and branching units, node stations are linked to junction boxes with 1 Gbps data rate. Junction boxes can be linked to both instruments and other junction boxes with different data channels. All the network channels transmit the data with 0.1 delay rate. Junction box to junction box and instrument to instrument connections are implemented with either 1 Gbps or 100 Mbps data rates. An important connection limit on the junction boxes is that more than 10 components cannot be connected to a single junction box.

Instruments can also be linked to either junction boxes or any other instrument with either 1 Gbps data rate or 100 Mbps data channel. The instruments are the final

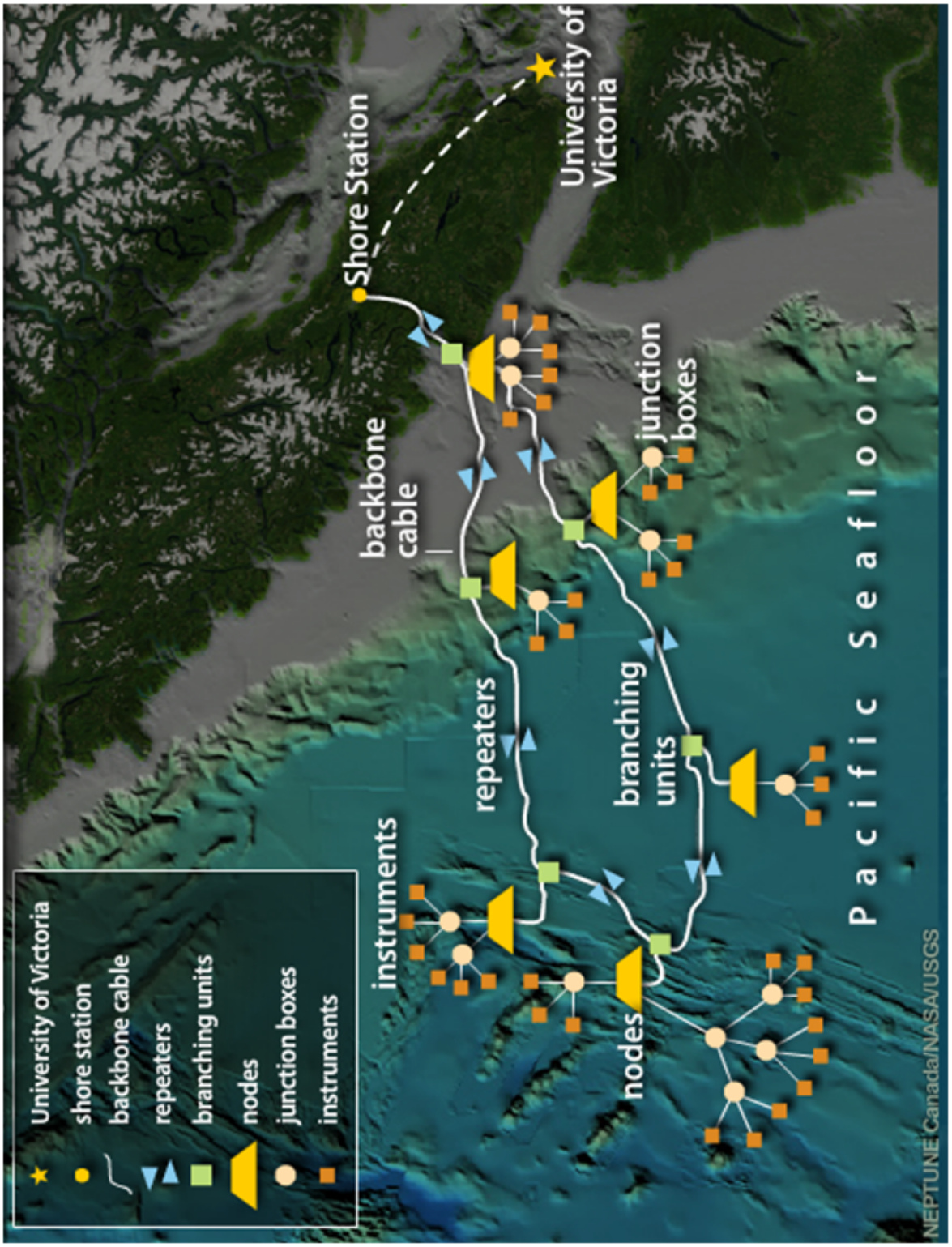

Figure 3.4: NEPTUNE Canada Network Architecture [23]

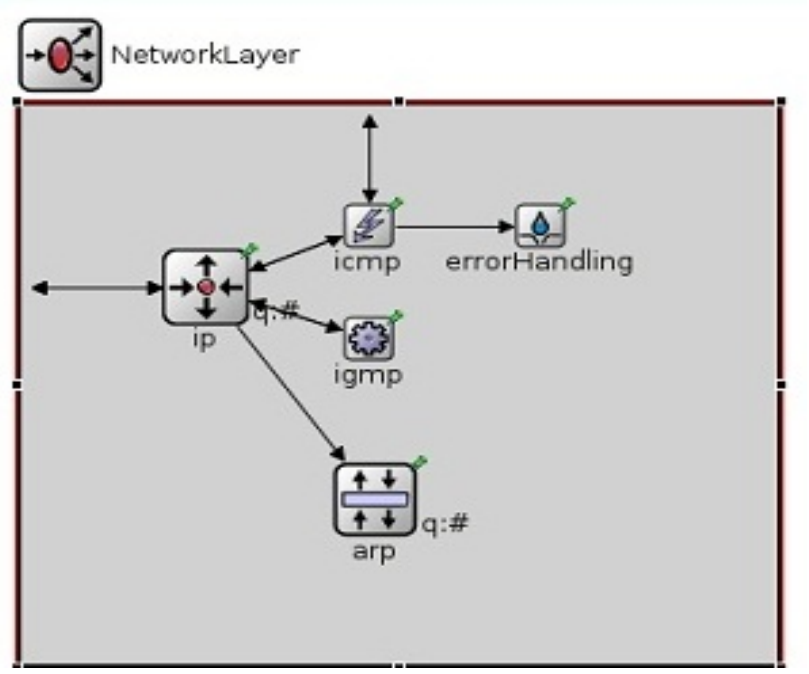

Figure 3.5: The Elements of The INET Network Layer

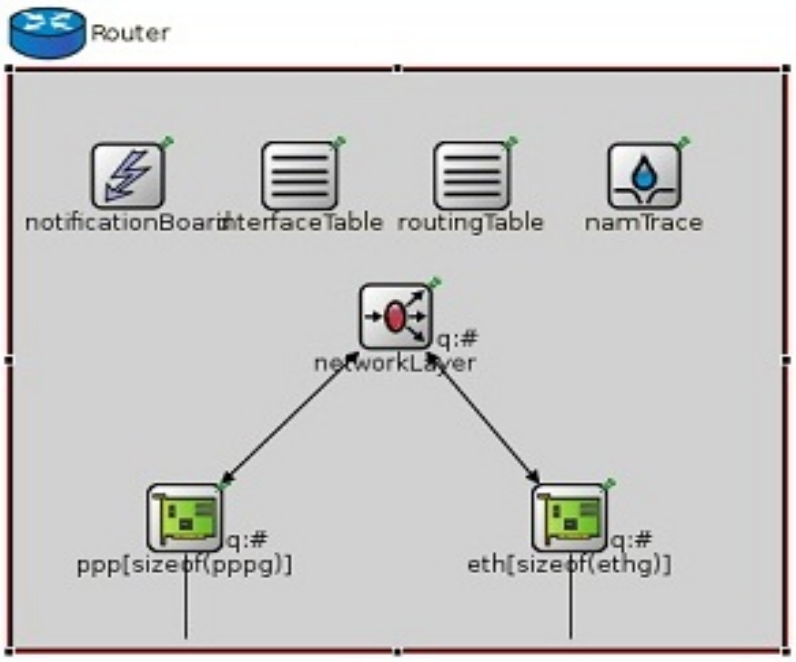

Figure 3.6: INET Router in OMNeT++ Simulation Tool

spot of the each regional network branches. The data flows from the instruments—where the edge of each regional branch is located—towards UVIC DMAS by follow-

ing a secure and the shortest path as configured by the network configurator used by NEPTSim. It runs Dijkstra's Shortest Path algorithm in order to compute the shortest path based on the predetermined IP nodes from the network topology [10]. The connections are implemented based on the NEPTSim modules' characteristics.

3.2.3 Instruments

The instruments are the data generators in the NEPTUNE network. The generated data are sent by the transmitters to the final destination point—UVIC Data Management and Archive Station at UVIC. Instruments use two types of transmission protocols: TCP and UDP. There are a number of basic differences between TCP and UDP communication protocols. TCP is connection—oriented and sends data on a transmit-acknowledge basis where transmission is guaranteed as long as the physical connection remains functional. On the other hand, UDP is not connection—oriented and does not require any acknowledgements from the receiver. In the NEPTUNE topology, most of the instruments are connected to junction boxes. However, some of them may be linked to another instrument, so-called 'piggy-back' instruments, in which case the data are aggregated before transmission. The major role of the instruments in the network generating data as they are configured in the NEPTUNE topology. Besides, some of them may be used as routers on the network. In other words, a minority of the instruments not only are capable of generating data but also can transmit data as being part of the network transmission process. These router-featured instruments are implemented to decrease cable length in between components which makes the network traffic more efficient and provides less delay in packet transmission. Therefore, any submodule could be defined as either an INET router or an INET standardhost. In this case, if a submodule is defined as an INET router, it can both generate and transmit data. However, an INET standardhost is not capable of contributing to the network traffic in any way other than generating data.

The main reason why the INET package was used in NEPTSim is that various useful applications are built-in features in INET nodes. Some of these applications can be listed as PingApp, TCPDump, TCP, UDP, SCTP, PPP[g], ETH[g] are included in INET package. These common applications are just introduced in this thesis without detailed information.

Figure 3.5 demonstrates that INET standardhost's network layer contains various

applications that are built in and compatible with the other INET nodes. The network layer includes an IP application that generates/operates IP addresses of the other submodules. INET standardhost node contains the error messages and the transport protocols controlled by ICMP (Internet Control Message Protocol) under its Internet layer. It basically selects what type of transmission protocol should be used in the network. Furthermore, ICMP determines the checkpoints of the data packages and checks whether or not the data is delivered to the next destination without any problems.

Internet Group Management Protocol (IGMP) is a communication protocol that is operated by data generators—*hosts* and routers—on ip networks in order to build multi-cast group networks. IGMP is commonly used for streaming on-line video data. Video data streaming is one of the greatest concerns of the NEPTUNE backbone because the video stream data have the biggest volume data on the backbone. These instruments are video cameras and hydrophones that use the UDP transmission protocol. So, using IGMP applications makes video streams transmission much easier to implement in the NEPTSim. 38 different instruments generate different sizes of data. For instance, hydrophones generate the largest volume of data on the network of roughly in between uniformly 0.9 Mbps and 2.9 Mbps in every single second. The smallest volume of data generated by a few instruments is 0.25 Mbps.

The Address Resolution Protocol (ARP) is an Internet protocol that is used to map IP network addresses to the link layer. ARP basically finds an address of a computer in a network. Therefore, ARP is used to translate between Medium Access Control (MAC) addresses to the link addresses of individual nodes in the network. ARP uses 4 types of messages: ARP request, ARP reply, RARP request and RARP reply. ARP has its own cache memory and it controls the requested and received data transmission in between data generation point(s) and the destination point(s). The duty of ARP model in the link layer is that controlling the message traffic of the data packages and configure them in terms of the type of the transfer protocols use [14].

3.2.4 Routers

INET routers are the relays of the NEPTSim simulation. Their primary role is to relay the generated data to the main receiver with as less as possible data loss. There are 5 different submodules that are defined as INET routers in NEPTSim: instruments, junction boxes, node stations, branching units and shore station.

The primary element of the NEPTSim data distribution process is junction boxes. Because they are defined as INET routers in the system and they do the same operations as the router do. Main role of a junction box is transmitting data— from most likely instruments—to the linked node station. However, besides data transmission, junction boxes can generate data as well if it is needed/configured. There are 11 junction boxes that can be linked to both instruments and node stations. Additionally, 3 junction boxes —located on Barkley Canyon—are connected to one junction box. Thus, junction boxes can be connected to another junction box as well [41].

The junction boxes contain the only network layer that has only gates. Figure 3.6 illustrates that these gates are *ETH* and *PPP* gates. Eth is for Ethernet connection that is not used in NEPTSim because of the ineffectiveness in SONET topology. Point-to-point (PPP) is a data link layer protocol which encapsulates only two nodes for transmission on synchronous and asynchronous communication lines. The protocol facilitates transmission of higher level protocols such as TCP/IP, across diverse communication links [42].

There are 6 node stations, 6 branching units and 1 shore station in NEPTSim that are implemented as INET routers like junction boxes. Unlike junction boxes, they are only capable of data transmission. Based on the topology, SONET runs bidirectionally in between 6 branching units. These INET routers contain a specific type of queue—Drop-Tail-Queue—that is used in NEPTSim. The queue type can be easily changed if required [30]. The shore station should have big stack size—in this case it has 10 Gigabyte stack, because the data transmitted from the backhaul is gathered and queued in the shore station in order to be delivered to UVIC DMAS [7, 2].

3.2.5 UVIC-Data Management and Archieve Station

UVIC Data Management and Archive Station is implemented as a standardhost but it acts like a server in the NEPTSim. It contains and is configured both TCP and UDP sink applications because UVIC DMAS is the final destination of the network traffic. In other words, if the data successfully transmitted by INET routers to UVIC DMAS, the DMAS should receive it then send reply to the source to prove that the sent data is received without any loss if the data generator uses TCP port. The whole received data is stored in the various databases at UVIC DMAS since 2009. The stored data can be used by scientists via the Internet when it is needed.

3.3 Summary

This chapter allows us to imagine how NEPTSim will be mapped the NEPTUNE Canada. As a network flow, NEPTUNE uses SONET network type that provides data can be transmitted bi-directionally. On the NEPTUNE SONET network, all the data channels and components were discussed in this chapter. In this chapter, it was explained that how the real-world system NEPTUNE Canada tried to be mimiced into a simulated arena. The NEPTSim split the network into 4 main sections: data channels and connections, instruments, routers and the main receiver—UVIC DMAS. Tables A.1to A.3(are located in Appendix section) illustrates that the characteristics of each instruments in the data transmission process. In this chapter, all the devices of NEPTUNE Canada were considered for the purpose of the research with the view of networking in computer science. Therefore, the complete features of the devices were not defined and explained based on what exactly their duties in the NEPTUNE Canada. The only concern is to observe the data transmission process of the network. In the next chapter, the implementation of the NEPTSim will be introduced by implementing a case study to be observed.

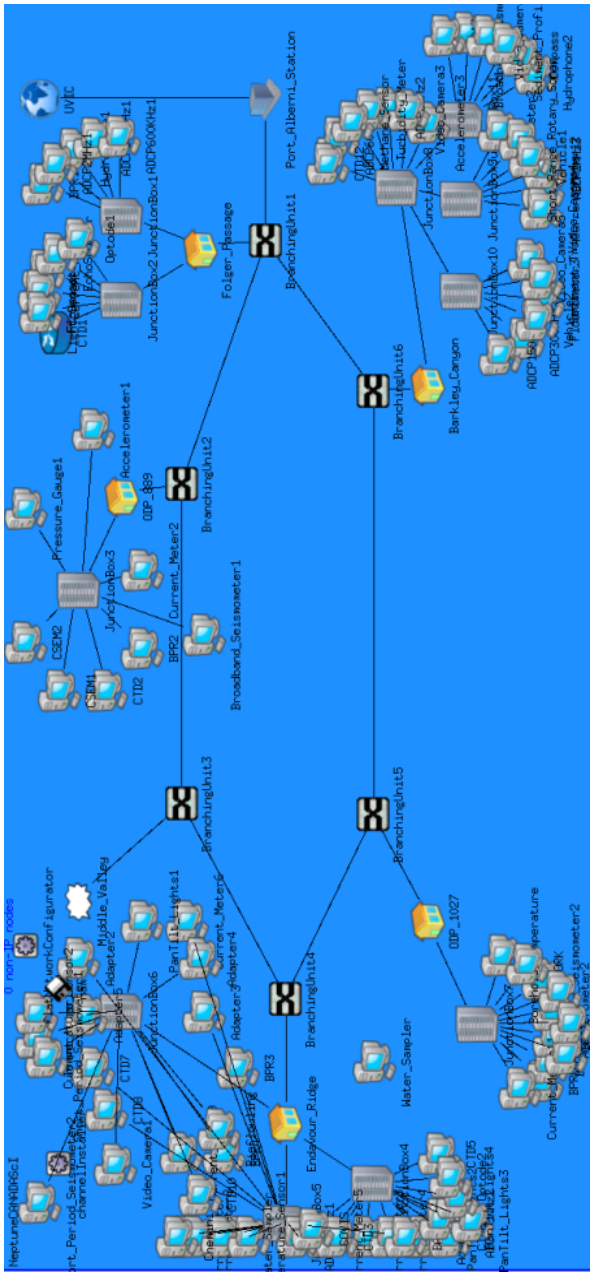

Figure 3.7: NEPTSim

Chapter 4

Implementation of NEPTUNE Canada Simulation

In this chapter, the implementation process of NEPTSim will be introduced by implementing a case study on it. In the Chapter 2, the characteristics of OMNeT++ Simulation Platform was explained in terms of how to create/code a simulation in the editors of OMNeT++. And in Chapter 3, the features of the components of NEPTUNE Canada was introduced with all details of the each components. This chapter will be the realization of the both chapters because in the further sections, it will be explained that how to add and modify the instrument into NEPTSim by using OMNEt++ Simulation Platform. Afterwards, the experiment—that observes how changing frame capacity effects the network traffic— will be discussed with the results.

