CHAPTER THREE

METHODOLOGY

In the previous chapter, we discussed the literature on TCP congestion control mechanisms. As stated in chapter one, one of the research objectives is to evaluate the proposed congestion control mechanism over LTE networks. In this chapter, the research methodology for evaluating the proposed modification is presented. The next section will introduced performance evaluation techniques in computer networks. The details of the selected network simulator are presented in section two. The rest of the chapter will discuss the simulation experimental design, the experimental hardware and software, validation and verification, and network performance metrics. Finally, we conclude the chapter in the summary section.

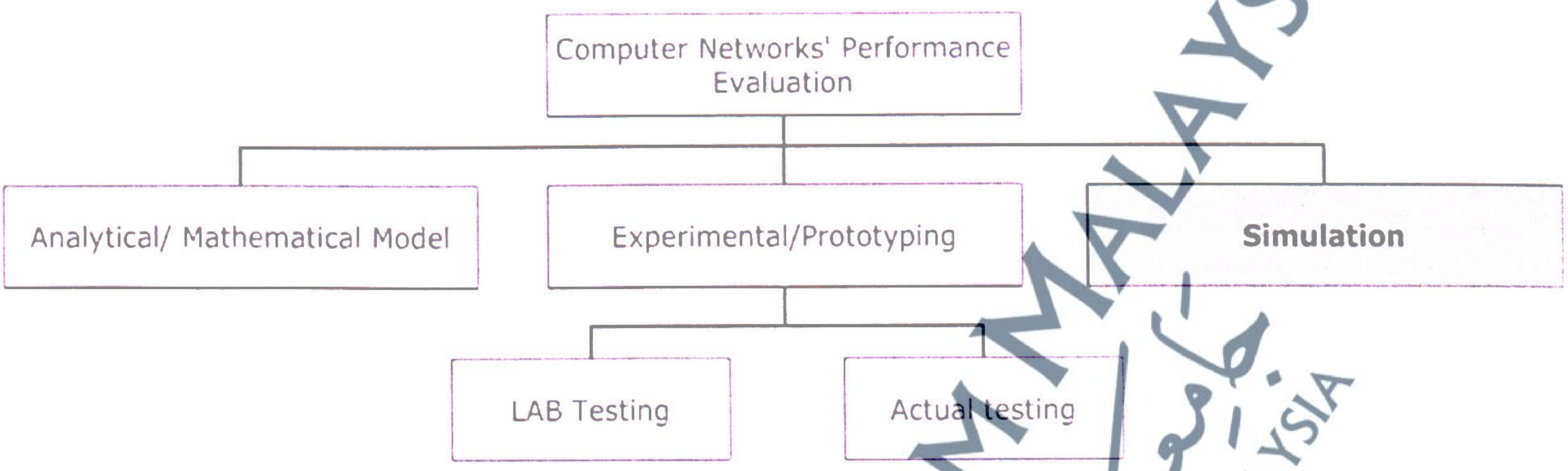
3.1 Introduction

Chapter one discussed the research framework. It is started by reviewing TCP congestion control mechanisms, followed by the selection of the network simulator. Finally, it considered the design, implementation and evaluation of the proposed enhanced congestion control mechanism. Chapter two introduced a comprehensive review of the TCP congestion control mechanism in the literature.

In this chapter, we will discuss the methods of selecting the simulation approach to evaluate our proposed modification. In chapter four we introduce the proposed modification design. Finally, in chapter five, we discuss the implementation and evaluate the proposed modifications.

According to Jain (2015), approaches to evaluating network performance are classified into three main techniques: measurement, simulation, and analytical model. Figure 3.1 illustrates these techniques.

Figure 3.1: Evaluation Approaches (Adopted from Jain, 2015)



3.1.1 Analytical Modelling

The pure definition of analytical modelling in computer systems is a set of equations that describes the performance of a computer system (Sayandev, 2014). Analytical modelling includes the process of building, solving, and validating the analytical model. The analytical model is used to analyse simple systems as well as complex systems. Some researchers conduct analytical modelling to understand the current activities of the system. Others may use it to predict the behaviour of a certain element within the system. Moreover, some use the analytical model to represent a system and use it as an input to another system (Mukherjee, 2014).

Using an analytical model to analyse computer networks has major drawbacks (Keshav, 1991; Law & Kelton, 1991). Firstly, using an analytical model involves many assumptions, which restrain the model from representing the behaviour of the real systems. Secondly, it ignores the interaction of the system components, which in

some systems could be the main factor that affects system performance. Therefore, using the analytical model alone to evaluate the communication system may not reflect the real system performance. For the purposes of this study, analytical modelling will be improbable to use regarding the complexity of the proposed model.

3.1.2 Measurement

The second approach for evaluating communication system performance is by measuring real system performance. Performance measurement requires monitoring the system while it is being subjected to a particular workload, which could be additional instruction, instruction mixes, kernels, a synthetic program, or application benchmarks. Therefore, the researcher can design a prototype and test it in a specific communication environment using a network emulator or real system.

