Thesis Project

Performance evaluation of routing protocols for Wireless Mesh Networks

Author: Spyridon MARINIS ARTELARIS
Supervisor: Ola FLYGT
Examiner: Johan HAGELBÄCK
Semester: HT2015
Subject: Computer Science

Abstract

Wireless Mesh Networks provide an organisation or a community with the means to extend or create a network independent of infrastructure. However, the network's dynamic topology along with the fact that devices in the network might be mobile and move randomly, brings to light various kind of problems on the network, with the most common being the routing. In this report, the problem of routing is examined in terms of throughput, routing overhead, end-to-end delay and packet delivery ratio on two chosen algorithms, namely the Dynamic MANET On-demand (DYMO) [1] and the Better Approach To Mobile Adhoc Networking (B.A.T.M.A.N.) [2]. Furthermore, this thesis examines also a Transmission Control Protocol (TCP) connection and compares it against several TCP congestion control mechanisms, two of which, were implemented, namely TCP-Illinois and TCP-FIT, to address the effects that different TCP congestion mechanisms have on an ad-hoc network, when reliable connections are needed.

The results show that DYMO is more stable, performs good overall and has the lowest routing overhead, however in a situation with limited mobility or no mobility (as in high mobility they perform poorly) proactive protocols like B.A.T.M.A.N. are worthy protocols, should the extra penalty of routing overhead in the network traffic is not causing any problems. Furthermore, regarding the TCP results, it was observed that TCP congestion algorithms designed specifically for Wireless networks, do offer better performance and should be considered, when designing an ad-hoc network.

Keywords: Wireless Mesh Networks, ad-hoc, DYMO, BATMAN, TCP, UDP, congestion, OMNET, INET, routing, performance, TCP-Illinois, TCP-FIT

ACKs

To Seva Meyer. For her support, discussions and understanding during the long months of writing and re-writing of the report.

To Ola Flygt my supervisor. For his valuable knowledge and support and for steering me in the right direction whenever it was needed.

To my family. For their support, encouragement and befief in me.

Contents

1 Introduction

Wireless networks transformed our lives and ways to connect to the Internet or another network seemingly and transparently, provided one is under the "umbrella" of an access point. Wireless Mesh Networks extend this connectivity to far greater distances.

Wireless Mesh Networks or WMNs for short are a rapidly growing and very promising technology. Similarly to the paradigm shift that created the Internet in 1960s, this technology is expected, due to its self-healing, self-configuration, self-organisation and broadband capacity, to play an important role in the direction of future wireless ad-hoc networks. Wireless Mesh Networks are desirable, as they are: easy to deploy and maintain, less expensive than other solutions with high scalability, resilience and reliable services. These characteristics are pushing the international organisations that create standards to actively look out for specifications, for instance, IEEE 802.11s (*WMN*), IEEE 802.11 (*WLAN*), IEEE 802.15 (*WPAN*), IEEE 802.16 (*WMAN*) and IEEE 802.20 (*MBWA*).

Overall, each experience we gain by implementing, studying and deploying Wireless Mesh Networks will provide us with the necessary tools in terms of knowledge to build a foundation for the evolution of future networks.

1.1 Background

Wireless Mesh Networks, due to their diverse nature are becoming good candidates for an increasing amount of applications. This cost effective, readily available and easy to configure and install technology, has found home in extreme environments where there is no communication infrastructure and aids in network coverage over long distances. Nowadays, almost all wireless capable devices are supporting ad-hoc networking and WMNs are looking fearless into the future.

Nodes in Wireless Mesh Networks can be any devices, that can play the role of a router or repeater for all other nodes in the network. Usually, only one node is required to be wired physically to an Internet gateway or other network provider. The nodes in a Mesh Network can be any device that supports wireless functionality or in other words any device of the existing network infrastructure or even mobile devices that enter the network. As a result, the network can reach great distances and create non-line-of-sight connections even on extreme environments like dams, quarries, construction zones and isolated areas. Hence, Wireless Mesh Networks are highly desirable in situations, where there is no communication infrastructure like natural disasters, army operations and health care professionals.

These reasons, make Wireless Mesh Networking an interesting and promising field of research, as its utilisation might bring the evolution of future networks. For instance, in some continents of the world there exists no support or there is limited support - it terms of infrastructure - for an Internet connection. Thus, with this technology, instead of building a network infrastructure an organisation or individuals could simply cover a specific area with inexpensive wireless devices, and provide its habitats with the benefits of an Internet connection or in broader terms, any kind of network.

Despite the fact that ad-hoc networks look very similar to the Wireless Mesh Networks, they also have many differences; mainly due to the fact that ad-hoc networks are built with the consideration of high-mobility, whereas WMNs are built with limited mobility in mind. As a result, a protocol designed for use in an ad-hoc network might perform poorly when deployed in a Wireless Mesh Network. Furthermore, WMNs have quite diverse topologies, which lead to the design or modification of protocols in order for a protocol to take advantage of them. What is more, WMNs are suffering from interference of other devices that share the same spectrum, which as a result limit considerably their capacity.

Research in this area is ongoing, and the evaluation in the performance of WMNs has been conducted with numerous network simulator environments (both commercial and free) [3], [4], [5]. As expected, the network simulator enviroments give different results, and unfortunately there are some factors like energy consumption, environment temperature, object modelling etc., that cannot be fully addressed or are difficult to address in such an enviroment.

1.2 Previous research

Performance comparisons on Dynamic MANET On-demand (DYMO), Ad hoc On-Demand Distance Vector (AODV), Dynamic Source Routing (DSR) and Destination-Sequenced Distance-Vector (DSDV) routing protocols have been conducted in the recent past. According to [6], DYMO proved to be a better routing protocol over the others. Also according to [7], a research was conducted on which routing protocol has better packet delivery ratio (PDR), average end-to-end delay and routing overhead.

The simulation results showed that DYMO, still performs better than AODV. It has less end-to-end delay than DSR and DSDV in the scenarios tested. However, DYMO has a slightly lower packet delivery ratio than DSR and DSDV.

1.3 Problem definition

Wireless Mesh Network protocols, are highly affected by the environment, obstacles like houses, cars and RF interference might hinder their performance and further introduce problems in seemingly connecting mobile clients that enter our networks or affect the coverage area and their capacity. Thus, by utilising a network simulator several protocols will be examined and a conclusion of which protocols best serve specific scenarios will be drawn.

It is good to mention that this conclusion will not be terminal, as due to the diversity of the environment a Mesh Network is deployed, there is no absolute certainty that a protocol that was performing well in the simulation, considering all the limitations, will still perform the same, in a similar real environment.

1.4 Purpose and research questions

There are many routing protocols developed for Wireless Mesh Networks with competing or similar features. As a result, choosing the right routing protocol is crucial, due to all the different qualities that might fit in a variaty of wireless routing scenarios. This report aims to address the following three questions.

1. *Which routing protocol in a Wireless Mesh Network performs better in terms of throughput, routing overhead, packet delivery ratio and end-to-end delay?* By answering this question the overall performance of the routing protocols will be addressed, and the best protocol will be chosen.

2. *Which factors might influence their performance?* This question will investigate, which factors like interference, noise etc., can influence the performance of a protocol.

3. *Do TCP congestion control algorithms provide better performance in an ad-hoc network in terms of throughput?* TCP congestion control is a crucial and integral part of TCP. As a result, this question will investigate, through the different scenarios, whether algorithms written for wireless networks can have any significant difference over algorithms written for wired connections.

1.5 Scope

This thesis will investigate two Mesh Networking protocols, namely DYMO [1] and B.A.T.M.A.N [2] and draw a statistical comparison between them in

order to understand which protocol performs better under specific circumstances. Those protocols were chosen since they utilise different routing strategies. DYMO was chosen as is considered a successor of AODV and can work as both pro-active and reactive routing protocol; in this report it will be used as reactive. B.A.T.M.A.N is a proactive protocol and was developed as a replacement to Optimized Link State Routing (OLSR) protocol.

Other limitations, that might be faced during research is that network simulators cannot give the most accurate results, since they cannot simulate real world conditions. However, there are some desirable features that this thesis will try to address, provided there is support from the network simulators. Those features could be the introduction of noise and obstacles in the mesh network, which will aid in drawing a more concrete comparison between the protocols. Another limitation, is that both protocols that are under investigation are quite outdated.

1.6 Outline

This report will be divided into 7 chapters including the introduction. In brevity, chapter 2, will describe the methodology of the research. Continuing from chapter 2, in chapter 3 the State of the Art will be presented by reviewing the theoretical concepts of mesh networking, TCP congestion control, the routing protocols and several important properties of Wireless Mesh Networks. The following chapter, chapter 4, will introduce several aspects related to the frameworks and the implementation of the algorithms. Chapter 5, will report and analyse the results as obtained in this study. chapter 6, is the discussion of the results, and lastly chapter 7 is where the thesis conclusion will be made.

2 Method

In this chapter, the thesis methodology and the different considerations behind its rationale will be presented.

2.1 Scientific approach

To produce this report three approaches were followed, namely literature study, implementation and testing.

2.1.1 Literature Study

By utilising online digital libraries like IEEE xplore, Association for Computing Machinery (ACM), and Google Scholar, several papers were collected and analysed.

The collection of those papers was done in several steps. Firstly by searching with relevant terms, such as "wireless mesh network routing", "adhoc routing protocols", "manet routing evaluation performance" and other similar. Secondly, the abstracts of the papers, which were the result of the search queries, were examined to narrow down possible candidate papers and thirdly, all candidate papers were read and either kept for later reference or discarded as not relevant. Overall, the study started with around 50 papers which were further reduced to 24.

Furthemore, the development and user manuals and online documentation of both OMNeT++ [8] and INET [9] frameworks were studied to obtain a better understanding on building simulations.

2.1.2 Implementation

To produce all the required results many scenarios had to be implemented and several modifications to be made to some of the existing models and are available online [10]. In addition, two TCP congestion control algorithms will be implemented and utilised. The development and testing environment is OMNeT++ [8] itself, which is based on Eclipse and provides all the necessary tools and facilities to a developer. Moreover, the statistical results and data produced by each of the scenarios will be the basis for a quantitative study.