4.1 Tuning NEPTSim Instruments

Our study reveals that, although building the simulator in OMNeT++ has many advantages, such as ease of tuning and calibration, capturing sufficient details regarding the working behavior of the actual NEPTUNE environment is still challenging. The NEPTUNE Canada network has been gradually enhanced by adding new instruments to the backhaul since 2009. In this example, we add an INET node '*New Instrument*' in NEPTSim in order to expose that how such alterations are implemented on NEPTSim platform.

Figure 4.1 displays the current ODP889 node station region that has 1 junction

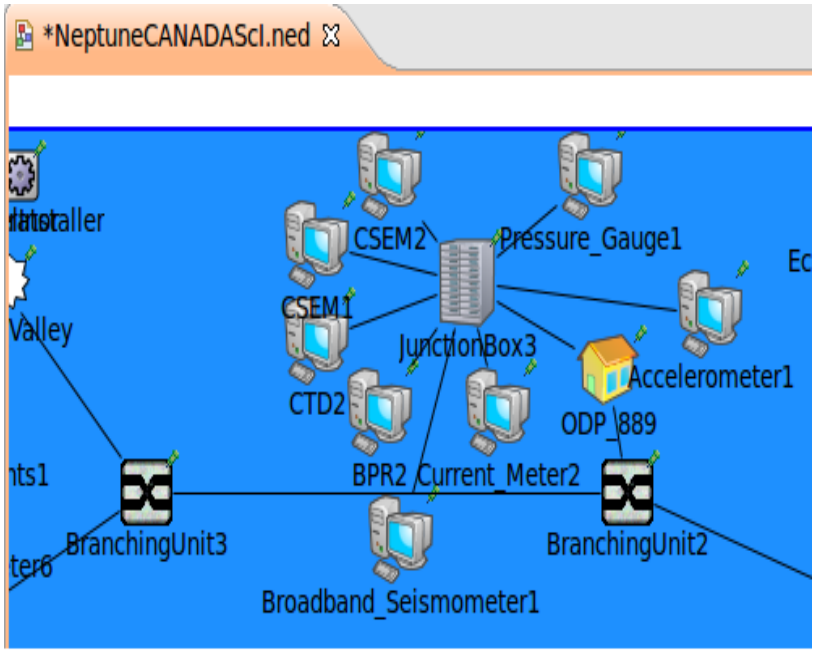

Figure 4.1: Before Adding *New Instrument* on ODP889 Branch of NEPTSim

box and 8 instruments. In this study, we want to add one instrument called *New Instrument* to junction box-3.

Figure 4.2: Defining *New Instrument* in the Network Description File(.ned)

Figure 4.2 indicates that New Instrument was defined as an INET standardhost under the submodules section of the network description file. After defining the new instrument of NEPTSim, we can connect it to any INET routers in the network—in this scenario, we linked it to the junction box-3. Figure 4.3 shows that the new instrument was connected to junction box by using PPP[g] gates via *C4* network channel—C4 channel distributes 100 Mbps data with 0.1 us delay rate which is explained in section 3.2.2.

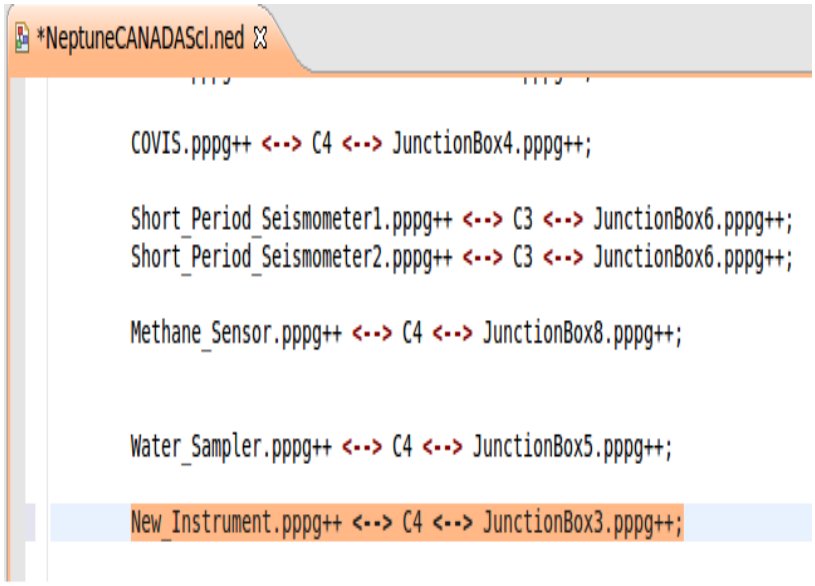

Figure 4.3: Connecting *New Instrument* to Junction Box-3

Figure 4.4 shows the configuration of INET standardhost node. In this scenario, we decided that our new instrument contains TCP port communication and generates 1 Mb data exponentially per second. In Figure 4.4 also shows that in the 0th second, the new instrument starts to generate data—*tOpen=0* and it generates data until the simulation time limit reached—*tClose=-1*.

After initializing *New Instrument* on NEPTSim, we can run the system to be sure if the adjointed instrument is working accurately. Figure 4.5 illustrates the ODP889 node station branch with the *New Instrument*. NEPTSim provides valuable outputs to be tested and analyzed after each run based on the given parameters.

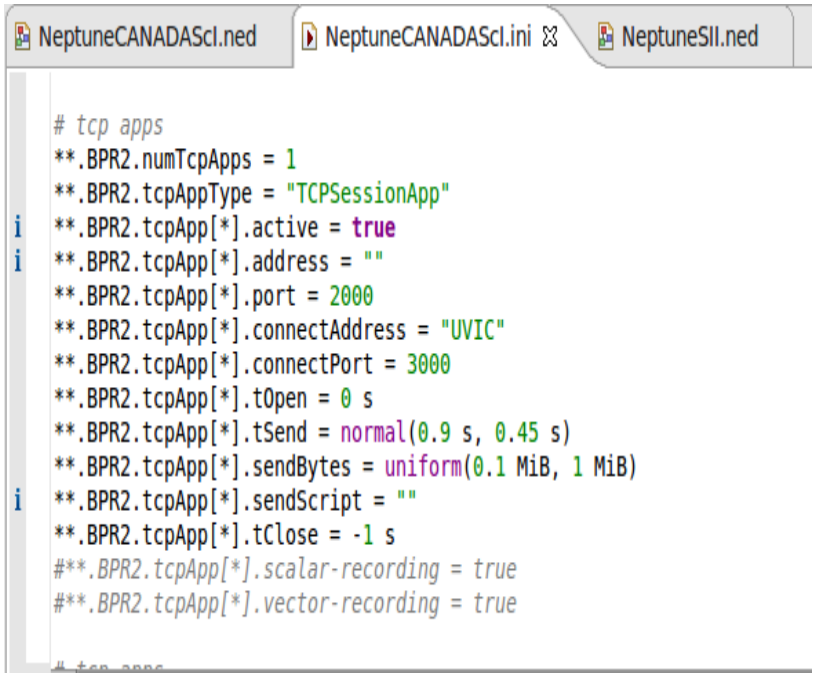

Figure 4.4: Configuring *New Instrument* in the Initialization File(.ini)

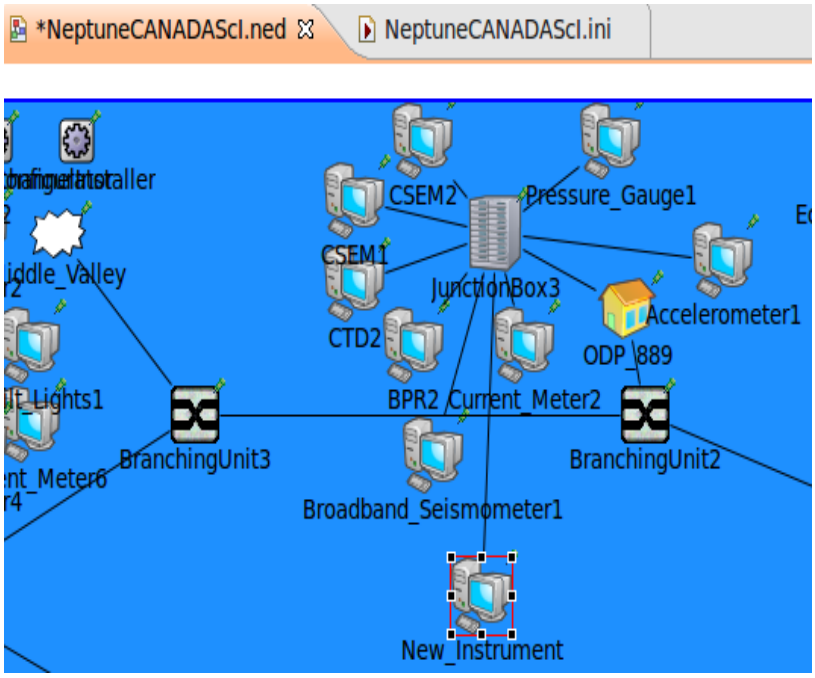

Figure 4.5: After Adding *New Instrument* on ODP889 Branch of NEPTSim

4.2 Adding an Instrument in NEPTSim

In this section, we added the instrument that uses UDP protocol into th NEPTSim then we will observe the results in terms of how the new instrument effects the network

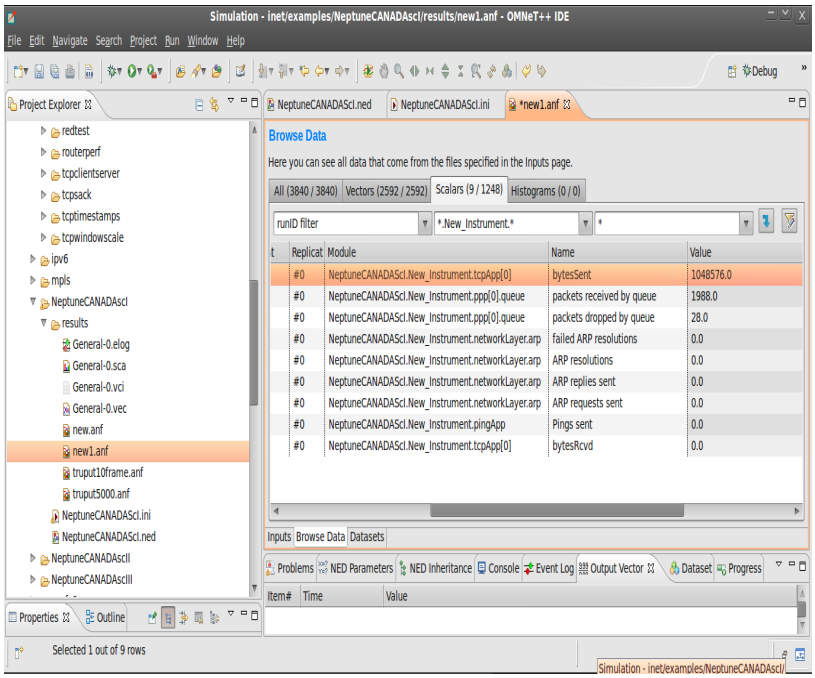

Figure 4.6: After Adding *New Instrument* on ODP889 Branch of NEPTSim

traffic based on the given parameters. The main purpose of this experiment is that to observe how the frame capacity of the routers effect the backhaul.

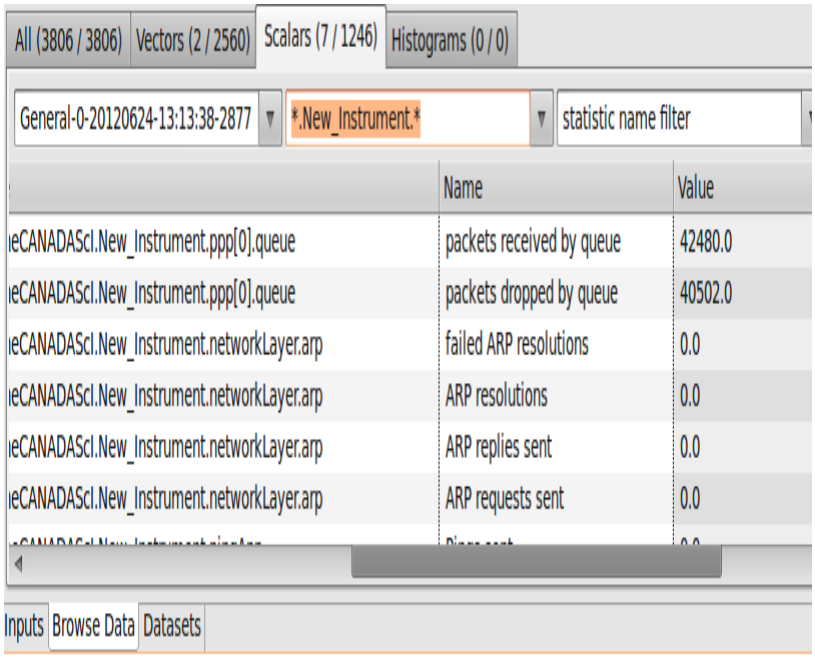

Figure 4.7: *New Instrument*, UDP Results with 10 Frame Capacity

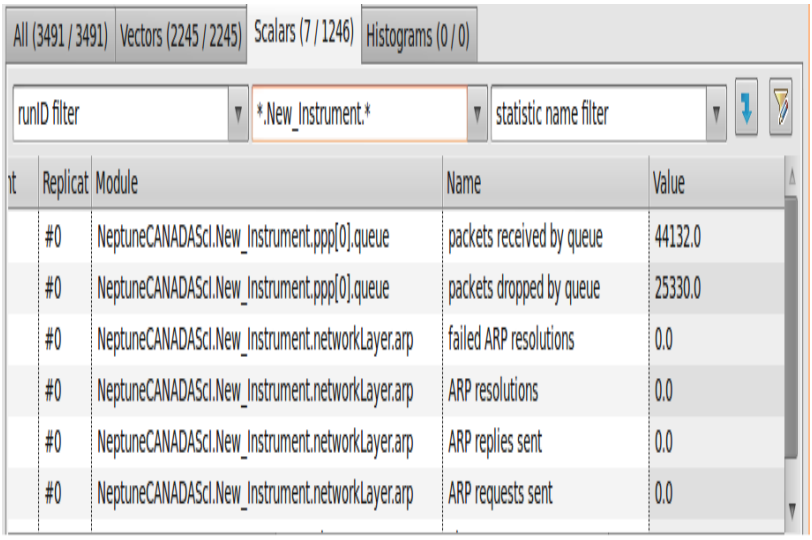

Figure 4.8: *New Instrument*, UDP Results with 100 Frame Capacity

Figure 4.6 displays that *New Instrument* node is accurately working in the network. Starting with a sample seed of 1000, the system runs the network with the new instrument in 300 simulation seconds. As a result, *New Instrument* sends 1048576 bytes and its queue—*DropTailQueue*—received 1988 packets—28 packets were dropped.

The *seed* is a number that controls whether the Random Number Generator produces a new set of random numbers or repeats a particular sequence of random numbers [18]. The Random Number Generator will produce a set of random numbers based on the value of the *seed*—in this case the given seed is 1000. The simulation could be run with different seed numbers but it must be ran several times with some specific mean values and for a while such as at least 4 or 5 simulation hours to be observed the distinguish between each seed numbers.

4.3 Modifying an Instrument in NEPTSim

Because of OMNeT++ ease-of-use, the submodules can be change instantly after the definition. For instance, we can initialize the new instrument with UDP port communication protocol in Figure 4.9. After the simulation run, Figure 4.7 shows

NeptuneCANADASci.ned | NeptuneCANADASci.ini | NeptuneSII.ned

```
# udp app configuration
**Video_Camera5.numUdpApps = 1
**Video_Camera5.udpAppType = "UDPBasicApp"
**Video_Camera5.udpApp[*].localPort = 2001
**Video_Camera5.udpApp[*].destPort = 3001
**Video_Camera5.udpApp[*].messageLength = uniform(0.5 MiB, 1.5 MiB)
**Video_Camera5.udpApp[*].messageFreq = exponential(1 s)
**Video_Camera5.udpApp[*].destAddresses = "UVIC"
#**Video_Camera5.udpApp[*].scalar-recording = true
#**Video_Camera5.udpApp[*].vector-recording = true

# udp app configuration
**Hydrophone1.numUdpApps = 1
**Hydrophone1.udpAppType = "UDPBasicApp"
**Hydrophone1.udpApp[*].localPort = 2001
**Hydrophone1.udpApp[*].destPort = 3001
**Hydrophone1.udpApp[*].messageLength = uniform(0.9 MiB, 2.5 MiB)
**Hydrophone1.udpApp[*].messageFreq = exponential(1 s)
**Hydrophone1.udpApp[*].destAddresses = "UVIC"
#**Hydrophone1.udpApp[*].scalar-recording = true
#**Hydrophone1.udpApp[*].vector-recording = true
```

Figure 4.9: *New Instrument* with UDP Configuration

the results based on the packet receive/drop operations when *New Instrument* uses UDP protocol within generating 1 Mb data exponentially per 5 seconds. The network traffic is ON/OFF model in the process of data generation. Thus, the device turns ON for 5 seconds and in these time span, the instrument generates the set size of data. After completing a single generation process, the device turns OFF. Then the process continues by doing so until the simulation time is reached [5]. Consequently, the simulation run with the same parameters as the *New Instrument* uses TCP protocol—with seed number 1000 and 300 simulation seconds. The results point out that *New Instrument*'s queue—PPP[0]— received 42480 packets in the queue and 40502 packets were dropped in the queue. Because the frame capacity of network configurators is 10; thus, 95 percent of the generated packets were dropped in the queue. In this example, we changed the frame capacity of network configurator from 10 to 100. Figure 4.8 shows that *New Instrument* received 44132 packets and dropped 25330 packets. So 57 percent of the packets were dropped with use of 100 frame capacity. So, frame capacity of network configurator has a crucial role in network traffic; like, using different port communications.