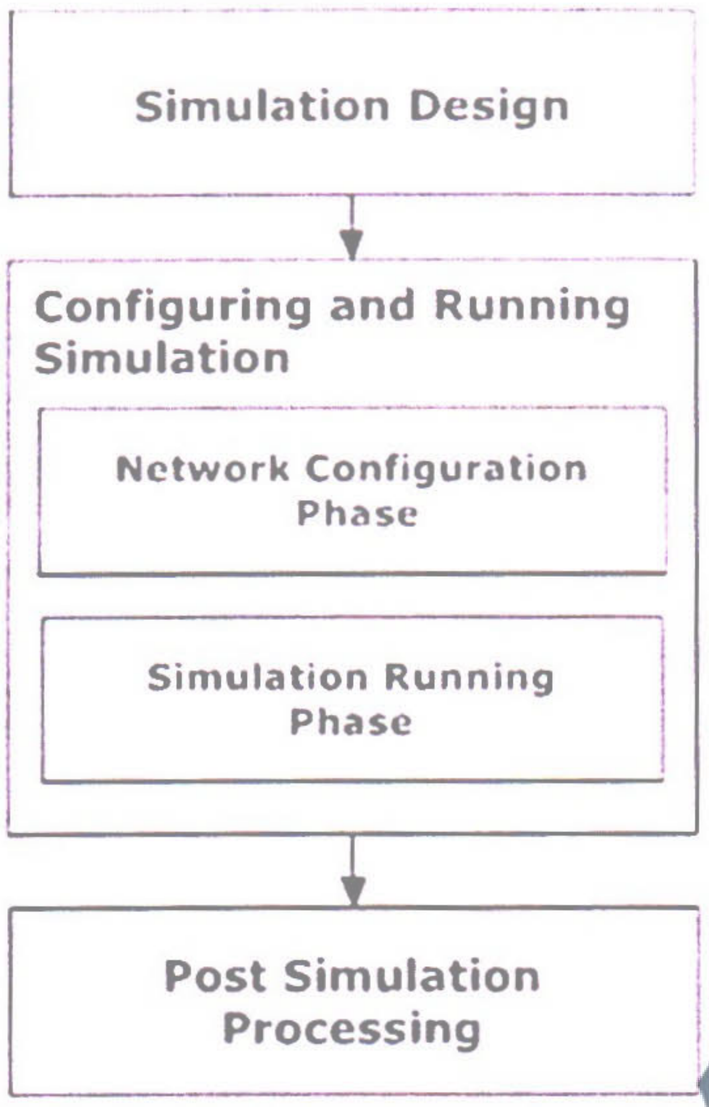
Measuring the performance of a workload requires real equipment and programs, so that the results reflect the actual performance of the tested system. However, the time required for measurement is the most variable in analytical modelling and simulation, as the process of preparing the test bed or the real network equipment is time consuming. Moreover, the measurement approach lacks strength in comparison with alternatives, and when finding the optimal parameters that affect system performance (Ince et al., 2007). Furthermore, measurement techniques are very costly concerning providing the equipment, instruments, and time. Despite all the above limitations, it is much easier to convene other findings using a measurements approach as it is using a real measure. However, in this research, it is beyond our capacity to use this approach even though it can reflect the actual behaviour of the proposed modifications.

3.1.3 Simulation

Simulation is a method of using computer software to model the operation of a real world process, system, or event (Law, 2006). Many problems can be formulated into models which act as simplified representations of a system. According to Davis et al. (2007), simulation is an increasingly significant methodology approach to theory development. The ACM symposium reported that based on 151 articles from a five year period, 76% of the works used network simulation methods. Numerous effective research studies have used simulation as their primary research methodology (Ivanov et al., 2007). In addition, simulation provides better insight into complex environments without the need to construct these expensive, time consuming and sometimes risky real environments. Moreover, simulation provides an analytically precise means to understand the behaviour of an actor, factor, or any element in the simulated environment (Michael, 2001).

A simulation approach for computer networks consists of three main steps, as in Issariyakul and Hossain (2012). Firstly, there is the planning phase, which includes defining the study case, designing the tentative model, and devising the experiments scenarios. Secondly, the implementation phase includes defining the simulation initial state, creating and executing the simulation events, and collecting the data for the post simulation processing. Finally, the post simulation process is started. This phase includes analysis the simulation results to verify and evaluate the performance of the proposed algorithm, and it may include comparing the results of the proposed design with the current implementation. Figure 3.2 shows the simulation framework.

Figure 3.2: Simulation framework (Adopted from Issariyakul & Hossain, 2012)



Network simulation can be classified into two main types: time-clocked simulation and discrete-event simulation (Faulin et al., 2010). With time-clocked simulators, simulation progresses through the iterative progressing of a pre-defined time slots. During the simulation, time events with the iterated time slot are executed. The time-clocked simulator is updated at fixed time intervals, and the system is examined if any event happens during these intervals. All events occurring during this period are treated as if they happened simultaneously. On the other hand, with discrete-event, the simulation progresses by the execution of the scheduled next event. Therefore, the simulation time is updated after the next event execution. It simply shifts from event to event and the system state does not change in between. It only keep track of the times when something of interest occurs during the simulation period.