To measure the throughput of the network both TCP and UDP protocols will be utilised. Regarding the end-to-end delay and packet loss measurements, it is recommended to use the User Datagram Protocol (UDP), mainly because the protocol's simplicity allows the measurement of the end-to-end

delay or one-way delay as it also called. However, an effort to measure this delay with TCP will be made by halving the Round Trip Time, which results in the average end-to-end delay. Further, to measure the routing overhead, most of the ready implemented routing protocols provide statistics for those measurements and it is only required to capture them, unless adding extra code in any of the implementations seems that is needed. For the complete configuration of the different simulation parameters refer to Appendix 1.

2.1.3 Testing

The testing will be mostly empirical as it will rely on observing the behaviour of the network's nodes and adjusting the configuration were needed.

The testing of the two TCP congestion control algorithms will be made by using a simple Client-Server simulation application already provided by INET [9] for observing other already implemented TCP congestion control algorithms, and the data produced will be examined against the relevant papers, which describe those algorithms.

2.2 Analysis

OMNeT++ [8] provides all the necessary tools for data acquisition by the form of vector and scalar measurements. After all the scenarios produce their results, further calculations will be made if required prior to forming a data collection. This data collection can be further exported into Comma-Separated Values (CSV) and processed by a spreadsheet application, to visualise it in the form of graphs, should this required.

2.3 Reliability

Using a simulation environment for conducting research is by itself limiting, due to the lack of a real environment and equipment. Consequently, the results produced in this report are accurate and valid up to a certain point, and should be considered as a basis and not the rule for further research experiments.

2.4 Reproducibility

The experiments are fully reproducible, and it is possible to re-create them by utilising the frameworks and the given project's code.

2.5 Ethical Considerations

This thesis project adheres to all regulations, as stated by Linnaeus University, concerning use of publishing research findings, patented work and copyright. Meticulous citing and referencing has been employed to signify use of other people's ideas and results.

All the results and data collected for this thesis is being carefully reported and has not been altered in any way to produce false findings.

3 Networking Basics

In this chapter the different notions and concepts of ad-hoc networks will be presented.

3.1 Introduction

It is important to define Wireless Mesh Networks, in order to dissolve and avoid any misconceptions, later during the research and implementation. Thus, the reader will be introduced into ad-hoc networks and their important properties, the different types of routing protocols along with a description of the routing protocols that will be simulated, the evaluation criteria, the simulation environment and finally, in brevity, the TCP congestion control mechanisms.

3.2 Ad-hoc Networks

Ad-hoc is a Latin phrase meaning "for this" or "for this situation", its relation to networks is the combination of internetworking devices to form a network without any required planning. Ad-hoc networks and Mobile ad-hoc networks (MANETs) are considered a subset of Wireless Mesh Networks (WMNs) and it could be said that, ad-hoc networks are multi-hop, infrastructure-less and decentralised wireless networks.

Further, due to the vision of ad-hoc networking in challenging communication channels, where problems such as hidden and exposed nodes, shared bandwidth and transmission impairments, communication is not always guaranteed to be bi-directional. Even if a router A can communicate with a router B and the router B with a router C, that does not guarantee that the router A and the router C can communicate with each other directly. Moreover, ad-hoc networks can utilise more than one communication channels; that is internetworking devices could make use of one or more wireless channels [11].

This type of networking is particularly useful in situations where providing the necessary infrastructure is not practical, either due to network's nature or due to environment restrictions. As a result, it is now very common to find ad-hoc networks in large number of scenarios, such as military (which is where WMNs have emerged), emergency and rescue services, rural environments, etc.

3.3 Standards and amendments

Wireless Mesh Networks were first defined in the 802.11s standard, which superseded by 802.11 WLAN standard at 2012.

3.3.1 802.11s amendment – Mesh Networking

With the need of the organisations to expand their networks, beyond the limits of the corporate building, along with the Ethernet cable limitation to 100 meters, organisations lead to adopt Wireless Networking in an exponentially growing rate. However, Wireless Access Points, paradoxically, need a wired connection to a switch or other similar device. As a result, the coverage of indoor areas like warehouses, etc. introduced more problems as either the needed infrastructure was missing or the cables could not reach in specific distances.

This problem worsens with the need to cover wirelessly outdoor areas like parking lots, campuses or even industrial complexes or entire cities where there might be a demand of supporting municipal, public and emergency services. In consequence, in order to address those limitations a combined effort created the 802.11s amendment at September 2003 which superseded many years later by the 802.11 standard in 2012.

The 802.11s amendment defined the concept of a Mesh Station [12]. A Mesh station is a station, capable of joining the Mesh Basic Service Set (MBSS) and that supports the Mesh Facility. The Mesh Facility, simply put is the set of all functions, frame formats and features that allow mesh operation of a device. In theory, the term Mesh Station indicates an access point, however, even a non-AP STA (client device) can be a Mesh Station.

3.3.2 802.11 Standard – Wireless Local Area Networks

This is the de facto standard for WLANs and it covers any small or big detail of their operation on all layers. 802.11 Standard describes from 2012 onwards all the MAC changes required to support wireless LAN mesh topologies. According to the standard [13], a group of STAs (Stations) form a mesh BSS (Basic Service Set) or a 802.11 LAN in broader terms.

In this LAN, all the STAs establish links with neighbour STAs in order to exchange information. Since, wireless networks are commonly multi-hop, information can be transmitted over a single instance of the medium to other STAs which are not in direct communication to each other. The multi-hop capability aids and enhances the range of STAs. Superficially, a mesh BSS

appears as all the STAs, even those not in range, are directly connected to each other at the MAC layer.

A mesh STA or "mesh station" supports mesh services, implements a subset of QoS functionalities, and participates in the formation and operation of a mesh BSS. Further, a mesh STA utilises the Mesh Coordination Function (MCF) in order to access the channel [13].

3.3.3 Mesh BSS and 802.11 components

For the formation of a MBSS and its functionalities such as path selection and forwarding, only Mesh STAs can participate and cannot communicate with non-Mesh STAs. However, in case an MBSS has access to the distribution system (DS) then there can be communication between non-Mesh STAs and Mesh STAs.

The component that acts as the logical point for the integration of the MBSS with the DS is called the Mesh Gate, and there can exist more than one. Further, a MBSS can be integrated through a Mesh Gate, with other infrastructure BSSs should the APs connect also to the DS, and other non-IEEE 802.11 LANs. An Access Point (AP) can function as any combination from a portal and a Mesh Gate [12].

3.4 Important Properties

Wireless Mesh Networks have several properties which make them attractive and desirable in many applications. Those properties are self-configuration, self-healing and self-forming.

3.4.1 Self-Configuration

Wireless Mesh networks reduce the effort required by a network administration to adjust and configure new nodes that enter the network, due to the fact that mesh networks learn and incorporate new nodes automatically [14].

3.4.2 Self-Healing

As stated before, since Mesh Networks learn, adjust and adapt to network changes. Network changes can be considered: a link to a node that fails or a node itself fails and the route is not valid or that of a transmission impairment (signal attenuation, dispersion, noise etc.), which as a result, all the nodes will seek alternate paths in order to converge the network. This capability, to seek alternate paths, in a Wireless Mesh Network is called self-healing [14].

3.4.3 Self-forming/Self-organising

The previous two capabilities result in a third important capability of Mesh Networks that of Self-forming or Self-organising. A Mesh Network has implemented functions for network and topology discovery, which allows the extension of the area of coverage. In other words, nodes can rapidly detect other newly introduced nodes and consequently their paths, which make the network really flexible and scalable [14].

However, there might be constraints regarding the number of nodes a network can incorporate and it is based on the physical and technical environment of an organisation. By physical environment it is meant obstacles like houses, trees, hills etc., and that of RF interference which may lead to transmission impairments. As a result, the capabilities of the routing protocol and processor of each node will dictate their number.

3.5 Overview of routing protocols

Wireless Mesh Networks have mainly three types of topology-based routing protocols, reactive, proactive and hybrid. There are also other types, such as position-based routing protocols, which will not be discussed as they are out of the scope of this report.

3.5.1 Reactive

This type of protocols will create the necessary routes towards a destination, only when a source requests a route. In other words, when a source node asks for the address of another node, the route discovery mechanism will be called and the network will be flooded with special Route Request packets.

Each node with this mechanism will learn about its immediate neighbours and obtain one or multiple routes towards the destination. When this mechanism is fulfilled the route discovery will conclude and the source will be able to communicate with the destination node [15].

3.5.2 Proactive

As the name suggests nodes with proactive protocols will store and maintain a routing table with up-to-date information about their immediate neighbourhoods. Each node will propagate route updates to other nodes in order to keep the routing information as current as possible.

The main goal of those protocols is to keep the overhead of route maintenance to a minimum, if possible; and each protocol will record different route state information, like broken links, etc [15].

3.5.3 Hybrid

Hybrid protocols are a combination of reactive and proactive protocols and try to bridge the advantages of both protocols and improve the overall performance. At the edges of a network, were changes in the routing information is not very frequent, a proactive mechanism is chosen. Whereas, at the heart (or core) of the network where changes can happen rather drastically a reactive approach is much more desirable [15].

3.6 Overview of DYMO and B.A.T.M.A.N

In this section, an overview of the routing protocols that this report will examine will be presented.

3.6.1 DYMO

DYMO [1] is a reactive routing protocol and the successor of AODV protocol often called AODVv2, developed to provide on-demand multi-hop unicast routing. DYMO has a pair of basic operations, namely Route Discovery and Route Maintenance.

When an internetwork device, such as a router, has a packet ready for transmission towards one or many destinations it will utilise Route Discovery in order to find the best path towards the destination(s). Whereas Route Maintenance operation, keeps the route table updated by avoiding dropping packets during link breaks, and to avoid deleting routes prematurely [16].

Route Discovery DYMO multicasts (one sender sends a single copy of data to multiple recipients) a Route Request Message (RREQ), in order to search for a path towards a destination. Each internetworking device will perform reverse routing, or in other words each internetworking device receiving this message will record the route towards the sender. When the destination finally receives the RREQ will record the path back to the sender(s), and it will generate a Route Reply (RREP) message which will be send unicast to the originator(s). After both sender and destination devices have exchanged the two messages, RREQ and RREP it means the link has been established bi-directionally [16]. To illustrate Route Discovery a simple example was drawn and presented in Figure 3.1.