The simulator shows us that, using TCP and UDP port communication protocols distinguishes the number of generated and dropped packets.

All (3838 / 3838) | Vectors (2590 / 2590) | Scalars (12 / 1248) | Histograms (0 / 0)

runID filter | *.BranchingUnit1.* | statistic name filter

Module	Name	Value
NeptuneCANADAScI.BranchingUnit1.ppp[0].queue	packets received by queue	214628.0
NeptuneCANADAScI.BranchingUnit1.ppp[1].queue	packets received by queue	134031.0
NeptuneCANADAScI.BranchingUnit1.ppp[2].queue	packets received by queue	48631.0
NeptuneCANADAScI.BranchingUnit1.ppp[3].queue	packets received by queue	16915.0
NeptuneCANADAScI.BranchingUnit1.networkLayer.arp	ARP requests sent	0.0
NeptuneCANADAScI.BranchingUnit1.networkLayer.arp	ARP replies sent	0.0
NeptuneCANADAScI.BranchingUnit1.networkLayer.arp	ARP resolutions	0.0
NeptuneCANADAScI.BranchingUnit1.networkLayer.arp	failed ARP resolutions	0.0
NeptuneCANADAScI.BranchingUnit1.ppp[0].queue	packets dropped by queue	0.0
NeptuneCANADAScI.BranchingUnit1.ppp[1].queue	packets dropped by queue	0.0
NeptuneCANADAScI.BranchingUnit1.ppp[2].queue	packets dropped by queue	0.0
NeptuneCANADAScI.BranchingUnit1.ppp[3].queue	packets dropped by queue	0.0

Figure 4.10: Branching Unit-1, Main Bottleneck of the *Spur Cable*

Like other simulators, the measurement of time must be analyzed based on the simulation time. The simulation clock must be considered during the output process. Because OMNeT++ is a discrete-event simulator, NEPTSim generates discrete events; thus, time *hops* because events are occurring or completed without perceptible delay. So, the clock steps up to the next event start time as the simulation continues.

Figure 4.10 illustrates the connections and related links from the spur cable to branching unit-1 where the bottleneck of the spur cable is. It has 4 different connections—PPP[0], PPP[1], PPP[2] and PPP[3]; with branching unit-2, branching unit-6, Folger Passage node station and the shore station—and its each queue receives certain number of packets to be transmitted. The role of the branching unit-1 is vital because the whole network except the main receiver—UVIC and shore station—is linked to branching unit-1. In other words, the *spur cable*, bi-directional and 818 meter long cable, ends up at the queue of branching unit-1. Thus, we can evaluate the network traffic based on the given parameters by observing the branching unit-1's data results. By looking at the results as shown in Figure 4.10, NEPTUNE experts can start to better determine if the network is working accurately with the given parameters. Consequently, the NEPTUNE Canada network system with its all instruments with the data channels can be implemented based on the using real-time

data and desired scenarios can be tested easily in NEPTSim if they requested.

4.4 Summary

The purpose of this chapter is to prove that NEPTSim can be easily simulated in OMNeT++ Simulation Platform. The case study was implemented in three way: adding a new instrument that uses TCP protocol into one of the pre-initialized scenarios of NEPTSim, adding a new instrument that uses UDP protocol in a same way in the previous way and modifying a new instrument by changing its parameters that are ready to implement in the initialization file. Additionally, we observed the frame capacity changes in the case study. By implementing the case study, it is shown that working on OMNeT++ is easy to tune and calibrate the characteristics of NEPTSim simulation.

In the next chapter, we will observe the results that were implemented on NEPTSim by using different scenarios based on the statistical analysis.

Chapter 5

Evaluation and Performance Analysis

5.1 Introduction and Motivation

The observation of a network system—depending on its design and components—needs to be compared between usually different what-if scenarios in order to give us an idea in terms of which one(s) is/are the most effective for certain circumstances. What-if scenarios usually perform with the different parameters or designs based on statistical analysis in terms of determining the limits of the systems. The performance analysis are interpreted based on the various models that are used today, for instance throughput performance test. Each simulation has been run for 900 simulation seconds with the first 10 seconds as a warm up period. During the warm up period, the statistics are ignored. Both data transmission and data generation points are recorded statistically in terms of not only their scalar parameters, but also vector parameters. Each simulation is run with 10 different seeds for each performance test.

5.1.1 NEPTUNE Canada Scenarios

Two different scenarios where implemented in the evaluation process by performing in the OMNeT++ simulations to collaborate the analysis. The current NEPTUNE Canada network system has been implemented as one of the scenarios to be evaluated. The current system contains 96 different instruments that are generating data using either TCP or UDP data transmission protocols and 23 routers that are connected to each other in accordance to the certain topology pattern. First scenario exactly

mimics the current network system.

In the second scenario, one more node station is added into the Middle Valley region of the network. Furthermore, two more junction boxes are integrated into the network system with 5 more instruments that are linked. 5 more instruments that are linked to 2 additional junction boxes that are integrated into the network system. The second scenario, we try to mimic the NEPTUNE Canada network system that has been installed after the last enhancement to test and analyze the effects of the additional instruments on the network traffic.

In the evaluation process, two scenarios are implemented in the different simulations with different seeds. The main purpose of the comparison between the existing network and the purposed network is to observe the behavior of the data transmission components as well as the network traffic.

Enhancing network performance by the network is a significant design and operation challenge that is encountered by scientists today. Network performance depends on the design of the data transmission protocols and the type of queue management in routers. Queue management systems provide feedbacks to end-points of the network by either dropping packets or marking the packets with Explicit Congestion Notification(ECN) marks. Therefore, routers play a vital role in data transmission based on the buffer sizes set have a direct impact on the network traffic.

In routers, the challenge is that buffers need to be fast and large. The capacities of Internet routers are limited by the buffers they must use to hold packets. The challenge is that buffers need to be both large and fast.

5.2 Calibration of NEPTUNE Canada

In the initialization files of the both scenarios, the parameters and models that are chosen, have a crucial role in the work process of the bottleneck. Instruments use either TCP or UDP transmission protocols and each instrument has specific parameters according to its type and role in the backhaul. As a role in the network, University of Victoria Data Management and Archive Station is defined as an INET standardhost but it has two sinks on it unlike the other standarthosts don't have them. A sink is a function that receives the incoming events from the other transmitters or sources. In other words, the device that has a sink on is the receiver of the network system. Therefore, UVIC DMAS has two types of sink applications: *TCPSinkApp* with 3000 port number and *UDPSink* with 3001 as a destination port number to listen on.

TCPSinkApp accepts any number of incoming TCP connections, and discards whatever arrives on them [20]. And UDPSink consumes and prints packets received from the UDP module(s).

5.2.1 Characteristics of NEPTUNE Instruments

The main characteristic of TCP instruments is to use *TCPSessionApp* for the data generation process and the application can be implement with regard to the given parameters. TCPSessionApp is a single-connection TCP application; according to NEPTUNE network topology, it opens a connection at the beginning of the simulation run which means that at the 0th second, sends the given number of bytes which is uniformly distributed between 0.1 Mb and 1 Mb as one event within the time that is normally distributed in 0.9 second as a mean value and 0.45 second as its variance in each simulation second until the simulation time ends. The instruments act as a client; thus, they work with TCPVirtualBytesSendQueue/RcvQueue as sendQueue/receiveQueue setting for TCP. When TCPApp is active=false, the application will listen on the given local port that is 3000 as a port number, and wait for an incoming connection. In this simulation, ephemeral port is not used in any scenario; thus, all the ports are defined and initialized manually.

On the other hand, *UDPBasicApp* is used by hydrophones and video cameras that are only instruments which send data via UDP transmission protocol. UDPBasicApp sends UDP packets to the given IP address at the given interval. In NEPTUNE simulation hydrophones and video cameras send uniformly distributed message length in exponentially per second interval time via 2001 local port.

These instruments generate data that are transmitted via the routers of the system. During the transmission process, the features of the routers and the density of the data packets effect the efficiency of the network. Efficiency can be measured by throughput performance analysis and packet drop rate in the networking systems. Thus, we performed these two performance tests to analyze how to differentiate the additional instruments to the current NEPTUNE network. In the next section, queue management of the routers will be introduced in terms of queue classification and the process of the queueing perspective in NEPTUNE system.

5.3 Queue Management

In routers of the network—including junction boxes, node stations, branching units and the shore station, Drop-Tail queue has been used with a 10 frame capacity buffer. Drop-Tail is the simplest queue among the others because it uses only one simple policy: it drops all the incoming packets after the buffer is full [33] [45]. Furthermore, the routers are connected to each other point-to-point style. Thus, the data packets can be dropped only in the queues of the router owing to many reasons. In the simulations, Drop-Tail queue has been used because the work process of the NEPTUNE routers is identical with the NEPTUNE data transmission devices being used today. The buffer size of the routers manages and controls the network traffic based on the packet drop rate of the queue. In order to evaluate a network system like NEPTUNE, the queue behaviors of the routers have a crucial impact on the efficiency of the system. Therefore, the rate of the number of dropped packets in the queues determines which scenario would be more efficient than the others which allows us to observe, test and compare the results based on chosen statistical analysis.

5.4 Performance Analysis

To observe the performance analysis of the two different scenarios, we ran two different simulations with different seeds in order to examine how the instruments have an impact on the differentiation of the network systems. Then we started performing a throughput performance test to the two pre-implemented scenarios of NEPTUNE. Throughput performance analysis can be calculated by the ratio of the total successfully received packets to the total generated packets. For the purpose of this statistical analysis, the main receiver is UVIC DMAS and the 'universal' generators are all of the instruments that are in the network.

And the other performance analysis is packet drop rate analysis. Packet drop rate analysis provides us with the information to ensure which data transmission device(s) drop(s) the highest number of packets. The results show that there are problematic router(s) in the network system; thus, computer scientists are able to develop specific operations to prevent data collision. To calculate the packet drop rate analysis, the number of dropped packets is calculated by OMNeT++. So, for each scenario, the different simulations are run with the different seeds based on which device(s) will be compared such as branching units or node stations.

5.5 Evaluation

Figure 5.1 illustrates that two NEPTUNE scenarios were run with different seeds to calculate the throughput performance analysis and the lines show us that the ratio of the packets that are successfully received by UVIC DMAS for current NEPTUNE system is significantly more than the NEPTUNE system after the installation. With the numbers, average ratio of the throughput of current NEPTUNE is approximately %.02 and the throughput of NEPTUNE after the installation is %.017. So Figure 5.1 states that after the installation in NEPTUNE network system, the data traffic will be %.003 less efficient than before.

Figure 5.2 displays the packet drop rate of two scenarios. In the Figure 5.2, NEPTUNE network after the installation in UVIC DMAS packet drop rate is approximately 6900 packets per run; however, the number of packets that are drooped in current NEPTUNE is around 5000 packets per run. Therefore, the additional devices causes a notable increase in the number of dropped packets per run.

Because NEPTUNE uses the hierarchy of SONET, there is no dropped packets in the branching units. When the buffer is full in the branching units, the packets that are potentially dropped is sent to another branching unit, bi-directionally. Thus, the data packets may be circulated through the available(be potentially transmitted with the least data collision) direction until the whole received data is sent to the desired destination.

In the evaluation process, the most open to comment part, is the analysis of the node stations. Because in the second scenario, a new node station was added in the Middle Valley region. Therefore, Figure 5.3 illustrates that the additional node station does effect the network traffic but the most affected region is Endeavour Ridge node station. The number of dropped packets in the current NEPTUNE is approximately 20 per run but in the second scenario the rate of the dropped packets is around 26. Thus, the node station in Endeavour Ridge may be the most problematic node station among the others in terms of the data collision.

The packet drop rate of the junction boxes is illustrated in Figure 5.4 and it displays the comparison the number of dropped packets rate between the current NEPTUNE and the NEPTUNE network after the installation. In this test, we chose the junction box which has the highest drop rate and junction box 11 had the highest number in 8 of 10 different seeds. However, junction box 8 and junction box 10 had the peaked rate among the other results.

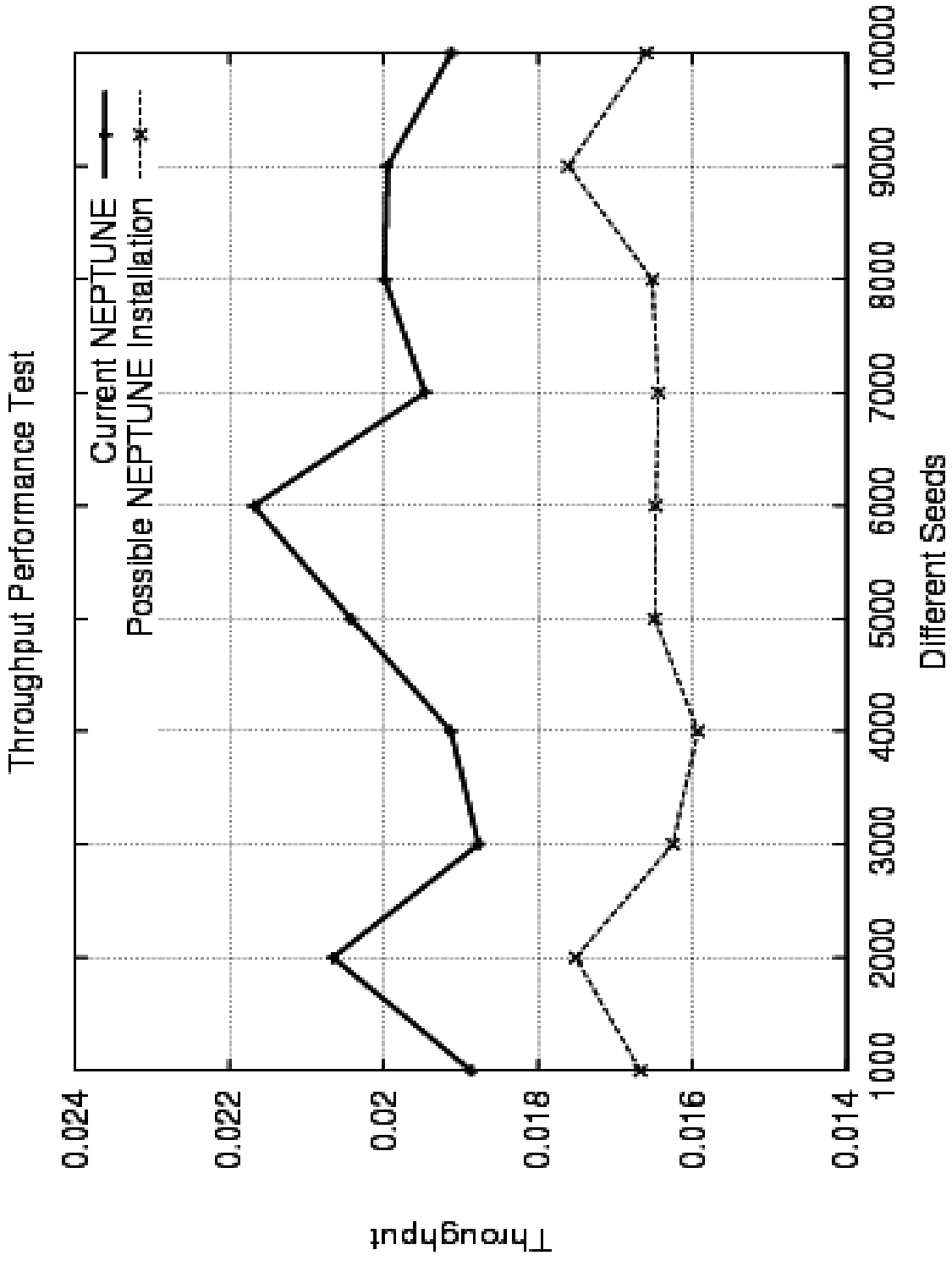

Figure 5.1: Comparison of Throughput Performance Test of Current NEPTUNE Network and NEPTUNE Network After The Installation