Precisely, simulation is easier, faster, safer, simpler, and cheaper than measurement. For the purpose of this research, the simulation approach was

conducted to test the proposed algorithms and validate their efficiency. The following section discusses the criteria for selecting a network simulator.

3.2 Selecting a Network Simulator

Nowadays, there are many network simulators that have different features and different aspects. Selecting a network simulator is a very important decision, because a proper choice will save time and effort, and will lead to efficient results. As presented in Sarkar and Halim (2011), four main categorizes are defined to select a network simulator. The categories are hardware and software consideration, modelling capabilities, simulation capabilities, and input/output issues.

The hardware and software considerations include the flexibility of the software to model complex systems. Firstly, these criteria include coding aspects such as the ability of the software to allow additional programming. Also, some programming features can be added to the strength of a selected package, including the ability to access source code, global variable support, built-in functions, and others. Furthermore, software compatibility can enhance the package's ability to interface with other software to exchange data with the simulated system. Moreover, the quality and type of the user support services plays major role in selecting a network simulator. In addition, the total cost of purchase, installation and maintenance of a certain package is the main consideration here.

On the other hand, the modelling capabilities include how well and precisely the simulator can represent the selected module. One main consideration here is the availability of the selected module. Other considerations include the educational and

experience level required to use a certain simulator, the ease of learning, and friendless issues.

Simulation capabilities include the way an experiment is performed and which attributes and parameters are used during the simulation period. These features include visual aspects (e.g. animation capabilities, and the quality and expressiveness of graphics, etc.), testability aspects (e.g. error messages, debugging tools, trace and echo files, etc.), running aspects (e.g. warming up period, breakpoints, automatic batch run, etc.), and the statistical facilities provided by the simulation package.

The last category in choosing a network simulator is the input/output capabilities. In this category, we investigate how the user can present the data to the simulator and the type and quality of the output reports provided by the simulator. Moreover, analysis capabilities (e.g. what-if analysis, conclusion making support, optimization, etc.) are included in this group.

To select a credible simulator, it is important to have a good knowledge of the available simulation tools. While various simulators are available for building and simulating network models, we have nominated four network simulators for the purpose of our study: OPNET, OMNeT++, ns-2, and ns-3. These simulators were selected according to their popularity, published results, and modelling capabilities. The following sections briefly describe each one.

3.2.1 OPNET Modeller

OPNET ("OPNET", n.d.) stands for OPtimized Network Engineering Tools and was created by OPNET Technologies Inc. OPNET is a network simulation tool set which provides product solutions to address many aspects of communication networks,

including application performance management, planning tools, engineering, operations, research and development.

The OPNET Modeller is a product that provides the best network modelling and simulation solution of all the OPNET tools. It is a dynamic discrete-event simulator with graphical user interface, providing comprehensive development environment tools, including model design, simulation, data collection, and data analysis. It is supported by object-oriented and hierarchical modelling, debugging, and analysis tools. Also, its system-in-the-Loop interface allows simulation with live systems which feed real-world data into the simulation. It provides an open interface for integrating external files, extended libraries, and other simulators. It incorporates a wide range of protocols and technologies, and enables modelling of a wide range of network technologies and types. Moreover, the OPNET Modeller provides models of hundreds of devices and protocols with source code access capabilities.

Furthermore, the OPNET Modeller provides various tools for processing, visual representing, and analyzing the simulation output data. Using the OPNET Modeller, you can configure the simulation to automatically generate animations of the modelled system at various levels. Even more, simulation results can be plotted as time series graphs, scatter graphs, histograms, and probability functions.

In spite of its strengths, the OPNET Modeller is very expensive software (according to OPNET Technology Inc., a licence costs about $28,916.88). Even though OPNET provides a "Modeller University Program" for free to qualifying universities worldwide for academic research, it is still a very limited licence which offers only the most popular modules. In addition, if a Modeller university program request is approved, then one licence will be installed in one university machine.

Furthermore, the OPNET Modeller parameter categorization is not very transparent (Sarkar & Halim, 2011). In sum, given all the above limitations, the OPNET Modeller is not feasible to use as a simulator in this research.

3.2.2 OMNeT++

OMNeT++ ("OMNeT++", n.d.) is a discrete event network simulation platform. It has provided an open source simulation package since 1992. OMNeT++ started as a programming assignment at the Technical University of Budapest. Since that time, many researchers have contributed to OMNeT++ open source and the first public release was in 1997. OMNeT++ provides module architecture for models programmed in C++. These modules can be assembled into complex models using Network Description programming language (NED) to describe the simulation topology.

Three distinctive elements are described in NED. The first element is the module definitions. This block of definitions describes the modules and sets its parameters. The second element is the channel definitions, which describe all the link details. The third element is the network definitions. This block describes the whole simulation system including descriptions of the top level module for the network and all its instances modules.

The main strengths of the OMNeT++ simulator include GUI, object inspectors for zooming into the components level and displaying the state of each model during the simulation time, modular abstraction, configurability, and a detailed implementation of the modules and protocols.

On the other hand, OMNeT++ is a bit slow as it requires long simulation running time and high memory consumption. Moreover, some argue (Sakar, 2011) about the difficulty of using OMNeT++. Even more, OMNeT++ does not support modelling many new wireless technologies, including LTE and LTE-Advance, which this research is about.