Figure 3.1 depicts Node 1 that wants to reach node 5. Node 1 multicasts a RREQ containing the source address and node's 5 address, asking all listening nodes how to reach node 5. Each of the intermediate nodes in their turn record in their routing table, the route back to Node 1 and forward the

message. When the node 5 is reached it will reply with an unicast RREP, which is sent back to node 1 through the path or route that was created by the RREQ. Again each intermediate node receiving the RREP will record the route towards node 5 and forward the message to the next hop. When node 1 receives the RREP then a connection is established between node 1 and 5 and message exchanging can take place.

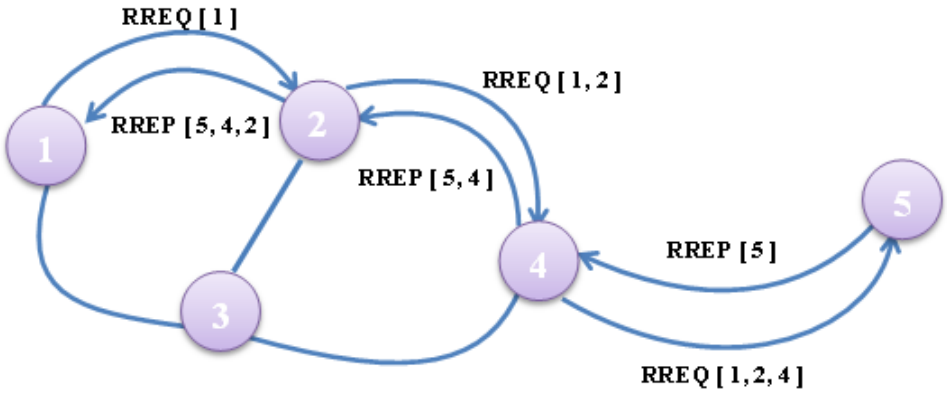


Figure 3.1: DYMO exemplified Route Discovery

Route Maintenance On the other hand, Route Maintenance is a more complex operation. Route Maintenance's role is to monitor and maintain the routes and links, of which traffic flows at an effort to respond to changes in the network topology. By extending the lifetime of the routing table entries and by monitoring the active links, DYMO is maintaining its paths.

In the case a data packet is to be forwarded further but no valid links can be found by a router or similar device, either due to that there is no route to destination or due to a broken link; it must notify the sender by broadcasting a Route Error (RERR) message. When an RERR is received, each device compares the entries in the routing table with the entries contained in the RERR message and invalidates the routing table entries that match. The nodes receiving the RERR will perform in their term Route Discovery to search for other possible paths [16].

3.6.2 B.A.T.M.A.N

The B.A.T.M.A.N. [2] routing protocol or in its proper name Better Approach To Mobile Ad-hoc Networking, was developed as an alternative to the popular and largely deployed OLSR (Optimised Link State Routing Protocol) routing protocol, in an effort to overcome several limits of OLSR's algorithm, such as the recalculation of a whole topology graph [17].

Route Discovery Each node in the network will broadcast messages to other nodes or an unknown network via HNA (Host and Network Association) messages, to inform them of their existence. Those messages are called Originator Messages (OGM) in B.A.T.M.A.N's terminology.

The OGMs are typicaly small in size and contain at least the node's address, a sequence number, and a TTL (Time To Live). The source address and the sequence number provide a source of identification for a packet and allow the detection of duplicate packets, a simplified dipticion follows in Figure 3.2.

Any node receiving an OGM, updates its routing table, which mainly consists of [17]:

- The originator's address (the address of the node the OGM came from).
- The sequence number of the last OGM.
- Sliding window [18]: recorded and not out-of range (recently received) sequence numbers stored in a dedicated sliding window.

Figure 3.2: B.A.T.M.A.N. exemplified route discovery

Route Maintenance Each node monitors any incomming OGM and maintains a list of every node that has sent OGM messages and also records any known HNA announcements. A node deletes an entry, should a known node surpasses a time longer than the Window Size and the interval defined by the PURGE_TIMEOUT. Further, the B.A.T.M.A.N deamon in a node has an opportunistic routing deletion policy and policy consideration [17].

3.7 TCP Congestion Control

TCP is an important protocol in a network, especially, when reliable end-to-end communication is required. This reliability has its roots deep in the TCP protocol implementation, and the congestion control algorithms that continue to evolve until today. In brevity, in can be said that the TCP Congestion Control is an integral part and a clever way for TCP to determine the capacity of a network and prevent it from overloading. By the term congestion it is meant that when devices such as routers are overloaded, they drop datagrams, and should those datagrams contain TCP segments, they left unacknowledged, expire after a period of time and get retransmitted. As a result, the traffic on the internetwork would increase between the different devices, which will further greatly reduce its performance and eventually will lead to a condition known as congestion collapse [19].

Congestion collapse was first observed in October 1986, which resulted in bandwidth drops from 32 Kbps to 40 bps. Investigations were commenced in order to discover what has gotten wrong and from that point on, several TCP congestion control mechanisms were born and introduced [20].

Four basic algorithms for congestion control are known, namely *Slow Start*, *Congestion Avoidance*, *Fast Retransmit* and *Fast Recovery*, and are defined in RFC5681 [21] which supersedes RFC2001. Despite the fact that Slow Start and Congestion Avoidance are two different algorithms, they are implemented together through the utilisation of another mechanism called Congestion Window. The Congestion Window (*cwnd*) can be thought as the boundaries of in-transit data between two endpoints. In other words, TCP always allows a certain amount of outstanding data (unacknowledged) to be transferred, which must be acknowledged from the receiver, before any other data is transmitted. To determine the size of the *cwnd*, RFC5681 [21] mandates that the Initial Window (*IW*) of a connection or in other words, the size of the congestion window (*cwnd*) of the sender must be set according to the following guidelines after the completion of the 3-way handshake:

If SMSS > 2190 bytes:
  IW = 2 * SMSS bytes and MUST NOT be more than 2 segments
If (SMSS > 1095 bytes) and (SMSS <= 2190 bytes):
  IW = 3 * SMSS bytes and MUST NOT be more than 3 segments
if SMSS <= 1095 bytes:
  IW = 4 * SMSS bytes and MUST NOT be more than 4 segments

The receiver also has its own window (rwnd), which also affects how much data can be sent. Hence, TCP flow control is governed also by the advertised receiver's window (rwnd) which is part of a TCP datagram as shown below:

SRC PORT		DST PORT	
SEQUENCE NUMBER			
ACKNOWLEDGEMENT NUMBER			
DATA OFFS	(RESERVED)	FLAGS	*WINDOW*
CHECKSUM		URGENT POINTER	
OPTIONS		PADDING	
DATA			

Table 3.1: TCP Datagram

Fast Retransmit and Fast Recovery are enhancements to the TCP congestion control, devised at late 80s [20], and were firstly introduced in 4.3 BSD Reno. Both algorithms are implemented together. Nowadays, the most common congestion control mechanisms used are TCP Reno and TCP NewReno, both part of the TCP Tahoe (first mechanism) family, with the latter addressing and correcting several issues of the former.

Despite the fact that those mechanisms work adequately in wired networks like Ethernet, their performance degrades over noisy wireless links. TCP congestion control is a rather complex and technical process which is best described in RFC5681 [21]. What follows is a rather abrupt explanation of the TCP congestion algorithms.

3.7.1 Slow Start

At the early days of TCP, when a connection was established by the 3-way handshake between two end-points, they were start exchanging packets until the depletion of the receiver's window, which provided that, there was some congestion in the internetwork this tactic would make it worse. To mitigate its effects the Slow Start mechanism was introduced. Slow Start observes the traffic and increases the cwnd by one *MSS* (Maximum Segment Size) for each successful acknowledgement it receives. Consequently, it limits the rate the devices exchange packets and further reduces congestion related problems. The value of the MSS can be either advertised by the other end or be a default value, such as 512 or 536 [21].

To illustrate, suppose there is a client in Sweden, which wants to retrieve a file from a server in Greece. Firstly, the two endpoints will perform the TCP

3-way handshake and also advertise the size of their *rwnd* (receiver window) and when the last *ACK* is received, the endpoints can start exchanging data. The server will initialise its *cwnd* to a specified conservative value or to the *MSS* as advertised by the other end. The value of cwnd is neither advertised nor exchanged and the amount of outstanding data will be the minimum of both *rwnd* and *cwnd* or *min(rwnd, cwnd)*.

At this point it Slow Start by doubling the *cwnd* size in the sender, for each successfully acknowledged packet, until it reaches the receiver's congestion threshold or slow start threshold (*ssthresh*) or until a packet is lost. Then congestion avoidance will take over. Despite the fact that the *cwnd* was doubled at each round trip time, it will not result in a precisely exponential growth as there are delays between the endpoints on the internetwork, meaning that *ACKs* might not arrive in time. This is better shown in the following graph, captured with Wireshark while downloading a file of approximate size 1.4MB:

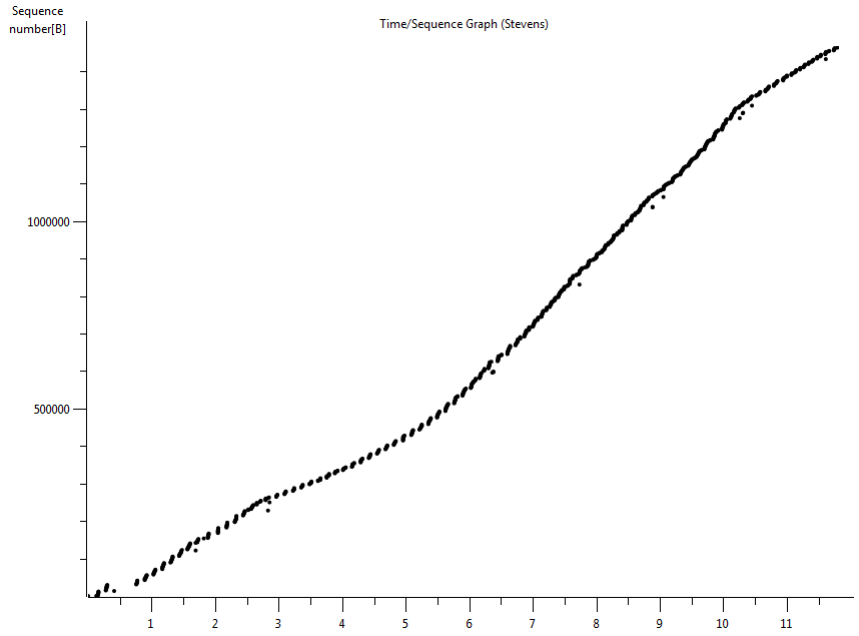


Figure 3.3: TCP Slow Start while downloading a file

The most interesting points from observing this graph are:

1. The cluttered packets every few milliseconds denoted by a small gap. This gap represents the round trip time between a client and a server. In this time the server sends the maximum allowed number of packets as indicated by the *Congestion Window* (CWND) and waits for an acknowledgement.