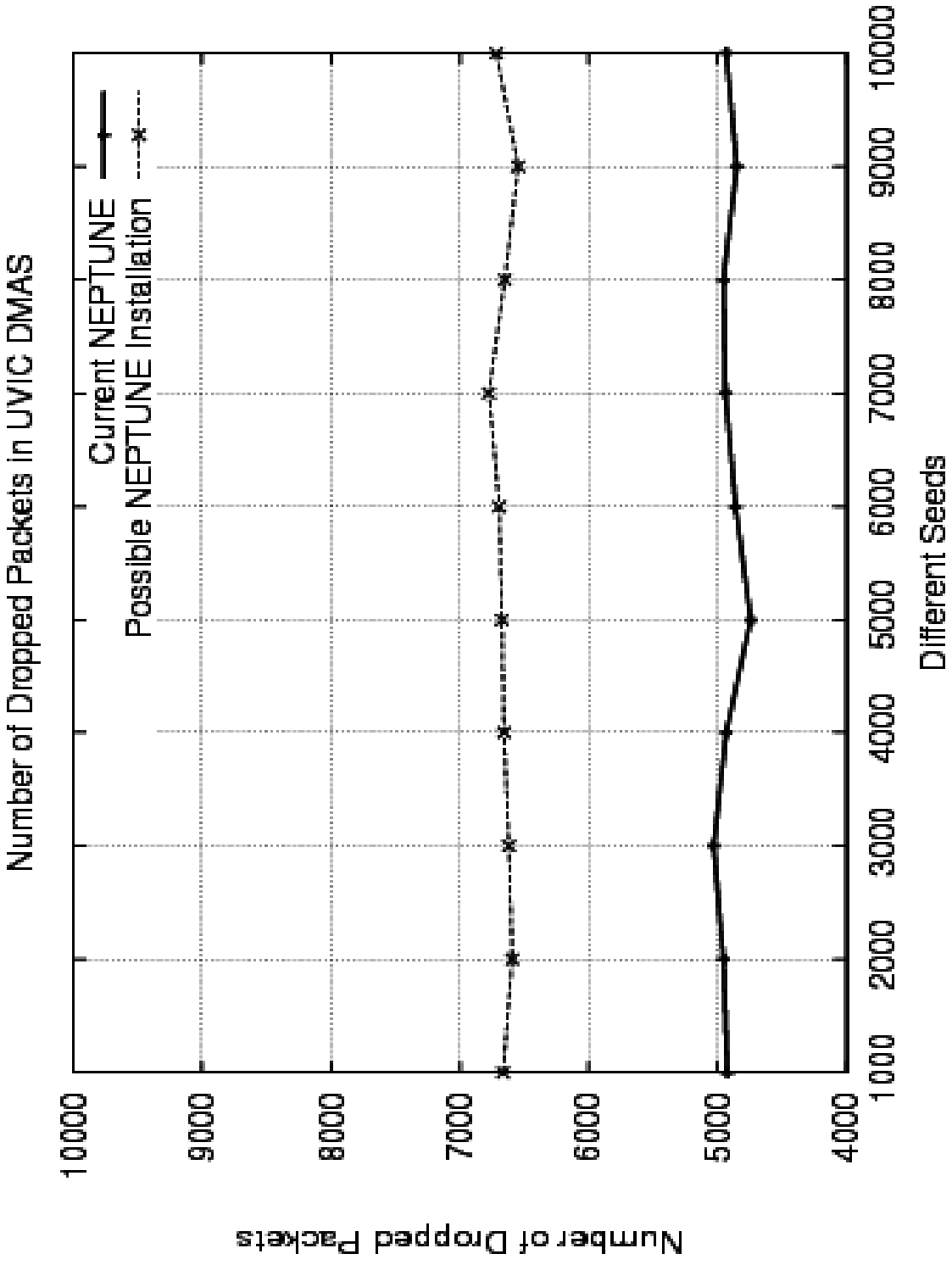

Figure 5.2: Comparison of Packet Drop Rate of UVIC DMAS Between Current NEPTUNE and NEPTUNE After The Installation

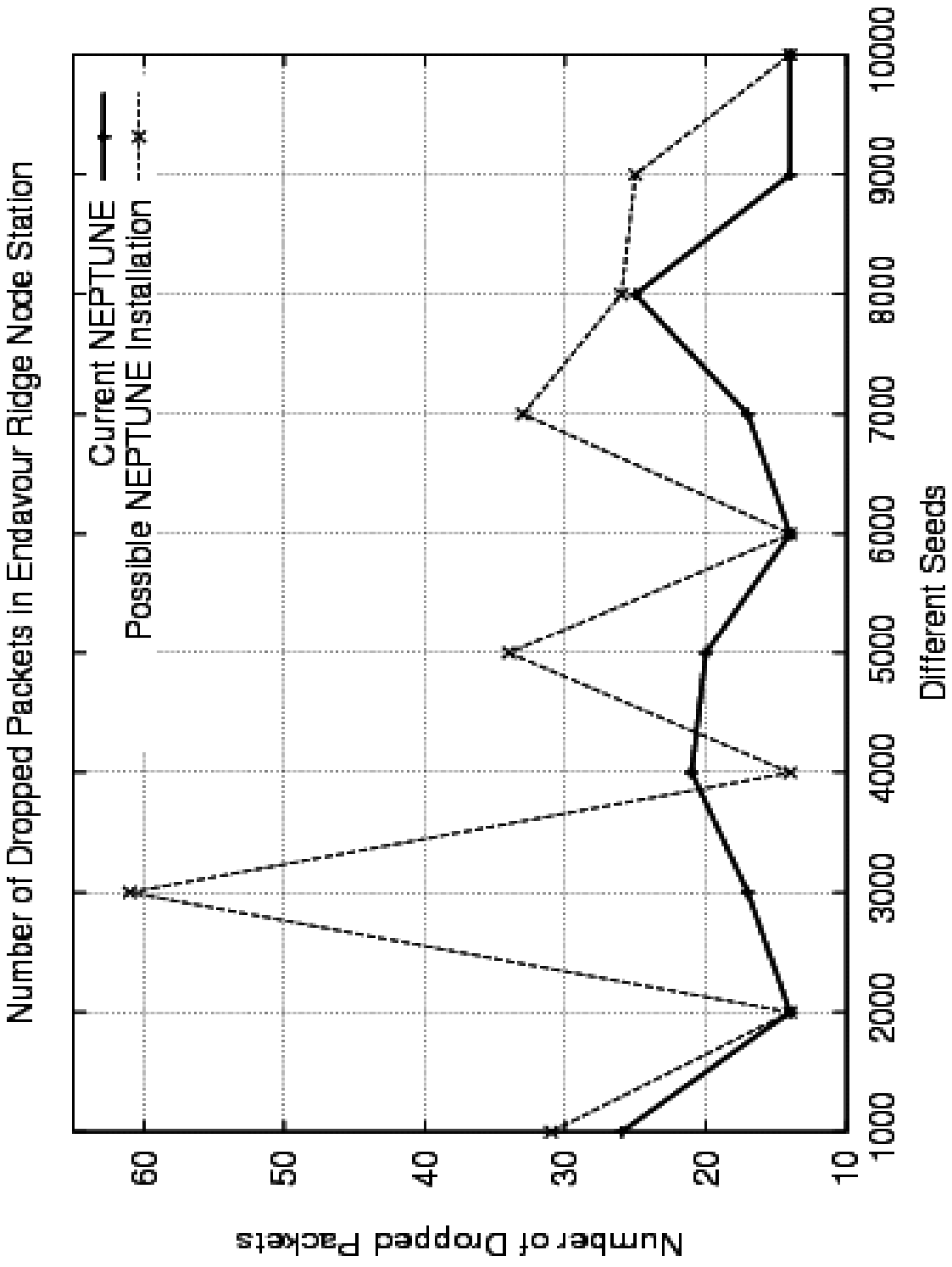

Figure 5.3: Comparison of Packet Drop Rate of Endeavour Ridge Node Station Between Current NEPTUNE and NEPTUNE After The Installation

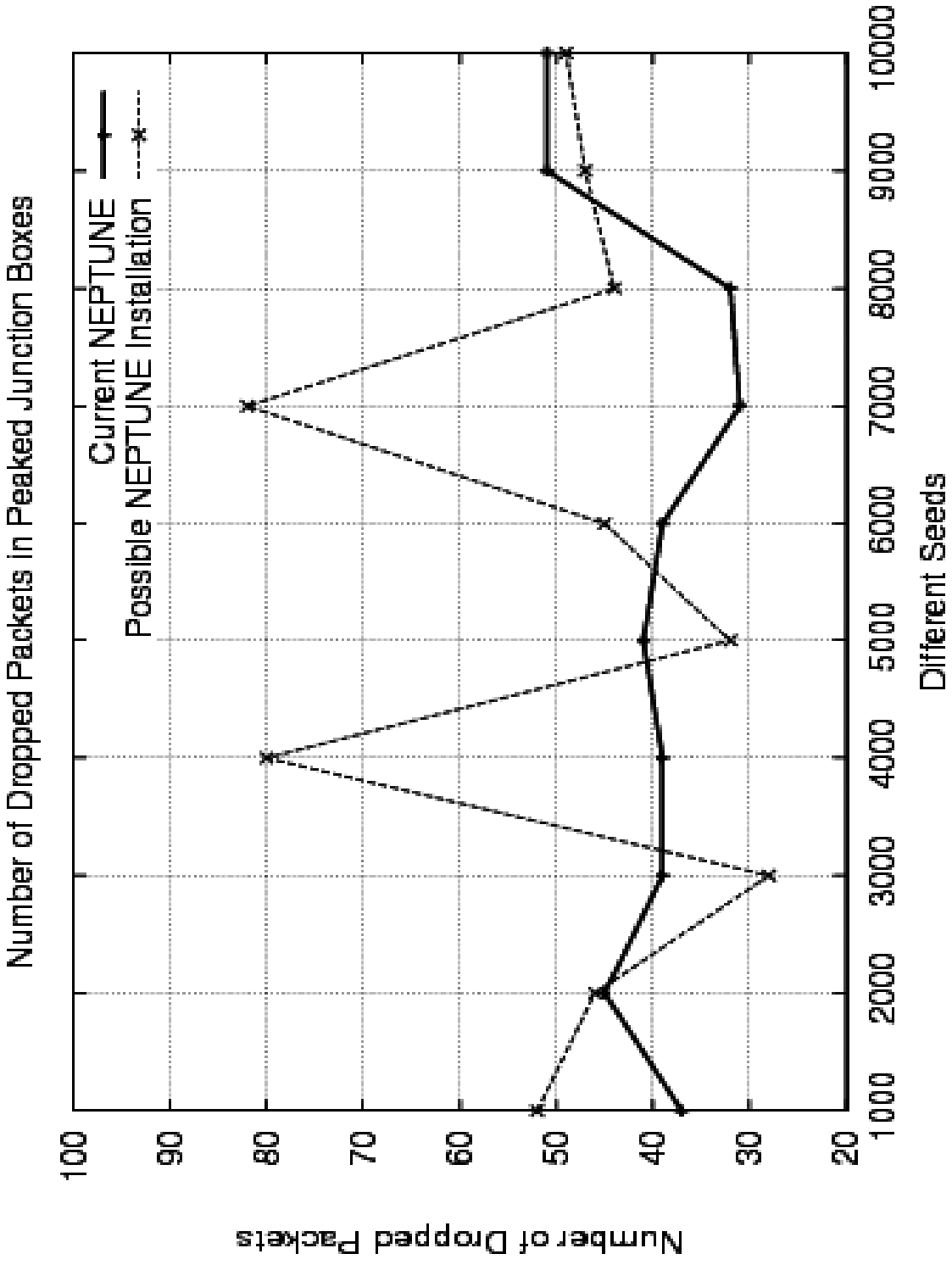

Figure 5.4: Comparison of Packet Drop Rate of The Junction Boxes Between Current NEPTUNE and NEPTUNE After The Installation

Performance Analysis of The Instruments based on used TCP and UDP Protocols

In this statistical analysis, we observe the instruments that are using UDP protocol—hydrophones and video cameras. The instruments that are using UDP protocol causes the highest data collision in the network traffic. Because these instruments generate the biggest sized data and send them continuously—without receiving any confirmation from the source in terms of the successful data packet delivery.

As mentioned above and previous chapters, in scenario 1, there are two hydrophones that are connected to the network and there are three hydrophones linked to the network in scenario 2. Figure 5.5 shows us that the additional hydrophone has the highest packet lose in the network. The number of packet drops of hydrophone 1 and hydrophone 2 in both scenarios are approximately similar. But hydrophone 3 has the peak number of packet lose as oppose to be expected the least packet lose. Because, the hydrophone 3 is linked to the new node station where is installed on the Middle Valley region. Furthermore, the additional node station has two junction boxes and 5 instruments; as compared to the other node station branches, the Middle Valley region has the minimum number of instruments among the other node station branches.

The Figures 5.6 and 5.7 illustrate the packet drop rate between five video cameras that are implemented in the current NEPTUNE network and one more video camera has been installed after the enhancement of the system. Two Figures display that after the installation, number of dropped packets are slightly increased with the additional camera. The additional junction boxes and the node station causes the increase of the packet loss.

The majority of the NEPTUNE instruments use TCP protocol during the data transmission process. The number of dropped packets of the instruments that use TCP protocol is significantly less than the instruments that use UDP protocol. The main reason is that the devices use TCP generate less sized data packets than the instruments use UDP protocol. Figure 5.8 displays that number of dropped packets of the instruments that use TCP protocol in the data transmission is slightly less than the values in NEPTUNE network after the installation. This slight difference does not really effect the performance of the whole network traffic but this does not mean that it may not effect if more instruments that use TCP would be added in advance.

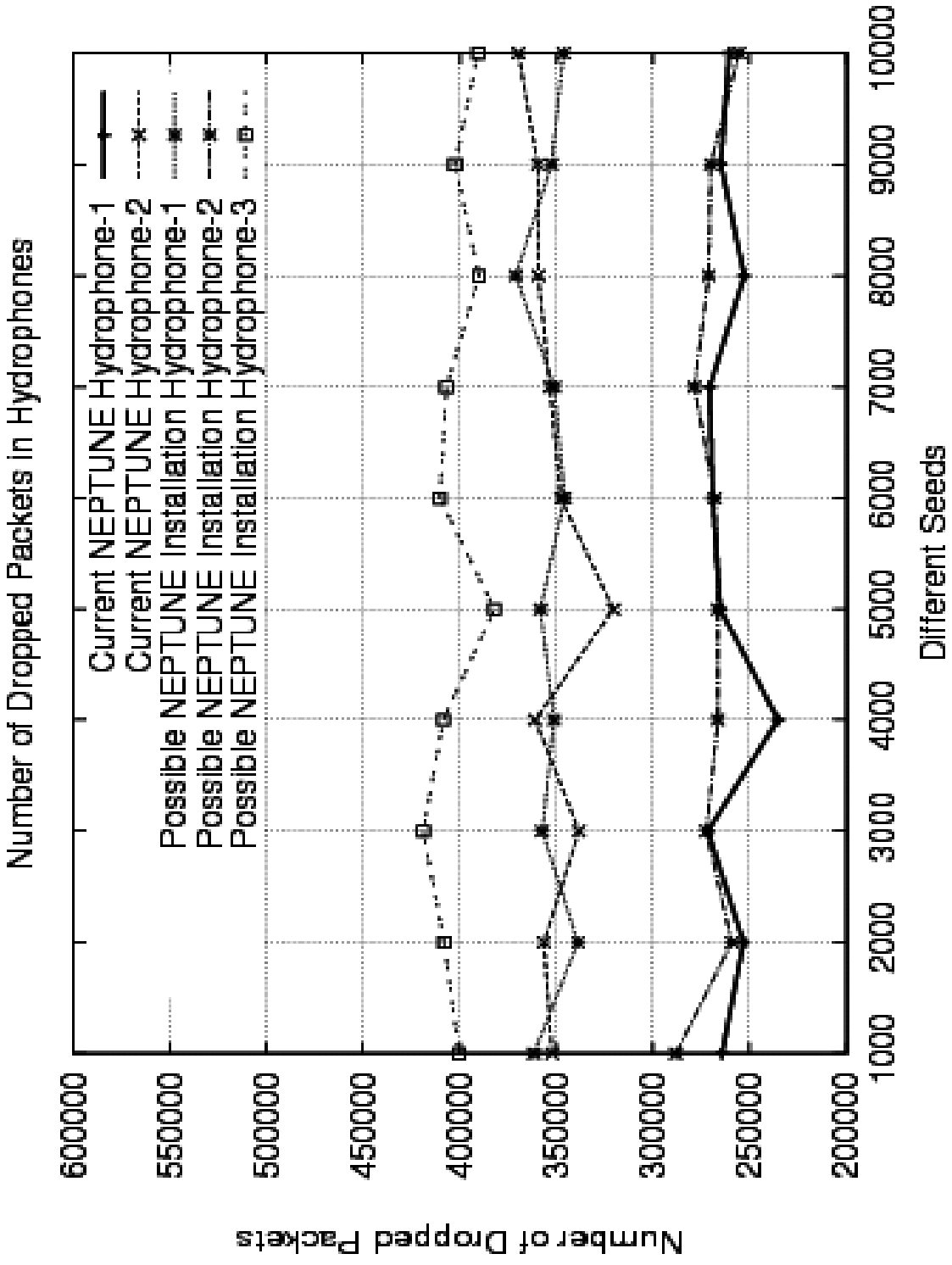

Figure 5.5: Comparison of Packet Drop Rate of The Hydrophones Between Current NEPTUNE and NEPTUNE After The Installation