3.2.3 Network Simulator-2 (ns-2)

ns-2 ("VINT Project, ns-2", n.d.) is a discrete event simulator targeted to networking research. ns-2 provides substantial support for TCP, routing, and multicast protocol over wired and wireless networks. ns-2 began in 1989 as a variant of the REAL (Realistic and large) network simulator. It has always had substantial contributions from other researchers and been used to develop and investigate various effective research studies, including:

- TCP behaviour: selective acknowledgement, explicit congestion notification, etc.
- Router queuing polices: random early dedication, and class-based queuing etc.
- Multicast transport multimedia: layer video, audio and video quality of service etc.
- Wireless networks: snoop and split-connection TCP, and multi-hop routing techniques etc.
- Application level protocol: web cache consistency protocol.

ns-2 uses a split programming model, where the simulation kernel is implemented in C++ language while the simulation is defined, configured, and controlled using OTCL (Objected-Oriented Tool Command Language) scripts.

The main strengths of ns-2 include being able to download it freely on a variety of platforms. Moreover, there are many available online resources, handbooks, tutorials, and a comprehensive list of ns-2 codes is useful for learning and simulating simple and complex models in ns-2.

However, ns-2 lacks graphical presentation tools and the raw data of the output file must be processed using other tools (e.g. Perl, awk) to produce a suitable format of data. Another weakness of ns-2 is its command line interface which makes the simulator unfriendly for users compared to OPNET and OMNeT++. In spite of the comprehensive documentations and tutorials of ns-2, its split-programming model remains a barrier to many developers.

3.2.4 Network Simulator-3 (ns-3)

ns-3 ("ns-3 Project", n.d.) is a discrete-event simulator targeted for research and educational use. It is a publicly available simulator for research, development, and use. The main goal of the ns-3 project is to develop an open simulation environment for modern networking research needs. Thus, it should encourage the research community's contribution, peer review, and validation of the package.

The ns-3 project builds a solid simulation core that is easy to use and debug (compared to ns-2), is well documented, and caters to the needs of the simulation workflow, starting from configuring the simulation experiment until the data collection and analysis. ns-3 infrastructure encourages the development of realistic simulation models which allows many real-world protocols and technologies to be reused within ns-3 simulation. Furthermore, ns-3's core supports both IP and non-IP based networks. It is focused on such wireless/IP simulations as Wi-Fi, WiMAX, and

LTE for layer 1 and 2. In addition, ns-3 provides the capability of reusing the real application and kernel code (e.g. Linux kernel networking stack, TCP/IP stack, and etc.).

Every three months, the ns-3 project ships a new stable version of ns-3 simulation with new models developed, documented, validated, and maintained by enthusiastic researchers speared over a large community of users and developers. However, ns-3 is not an updated version of ns-2, although several mechanisms have been redesigned based on successful and unsuccessful aspects of ns-2. The main features of ns-3 are listed below:

- The core of ns-3 is written in C++ with Python scripting interfaces
- Most of the protocols implementations in ns-3 are designed to be more authentic
- It supports the incorporation of more open-source networking software
- It supports the integration of virtualization and a testbed
- It enables a customizing tracing and statistics gathering without rebuilding the simulation core
- It permits callback-driven events and connections
- It has a flexible core with a helper layer.

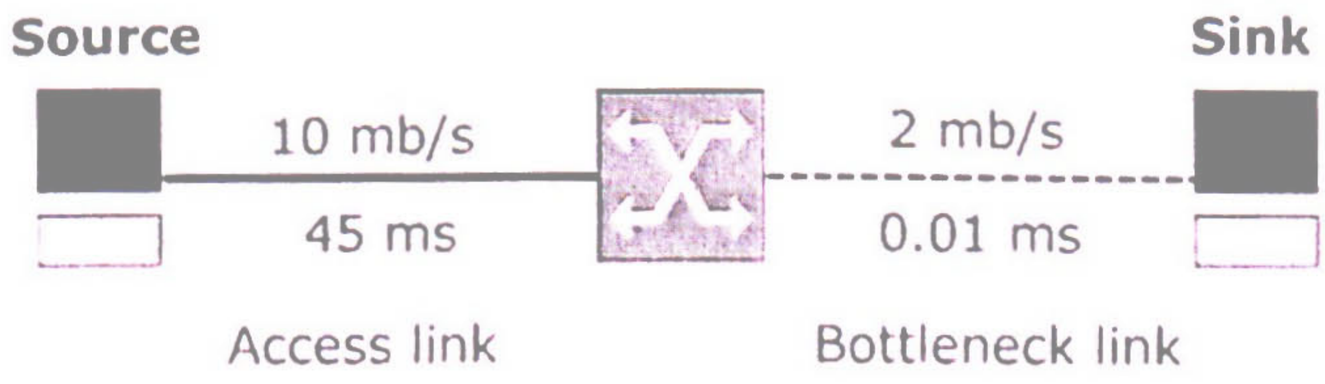
However, ns-3 lacks an Integrated Development Environment (IDE) to configure, debug, execute, and visualize simulations in a GUI window. However, for the purpose of this study, we have selected ns-3 to conduct all the simulations presented. We found that ns-3 simulation fulfils the criteria of selecting a network simulator as mentioned in section 3.2.