2. The graph shows that the curve is not trully exponential and doubles itself every transmission time.

Early implementations were Slow Starting only if an endpoint was part of another network, whereas today the implementations always Slow Start.

3.7.2 Congestion Avoidance

When the specified cwnd threshold is reached, denoted by packet loss TCP will shift into the Congestion Avoidance algorithm. Packet loss can be either packet time expiration or the reception of duplicate acknowledgements and it is a matter of **when** and not **if**, it will happen. Hence, Congestion Avoidance can be defined as an algorithm that deals with packet loss. After loss is signalled, the Congestion Avoidance algorithm will drop the rate (by adjusting the *window*) of which segments are sent and then Slow Start will be invoked in order to gradually increase this rate and restore the throughput while avoiding congestion. Congestion Avoidance requires two variables: the *cwnd* and the *ssthress*. Initially, *cwnd* is equals one segment and *ssthress* will be set to an arbitrarily high value, such as *65535*. When congestion occurs the *ssthress* will hold half of the current *window* value. The variable *ssthress*, depending on its value will aid in the distinction of which algorithm (Slow Start or Congestion Avoidance) should be used to control the data transmission. Although, Slow Start and Congestion Avoidance are independent they usually implemented side by side [21].

3.7.3 Fast Retransmit

Fast retransmit allows TCP to send a duplicate acknowledgement upon receiving an out-of-order segment without waiting for a timeout to occur. This algorithm should be used in order to detect and repair loss and is uses as a signal 3 duplicate packets, which if they are received it retransmits the missing segment; and by this the sender will know the sequence number and which segment was expected [21].

3.7.4 Fast Recovery

After Fast Retransmit finishes resenting a lost datagram, fast recovery is taking over until there are no more duplicate packets. In other words, Congestion Avoidance will be invoked right after Fast Retransmit but without Slow Start for increased performance, mainly due to that duplicate acknowledgements denote out-of-order datagrams and the rate can be increased faster than congestion occurs [21]. It is worth noting that Fast Retransmit and Fast Recovery are usually implemented together.

3.7.5 Taxonomy of TCP Congestion Control algorithms

As stated before, TCP Congestion Control is an integral and important part of TCP and its performance; a good congestion control algorithm utilises the network's bandwidth, while mitigating potential packet losses. TCP Congestion Control, according to what it was developed to consider as congestion, could be categorised into: loss-, delay- and hybrid- based algorithms.

- Loss-based – Considers packet loss as a network congestion symptom and reduces the congestion window (*cwnd*), when packet loss occurs or increases it otherwise.
- Delay-based – Considers as a symptom of network congestion the queuing delay or in other words the actual time a packet requires in order to be transmitted from the sender to the receiver.
- Hybrid-based – Considers both delay and loss a network congestion symptom.

3.8 Evaluation Criteria

To evaluate the routing protocols, one can choose many different metrics; in this report the following metrics will be considered: throughput, routing overhead, packet delivery ratio and end-to-end delay. Those metrics were chosen as they are the most common metrics used to evaluate the performance of networks and they are used by many researchers, for instance in [4] and [5], which make them ideal candidates as they are really understood metrics with readily available information. A description of the chosen metrics follows in order to facilitate the reader.

3.8.1 Throughput

Throughput is the average rate of delivering successfully a packet (or message) over a communication channel which could be Ethernet, radio, etc. Throughput is usually being measured depending on context as bits per second (bps), or data packets per second or data packets per time slot. Throughtput is a very important metric for a network and its value should be as high as possible.

3.8.2 Packet Delivery Ratio

The result of the computation of the successfully delivered packets to a destination, by the number of packets sent. The greater this figure is, the better a routing protocol performs. A general formula for calculating packet delivery ratio as a percentage is as follows:

$$\frac{\sum packets\_received}{\sum packets\_sent} x100$$

3.8.3 Routing Overhead

Routing overhead is the sum of all routing packets. Routing packets mean all the packets send by a source, in order to find a destination node or inform about link failures. Routing overhead has an immediate impact on a network's scalability; it is usually measured in bits per second, or packets per second. As a result, when a network expands its routing traffic increases as well.

3.8.4 End-to-End delay

End-to-End delay is the amount of time required for a packet to travel from the source until it reaches its destination. Typically, it can be calculated by:

$$\frac{\sum(arrival\_time - sent\_time)}{\sum number\_of\_connections}$$

The most common measurement unit is *milliseconds* and it is really important, depending on the use of a network, for this value to be small. In regards to routing protocols the smaller the end to end delay is the better a protocol is perfoming. For instance, the International Telecommunication Unit (ITU) recommends in a Packet Voice network with adequetly handled echo or in other words with echo cancellers utilised, the following one-way delays[22]:

Milliseconds	Comment
0-150	All users are satisfied.
150-400	Some users will be dissatisfied.
400 and above	Unacceptable for general network planning purposes. However there might be some exceptional cases.

Table 3.2: ITU One-way delay recommendation

4 Implementation of TCP Congestion Control Mechanisms

In this chapter, the reader will be given a short introduction to the implementation of TCP-FIT and TCP-Illinois.
Both TCP-FIT and TCP-Illinois, have been developed for wireless networks which are subject to challenging network conditions. As a result, they are expected to function and achieve better results over mechanisms that are developed for wired networks.

4.1 Development Environment

The development environment for the implementation is OMNeT++[8] itself. OMNeT++ IDE is based on the Eclipse platform, which is stripped from all Java related development tools. This allows the size distribution to be small, and any developer to make his/her own plugins for the framework through the extensible plugin system of Eclipse.
OMNeT++ contains the following tools pre-installed [23]:

- All OMNeT++ specific tools you use such as NED, MSG and INI file editor, result analysis tools, documentation generator etc.
- The C/C++ Development Tooling (CDT), used for C++ development and debugging, which further integrates with the standard gcc toolchain and the gdb debugger.

4.1.1 OMNeT++

OMNeT++ [8] is a C++ based discrete event simulation environment developed for modelling networks, multiprocessors, etc. It operates under the Academic Public License, which makes the software free for non-profit use. OMNeT++ recognition as a platform for simulating networks and distributed systems among the Academic Community and researchers is rising. OMNeT++ in its core is following a framework-wise philosophy and provides just the necessary mechanisms and tools to the user in order to write simulations or create his own simulator.
In other words, OMNeT++ can be seen as a platform for simulating computer networks and distributed systems, meaning that OMNeT++ is not a simulator and it merely provides the necessary infrastructure for writing simulations. Depending on the application area or problem domain a researcher can utilise other frameworks like INET and VEINS through OMNeT++ and build his own simulation environment.

4.1.2 INET Framework

The INET framework was built as an extension model library to OMNeT++. It provides to OMNeT++ a set of agents, protocols, models for the Internet Suite, such as TCP, UDP, IPv4, BGP, etc. Moreover, it has also mobility support and wireless radio communication and distribution models along with several implementations of MAC protocols, and a lot of other useful features, which make INET ideal for students and researchers of communication networks [24].

INET above all is open-source and many of its components have their own licences (either LGPL or GPL). INET is well-supported and community driven. At the moment of writing, the developers of INET are incorporating most of the features found in MiXiM framework, but lacking or were incomplete in INET. This will result in more meaningful environment implementations with objects, better environment control etc. As OMNeT++, INET is following the logic of modules, which one can simply put together and create a "realistic" scenario.

4.1.3 Modelling a simulation

A simulation in OMNeT++ can be seen as a structure of independent but closely related modules, which communicate by message passing and together they form an OMNeT++ model. The equivalent is to think of those modules as LEGO pieces, which should you combine them; you obtain a truly unique item with specific characteristics. Modules can be categorised as simple modules or compound modules, where compound modules are a group of simple modules. This idea is illustrated in Figure 4.1.

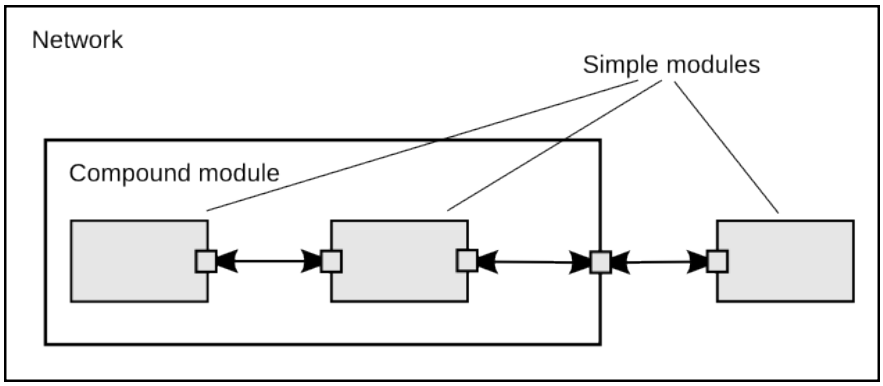


Figure 4.1: Module Structure in OMNeT++

For instance, to measure in this report the throughput of the network, the module NodeBase was extended and used as its basis the StandardHost module implementation, which was modified further to include two ThruputMeter modules. It is required to use two ThruputMeter modules in order to collect both incoming and outgoing traffic.