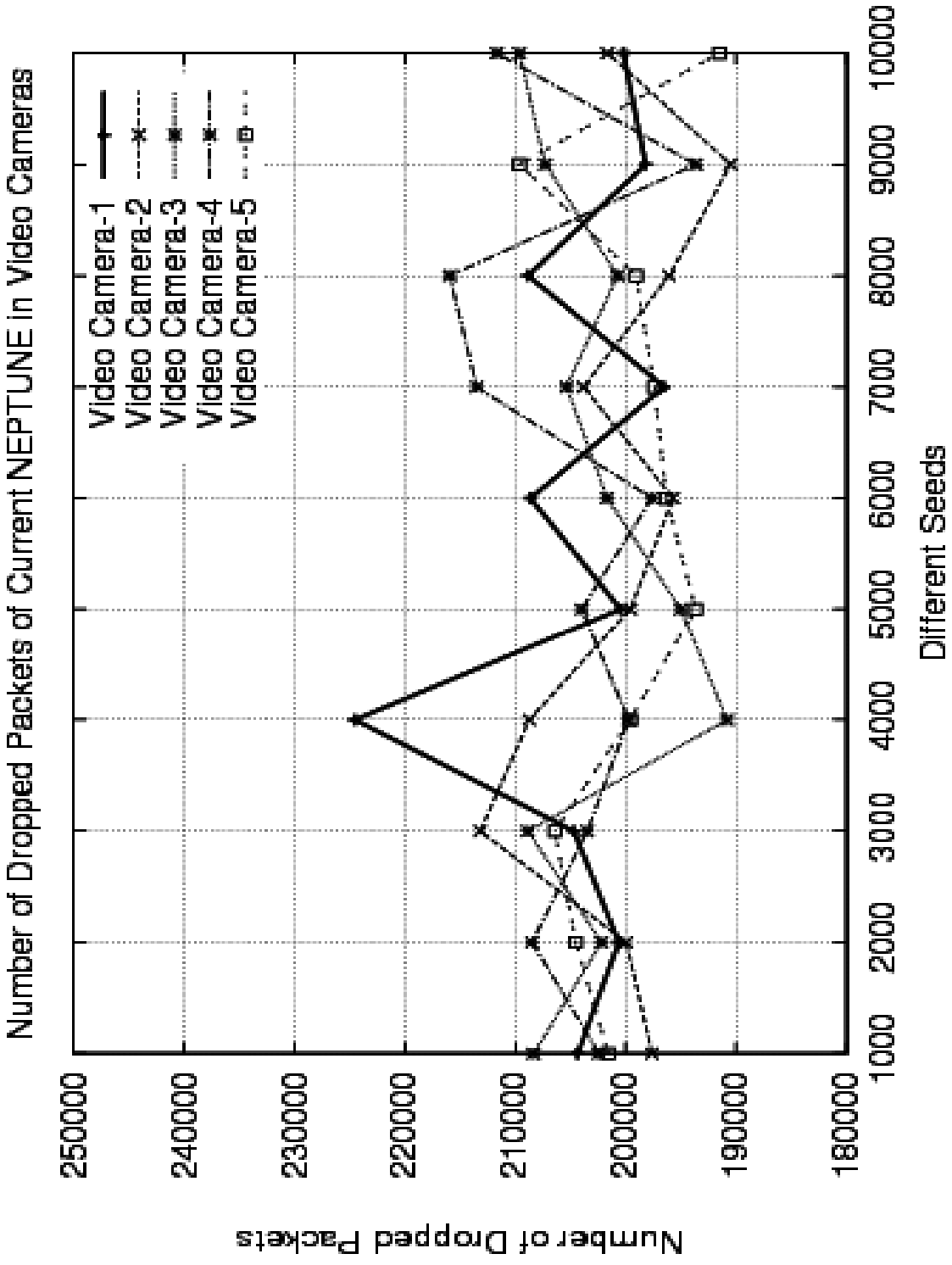

Figure 5.6: Comparison of Packet Drop Rate of The Video Cameras Between Current NEPTUNE and NEPTUNE After The Installation

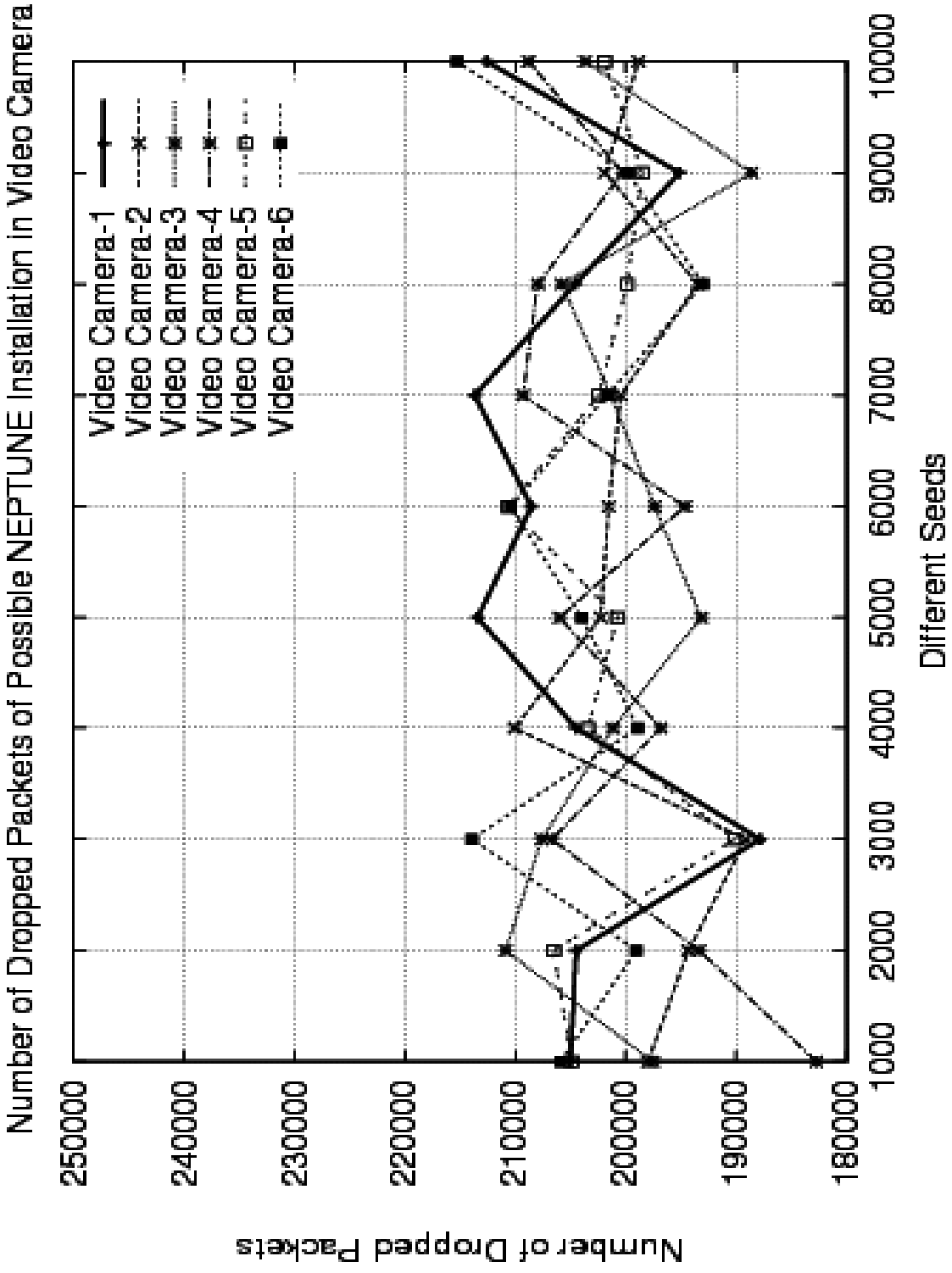

Figure 5.7: Comparison of Packet Drop Rate of The Video Cameras Between Current NEPTUNE and NEPTUNE After The Installation

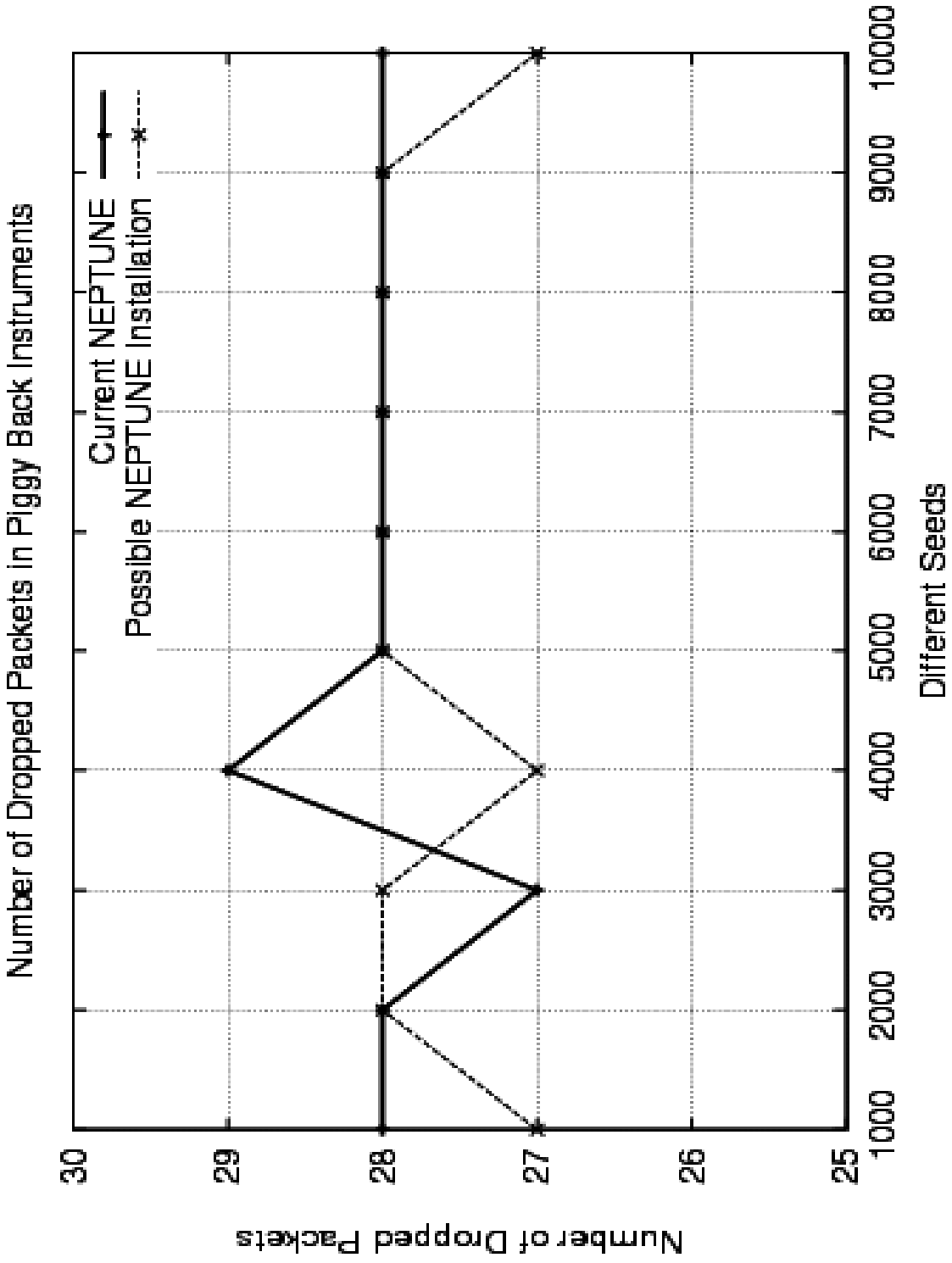

Figure 5.8: Comparison of The Peaked Packet Drop Rate of The TCP Instruments Between Current NEPTUNE and NEPTUNE After The Installation

5.6 Summary

In the evaluation of the performance analysis of both NEPTUNE scenarios, we observe that the additional branch installed on the Middle Valley region with one node station, two junction boxes and 5 instruments (including one hydrophone and one video camera that use UDP) have a crucial impact on the network traffic. The Figures show that all the number of dropped packets in the queues of the current network are higher than the values of the network after the installation. In this section, we performed a throughput performance test and a packet drop rate test to observe the network behaviors of the two different NEPTUNE simulations.

Chapter 6

Discussion and Future Work

By definition, models are approximations of real systems. That means we suffer information loss, and as a result, do not capture exact system dynamics. This is definitely true in the case of our simulation. NEPTSim has value at the level of exploring high-level relationships associated with corresponding changes in the configuration of the system. Though we believe this model is accurate enough to provide important information to the NEPTUNE team in terms of statistically sound results, future work in calibrating the system by experts will ultimately determine its value. There are many things not captured, such as subtle features associated with specific instruments or the database itself, which do not factor into this simulation.

Table 6.1 shows what we have captured in the simulation, and what we have not. Bi-Directional line switched ring of SONET topology is easy to implement in OMNeT++ because the simulation provides in and out gates that can be easily implement in the NED file due to the fact that OMNeT++ provides an excellent GUI in order to implement even complex networks by dragging and dropping the data channels that are already built in several OMNeT++ packages. However, the data transmission process and the network traffic is much more simple than we thought. Because the data that are generated by the instruments are most likely identical with the data that were generated beforehand. Owing to NEPTUNE Canada most likely is an underwater surveillance system, the data that are generated are transmitted to be stored in the main receiver. Thus, the data loss is not a vital problem according to the main duty of NEPTUNE Canada. SONET topology is used in NEPTUNE Canada because SONET is the most safest topology in terms of the data loss. All the data channels that are used in NEPTUNE Canada are successfully simulated in NEPTSim.

There are 38 different types of instruments linked to the network that have different roles in NEPTUNE. However, NEPTSim only cares network activities in the network. Thus, even though there are 38 different instruments that are linked to the network, according to the size of data and use of transmission protocols, to simulate NEPTUNE in NEPTSim does not need significant effort. Because the sizes of data that are generated by the instruments take the parameters that are generated by OMNeT++'s random number generator. This supports the reliability of NEPTSim in essence of tuning the parameters independently. For the data transmission protocol perspective, there are two types of transmission protocols are used by instruments: TCP and UDP. In the instrument calibration, it is easy to implement either TCP or UDP applications for each instruments in OMNeT++. Because INET package provides nodes that are perfectly fit for the NEPTUNE instruments as well as their network layer are already implemented under the characteristics of INET node agents. Thus, the user only tunes the application that is related to purposed instrument that will be implemented and enters the parameters in.

In NEPTUNE Canada network, branching units are implemented as switches and their role in the backhaul is to link the other branching units that are next to them bi-directionally. Thus, each branching units has three *in* and three *out* gates: two of them are linked to the closest branching units and the other one connects it to the node station where its region is. In NEPTSim, the branching units are considered as router due to having similar features and roles in the network. There is no obstacle in defining and implementing branching units in NEPTSim in terms of providing pre-determined data channels that could be modified if needed.

Node stations in NEPTUNE have much more characteristics than the node stations that are implemented in NEPTSim. In real system, they are the power generators of the whole backbone. Also they have a self control mechanism that makes self decision under several circumstances. But in NEPTSim, node stations have the same role as branching units: data transmitting. Furthermore, they are implemented as routers in NEPTSim and that are connected to the branching unit and junction boxes that are located in the same region of the branching unit. In NEPTSim, node stations mimic only the data transmitting ability of the node stations of the real system.

Junction boxes in NEPTUNE Canada are implemented as for both data transmission operation and data generation process. Which are implemented as the same purpose in NEPTSim. Junction boxes like in the real system are linked to both instruments and node stations as well as they can be linked to another junction boxes

in purpose. They are implemented as routers both in real system and in NEPTSim. Junction boxes are successfully mimicked in NEPTSim based on their role in the real system.

Shore Station in Port Alberni has a vital role that can be described(or exemplify) as a bottleneck of the network in both NEPTUNE and NEPTSim. Shore Station acts like a filter of the whole network that sucks up the whole data that are collected by branching units and transmits the collected data to the main receiver—UVIC DMAS. In NEPTSim, Shore Station has a same role as in NEPTUNE, the only difference is that Shore Station is also connected to the Data Center in Saskatchewan as a fault tolerance channel. In NEPTSim, we did not implement the fault tolerance channel because the Internet is used as the data channel. And creating the Internet simulator would be such challenging experiment for this research but we could not trust that the results may be deceptive due to unlimited reasons related to the Internet.

UVIC DMAS is the main collector in NEPTUNE as it is in NEPTSim. However, in the real system, it has a database that controls and queries the data that were collected via the network. Furthermore, it is also linked to the Data Center in Saskatchewan for being used as a fault tolerance channel which is not implemented in NEPTSim due to the facts that are the same as the Shore Station has. We could not mimic the calibration of the database of DMAS UVIC due to the lack of information that could not provided by NEPTUNE team. Because the data that we need is the classified data which might be used by military. DMAS UVIC has a server in it in the real system. However, in NEPTSim, DMAS UVIC was implemented as a standard INET node that has UDP and TCP sinks on it.

Even though we tried to mimic NEPTUNE Canada network system based on the network components of it, we cannot simulate the natural conditions such as the depth or the ocean floor temperature that might effect the efficiency of the data transmission in NEPTSim. Because OMNeT++ does not support any simulation model that can simulate natural factors. We can only simulate the failure of the components during the run which NEPTSim is capable of obtaining such results if they are needed as long as the topology is implemented appropriately.

Future work could possibly consider integrating emulation with simulation to address the discrepancies that exist between NEPTSim and NEPTUNE. This integration technique is not uncommon in modeling approaches, and enhances the ability to predict and reproduce accurate system behavior [19].

In general, simulation models are good for feedback and testing what-if scenarios

within a restricted set of circumstances. Typically, the system is abstracted to focus on particular elements in question. Emulation is different, in that it typically contains some functional part of the real system, sometimes coupled with other simulated elements. As NEPTUNE uses standard network protocols, it may also benefit from emulation in addition to the "pure" simulation model of NEPTSim.