3.3 Experimental Design

As presented in previous sections, the evaluation process was conducted using a simulation approach. The network simulator ns-3 was selected to perform all the simulation experiments in this study. The design of the simulation experiment plays a major role in gaining useful results, which are well structured and valid. The experiments were carried out by defining the simulation topologies, selecting the statistics, running the simulation scenarios, and collecting, processing, and analyzing the results.

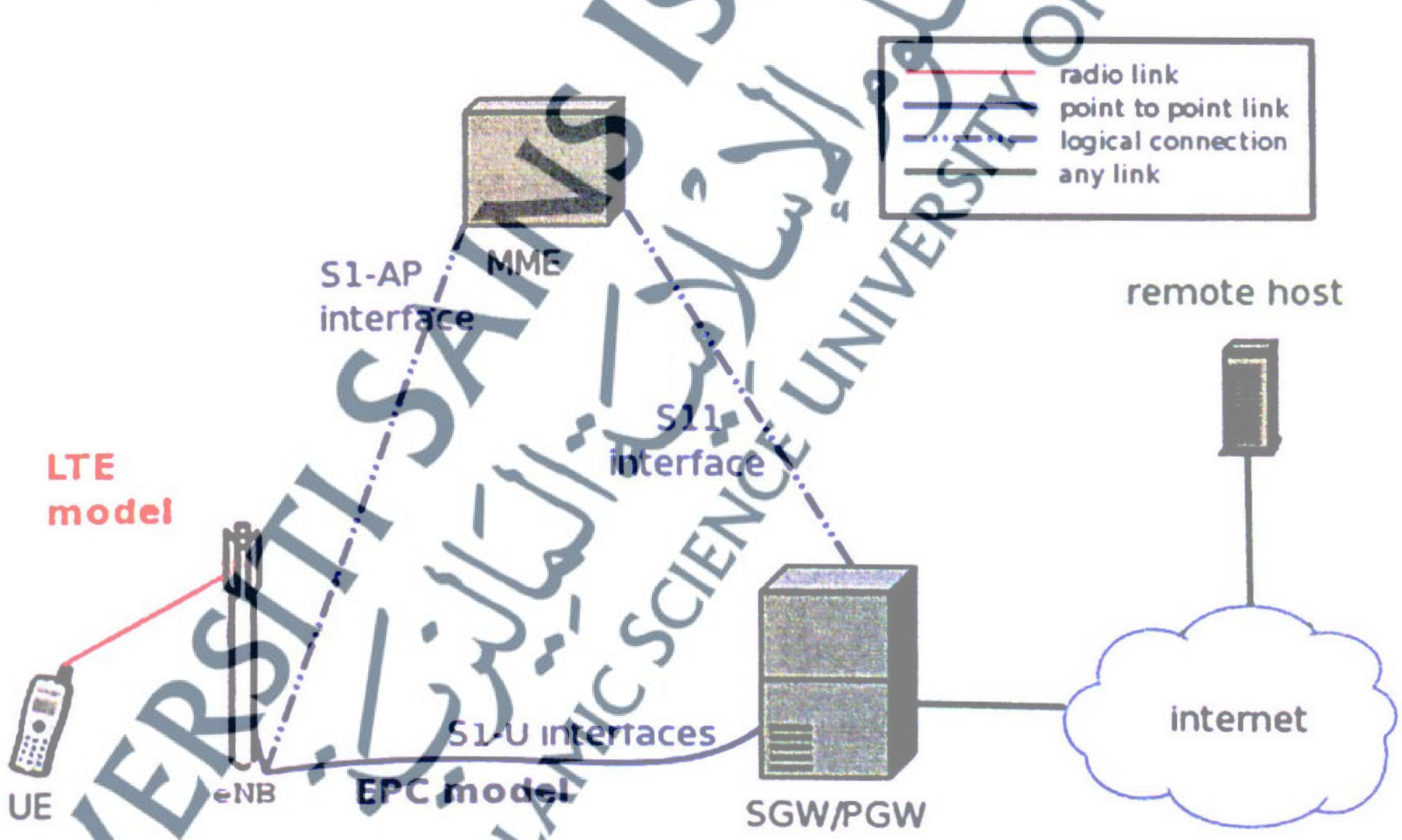
For the simulation topologies, we used four different topologies to evaluate the proposed new modifications PETRA. For a consistent comparison of the first topology, we used the same experimental scenario used to introduce TCP Westwood (Mascolo et al., 2001). The topology is shown in figure 3.3. A single source and sink were connected via a gate-way (PGW). A PointToPointHelper (ns-3 Documentation, n.d.) was used to connect the source and sink to the PGW. ns-3's BulkSendApplication class was used to generate continuous traffic started at the source and ended at the sink along the simulation period. To simulate the wireless link loss environment, we used the RateErrorModel class to generate uniform distributed errors along the link between the sink and PGW.