To evaluate the throughput of the TCP protocol it is required to place the ThruputMeter modules between the TCPApp (Application Layer) and the TCP layer. This is a key step, as TCP, due to its nature will generate many packets for its operation after the TCP layer, in response to out of order deliveries, lost packets, acknowledgements (duplicate or not) etc. and this will produce false measurements. In networking terms, when the application level data is being measured, it is called more accurately goodput. The modification is shown in Figure 4.2 (surrounded in red):

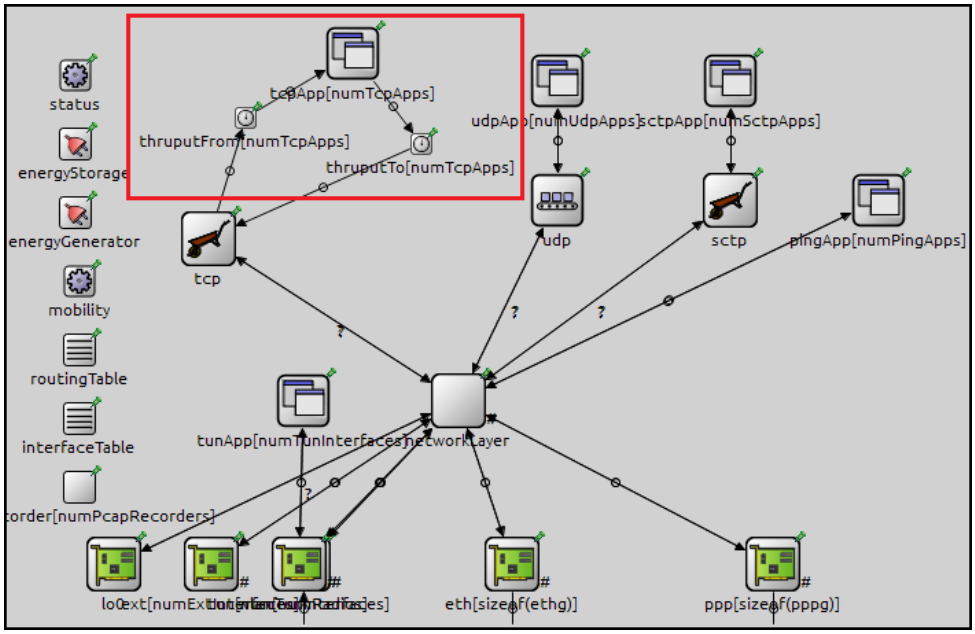


Figure 4.2: Modified Mesh Node with 2 thruputMeters

There has been other modifications at the source of several components such as TCPSessionApp, Batman protocol and others, see github [10].

Furthermore, OMNeT++ provides the user with a complete Graphical User Interface, which allows the person, who runs the simulation, to have a complete view of the simulation internals; it shows the network topology and graphics, it displays the message flow between the different components as an animation, and at any given moment a user can peek inside the compo-

nents and variables, like for instance, to determine if a routing table is as it should [23].
A simulation is formed by the following parts [23]:

1. The Network, which is a file or files that describe the structure of simple or compound modules along with their parameters, gates, channels etc. This file is written using the NED language.

2. A definition of the message types and data structures, which OMNeT++ will do the necessary translation into C++ equivalent classes. These definitions come with the .msg extension.

3. The Modules, the actual implementation of the modules, done in C++ and usually have the suffix .cc. A module can be a router, an HTTP server or in a more general sense anything that has networking capabilities and can be connected in a network.

4. The gates, which are the connection end-points of the modules. There are three types of gates: *input*, *output* and *inout.*

The steps required to create a new project in OMNeT++/INET are described in Appendix 2.

4.2 TCP Congestion Flavours in OMNET++/INET

The INET Framework provides several implementations of TCP Congestion Control Mechanisms, including TCP Westwood which targeting wireless networks, all can be found under the directory *src>inet>transportlayer>tcp> flavours*. In order to implement a TCP Congestion flavour in INET, a specific structure needs to be followed. Classes should extend the functionality of TCPBaseAlg, which itself is an extension of the TCPAlgorithm class and override some or all of the provided methods, should different behaviour is required. For a brief reference some of the methods one can override are:

- *dataSent(uint32 fromseq)* - Invoked after data was sent. This hook can be used to schedule the retransmission timer, to start round-trip time measurement, etc. The argument is the seqno of the first byte sent.

- *segmentRetransmitted(uint32 fromseq, uint32 toseq)* - Called after a retransmitted segment. The argument fromseq is the seqno of the first byte sent. The argument toseq is the seqno of the last byte sent+1.

- *receivedDuplicateAck()* - Per class documentation, this method will be invoked after a received duplicate ACK (that is: ackNo == snd_una, no data in segment, segment there is unacked data). The dupack counter got already updated when calling this method (i.e. dupacks == 1 on first duplicate ACK.)

4.3 Overview of TCP-FIT

TCP-FIT is a hybrid congestion control network mechanism, which considers both delay and packet loss as a signal for congestion. For its implementation an article describing the algorithm's pseudocode [25] along with several other important properties was used.

4.3.1 Keypoints on the implementation

- TCP-Fit is using the same mechanisms for Fast-Retransmit and Fast Recovery as TCP-Reno.
- The pseudocode provided was implemented as is, and by following the several hints, from the authors in the paper.
- TCP-Fit, now TCP-Max, is following a commercialisation path and as such, there is a possibility that the authors did not disclose modifications to the mechanisms of TCP-Reno or others, which might produce better results.

4.4 Overview of TCP-Illinois

TCP-Illinois is also a hybrid congestion control algorithm. It is using packet losses as a way to determine, whether the congestion window should be increased or decreased; and queueing delay to determine how much the amount of increment or decrement should be.

TCP-Illinois is an algorithm developed to achieve high throughput, and fairly utilise network resources. Further, TCP-Illinois is also highly compatible with the TCP standard [26]. TCP-Illinois is already implemented into the Linux kernel and it was used as a guide for implementing/porting TCP-Illinois to OMNeT++/INET.

4.4.1 Keypoints on the implementation

- The source code of the linux implementation of TCP-Illinois [27] was used as a guide to port the algorithm into OMNET++/INET.

- There have been only minor changes in the code, which do not affect the performance, but allows the algorithm to function inside OMNET++/INET.

4.5 Congestion Windows for both algorithms after implementation.

In this chapter a comparison will be drawn, between the graphs produced from the tests run in this report, and those of the authors for both algorithms.

The scenario *tcpclientserver* provided with the INET examples was used, as it is a simple TCP Client-Server scenario where the client is sending 50 megabytes to a server through a *wired* connection, which allows smooth results with minimal packet collisions.

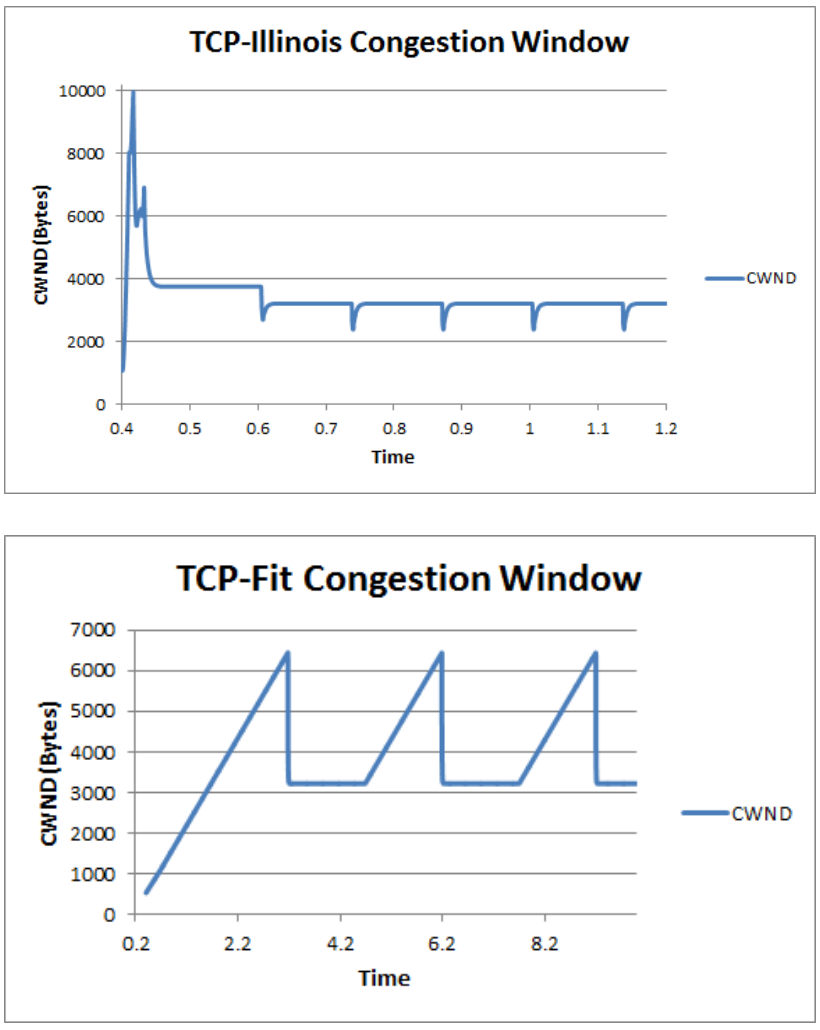


Figure 4.3: CWND of TCP-Illinois and TCP-Fit (our testing)

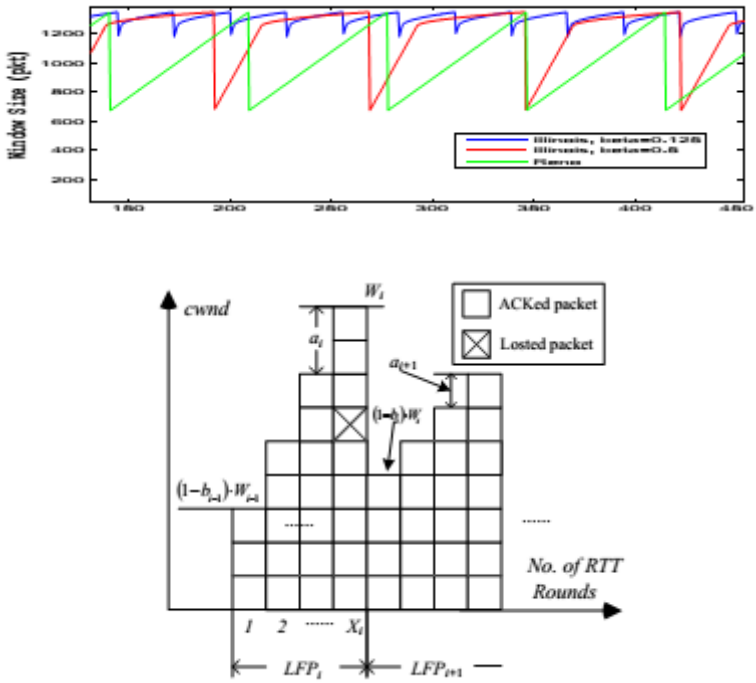


Figure 4.4: CWND of TCP-Illinois (in blue) top, and TCP-Fit bottom, as measured by the their creators.