NEPTUNE Artefact	**NEPTSim Implementation**
1) Bi-Directional Line Switched Ring of SONET	-Easy to implement in OMNeT++ -The data flow in the real system is much more simple than we thought -Has in and out gates to provide bi-directionality -Identical with the real system
2) Instruments	-Implemented 38 types as standardhost -Use TCP and UDP transmission protocols -Parameters are generated by RNG to stimulate reliability -Network layers are built in INET package
3) Junction Boxes	-Implemented as routers -Connected to instruments and node stations -Can be linked to other junction boxes -Can generate data
4) Node Stations	-Implemented as routers -Connected to the branching unit and junction boxes -Only transmits the data -Cannot mimic all the features
5) Branching Units	-Implemented as router -Connected to branching units and Shore Station -Formed SONET network -Have no data collison in the ring
6) Shore Station	-Bottleneck of the network -Implemented as a router -Acts like a filter -Connected to 2 branching units and the main receiver—UVIC DMAS
7) UVIC DMAS	-The main receiver -Has TCP and UDP application sinks -Implemented as a standardhost -Couldn't mimic its all features such as database

Table 6.1: Correlations Between NEPTUNE and NEPTSim

Chapter 7

Conclusions

In this thesis, we demonstrated one possible simulation for NEPTUNE Canada using OMNeT++. We believe this approach has benefits of ease-of-use and portability. Because of a broad range of environmental extremes under the ocean, simulating the system could be the cheapest and safest way to observe the network behavior. In other words, a simulation environment does bring the real-time systems into the virtual arena. This could have a vital importance for such real-time systems like NEPTUNE.

Simulators such as ours can provide elaborate reports to be analyzed scrutinizingly before the future operations because desired scenarios can be evaluated in the simulator without any physical effort. NEPTSim can be implemented and tested on the academic/non-commercial version of OMNeT++, freely downloaded from OMNeT++ official website [25]. Consequently, building NEPTSim may assist financially and save considerable time for cabled ocean real-time systems.

NEPTSim can work with a variety of configurations of instruments, junction boxes, node stations, branching units and traffic patterns. Our study reveals that, although building the simulator in OMNeT++ has many advantages—such as ease of tuning and calibration, capturing sufficient details regarding the working behavior of the actual NEPTUNE environment is still challenging. Relative to other simulators, OMNeT++ is an appropriate simulation software based on its functions for the requirements of NEPTUNE Canada.

In the evaluation chapter, two different scenarios were performed to investigate the behavior of the network traffic in between the current system and the system that after adding an additional branch Middle Valley region node station. Throughput performance test and packet drop rate tests were performed using different seeds in

two different scenarios. NEPTSim outputs statistical analysis using the parameters that are given by NEPTUNE team and the results show that additional node station could cause a significant effect on the backhaul in terms of the packet loss. Although NEPTUNE uses SONET topology that provides minimum data collision, additional instruments, especially new hydrophone and video camera, narrows the data flow on the spur cable which stimulates the data collision on the backbone. Thus, less data packets are transmitted to UVIC DMAS where the main receiver is. Therefore, NEPTSim shows us the ways in which an additional node station branch in Middle Valley could significantly impact data transmission on the network.

In this thesis, NEPTSim illustrates how NEPTUNE s multi- node ocean cabled observatory network system can be simulated using the OMNeT++ simulation platform. The OMNeT++ simulation platform is potentially a useful tool to simulate important elements of NEPTUNE Canada due to its ease-of-use, portability and configurability.

Bibliography

[1] Simulator of an adhoc network originally developed using omnet++ v2.2 [online]. *Available: http://www.omnetpp.org/omnetpp/docdetails/2118adhocsim.*

[2] Neptune canada: Transforming ocean science, September 2010.

[3] Roberto Aringhieri and Mauro Dell Amico. Comparing metaheuristic algorithms for sonet network design problems. *Journal of Heuristics, 11: 35-57*, 2005.

[4] Jerry Banks, John S. Carson, Barry L. Nelson, and David M. Nicol. *Discrete Event System Simulation, 5th Edition.* Prentice Hall, 2009.

[5] Balakrishnan Chandrasekaran. Survey of network traffic models.

[6] Peter Fairley. A tale of two neptunes. *IEEE Spectrum*, 49:11–12, February, 2012.

[7] Paolo Favali, Roland Person, Chris R. Barnes, Yoshiyuki Kaneda, John R. Delaney, and Shu-Kun Hsu. Seafloor observatory science. *OceanOBS*, 2009.

[8] Open-Source Communication Networks Simulation Package for the OMNeT++ Simulation Environment [Online]. Available: http://inet.omnetpp.org/.

[9] Simulation Framework for Wireless and Mobile Networks using the OMNeT++ Simulation Engine. [Online]. Available: http://mixim.sourceforge.net/.

[10] Yusuke Fujita, Yoshihiko Nakamura, and Zvi Shiller. Dual dijkstra search for paths with different topologies. *Proceedings of IEEE Taipei, Taiwan, September 14-19*, 2003.

[11] I. S. Hammoodi, B. G. Stewart, A. Kocian, and S. G. McMeekin. A comprehensive performance study of opnet modeler for zigbee wireless sensor networks. *Third International Conference on Next Generation Mobile Applications, Services and Technologies*, 2009.

[12] Java in Simulation Time / Scalable Wireless Ad hoc Network Simulator [Online]. Available: http://jist.ece.cornell.edu/.

[13] DML Model Configuration Database Java Software Library Containing the Fast Parallel Discrete-Event Simulation SSF Kernel and Any Other Included Software Developed by Renesys Corporation [Online]. Available: http://www.ssfnet.org/download/license.html.

[14] M. Junaid, Arshad Qureshi, and M. Saleem. Simulation and visualization of transmission control protocol's(tcp) flow-control and multi-home options. *Proceedings of International Bhurban Conference on Applied Sciences and Technology*, January 8–11, 2007.

[15] Stephan T. Lentz. Neptune canada communications network. *MTS*, 2007.

[16] Michael Liljenstam, David M., Nicol Vincent, H. Berk, and Robert S. Gray. Simulating realistic network worm traffic for worm warning system design and testing. *WORM*, 2003.

[17] Gilberto Flores Lucio, Marcos Paredes-Farrera, Emmanuel Jammeh, Martin Fleury, and Martin J. Reed. Opnet modeler and ns-2: Comparing the accuracy of network simulators for packet-level analysis using a network testbed.

[18] Dan C. Marinescu, Chen Yu, and Gabriela M. Marinescu. A secure self-organizing sensor network. *Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops*, 2008.

[19] Ian McGregor, E. Ycesan, C.H. Chen, J.L. Snowdon, and J. M. Charnes. The relationship between simulation and emulation. *Proceedings of the Winter Simulation Conference*, 2002.

[20] OMNET++. *OMNET++Discrete Event Simulation System Version 4.2 User Manual*, 2012.

[21] Discrete-Event Network Simulator [Online]. Available: http://www.nsnam.org/.

[22] Discrete-Time Process-Oriented Simulation [Online]. Available: www.j-sim.zcu.cz.

[23] NEPTUNE Canada [Online]. Available: http://neptunecanada.ca.

[24] Network Simulator [Online]. Available: http://www.isi.edu/nsnam/ns/.

[25] OMNET++ Network Simulation Framework [Online]. Available: http://www.omnetpp.org/.

[26] Scalable Simulation Framework Network Models [Online]. Available: www.ssfnet.org.

[27] Simulation Platform [Online]. Available: http://www.scalable-networks.com/content/products/qualnet.

[28] SONET Synchronous Optical Networking [Online]. Available: http://www.technologyuk.net/.

[29] P. Phibbs and S. Lentz. Implementation of neptune canada backbone. *SIMU-Tools*, 2008.

[30] Peter Phibbs, Steve Mihaly, and Rob Jones. System engineering at the edge of a cabled ocean observatory. *IEEE*, 2009.

[31] Conservatively Synchronized Parallel Simulator [Online]. Process-Oriented. Available: http://users.cis.fiu.edu/ liux/research/projects/dassf/index.html.

[32] Alfonso Ariza Quintana, Eduardo Casilari, and Alicia Trivio. Implementation of manet routing protocols on omnet++. 2008.

[33] K.K. Ramakrishnan, S. Floyd, and D. Black. The addition of explicit congestion notification (ecn) to ip. *RFC 3168, Proposed Standard*, 2001.

[34] Karsten M. Reineck. Evaluation and comparison of network simulation tools. August, 2008.

[35] Munam Ali Shah, Ghazanfar Ali Safdar, Carsten Maple, and Khurram Sardar. Network performance optimization: A case study of enterprise network simulated in opnet. *Frontiers of Information Technology, IEEE*, 2011.

[36] Body Area Networks (BAN) Simulator for Wireless Sensor Networks (WSN) and Generally Networks of Low-Power Embedded Devices. [Online]. Available: http://castalia.npc.nicta.com.au/.

[37] Ahmed Sobeih, Wei-Peng Chen, Jennifer C. Hou, Lu-Chuan Kung, Ning Li, Hyuk Lim, Hung-Ying Tyan, , and Honghai Zhang. J-sim: A simulation environment for wireless sensor networks. *IEEE*, 2005.

[38] Andrew S. Tanenbaum. *Modern Operating Systems, 3rd Edition.* Prentice Hall, 2007.

[39] Andras Varga and Rudolf Hornig. An overview of omnet++ simulation environment. *SIMUTools*, 2008.

[40] Elias Weingartner, Hendrik vom Lehn, and Klaus Wehrle. A perfomance comparison of recent network simulators. *IEEE ICC*, 2009.

[41] A. M. Woodroffe, S. W. Pridie, and G. Druce. The neptune canada junction box - interfacing science instruments to sub-sea cabled observatories. *IEEE*, 2008.

[42] Xudong Wu and Ioanis Nikolaidis. On the average of lifetime and rtt classification schemes. *IEEE*, 2003.

[43] Xiaodong Xian, Weiren Shi, and He Huang. Comparison of omnet++ and other simulator for wsn simulation. *IEEE*, 2008.

[44] Sunghyun Yoon and Young Boo Kim. A design of network simulation environment using ssfnet. *First International Conference on Advances in System Simulation*, 2009.

[45] Zhongwei Zhang, David Lai, and Shan Suthaharan. A unified approach to ameliorate active queue management of network routers. *Proceedings of International Symposium on Intelligent Signal Processing and Communication Systems*, December 13–16, 2005.

Appendix A

Appendix

A.1 Coding NEPTSim

A.1.1 Scenario-I

Network Description File (NED)

```
package inet.examples.NeptuneCANADAscI;
import inet.networklayer.autorouting.FlatNetworkConfigurator;
import inet.nodes.inet.Router;
import inet.nodes.inet.StandardHost;
import inet.world.ChannelInstaller;
import inet.world.NAMTrace;
import ned.DatarateChannel;
network NeptuneCANADAScI
parameters:
@display("bgb=,,dodgerBlue,blue");
types:
channel C1 extends DatarateChannel
datarate = 10Gbps;
delay = 0.1us;
channel C2 extends DatarateChannel
datarate = 4Gbps;
delay = 0.1us;
channel C3 extends DatarateChannel
```

datarate = 1Gbps;
delay = 0.1us;
channel C4 extends DatarateChannel
datarate = 100Mbps;
delay = 0.1us;
submodules:
//———————————//

UVIC: StandardHost {
@display(”p=1088,33;i=misc/globe”);
}
Port_Alberni_Station: Router {
@display(”p=1088,238;i=misc/building”);
}
//Branching Units–There are 6 Branching Units that transmit the data coming from Node agents and connected through Shore Station
BranchingUnit1: Router {
@display(”p=956,238;i=old/xconn”);
}
BranchingUnit2: Router {
@display(”p=730,162;i=old/xconn”);
}
BranchingUnit3: Router {
@display(”p=433,162;i=old/xconn”);
}
BranchingUnit4: Router {
@display(”p=251,256;i=old/xconn”);
}
BranchingUnit5: Router {
@display(”p=423,336;i=old/xconn”);
}
BranchingUnit6: Router {
@display(”p=816,336;i=old/xconn”);
}
// Node Agents–There are 6 Node agents and one of them which is Middle Valley is

empty and the others are connected to Junction Boxes
Folger_Passage: Router {
@display("p=948,179;i=misc/house");
}
ODP_889: Router {
@display("p=718,107;i=misc/house");
}
Middle_Valley: Router {
@display("p=342,68;i=misc/cloud2");
}
Endavour_Ridge: Router {
@display("p=135,256;i=misc/house");
}
ODP_1027: Router {
@display("p=324,386;i=misc/house");
}
Barkley_Canyon: Router {
@display("p=822,386;i=misc/house");
}
//Junction Boxes–There are 11 Junction Boxes that are connected to 5 Nodes one by one.
JunctionBox1: Router {
@display("p=979,107;i=device/router");
}
JunctionBox2: Router {
@display("p=901,107;i=device/router");
}
JunctionBox3: Router {
@display("p=631,68;i=device/router");
}
JunctionBox4: Router {
@display("p=79,336;i=device/router");
}
JunctionBox5: Router {
@display("p=42,256;i=device/router");

```
}
JunctionBox6: Router {
@display("p=239,107;i=device/router");
}
JunctionBox7: Router {
@display("p=227,432;i=device/router");
}
JunctionBox8: Router {
@display("p=1005,358;i=device/router");
}
JunctionBox9: Router {
@display("p=995,417;i=device/router");
}
JunctionBox10: Router {
@display("p=909,417;i=device/router");
}
JunctionBox11: Router {
@display("p=1068,417;i=device/router");
}
//————————————-//
//Instruments–There are 38 Different Type of Instruments which are connected to Junction Boxes and they generate data which is sent throught UVIC.
CTD1: Router;
Hydrophone1: StandardHost;
Hydrophone2: StandardHost;
Water_Sampler: StandardHost;
BPR1: StandardHost;
BPR2: StandardHost {
@display("p=577,126");
}
BPR3: StandardHost {
@display("p=187,214");
}
BPR4: StandardHost {
@display("p=108,201");
```

```
}
BPR5: StandardHost;
BPR6: StandardHost;
BPR7: StandardHost;
Current_Meter1: StandardHost;
Current_Meter2: StandardHost {
@display("p=650,126");
}
Current_Meter3: StandardHost;
Current_Meter4: StandardHost;
Current_Meter5: StandardHost;
Current_Meter6: StandardHost {
@display("p=324,167");
}
Current_Meter7: StandardHost;
Current_Meter8: StandardHost;
Current_Meter9: StandardHost {
@display("p=108,162");
}
Current_Meter10: StandardHost;
Current_Meter11: StandardHost;
Current_Meter12: StandardHost;
Accelerometer1: StandardHost {
@display("p=781,82");
}
Accelerometer2: StandardHost;
Accelerometer3: StandardHost;
ADCP300KHz1: StandardHost;
ADCP300KHz2: StandardHost;
ADCP150KHz: StandardHost;
ADCP600KHz1: StandardHost;
ADCP600KHz2: StandardHost;
ADCP75KHz1: StandardHost;
ADCP75KHz2: StandardHost;
ADCP2MHz1: StandardHost;
```

```
ADCP2MHz2: StandardHost;
ADCP2MHz3: StandardHost;
Adapter1: StandardHost;
Adapter2: StandardHost {
@display("p=289,68");
}
Adapter3: StandardHost {
@display("p=239,183");
}
Adapter4: StandardHost {
@display("p=289,179");
}
Adapter5: StandardHost {
@display("p=236,75");
}
Broadband_Seismometer1: StandardHost {
@display("p=590,183");
}
Broadband_Seismometer2: StandardHost;
Broadband_Seismometer3: StandardHost;
Borehole_Temperature: StandardHost;
Compass: StandardHost;
CTD2: StandardHost {
@display("p=538,94");
}
CTD3: StandardHost;
CTD4: StandardHost;
CTD5: StandardHost;
CTD6: StandardHost;
CTD7: StandardHost {
@display("p=187,82");
}
CTD8: StandardHost {
@display("p=148,94");
}
```

```
CTD9: StandardHost;
CTD10: StandardHost {
@display("p=69,179");
}
CTD11: StandardHost;
CTD12: StandardHost;
CTD13: StandardHost;
CORK: StandardHost;
Bioflouling: StandardHost {
@display("p=130,196");
}
Flourometer1: StandardHost;
Flourometer2: StandardHost;
Flourometer3: StandardHost;
BARS: StandardHost;
Chemini: StandardHost;
EchoSounder: StandardHost;
PanTilt_Lights1: StandardHost {
@display("p=324,126");
}
PanTilt_Lights2: StandardHost;
PanTilt_Lights3: StandardHost;
PanTilt_Lights4: StandardHost;
Temperature_Sensor1: StandardHost;
Temperature_Sensor2: StandardHost;
Optode1: StandardHost;
Optode2: StandardHost;
Video_Camera1: StandardHost {
@display("p=97,102");
}
Video_Camera2: StandardHost;
Video_Camera3: StandardHost;
Video_Camera4: StandardHost;
Video_Camera5: StandardHost;
Piezometer: StandardHost;
```

```
Pressure_Gauge1: StandardHost {
@display("p=707,20");
}
Pressure_Gauge2: StandardHost;
Light_Sensor: StandardHost;
Sediment_Profiler: StandardHost;
Sediment_Trap: StandardHost;
Short_Range_Rotary_Sonar: StandardHost;
Turbidity_Meter: StandardHost;
Vehicle1: StandardHost;
Vehicle2: StandardHost;
CSEM1: StandardHost {
@display("p=538,51");
}
CSEM2: StandardHost {
@display("p=583,20");
}
COVIS: StandardHost;
Short_Period_Seismometer1: StandardHost;
Short_Period_Seismometer2: StandardHost {
@display("p=62,33");
}
Methane_Sensor: StandardHost;
Water_Sampler: StandardHost {
@display("p=135,303");
}
channelInstaller: ChannelInstaller {
parameters:
channelClass = "ThruputMeteringChannel";
channelAttrs = "format=u";
@display("p=98,50");
}
flatNetworkConfigurator: FlatNetworkConfigurator {
@display("p=301,20");
}
```

nam: NAMTrace {
parameters:
@display(”p=260,50”);
}
connections:
UVIC.pppg++ <–> C1 <–> Port_Alberni_Station.pppg++;
Port_Alberni_Station.pppg++ <–> C1 <–> BranchingUnit1.pppg++;
//Branching Unit connections
BranchingUnit1.pppg++ <–> C2 <–> BranchingUnit2.pppg++;
BranchingUnit2.pppg++ <–> C2 <–> BranchingUnit3.pppg++;
BranchingUnit3.pppg++ <–> C2 <–> BranchingUnit4.pppg++;
BranchingUnit4.pppg++ <–> C2 <–> BranchingUnit5.pppg++;
BranchingUnit5.pppg++ <–> C2 <–> BranchingUnit6.pppg++;
BranchingUnit6.pppg++ <–> C2 <–> BranchingUnit1.pppg++;
//Node Agent connections
Folger_Passage.pppg++ <–> C3 <–> BranchingUnit1.pppg++;
ODP_889.pppg++ <–> C3 <–> BranchingUnit2.pppg++;
Middle_Valley.pppg++ <–> C3 <–> BranchingUnit3.pppg++; //Empty Node Agent!!!
Endavour_Ridge.pppg++ <–> C3 <–> BranchingUnit4.pppg++;
ODP_1027.pppg++ <–> C3 <–> BranchingUnit5.pppg++;
Barkley_Canyon.pppg++ <–> C3 <–> BranchingUnit6.pppg++;
//Junction Box connection –> in here Junction Box8 is connected to Junction boxes 9,10 and 11.
JunctionBox1.pppg++ <–> C3 <–> Folger_Passage.pppg++;
JunctionBox2.pppg++ <–> C3 <–> Folger_Passage.pppg++;
JunctionBox3.pppg++ <–> C3 <–> ODP_889.pppg++;
//There is no instrument connections to Middle Valley Node!!!
JunctionBox4.pppg++ <–> C3 <–> Endavour_Ridge.pppg++;
JunctionBox5.pppg++ <–> C3 <–> Endavour_Ridge.pppg++;
JunctionBox6.pppg++ <–> C3 <–> Endavour_Ridge.pppg++;
JunctionBox7.pppg++ <–> C3 <–> ODP_1027.pppg++;
JunctionBox8.pppg++ <–> C3 <–> Barkley_Canyon.pppg++;
//————————————————————-//
JunctionBox8.pppg++ <–> C4 <–> JunctionBox9.pppg++;
JunctionBox8.pppg++ <–> C4 <–> JunctionBox10.pppg++;