Figure 3.3: First Simulation Topology



For the second simulation model, we adopted the LENA model (ns-3 LTE module, n.d.) to implement the LTE-EPC simulation model. There are two main components as we can see in figure 3.4, the LTE model. This model includes the LTE radio protocol stack (RRC, PDCP, RLC, MAC, and PHC) and these entities reside in the UE and eNB. In the second component there is the Evolved Core Packet (EPC). This model includes the core network interfaces, protocols and entities which reside in the SGW, PGW and MME node, and partially within the eNB node. We used this topology to test the new TCP congestion control modification PETRA, and then to compare the results to the NewReno and TCPW implementations. This topology is discussed in more detail in chapter 5.

Figure 3.4: LTE LENA Model (adopted from ns-3 LTE Module)



For the third topology, we have extended the previous LTE-EPC simulation model to have a somewhat dumbbell topology. We tested PETRA performance over a congestion LTE network, and we used the same topology to test the fairness of PETRA among various connections flows. More details about this topology and its parameters are presented in chapter 5, section 5.4.2.

For the last topology, we tested PETRA over a mobility environment. We used the same LTE-EPC simulation model in the second topology, but we include the mobility feature to the User Equipment (UE). The details of the simulation model are discussed in depth in chapter 5, section 5.4.3.

3.4 Experimental Hardware and Software

The hardware used for the simulation was OPTIPLEX 740 Dell desktop with an AMD Athlon Dual Core Processor – 4200 with a 1GB and hard disk of 300GB. The installed Operating System is Fedora Release 12 with Kernel Linux 2.6.31.5-127.fc12.x86_64. The simulator used for all the experiments was ns-3 version 3.21.

C++ was used to implement all the modifications to the algorithms presented in this research. We used gedit version 2.28.0 as a text editor where the C++ code was written. GNUPLOT version 4.2 was used to plot all the graphs presented in this study.

3.5 Validation and Verification

The essential purpose of testing is to make sure that a simulation model behaves "correctly". Generally, there are two perspectives from which one can view correctness in network simulation (Hassan & Jain, 2003; Michael, 2001): the model verification, and the model validation.

Every simulation model has to follow its target model specifications. The first step in creating a simulation model is defined its target model, the details of the target model, and the accuracy that the simulation seeks to reproduce. At the end of this step, the abstract model was designed. The process of developing the abstract model from its target model is called model qualification. The abstract model is then developed into a computer program to produce a simulation model. This process of getting the

implementation to agree with the abstract model is called model verification. Once the simulation model is verified, then another process is started which validates the model's behaviour to ascertain whether it agrees with the desired target system behaviour as defined in the model qualification process. This process is called model validation in the literature.

ns-3 provides testing environment tools to allow both model verification and testing, and encourages the publication of validation results. All ns-3 tests are classified into either test suites or test cases. A test suite is a collection of test cases that exercise a kind of functionality. Test suites are classified into:

- Build Verification Tests: these tests are used to make sure that the build is working
- Unit Tests: these tests are used to make sure that a piece of code works as advertised in isolation
- System Tests: these involve more than one module in the system. For example, the NSC TCP test is used to test the ns-3's TCP implementation
- Example tests: used by the framework to ensure that all the built examples will run
- Performance Tests: used to determine if a particular part of the system is executed within a reasonable time
- Running Tests: used to provide, add, and obtain test suites for a collection of tests.

On the other hand, test cases are used to create new test cases in the system. These tests can be a one test case per feature, one test case per method, or mixtures of these models can be used. To create a new test case in ns-3, inherent from TestCase

base class, we override a constructor to give the test case a name, and override the DoRun method to run the test.

In this research, we used the ns-3 verification and validation tools to confirm the correctness of our simulation model. Then we validated our proposed modification by comparing the results of network performance metrics with the results of two TCP implementations: TCPW and NewReno.

3.6 Validation of the Network Simulator

After setting-up the sn-3 simulator, many testing tools can be used to validate the installed simulator. One tool is 'test.py'; this tool triggers the validation process by running specific simulations with predefined input values and then compares the results with the known output values. The process will loop until all available tests have been run and then a status report is sent back. The report is a result in a number of PASS, FAIL, CRASH, or SKIP indications followed by the kind of the test run and its display name. Figure 3.5 shows a portion screenshot of the Test.py report for the research simulator ns-3 version.

Figure 3.5: ns-3 Test.py Report

```
[root@Mohanad ns-3.19]# ./test.py
Waf: Entering directory `/home/Mohanad/ns-allinone-3.19/ns-3.19/build'
Waf: Leaving directory `/home/Mohanad/ns-allinone-3.19/ns-3.19/build'
'build' finished successfully (11.167s)

Modules built:
antenna                  aodv                     applications
bridge                   buildings                config-store
core                     csma                     csma-layout

      :

PASS: TestSuite lte-ue-measurements
PASS: TestSuite test-asn1-encoding
PASS: TestSuite lte-handover-delay
PASS: TestSuite lte-x2-handover
PASS: TestSuite lte-rrc
PASS: TestSuite lte-harq
PASS: TestSuite lte-mimo
PASS: TestSuite lte-phy-error-model

        :

281 of 284 tests passed (281 passed, 3 skipped, 0 failed, 0 crashed, 0 valgrind
errors)
[root@Mohanad ns-3.18]#
```

We have used this tool to validate the version we used in this research. All the tests as seen in the above figure were "PASS"; however, three tests were skipped (wimax-qos, wimax-phy, and wimax-tlv).