The similaries are clearly visible, and at this point it is considered that the implementation reached its final phase. The source code of the implementation is provided through GitHub [10]under the LGPL licence; further research and refinement of the source is encouraged.

5 Results

This chapter presents and discusses the results of the different scenarios for UDP and TCP.

5.1 Scenario structure for collecting data

The results will be presented by their performances namely the routing overhead, the end-to-end delay, the goodput and finally by the packet delivery ratio. Also in the TCP results the Round Trip Time will be discussed but will not be taken into serious consideration as some of the connected clients (TCP Sess.) did not produce any data recordings.

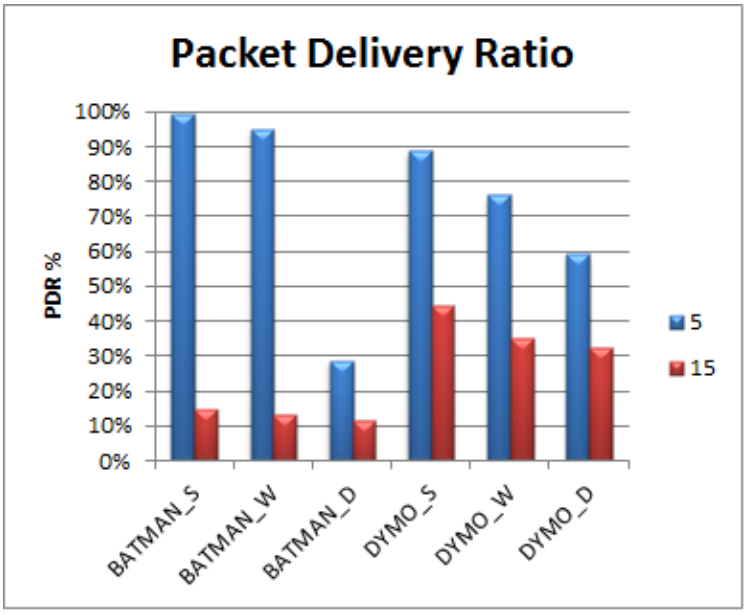
Scenario	**Comment**
UDP DYMO & BATMAN	5 clients static, walking and driving
UDP DYMO & BATMAN	15 clients static, walking and driving
TCP-NewReno DYMO	5 clients static, walking and driving
TCP-NewReno BATMAN	5 clients static, walking and driving
TCP-Illinois DYMO	5 clients static, walking and driving
TCP-Illinois BATMAN	5 clients static, walking and driving
TCP-FIT DYMO	5 clients static, walking and driving
TCP-FIT BATMAN	5 clients static, walking and driving
TCP-NewReno DYMO	15 clients static, walking and driving
TCP-NewReno BATMAN	15 clients static, walking and driving
TCP-Illinois DYMO	15 clients static, walking and driving
TCP-Illinois BATMAN	15 clients static, walking and driving
TCP-FIT DYMO	15 clients static, walking and driving
TCP-FIT BATMAN	15 clients static, walking and driving

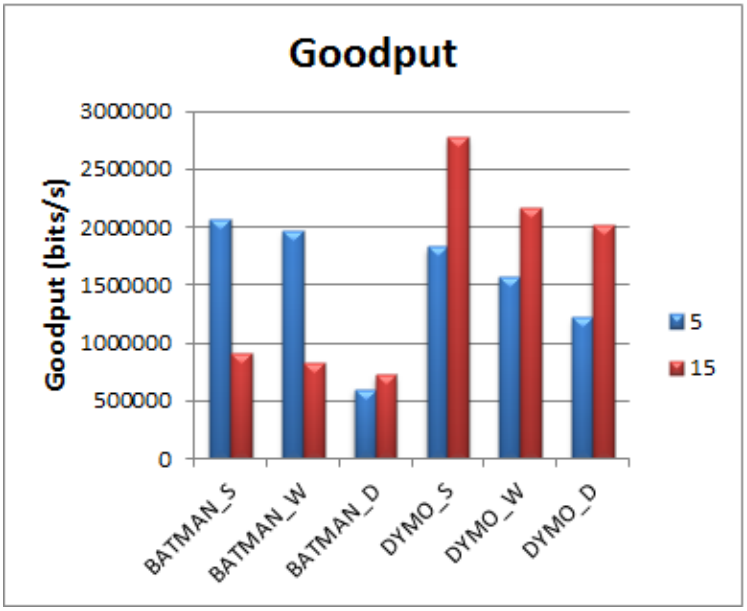
Table 5.1: Scenarios run to accumulate the data.

Table 5.1, lists in brief all the different scenarios that were considered and run, in order to accumulate the data that will serve as the basis of our analysis. More scenarios could be run with more clients, however, that was not possible due to resources limitation. Resources limitation is this context means the laptop, where the scenarios were run on, was not powerful enough to accomodate larger networks and more clients. To review most of the settings used in the scenarios, refer to Appendix 1. Furthermore, many recording statistics were disabled as they were not very relevant, were slowing down our simulations and lastly requiring a really large amount of space, that was not available at the moment.

5.2 UDP Results

Figure 5.1, describes the results accumulated after all the UDP scenarios were concluded. The blue bars in the figure describe the scenarios with 5 clients, whereas the red bars the scenarios with 15 clients. Furthermore, the bottom axis is named after the protocol and the scenario as such *protocol_scenario.* For instance BATMAN_S means BATMAN as routing protocol and static (**S**) scenario. For complicity **W** means Walking and **D** driving.





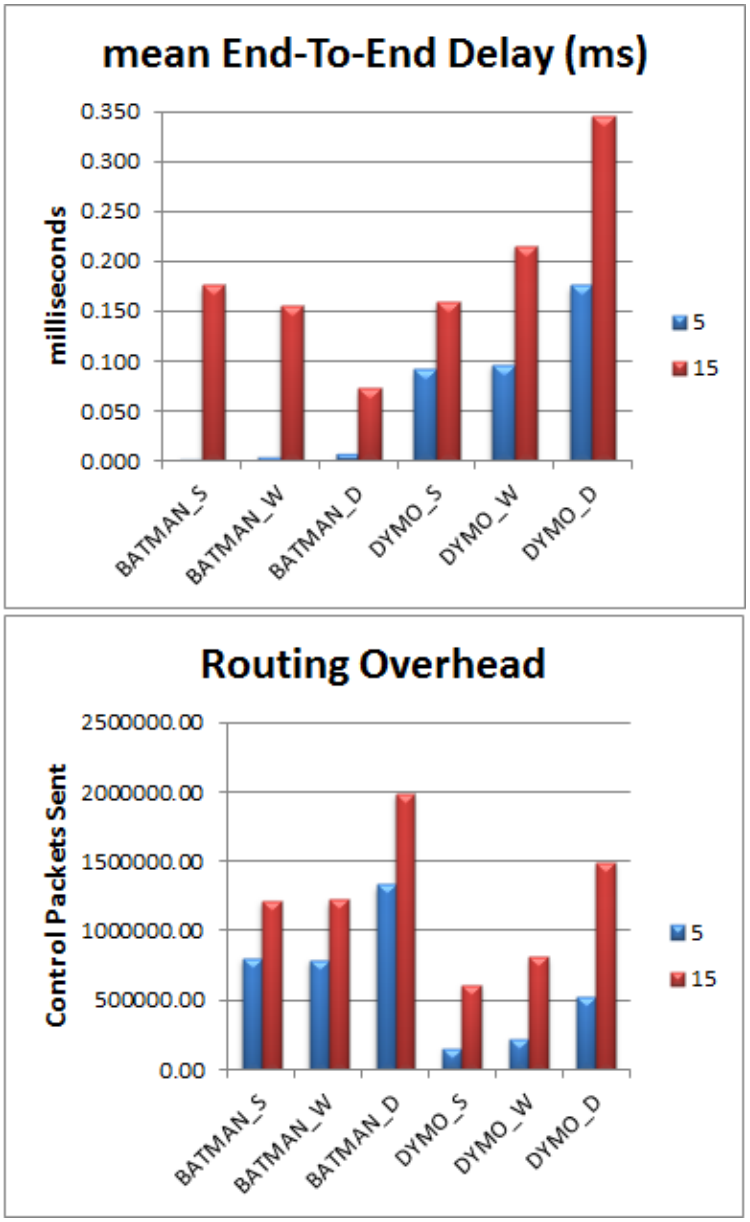


Figure 5.1: UDP results for B.A.T.M.A.N. and DYMO

5.2.1 Routing overhead

According to the data collected in Figure 5.1, the B.A.T.M.A.N. protocol transmits the highest amount of routing traffic in the network, followed by DYMO, which transmits the least. This is true for all the scenarios tested with either 5 or 15 traffic sources in static, walking and driving scenarios.

Hence, in terms of routing overhead DYMO, outperforms B.A.T.M.A.N and in the case of a network with low resource requirements, DYMO would be a better choice as it would perform better.

5.2.2 End-To-End delay

It was observed that B.A.T.M.A.N. has the lowest end-to-end delay compared to DYMO in all scenarios with 5 traffic sources. When the traffic sources increase and mobility is introduced, B.A.T.M.A.N's delay performance competes with that of DYMO.

B.A.T.M.A.N manages to maintain a quite stable end-to-end delay and performs much better than DYMO in all scenarios, even with higher mobility speeds.

5.2.3 Goodput

From Figure 5.1, B.A.T.M.A.N. in the static and walking scenarios with 5 clients record a higher goodput, which as was mentioned earlier, is an indicator that a protocol performs better over others or in our case it performs better than DYMO.

5.2.4 Packet Delivery Ratio

DYMO is following B.A.T.M.A.N. in both static and walking scenarios with 5 clients, which has a higher packet delivery ratio. In the driving scenario B.A.T.M.A.N. performs poorly, whereas DYMO even though it is affected by the higher speeds, still performs better with a PDR of approximately 60%.

On the other hand, when the clients (traffic sources) are increased, DYMO has a highest packet delivery ratio than B.A.T.M.A.N. and consequently outperforms it.