JunctionBox8.pppg++ <–> C4 <–> JunctionBox11.pppg++;
Accelerometer1.pppg++ <–> C3 <–> JunctionBox3.pppg++;
Accelerometer2.pppg++ <–> C3 <–> JunctionBox7.pppg++;
Accelerometer3.pppg++ <–> C3 <–> JunctionBox8.pppg++;
ADCP300KHz1.pppg++ <–> C4 <–> JunctionBox1.pppg++;
ADCP300KHz2.pppg++ <–> C4 <–> JunctionBox10.pppg++;
ADCP75KHz1.pppg++ <–> C4 <–> JunctionBox5.pppg++;
ADCP75KHz2.pppg++ <–> C4 <–> JunctionBox4.pppg++;
ADCP600KHz1.pppg++ <–> C4 <–> JunctionBox1.pppg++;
ADCP600KHz2.pppg++ <–> C4 <–> JunctionBox8.pppg++;
ADCP150KHz.pppg++ <–> C4 <–> JunctionBox10.pppg++;
ADCP2MHz1.pppg++ <–> C4 <–> JunctionBox1.pppg++;
ADCP2MHz2.pppg++ <–> C4 <–> JunctionBox8.pppg++;
ADCP2MHz3.pppg++ <–> C4 <–> JunctionBox9.pppg++;
BPR1.pppg++ <–> C4 <–> JunctionBox1.pppg++;
BPR2.pppg++ <–> C4 <–> JunctionBox3.pppg++;
BPR3.pppg++ <–> C4 <–> JunctionBox5.pppg++;
BPR4.pppg++ <–> C4 <–> JunctionBox6.pppg++;
BPR5.pppg++ <–> C4 <–> JunctionBox7.pppg++;
BPR6.pppg++ <–> C4 <–> JunctionBox7.pppg++;
BPR7.pppg++ <–> C4 <–> JunctionBox7.pppg++;
Current_Meter1.pppg++ <–> C4 <–> JunctionBox2.pppg++;
Current_Meter2.pppg++ <–> C4 <–> JunctionBox3.pppg++;
Current_Meter3.pppg++ <–> C4 <–> JunctionBox4.pppg++;
Current_Meter4.pppg++ <–> C4 <–> JunctionBox4.pppg++;
Current_Meter5.pppg++ <–> C4 <–> JunctionBox4.pppg++;
Current_Meter6.pppg++ <–> C4 <–> JunctionBox5.pppg++;
Current_Meter7.pppg++ <–> C4 <–> JunctionBox5.pppg++;
Current_Meter8.pppg++ <–> C4 <–> JunctionBox5.pppg++;
Current_Meter9.pppg++ <–> C4 <–> JunctionBox6.pppg++;
Current_Meter10.pppg++ <–> C4 <–> JunctionBox6.pppg++;
Current_Meter11.pppg++ <–> C4 <–> JunctionBox7.pppg++;
Current_Meter12.pppg++ <–> C4 <–> JunctionBox9.pppg++;
CTD1.pppg++ <–> C4 <–> JunctionBox2.pppg++;
Hydrophone1.pppg++ <–> C3 <–> JunctionBox1.pppg++;

Hydrophone2.pppg++ <–> C3 <–> JunctionBox11.pppg++;
Adapter1.pppg++ <–> C3 <–> JunctionBox4.pppg++;
Adapter2.pppg++ <–> C3 <–> JunctionBox5.pppg++;
Adapter3.pppg++ <–> C3 <–> JunctionBox5.pppg++;
Adapter4.pppg++ <–> C3 <–> JunctionBox6.pppg++;
Adapter5.pppg++ <–> C3 <–> JunctionBox6.pppg++;
Broadband_Seismometer1.pppg++ <–> C4 <–> JunctionBox3.pppg++;
Broadband_Seismometer2.pppg++ <–> C4 <–> JunctionBox7.pppg++;
Broadband_Seismometer3.pppg++ <–> C4 <–> JunctionBox11.pppg++;
Borehole_Temperature.pppg++ <–> C4 <–> JunctionBox7.pppg++;
Compass.pppg++ <–> C4 <–> JunctionBox11.pppg++;
CTD2.pppg++ <–> C4 <–> JunctionBox3.pppg++;
CTD3.pppg++ <–> C4 <–> JunctionBox4.pppg++;
CTD4.pppg++ <–> C4 <–> JunctionBox4.pppg++;
CTD5.pppg++ <–> C4 <–> JunctionBox4.pppg++;
CTD6.pppg++ <–> C4 <–> JunctionBox4.pppg++;
CTD7.pppg++ <–> C4 <–> JunctionBox5.pppg++;
CTD8.pppg++ <–> C4 <–> JunctionBox5.pppg++;
CTD9.pppg++ <–> C4 <–> JunctionBox5.pppg++;
CTD10.pppg++ <–> C4 <–> JunctionBox6.pppg++;
CTD11.pppg++ <–> C4 <–> JunctionBox7.pppg++;
CTD12.pppg++ <–> C4 <–> JunctionBox8.pppg++;
CTD13.pppg++ <–> C4 <–> JunctionBox11.pppg++;
CORK.pppg++ <–> C4 <–> JunctionBox7.pppg++;
Flourometer1.pppg++ <–> C4 <–> JunctionBox2.pppg++;
Flourometer2.pppg++ <–> C4 <–> JunctionBox9.pppg++;
Flourometer3.pppg++ <–> C4 <–> JunctionBox10.pppg++;
BARS.pppg++ <–> C4 <–> JunctionBox5.pppg++;
Bioflouling.pppg++ <–> C4 <–> JunctionBox6.pppg++;
Chemini.pppg++ <–> C4 <–> JunctionBox5.pppg++;
EchoSounder.pppg++ <–> C4 <–> JunctionBox2.pppg++;
PanTilt_Lights1.pppg++ <–> C4 <–> JunctionBox6.pppg++;
PanTilt_Lights2.pppg++ <–> C4 <–> JunctionBox4.pppg++;
PanTilt_Lights3.pppg++ <–> C4 <–> JunctionBox4.pppg++;
PanTilt_Lights4.pppg++ <–> C4 <–> JunctionBox4.pppg++;

Temperature_Sensor1.pppg++ <–> C3 <–> JunctionBox5.pppg++;
Temperature_Sensor2.pppg++ <–> C3 <–> JunctionBox6.pppg++;
Optode1.pppg++ <–> C4 <–> JunctionBox2.pppg++;
Optode2.pppg++ <–> C4 <–> JunctionBox4.pppg++;
Video_Camera1.pppg++ <–> C4 <–> JunctionBox6.pppg++;
Video_Camera2.pppg++ <–> C4 <–> JunctionBox11.pppg++;
Video_Camera3.pppg++ <–> C4 <–> JunctionBox8.pppg++;
Video_Camera4.pppg++ <–> C4 <–> JunctionBox9.pppg++;
Video_Camera5.pppg++ <–> C4 <–> JunctionBox10.pppg++;
Piezometer.pppg++ <–> C4 <–> JunctionBox7.pppg++;
Pressure_Gauge1.pppg++ <–> C4 <–> JunctionBox3.pppg++;
Pressure_Gauge2.pppg++ <–> C4 <–> JunctionBox7.pppg++;
Light_Sensor.pppg++ <–> C4 <–> JunctionBox2.pppg++;
Sediment_Profiler.pppg++ <–> C3 <–> JunctionBox11.pppg++;
Sediment_Trap.pppg++ <–> C4 <–> JunctionBox10.pppg++;
Short_Range_Rotary_Sonar.pppg++ <–> C4 <–> JunctionBox9.pppg++;
Turbidity_Meter.pppg++ <–> C4 <–> JunctionBox8.pppg++;
Vehicle1.pppg++ <–> C4 <–> JunctionBox9.pppg++;
Vehicle2.pppg++ <–> C4 <–> JunctionBox10.pppg++;
CSEM1.pppg++ <–> C4 <–> JunctionBox3.pppg++;
CSEM2.pppg++ <–> C4 <–> JunctionBox3.pppg++;
COVIS.pppg++ <–> C4 <–> JunctionBox4.pppg++;
Short_Period_Seismometer1.pppg++ <–> C3 <–> JunctionBox6.pppg++;
Short_Period_Seismometer2.pppg++ <–> C3 <–> JunctionBox6.pppg++;
Methane_Sensor.pppg++ <–> C4 <–> JunctionBox8.pppg++;
Water_Sampler.pppg++ <–> C4 <–> JunctionBox5.pppg++;

}

A.1.2 Scenario-II

Network Description File (NED)

package inet.examples.NeptuneCANADAscII;
import inet.networklayer.autorouting.FlatNetworkConfigurator;

```
import inet.nodes.inet.Router;
import inet.nodes.inet.StandardHost;
import inet.world.ChannelInstaller;
import inet.world.NAMTrace;
import ned.DatarateChannel;
network NeptuneCANADAScII
{
parameters:
@display("bgb=,,dodgerBlue,blue");
types:
channel C1 extends DatarateChannel
{
datarate = 10Gbps;
delay = 0.1us;
}
channel C2 extends DatarateChannel
{
datarate = 4Gbps;
delay = 0.1us;
}
channel C3 extends DatarateChannel
{
datarate = 1Gbps;
delay = 0.1us;
}
channel C4 extends DatarateChannel
{
datarate = 100Mbps;
delay = 0.1us;
}
submodules:
//————————————————//

UVIC: StandardHost {
@display("p=1088,33;i=misc/globe");
```

```
}
Port_Alberni_Station: Router {
@display("p=1088,238;i=misc/building");
}
//Branching Units–There are 6 Branching Units that transmit the data coming from
Node agents and connected through Shore Station
BranchingUnit1: Router {
@display("p=956,238;i=old/xconn");
}
BranchingUnit2: Router {
@display("p=730,162;i=old/xconn");
}
BranchingUnit3: Router {
@display("p=433,162;i=old/xconn");
}
BranchingUnit4: Router {
@display("p=251,256;i=old/xconn");
}
BranchingUnit5: Router {
@display("p=423,336;i=old/xconn");
}
BranchingUnit6: Router {
@display("p=816,336;i=old/xconn");
}
// Node Agents–There are 6 Node agents that are connected to Junction Boxes
Folger_Passage: Router {
@display("p=948,179;i=misc/house");
}
ODP_889: Router {
@display("p=718,107;i=misc/house");
}
Middle_Valley: Router {
@display("p=342,68;i=misc/cloud2");
}
Endavour_Ridge: Router {
```

@display(”p=135,256;i=misc/house”);
}
ODP_1027: Router {
@display(”p=324,386;i=misc/house”);
}
Barkley_Canyon: Router {
@display(”p=822,386;i=misc/house”);
}
//Junction Boxes–There are 13 Junction Boxes that are connected to 5 Nodes one by one.
JunctionBox1: Router {
@display(”p=979,107;i=device/router”);
}
JunctionBox2: Router {
@display(”p=901,107;i=device/router”);
}
JunctionBox3: Router {
@display(”p=631,68;i=device/router”);
}
JunctionBox4: Router {
@display(”p=79,336;i=device/router”);
}
JunctionBox5: Router {
@display(”p=42,256;i=device/router”);
}
JunctionBox6: Router {
@display(”p=239,107;i=device/router”);
}
JunctionBox7: Router {
@display(”p=227,432;i=device/router”);
}
JunctionBox8: Router {
@display(”p=1005,358;i=device/router”);
}
JunctionBox9: Router {

@display(”p=995,417;i=device/router”);
}
JunctionBox10: Router {
@display(”p=909,417;i=device/router”);
}
JunctionBox11: Router {
@display(”p=1068,417;i=device/router”);
}
//————————// Additional Junction Boxes
JunctionBox12: Router;
JunctionBox13: Router;
//————————-//
//Instruments–There are 38 Different Type of Instruments which are connected to Junction Boxes and they generate data which is sent throught UVIC.

//————————————-//Additional Instruments

Hydrophone3: StandardHost;
Video_Camera6: StandardHost;

Water_Sampler1: StandardHost;
Vehicle3: StandardHost;
Chemini1: StandardHost;