3.7 Performance Metrics

As a step toward evaluating our findings in this research, we experimented and compared the performance of the new modification with two TCP implementations: TCP Westwood and NewReno. To do the comparison, a set of network performance metrics were selected by referring to Floyd (2008). These metrics include throughput, packet loss, and average delay. Also, we have included more metrics such as congestion window size, and fairness to confirm the performance enhancement of the new modification.

3.7.1 Congestion Window Size

A TCP connection uses flow control and congestion control mechanisms to govern the data transmission. In flow control, the TCP controls the data flow according to the receiver advertise window (*rwnd*). On the other hand, congestion control limits the amount of data injected to the network to the amount of data the network can transmit. Therefore, the sender side maintains a variable, the congestion window (*cwnd*) that determines the number of packets that can be outstanding at any time. This is the minimum value of both windows bounding the sending rate (Sharma et al., 2014).

The congestion window size is an important factor that affects transport protocol performance, and the bigger the congestion window, the better the bandwidth utilization. Permanent enhancements and perfections of the TCP congestion control mechanisms aim to preserve a bigger congestion window size.

In this research, we record and monitor the *cwnd* behaviour of the new modified algorithms using the ns-3 simulator. Then, we will compare the results with other TCP implementations using plotted graphs.

3.7.2 Throughput

According to RFC 1242, throughput is defined as the maximum rate at which none of the offered frames are dropped by the device (Bradner, 1991). Throughput usually measures in bits per second (bps) or packets per time slot (usually per seconds). A network throughput or an aggregate throughout is the sum of data that is successfully transmitted and received to all the network's terminals. It is a clear goal of most new congestion control mechanisms to maximize the system overall throughput.

There are two ways to calculate the throughput; the indiscriminant throughput and the goodput. The indiscriminant throughput is the amount of bytes received by the destination regardless of whether it is a retransmitted data or not. On the other hand, goodput is the useful traffic amount excludes the retransmitted packets, downstream dropped packets, and packet fragments etc. In this research, we used the second type of throughput calculation. Throughput is used to measure the communication channel performance. Various factors may affect a communication system throughput, including the limitations of the physical medium, terminals and intermediate nods specifications, as well as the limitations of the transport protocol itself.

In this research, we measured the throughput as the total bytes transmitted and successfully received at destination within the simulation period in a unit of MByte per second. For any given time, (t) and $t>0$, and N is the number of bytes transmitted in $[0,t]$; we calculate the throughput ($B_{t)}$ as in equation (3.1).

$$Bt = \frac{(N\ /1024*1024)}{t} \tag{3.1}$$

To measure the value of N we used the flow monitor module (ns-3, n.d.) provided by ns-3. The flow monitor provides a flexible system to measure the performance of network protocols. The module can be installed in any node to track the packets exchanged by the nodes and measure various statistics to be exported in XML format. The user can also access these statistics directly to request specific stats. We used the ns-3's flow monitor to trace every packet received by the sink using a void function called SinkRxTrace to calculate the rxBytes. More details about the flow monitor could be found at (ns-3, n.d.).

3.7.3 Packet Loss

Another important network performance metric is packet loss. In this research, we refer to packet loss as the number of dropped packets along the simulation period. There are two ways to calculate the total number of dropped packets. This is done in two ways: by tracing every retransmission event at the sender side either a timeout events or duplicate acknowledgments causes, and by tracing both the total number of packets sent and received. We used the second method to calculate the total number of dropped packets as in equation 3.2:

$$DP = TotalPacketSent - TotalPacketReceived \tag{3.2}$$

To measure the *TotalPacketSent*, and the *TotalPacketReceived* we used the flow monitor module to trace both the txBytes and rxBytes respectively. Where txBytes is the total number of transmitted bytes and rxBytes is the total number of received bytes.

3.7.4 Average Delay

Delay can be measured as a router-based metric of queuing delay over time or in term of per-packet transfer times as a flow-based metric (Floyd, 2008). A flow-based delay includes nodal processing, transmission delay, and propagation delay. Nodal processing includes check bit errors, delay at the sender waiting for transport protocol to send packets, determine output links, etc. On the other hand, transmission delay includes acknowledgments store and forward delay, link bandwidth, and time to send bits into the link, etc. Finally, propagation delay is the medium speed of transmission of a packet which is equal to the length of a physical link divided by the propagation speed in medium (Demichelis & chimento, 2002).