5.3 TCP Results (static scenario)

Table 5.2 describes the data collected from the TCP static scenario with all TCP variants. The first column is the scenario run, the second is the measured goodput, the third is the packet delivery ratio (PDR), the fourth is average mean round trip time, the fifth the routing overhead and lastly the TCP Sessions.

There are two columns that reserve explanation, the Average mean RTT, which is the sum of the mean round trip times values recorded from the clients and then divided by the number of the recorded values; and the TCP Session, which represents how many clients managed to connect with the gateway.

TCP NewReno					
Static Scenario	*Goodput (bits/s)*	*PDR*	*Avg. mean RTT*	*OHD*	*TCP Sess.*
DYMO 5 Clients	721267	21.6%	0.0246923	146844	5
DYMO 15 Clients	4824574	48.2%	0.0276579	267236	14
BATMAN 5 Clients	3114560	93.4%	0.0245930	764079	5
BATMAN 15 Clients	5388584	67.4%	0.0165166	1285406	12

TCP Illinois					
Static Scenario	*Goodput (bits/s)*	*PDR*	*Avg. mean RTT*	*OHD*	*TCP Sess.*
DYMO 5 Clients	783953	23.5%	0.0021406	144912	5
DYMO 15 Clients	4931707	56.9%	0.0032317	232622	13
BATMAN 5 Clients	3333332	100.0%	0.0021237	769403	5
BATMAN 15 Clients	5980280	64.1%	0.0028525	1279475	14

TCP Fit					
Static Scenario	*Goodput (bits/s)*	*PDR*	*Avg. mean RTT*	*OHD*	*TCP Sess.*
DYMO 5 Clients	855865	25.7%	0.0031685	139497	5
DYMO 15 Clients	4876583	52.2%	0.0044332	233472	14
BATMAN 5 Clients	3333332	100.0%	0.0042324	765245	5
BATMAN 15 Clients	5888196	63.1%	0.0038250	1285407	14

Table 5.2: Results from the TCP static scenario.

5.3.1 Round Trip Time (RTT)

DYMO and TCP-Newreno keep the RTT inside acceptable limits. However, B.A.T.M.A.N. performs better with slightly lower RTT, around 0.4021% decrease with 5 clients, and a much lower decrease of 40.2825% with 15

clients. The same is observed with TCP-Illinois, B.A.T.M.A.N. records a 0.7895% decrease in RTT with 5 clients and 11.7338% with 15 clients. With TCP-FIT and 5 clients DYMO performs better with 25.137% decrease over B.A.T.M.A.N., which records a lower RTT with 15 clients of around 13.7192%.

The round trip time between the TCP variants, show that TCP-FIT and TCP-Illinois have a much smaller RTT than TCP-Newreno, which also result in a better goodput. Also B.A.T.M.A.N. outperforms DYMO by having a lower RTT in most of the scenarios.

5.3.2 Goodput

The developers from both TCP-FIT and TCP-Illinois promised in their corresponding papers a higher throughput, which is reflected in the obtained data and results in a higher goodput.

B.A.T.M.A.N. shows an increase of 331.8179% in goodput with 5 clients and TCP-Newreno and 11.6904% with 15 clients. 325.1954% increase with 5 clients and 21.2619% with 15 clients and TCP-Illinois as the algorithm. With TCP-FIT and increase of 289.4694% in the goodput for B.A.T.M.A.N. with 5 clients and 20.7443% with 15 clients. As a result, B.A.T.M.A.N. outperforms DYMO with every algorithm.

5.3.3 Packet Delivery Ratio

The packet delivery ratio (PDR) observed shows the TCP-FIT and TCP-Illinois outperforms TCP-Newreno. However, with 15 clients and B.A.T.M.A.N. as the routing protocol, TCP-Newreno shows better PDR. Also, B.A.T.M.A.N. performs better than DYMO.

5.3.4 Routing overhead

It is worth noting that when comparing TCP-Illinois and TCP-FIT with TCP-Newreno, the former two show a reduction in the overall routing overhead for both routing protocols. B.A.T.M.A.N. as a proactive protocol is performing poorly over DYMO.

5.4 TCP Results (Walking scenario)

Table 5 describes the data collected from the TCP walking scenario, also with all TCP variants. Since the data in the table is expressed as in the static scenario, one could refer to it.

TCP NewReno					
Walking Scenario	*Goodput (bits/s)*	*PDR*	*Avg. mean RTT*	*OHD*	*TCP Sess.*
DYMO 5 Clients	2920703	87.6%	0.0118832	136242	5
DYMO 15 Clients	5713815	65.9%	0.0127956	203534	13
BATMAN 5 Clients	3333330	100.0%	0.0189069	770229	5
BATMAN 15 Clients	6749509	77.9%	0.0167559	1304478	13

TCP Illinois					
Walking Scenario	*Goodput (bits/s)*	*PDR*	*Avg. mean RTT*	*OHD*	*TCP Sess.*
DYMO 5 Clients	3210890	96.3%	0.0011166	108579	5
DYMO 15 Clients	7383361	79.1%	0.0012739	207305	14
BATMAN 5 Clients	3333330	100.0%	0.0019557	774406	5
BATMAN 15 Clients	7071873	70.7%	0.0020500	1310749	15

TCP Fit					
Walking Scenario	*Goodput (bits/s)*	*PDR*	*Avg. mean RTT*	*OHD*	*TCP Sess.*
DYMO 5 Clients	2552230	76.6%	0.0031692	132533	5
DYMO 15 Clients	7835911	78.4%	0.0037528	189335	15
BATMAN 5 Clients	3333330	100.0%	0.0048314	766477	5
BATMAN 15 Clients	6532139	81.7%	0.0036081	1299265	12

Table 5.3: Results from the TCP walking scenario.

5.4.1 Round Trip Time (RTT)

In table 5, can be seen that TCP-Illinois has the lowest RTT with TCP-FIT following. In brief, DYMO shows with TCP-Newreno, a decrease in RTT with 5 clients of 37.1489% and 23.6353% with 15 clients.

Further with TCP-Illinois, a 42.9054% decrease with 5 clients and 37.8585% with 15 clients. TCP-FIT, show similar results with DYMO displaying a decrease of 34.4041% with 5 clients, except with 15 clients where B.A.T.M.A.N. performs better with DYMO recording a 4.0104% increase. In this case, DYMO outperforms B.A.T.M.A.N. by having a lower RTT overall.

5.4.2 Goodput

TCP-Illinois seems to perform much better and give a higher goodput when compared to the other two TCP variants. TCP-FIT is following with similar results.

With TCP-Newreno, B.A.T.M.A.N. has an increase of 14.1277% with 5 clients and a 18.1261% with 15 clients. With 5 clients and TCP-Illinois, B.A.T.M.A.N. is showing a small increase of 3.8133% and a decrease of 4.2188% with 15 clients. 30.6046% increase with 5 clients and TCP-FIT and 16.6384% decrease with 15 clients.

B.A.T.M.A.N. seems to perform better, except in the cases with TCP-Illinois and TCP-FIT with 15 clients.

5.4.3 Packet Delivery Ratio

TCP-Illinois and TCP-FIT have a higher PDR than TCP-Newreno. Further, both routing protocols perform equally well with TCP-Illinois and TCP-FIT, since there are a few TCP sessions less, in some scenarios.

5.4.4 Routing overhead

In table 5, TCP-FIT records low values for routing overhead in all but one scenario and such is winning at this measurement with surprisingly TCP-Newreno following. Also, as expected DYMO is outperforming B.A.T.M.A.N..

5.5 TCP Results (driving scenario)

The data collected from the TCP driving scenario with all TCP variants is described by table 6 in a similar fashion as in the static scenario, which one could refer to.

TCP NewReno					
Driving Scenario	*Goodput (bits/s)*	*PDR*	*Avg. mean RTT*	*OHD*	*TCP Sess.*
DYMO 5 Clients	1529055	45.9%	0.0254553	259569	5
DYMO 15 Clients	6016713	64.5%	0.0108939	398989	14
BATMAN 5 Clients	77167	2.3%	0.0449272	1371442	5
BATMAN 15 Clients	3460127	34.6%	0.0218555	2087110	15

TCP Illinois					
Driving Scenario	*Goodput (bits/s)*	*PDR*	*Avg. mean RTT*	*OHD*	*TCP Sess.*
DYMO 5 Clients	1583725	47.5%	0.0020532	254529	5
DYMO 15 Clients	5830933	62.5%	0.0010029	384607	14
BATMAN 5 Clients	142956	5.4%	0.0024014	1369909	4
BATMAN 15 Clients	3712318	39.8%	0.0029375	2076197	14

TCP Fit					
Driving Scenario	*Goodput (bits/s)*	*PDR*	*Avg. mean RTT*	*OHD*	*TCP Sess.*
DYMO 5 Clients	1579903	47.4%	0.0030182	257512	5
DYMO 15 Clients	6635451	66.4%	0.0027660	386345	15
BATMAN 5 Clients	98290	3.7%	0.0096568	1370718	4
BATMAN 15 Clients	3843335	44.3%	0.0062592	2079371	13

Table 5.4: Results from the TCP driving scenario.

5.5.1 Round Trip Time (RTT)

TCP-Illinois is outperforming the other variants in this scenario with TCP-FIT following and TCP-Newreno being far behind. Regarding the routing protocols DYMO outperforms B.A.T.M.A.N. in all scenarios by a decrease of 43.34% with 5 clients and a decrease of 50.15% with 15 clients.

DYMO and TCP-Illinois record a decrease of 14.5% with 5 clients and with 15 clients a decrease of 65.86%. With TCP-FIT and 5 clients and 15 clients, a decrease of 68.75% and 55.81%, respectivelly.

5.5.2 Goodput

TCP-FIT and TCP-Illinois seem to outperform TCP-Newreno and record higher goodput with the exception the difference between TCP-Newreno and TCP-Illinois with DYMO as the protocol and 15 traffic sources, where TCP-Illinois is not performing as good as TCP-Newreno. DYMO also outperforms B.A.T.M.A.N. an increase of 1881.49% and 73.89% with 5 and 15 clients respectively.

With an increase of 1007.84% with 5 traffic sources and 57.07% with 15 traffic sources with TCP-Illinois. Lastly, with TCP-FIT records an increase of 1507.39% and 72.65%.