//————————————//

CTD1: Router;
Hydrophone1: StandardHost;
Hydrophone2: StandardHost;
Water_Sampler: StandardHost;
BPR1: StandardHost;
BPR2: StandardHost {
@display(”p=577,126”);
}
BPR3: StandardHost {

```
@display("p=187,214");
}
BPR4: StandardHost {
@display("p=108,201");
}
BPR5: StandardHost;
BPR6: StandardHost;
BPR7: StandardHost;
Current_Meter1: StandardHost;
Current_Meter2: StandardHost {
@display("p=650,126");
}
Current_Meter3: StandardHost;
Current_Meter4: StandardHost;
Current_Meter5: StandardHost;
Current_Meter6: StandardHost {
@display("p=324,167");
}
Current_Meter7: StandardHost;
Current_Meter8: StandardHost;
Current_Meter9: StandardHost {
@display("p=108,162");
}
Current_Meter10: StandardHost;
Current_Meter11: StandardHost;
Current_Meter12: StandardHost;
Accelerometer1: StandardHost {
@display("p=781,82");
}
Accelerometer2: StandardHost;
Accelerometer3: StandardHost;
ADCP300KHz1: StandardHost;
ADCP300KHz2: StandardHost;
ADCP150KHz: StandardHost;
ADCP600KHz1: StandardHost;
```

```
ADCP600KHz2: StandardHost;
ADCP75KHz1: StandardHost;
ADCP75KHz2: StandardHost;
ADCP2MHz1: StandardHost;
ADCP2MHz2: StandardHost;
ADCP2MHz3: StandardHost;
Adapter1: StandardHost;
Adapter2: StandardHost {
@display("p=289,68");
}
Adapter3: StandardHost {
@display("p=239,183");
}
Adapter4: StandardHost {
@display("p=289,179");
}
Adapter5: StandardHost {
@display("p=236,75");
}
Broadband_Seismometer1: StandardHost {
@display("p=590,183");
}
Broadband_Seismometer2: StandardHost;
Broadband_Seismometer3: StandardHost;
Borehole_Temperature: StandardHost;
Compass: StandardHost;
CTD2: StandardHost {
@display("p=538,94");
}
CTD3: StandardHost;
CTD4: StandardHost;
CTD5: StandardHost;
CTD6: StandardHost;
CTD7: StandardHost {
@display("p=187,82");
```

```
}
CTD8: StandardHost {
@display("p=148,94");
}
CTD9: StandardHost;
CTD10: StandardHost {
@display("p=69,179");
}
CTD11: StandardHost;
CTD12: StandardHost;
CTD13: StandardHost;
CORK: StandardHost;
Bioflouling: StandardHost {
@display("p=130,196");
}
Flourometer1: StandardHost;
Flourometer2: StandardHost;
Flourometer3: StandardHost;
BARS: StandardHost;
Chemini: StandardHost;
EchoSounder: StandardHost;
PanTilt_Lights1: StandardHost {
@display("p=324,126");
}
PanTilt_Lights2: StandardHost;
PanTilt_Lights3: StandardHost;
PanTilt_Lights4: StandardHost;
Temperature_Sensor1: StandardHost;
Temperature_Sensor2: StandardHost;
Optode1: StandardHost;
Optode2: StandardHost;
Video_Camera1: StandardHost {
@display("p=97,102");
}
Video_Camera2: StandardHost;
```

```
Video_Camera3: StandardHost;
Video_Camera4: StandardHost;
Video_Camera5: StandardHost;
Piezometer: StandardHost;
Pressure_Gauge1: StandardHost {
@display("p=707,20");
}
Pressure_Gauge2: StandardHost;
Light_Sensor: StandardHost;
Sediment_Profiler: StandardHost;
Sediment_Trap: StandardHost;
Short_Range_Rotary_Sonar: StandardHost;
Turbidity_Meter: StandardHost;
Vehicle1: StandardHost;
Vehicle2: StandardHost;
CSEM1: StandardHost {
@display("p=538,51");
}
CSEM2: StandardHost {
@display("p=583,20");
}
COVIS: StandardHost;
Short_Period_Seismometer1: StandardHost;
Short_Period_Seismometer2: StandardHost {
@display("p=62,33");
}
Methane_Sensor: StandardHost;
Water_Sampler: StandardHost {
@display("p=135,303");
}
channelInstaller: ChannelInstaller {
parameters:
channelClass = "ThruputMeteringChannel";
channelAttrs = "format=u";
@display("p=98,50");
```

```
}
flatNetworkConfigurator: FlatNetworkConfigurator {
@display("p=301,20");
}
nam: NAMTrace {
parameters:
@display("p=260,50");
}
connections:
UVIC.pppg++ <–> C1 <–> Port_Alberni_Station.pppg++;
Port_Alberni_Station.pppg++ <–> C1 <–> BranchingUnit1.pppg++;
//Branching Unit connections
BranchingUnit1.pppg++ <–> C2 <–> BranchingUnit2.pppg++;
BranchingUnit2.pppg++ <–> C2 <–> BranchingUnit3.pppg++;
BranchingUnit3.pppg++ <–> C2 <–> BranchingUnit4.pppg++;
BranchingUnit4.pppg++ <–> C2 <–> BranchingUnit5.pppg++;
BranchingUnit5.pppg++ <–> C2 <–> BranchingUnit6.pppg++;
BranchingUnit6.pppg++ <–> C2 <–> BranchingUnit1.pppg++;
//Node Agent connections
Folger_Passage.pppg++ <–> C3 <–> BranchingUnit1.pppg++;
ODP_889.pppg++ <–> C3 <–> BranchingUnit2.pppg++;
Middle_Valley.pppg++ <–> C3 <–> BranchingUnit3.pppg++; //Empty Node Agent!!!
Endavour_Ridge.pppg++ <–> C3 <–> BranchingUnit4.pppg++;
ODP_1027.pppg++ <–> C3 <–> BranchingUnit5.pppg++;
Barkley_Canyon.pppg++ <–> C3 <–> BranchingUnit6.pppg++;
//Junction Box connection –> in here Junction Box8 is connected to Junction boxes
9,10 and 11.
JunctionBox1.pppg++ <–> C3 <–> Folger_Passage.pppg++;
JunctionBox2.pppg++ <–> C3 <–> Folger_Passage.pppg++;
JunctionBox3.pppg++ <–> C3 <–> ODP_889.pppg++;
//There is no instrument connections to Middle Valley Node!!!
JunctionBox4.pppg++ <–> C3 <–> Endavour_Ridge.pppg++;
JunctionBox5.pppg++ <–> C3 <–> Endavour_Ridge.pppg++;
JunctionBox6.pppg++ <–> C3 <–> Endavour_Ridge.pppg++;
JunctionBox7.pppg++ <–> C3 <–> ODP_1027.pppg++;
```

JunctionBox8.pppg++ <–> C3 <–> Barkley_Canyon.pppg++;
//————————————————————————————-//
JunctionBox8.pppg++ <–> C4 <–> JunctionBox9.pppg++;
JunctionBox8.pppg++ <–> C4 <–> JunctionBox10.pppg++;
JunctionBox8.pppg++ <–> C4 <–> JunctionBox11.pppg++;
Accelerometer1.pppg++ <–> C3 <–> JunctionBox3.pppg++;
Accelerometer2.pppg++ <–> C3 <–> JunctionBox7.pppg++;
Accelerometer3.pppg++ <–> C3 <–> JunctionBox8.pppg++;
ADCP300KHz1.pppg++ <–> C4 <–> JunctionBox1.pppg++;
ADCP300KHz2.pppg++ <–> C4 <–> JunctionBox10.pppg++;
ADCP75KHz1.pppg++ <–> C4 <–> JunctionBox5.pppg++;
ADCP75KHz2.pppg++ <–> C4 <–> JunctionBox4.pppg++;
ADCP600KHz1.pppg++ <–> C4 <–> JunctionBox1.pppg++;
ADCP600KHz2.pppg++ <–> C4 <–> JunctionBox8.pppg++;
ADCP150KHz.pppg++ <–> C4 <–> JunctionBox10.pppg++;
ADCP2MHz1.pppg++ <–> C4 <–> JunctionBox1.pppg++;
ADCP2MHz2.pppg++ <–> C4 <–> JunctionBox8.pppg++;
ADCP2MHz3.pppg++ <–> C4 <–> JunctionBox9.pppg++;
BPR1.pppg++ <–> C4 <–> JunctionBox1.pppg++;
BPR2.pppg++ <–> C4 <–> JunctionBox3.pppg++;
BPR3.pppg++ <–> C4 <–> JunctionBox5.pppg++;
BPR4.pppg++ <–> C4 <–> JunctionBox6.pppg++;
BPR5.pppg++ <–> C4 <–> JunctionBox7.pppg++;
BPR6.pppg++ <–> C4 <–> JunctionBox7.pppg++;
BPR7.pppg++ <–> C4 <–> JunctionBox7.pppg++;
Current_Meter1.pppg++ <–> C4 <–> JunctionBox2.pppg++;
Current_Meter2.pppg++ <–> C4 <–> JunctionBox3.pppg++;
Current_Meter3.pppg++ <–> C4 <–> JunctionBox4.pppg++;
Current_Meter4.pppg++ <–> C4 <–> JunctionBox4.pppg++;
Current_Meter5.pppg++ <–> C4 <–> JunctionBox4.pppg++;
Current_Meter6.pppg++ <–> C4 <–> JunctionBox5.pppg++;
Current_Meter7.pppg++ <–> C4 <–> JunctionBox5.pppg++;
Current_Meter8.pppg++ <–> C4 <–> JunctionBox5.pppg++;
Current_Meter9.pppg++ <–> C4 <–> JunctionBox6.pppg++;
Current_Meter10.pppg++ <–> C4 <–> JunctionBox6.pppg++;

Current_Meter11.pppg++ <–> C4 <–> JunctionBox7.pppg++;
Current_Meter12.pppg++ <–> C4 <–> JunctionBox9.pppg++;
CTD1.pppg++ <–> C4 <–> JunctionBox2.pppg++;
Hydrophone1.pppg++ <–> C3 <–> JunctionBox1.pppg++;
Hydrophone2.pppg++ <–> C3 <–> JunctionBox11.pppg++;
Adapter1.pppg++ <–> C3 <–> JunctionBox4.pppg++;
Adapter2.pppg++ <–> C3 <–> JunctionBox5.pppg++;
Adapter3.pppg++ <–> C3 <–> JunctionBox5.pppg++;
Adapter4.pppg++ <–> C3 <–> JunctionBox6.pppg++;
Adapter5.pppg++ <–> C3 <–> JunctionBox6.pppg++;
Broadband_Seismometer1.pppg++ <–> C4 <–> JunctionBox3.pppg++;
Broadband_Seismometer2.pppg++ <–> C4 <–> JunctionBox7.pppg++;
Broadband_Seismometer3.pppg++ <–> C4 <–> JunctionBox11.pppg++;
Borehole_Temperature.pppg++ <–> C4 <–> JunctionBox7.pppg++;
Compass.pppg++ <–> C4 <–> JunctionBox11.pppg++;
CTD2.pppg++ <–> C4 <–> JunctionBox3.pppg++;
CTD3.pppg++ <–> C4 <–> JunctionBox4.pppg++;
CTD4.pppg++ <–> C4 <–> JunctionBox4.pppg++;
CTD5.pppg++ <–> C4 <–> JunctionBox4.pppg++;
CTD6.pppg++ <–> C4 <–> JunctionBox4.pppg++;
CTD7.pppg++ <–> C4 <–> JunctionBox5.pppg++;
CTD8.pppg++ <–> C4 <–> JunctionBox5.pppg++;
CTD9.pppg++ <–> C4 <–> JunctionBox5.pppg++;
CTD10.pppg++ <–> C4 <–> JunctionBox6.pppg++;
CTD11.pppg++ <–> C4 <–> JunctionBox7.pppg++;
CTD12.pppg++ <–> C4 <–> JunctionBox8.pppg++;
CTD13.pppg++ <–> C4 <–> JunctionBox11.pppg++;
CORK.pppg++ <–> C4 <–> JunctionBox7.pppg++;
Flourometer1.pppg++ <–> C4 <–> JunctionBox2.pppg++;
Flourometer2.pppg++ <–> C4 <–> JunctionBox9.pppg++;
Flourometer3.pppg++ <–> C4 <–> JunctionBox10.pppg++;
BARS.pppg++ <–> C4 <–> JunctionBox5.pppg++;
Bioflouling.pppg++ <–> C4 <–> JunctionBox6.pppg++;
Chemini.pppg++ <–> C4 <–> JunctionBox5.pppg++;
EchoSounder.pppg++ <–> C4 <–> JunctionBox2.pppg++;

PanTilt_Lights1.pppg++ <–> C4 <–> JunctionBox6.pppg++;
PanTilt_Lights2.pppg++ <–> C4 <–> JunctionBox4.pppg++;
PanTilt_Lights3.pppg++ <–> C4 <–> JunctionBox4.pppg++;
PanTilt_Lights4.pppg++ <–> C4 <–> JunctionBox4.pppg++;
Temperature_Sensor1.pppg++ <–> C3 <–> JunctionBox5.pppg++;
Temperature_Sensor2.pppg++ <–> C3 <–> JunctionBox6.pppg++;
Optode1.pppg++ <–> C4 <–> JunctionBox2.pppg++;
Optode2.pppg++ <–> C4 <–> JunctionBox4.pppg++;
Video_Camera1.pppg++ <–> C4 <–> JunctionBox6.pppg++;
Video_Camera2.pppg++ <–> C4 <–> JunctionBox11.pppg++;
Video_Camera3.pppg++ <–> C4 <–> JunctionBox8.pppg++;
Video_Camera4.pppg++ <–> C4 <–> JunctionBox9.pppg++;
Video_Camera5.pppg++ <–> C4 <–> JunctionBox10.pppg++;
Piezometer.pppg++ <–> C4 <–> JunctionBox7.pppg++;
Pressure_Gauge1.pppg++ <–> C4 <–> JunctionBox3.pppg++;
Pressure_Gauge2.pppg++ <–> C4 <–> JunctionBox7.pppg++;
Light_Sensor.pppg++ <–> C4 <–> JunctionBox2.pppg++;
Sediment_Profiler.pppg++ <–> C3 <–> JunctionBox11.pppg++;
Sediment_Trap.pppg++ <–> C4 <–> JunctionBox10.pppg++;
Short_Range_Rotary_Sonar.pppg++ <–> C4 <–> JunctionBox9.pppg++;
Turbidity_Meter.pppg++ <–> C4 <–> JunctionBox8.pppg++;
Vehicle1.pppg++ <–> C4 <–> JunctionBox9.pppg++;
Vehicle2.pppg++ <–> C4 <–> JunctionBox10.pppg++;
CSEM1.pppg++ <–> C4 <–> JunctionBox3.pppg++;
CSEM2.pppg++ <–> C4 <–> JunctionBox3.pppg++;
COVIS.pppg++ <–> C4 <–> JunctionBox4.pppg++;
Short_Period_Seismometer1.pppg++ <–> C3 <–> JunctionBox6.pppg++;
Short_Period_Seismometer2.pppg++ <–> C3 <–> JunctionBox6.pppg++;
Methane_Sensor.pppg++ <–> C4 <–> JunctionBox8.pppg++;
Water_Sampler.pppg++ <–> C4 <–> JunctionBox5.pppg++;

//————————————————-//Additional Connections

JunctionBox12.pppg++ <–> C3 <–> Middle_Valley.pppg++;
JunctionBox13.pppg++ <–> C3 <–> Middle_Valley.pppg++;

Hydrophone3.pppg++ <–> C3 <–> JunctionBox12.pppg++;
Chemini1.pppg++ <–> C4 <–> JunctionBox13.pppg++;
Video_Camera6.pppg++ <–> C4 <–> JunctionBox13.pppg++;
Vehicle3.pppg++ <–> C4 <–> JunctionBox12.pppg++;
Water_Sampler1.pppg++ <–> C4 <–> JunctionBox13.pppg++;

}

A.2 Features and Parameters of NEPTSim Instruments

In this section, all the instruments including piggy-back, using TCP and UDP ones are informed in terms of their metrics on the network traffic. So each table includes certain number of device and these devices are columned according to their device name, time when they send data or we can say message frequency and what size they generate data or it is sometimes called message length. These tables are made because during the implemetation of NEPTSim, these information were used in the initialization files. As it is explained in the previous chapters, the initialization file includes the characteristics of the nodes. Thus, instead of displaying the code-lines of the initialization files, we prefered to create tables for each type of devices and it displays all the network features of each instrument.

Device Name	**Send Time**(in seconds)	**Bytes Send**(in MBytes)
BPR1	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
BPR2	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
BPR3	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
BPR4	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
BPR5	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
BPR6	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
BPR7	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
CTD1	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
CTD2	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
CTD3	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
CTD4	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
CTD5	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
CTD6	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
CTD7	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
CTD8	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
CTD9	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
CTD10	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
CTD11	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
CTD12	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
CTD13	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Current Meter1	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Current Meter2	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Current Meter3	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Current Meter4	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Current Meter5	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Current Meter6	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Current Meter7	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Current Meter8	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Current Meter9	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Current Meter10	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)

Table A.1: The Characteristics of The First 30 NEPTSim Instruments that use TCP Protocol

Device Name	**Send Time**(in seconds)	**Bytes Send**(in MBytes)
Current Meter11	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Current Meter12	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Current Meter13	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Accelerometer1	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Accelerometer2	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Accelerometer3	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
ADCP150KHz	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
ADCP75KHz1	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
ADCP75KHz2	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
ADCP300KHz1	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
ADCP300KHz2	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
ADCP600KHz1	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
ADCP600KHz2	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
ADCP2MHz1	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
ADCP2MHz2	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
ADCP2MHz3	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Adapter1	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Adapter2	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Adapter3	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Adapter4	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Adapter5	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Broadband Seismometer1	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Broadband Seismometer2	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Broadband Seismometer3	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Borehole Temperature	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Compass	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
CORK	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Bioflouling	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Flourometer1	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Flourometer2	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)

Table A.2: The Characteristics of The NEPTSim Instruments from 30 to 60 that use TCP Protocol

Device Name	**Send Time**(in seconds)	**Bytes Send**(in MBytes)
Flourometer3	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
BARS	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Chemini	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
EchoSounder	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
PanTilt Lights1	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
PanTilt Lights2	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
PanTilt Lights3	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
PanTilt Lights4	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Temperature Sensor1	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Temperature Sensor2	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Optode1	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Optode2	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Piezometer	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Pressure Gauge1	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Pressure Gauge2	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Light Sensor	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Sediment Profiler	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Sediment Trap	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Short Range Rotary Sonar	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Turbidity Meter	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Vehicle1	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Vehicle2	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
CSEM1	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
CSEM2	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
COVIS	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Short Period Seismometer1	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Short Period Seismometer2	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
SMethane Sensor	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)
Water Sampler	normal(0.9 s, 0.45 s)	uniform(0.1 MiB, 1 MiB)

Table A.3: The Characteristics of The Last 30 NEPTSim Instruments that use TCP Protocol

Device Name	**Message Frequency**(in seconds)	**Message Length**(in MBytes)
Video Camera1	exponential(1 s)	uniform(0.5 MiB, 1.5 MiB))
Video Camera2	exponential(1 s)	uniform(0.5 MiB, 1.5 MiB))
Video Camera3	exponential(1 s)	uniform(0.5 MiB, 1.5 MiB))
Video Camera4	exponential(1 s)	uniform(0.5 MiB, 1.5 MiB))
Video Camera5	exponential(1 s)	uniform(0.5 MiB, 1.5 MiB))
Hydrophone1	exponential(1 s)	uniform(0.9 MiB, 2.5 MiB))
Hydrophone2	exponential(1 s)	uniform(0.7 MiB, 1.9 MiB))

Table A.4: The Characteristics of The NEPTSim Instruments that use UDP Protocol

Device Name	**Protocol**	**Frequency**(in seconds)	**Length**(in MBytes)
Video Camera6	UDP	exponential(1 s)	uniform(0.5 MiB, 1.5 MiB))
Hydrophone3	UDP	exponential(1 s)	uniform(0.9 MiB, 2.9 MiB))
Chemini1	TCP	normal(0.8 s, 0.4 s)	uniform(0.6 MiB, 1.8 MiB)
Vehicle3	TCP	normal(0.7 s, 0.3 s)	uniform(0.5 MiB, 1.5 MiB)
Water Sampler1	TCP	normal(0.5 s, 0.25 s)	uniform(0.3 MiB, 1.2 MiB)

Table A.5: The Characteristics of The Additional NEPTSim Instruments that use either TCP or UDP Protocol for the Scenario-2