In this research, we measure the average delay as the summation of all end-to-end delays for all the received packets of the flow divided by the number of received packets at the receiver side, as in equation 3.3.

$$AverageDelay = \frac{\sum_{i=1}^{n} DelayofPacket}{n} \quad (3.3)$$

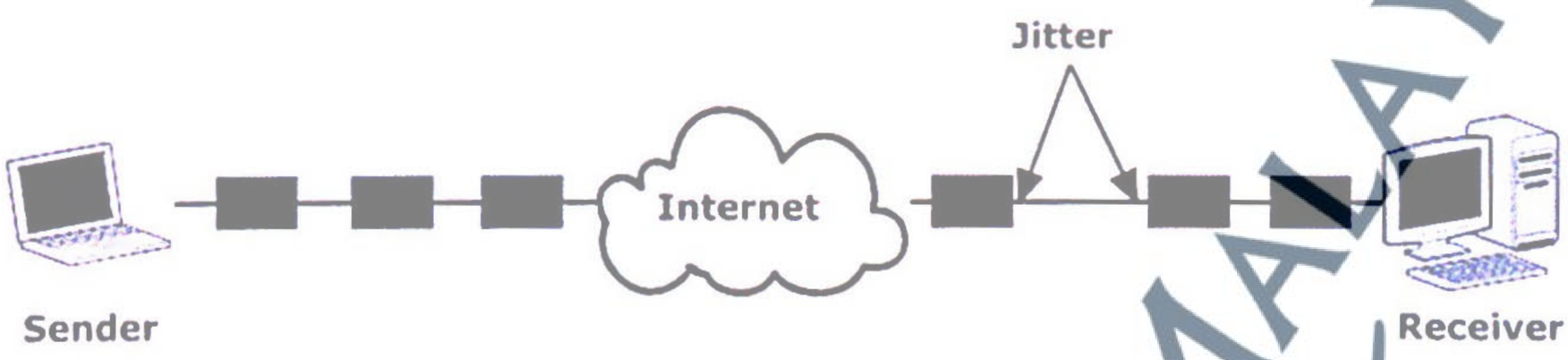
Where, the *DelayofPacket* is the summation of all the received packets delays calculated by the flow monitor module at the delaySum traced value.

3.7.5 Jitter

Jitter is the packet delay variation (Demichelis, 2002). The term "Jitter" has caused confusion because it is used in different ways by different people. This term generally has two meanings. Firstly, it means the variation of a packet delay with respect to the

clock signal (reference to synchronous signals). Secondly, it means the variation of some metrics with respect to some reference metric (e.g. the variant of packets delay to the average delay). However, a network with constant latency has no jitter (i.e. jitter= 0). Figure 3.6 describes the meaning of jitters.

Figure 3.6: Packet delay variant (Jitter by the internet)



In this research, we used the flow monitor module to measure the value of jitters as defined in RFC 3393, using the jitterSum build-in variable. jitterSum traces all the end-to-end delay variation values of all received packets of the flow.

3.7.6 Fairness

Fairness is an important issue to be considered when implementing a new TCP congestion control mechanism. According to Jaffe & Gerla (1981), a connection is fair if "each user's throughput is at least as large as that of all other users which have the same bottleneck". A Fairness test is used to ensure that users and applications access the available network resources equitably to provide a best-effort service. The TCP unfairness problem results when multiple connections with different round trip times share the same bottleneck, whereas long *RTT* connections increase their congestion window more slowly than short *RTT* connections.

In TCP congestion control mechanisms, there is limited agreement in the literature about what should be equalized, equal delay, equal throughput, or equal

power for all users to share a resource. However, in this research we tested the fair bandwidth sharing of the new congestion control mechanism using Jain's fairness index (Jain, 1984). This index is applicable to any number of users and connections, independent of the amount of resources, and it is bonded between 0 and 1 to provide intuitive understanding of the fairness index result. For any set of users throughputs (T_1, T_2, ..., T_n), Jain's fairness index is given as in equation 3.4.

$$Fairness\,(throughput) = \frac{\left(\sum_{i=1}^{n} T_i\right)^2}{n\left(\sum_{i=1}^{n} T^2{}_i\right)} \tag{3.4}$$

Where n is the total number of TCP users. If all users receive equal throughput then the fairness index will equal to 100%. On the other hand, a fairness index of 90% means 90% of the bottleneck bandwidth is shared fairly among the connections.

3.7.7 Friendliness

TCP-Friendliness is a generic term that describes a scheme which aims to use no more bandwidth that TCP uses (Polishchuk & Gurtov, 2010). In other words, a flow that used different transport protocol should have the same throughput as TCP flow if they are competing for the same bottleneck. TCP-Friendliness emerges as a measure of congestion control mechanism correctness. Usually, protocols meet this requirement by using some form of TCP congestion avoidance mechanism as AIMD (Additive Increase Multiplicative Decrease). A TCP-friendly scheme should be able to use the same bandwidth as TCP during the steady-state phase, aggressive enough to use the available bandwidth, and responsive to protect the flow from congestion (Widmer et al., 2001).

For the first moment, fairness and friendliness appears to be the same term. However, Fairness aims to validate the fair share of the available bandwidth among different flows that run the same transport protocol. On the other hand, friendliness aims to validate the fair share among different flows that run different transport protocols.

3.8 Conclusion

In this chapter, we have discussed the methodology of evaluating the proposed congestion control algorithms, PETRA. Firstly, we have introduced a brief background study about approaches to evaluating network performance. Then, we elaborated more about the network simulation approach selected to evaluate the performance of the proposed modifications. Moreover, we discussed the main network simulators. In addition, the designed experiments were also introduced in this chapter. Finally, we introduced the performance metrics used to evaluate the proposed modifications in this research. The next chapter will introduce three new modifications to TCPW congestion control mechanism.