5.5.3 Packet Delivery Ratio

Table 4, shows that TCP-FIT would outperform all other TCP variants in all but one scenario, however, the amount of clients connected does not give a clear and definitive answer. TCP-Newreno seems to also perform and handle the higher mobility equally well. DYMO clearly from the table performs better than B.A.T.M.A.N..

5.5.4 Routing overhead

TCP-FIT and TCP-Illinois record lower values for routing overhead in all scenarios. As expected, DYMO records the lowest routing overhead and outperforms B.A.T.M.A.N..

5.6 TCP Goodput change

Table 7 lists the change of the goodput in the form of percentage, and it was created in an effort to display the goodput difference between the congestion control algorithms tested in this report.

The reason for its creation was mainly the fact that all congestion controll algorithms developed for wireless networks promise an increase in the throughput, which consequently results in an increase in goodput. A negative number in the table means an algorithm performed worst.

Goodput change in Static scenarios (percentage)			
Scenario	**Illinois vs. NewReno**	**FIT vs. NewReno**	**FIT vs. Illinois**
DYMO 5 Clients	8.69%	18.66%	9.17%
DYMO 15 Clients	2.22%	1.08%	-1.12%
BATMAN 5 Clients	7.02%	7.02%	0.00%
BATMAN 15 Clients	10.98%	9.27%	-1.54%

Goodput change in Walking scenarios (percentage)			
Scenario	**Illinois vs. NewReno**	**FIT vs. NewReno**	**FIT vs. Illinois**
DYMO 5 Clients	9.94%	-12.62%	-20.51%
DYMO 15 Clients	29.22%	37.14%	6.13%
BATMAN 5 Clients	0.00%	0.00%	0.00%
BATMAN 15 Clients	4.78%	-3.22%	-7.63%

Goodput change in Driving scenarios (percentage)			
Scenario	**Illinois vs. NewReno**	**FIT vs. NewReno**	**FIT vs. Illinois**
DYMO 5 Clients	3.58%	3.33%	-0.24%
DYMO 15 Clients	-3.09%	10.28%	13.80%
BATMAN 5 Clients	85.26%	27.37%	-31.24%
BATMAN 15 Clients	7.29%	11.07%	3.53%

Table 5.5: Goodput differences between the TCP variants. A negative number means an algorithm performed not as well.

6 Discussion

In this chapter the report's discussion is drawn.

In this section the problems that occured during the implementation of the two congestion control algorithms will be discussed. Followed by a discussion on the configuration and lastly on the different results.

6.1 Problems during the implementation of congestion control mechanisms.

During the implementation of the two TCP congestion algorithms there were numerous problems encountered, that needed to be resolved before proceeding further. Those problems were mostly related, either with the implementation itself or in the case of TCP-Illinois with understanding the Linux kernel implementation of the algorithm.

6.1.1 Issues with the kernel implementation

In brevity, in the Linux kernel implementation there were some methods along with their parameters that lack any further explanation. Hence, an understanding of how different methods, parts and internal TCP functions work needed to be done. After reading numerous fragments of information regarding kernel patches, and the source codes of existing implementations in the INET framework, a picture was drawn and the implementation went further. In its final state only some variables were changed, which do not affect the algorithm's performance, and allow it to run in OMNET++/INET. For instance, one such small change was the current timestamp, which was changed to OMNET++ simulation time.

6.1.2 TCP-FIT algorithm considerations

TCP-FIT, although not widely used or adopted, its algorithm is given in pseudocode form, in[25] where the algorithm is described. In this paper there are several suggestions that were taken into consideration. One of them is the update interval (*update_epoch*) of the algorithm, which was set to 500 *ms*, as the developers did in their testing. However, due to that this value can produce different results, it worths more investigation. Another suggestion is the use of a slightly different formula, considering the *beta* value, which was added in the code and can be used, when the user provides a different value for *beta* and then re-compile the code.

Furthermore, the TCP-FIT algorithm's implementation provided [10] might not be absolutely correct as the developers are following a commercialised approach and their pseudocode describes only the algorithm's internals, and skips the fast-recovery and fast re-transmitt parts, by mentioning only that they are similar to TCP-Reno that the algorithm tries to improve. As a result, those mechanisms were kept as is from TCP-Reno.

6.1.3 Modifications into OMNET++/INET nodes

To retrieve the required results some files needed to be changed, like the thruputmeter, batman protocol and some tcp files. All these files are provided in the same github directory as the algorithms [10].

Nevertheless, both algorithms reached their final state for this report. The results from several tests seem to give satisfactory results and as such, the algorithms are freely available under the LGPL license in github [10].

6.2 Configuration

Concerning the obtained results, it is worth mentioning that all the scenarios were not set in an ideal world without noise and interference, and it is also worth noting that this fact is what affects the performance of both routing protocols in all the scenarios, and mostly those with 15 clients and those with mobility. In other words, in a moderate mobility scenario, due the client movement the interference, the noise and the possible packet collisions are minimised as the clients move and give space to others. The opposite is true with 15 or more clients where the network becomes crowded and the performance worsens.

In this report,both UDP and TCP nodes were configured as such: in the UDP scenarios the message length was set to 1024 bytes (1MB), the clients were set to send packets in one second burst with a slightly variable interval, which produces every second either heavy or medium traffic. Moreover, in the UDP configuration the sleep and the stop time were disabled, meaning that data will be sent at once, and the clients will keep sending data as long as the simulation runs. TCP proved a little trickier due to bigger amount of settings it needs. Nevertheless, most of the TCP features remained at their default values with the exception of the *advertised window*, *maximum segment size* and *Nagle algorithm*. The TCP clients were configured to make a connection with the gateway mesh station and each of them sent 300MB of data with a variable time of sending the data and opening the connection. The variable time for opening the connection and sending the data was chosen in an effort to mitigate collisions at the MAC layer or in other words to limit the

possibility of all clients trying to send data at the same time. What is more, another crucial factor in regards to the wireless receiver and transmitter of the nodes was the calibration of the *contention window*. The *contention window* was set according to the information found in several slides [28] and are specific to wireless “g” mode. The values that worked best in this case and gave a higher packet delivery ratio were:

- *slot time* = 9us (default)
- *Cwmin* = 31, and
- *Cwmax* = 1023

The equivalent names of these settings in OMNET++/INET are *cwMinData*, *cwMaxData* and *slotTime* and all are part of the MAC layer of the wireless device.

6.3 Results

6.3.1 UDP Overhead

Regarding the data obtained from the UDP results, the measured overhead is the total control packets sent from all involving nodes. In the case of DYMO it also incorporates the routing error messages. Routing error messages (RERR) are sent when a link or path towards a node fails. This message propagates itself to all nodes in the neighbourhood or vicinity of the node that sends it. This is done in DYMO due to the lack of alternative routes towards the destination, as the routing table contains only the routes required and not all possible routes.

Nevertheless, it was expected that DYMO would outperform B.A.T.M.A.N., since as it was described at chapter 3, on-demand or reactive routing protocols transmit control packets only when a source needs to send some data, which consequently reduces the routing overhead by not "polluting" the network with unnecessary routing traffic. Whereas proactive protocols like B.A.T.M.A.N. always send control packets to monitor and maintain its routing table.

In the considered simulations, the default interval of sending control packets was increased, 2 seconds from 1 for both B.A.T.M.A.N. and DYMO, as those numbers produced better results. Also it is important to mention, that by default DYMO relies on the link state information to discover faults in the paths. This behaviour in the simulations resulted with poor performance and as such it was overridden by setting the sending interval of the HELLO messages.

6.3.2 UDP End-to-end delay

The measured end-to-end delay shows that B.A.T.M.A.N. outperforms DYMO. The reason behind this observation is that B.A.T.M.A.N. maintains a routing table for every possible destination and has its routes ready for when data needs to be sent, unlike DYMO which always requests the route, if not in the table or in case it was deleted, failed or changed.

6.3.3 UDP Routing overhead

DYMO recorded a lower routing overhead, as better seen in the results from chapter 5. Since, DYMO is an on-demand routing protocol and this allows the protocol to be fairly consistent which despite the higher one-way delay (which is still inside acceptable limits), to have a much lower impact on the network traffic by regulating its routing overhead by need (On-demand). Moreover, it performs also better in a driving scenario and it is worth noting that in a highway or in a town, clients can also act as mesh stations and provide many different network gateways (not visible with our simulations), which might or might not allow for better performance or different results.

6.3.4 UDP Goodput

The results show that B.A.T.M.A.N. records a higher goodput and this can be explained as proactive protocols already know which path to follow to reach a destination, and since throughput (goodput is always less) is defined as the total amount of data received to the time the receiver got the very last packet. As a result B.A.T.M.A.N. which has lower delay in those scenarios has a higher goodput.

Moreover, as the traffic sources are increased the routing protocols have to work harder to compensate with network degradation, and problems like congestion, hidden terminals, etc. In the results, this is visible as B.A.T.M.A.N. displays higher delay, which results in a much lower goodput. This means that while the network is small and the mobility is low or completely static, B.A.T.M.A.N. outperforms DYMO. However, as the network grows whether there is mobility or not, the opposite is true.

6.3.5 TCP

On the other hand, from the TCP results it was observed that TCP congestion algorithms designed for wireless networks can offer greater performance.

6.3.6 TCP Static scenarios

In brief, from the static scenarios it became clear that congestion control algorithms designed for wireless networks perform much better and give as promised a higher throughput.

Regarding the Round Trip Time (RTT), TCP-Illinois seems to outperform the other two algorithms with TCP-FIT following. This is visible in our results with TCP -FIT and -Illinois having a similar performance. However, the similarities could be due to the number of the connected clients (as indicated by the TCP Sess. column).

According to this column, only 12 out of 15 clients managed to connect, which might be the reason for such differences during the simulation's running time. In other words, the smaller amount of connected clients record more data in the simulation's time, whereas in the other TCP variants the results show less packets sent, which is due to more collisions, re-transmittions, congestion etc. Nevertheless, TCP-FIT and TCP-Illinois seem to perform better in wireless networks.

In any case, a proactive protocol like B.A.T.M.A.N. seems to perform better than DYMO in the static scenarios, which was as expected, since TCP, unlike UDP, in order to operate needs a stable route as it is an end-to-end connection oriented communication protocol. The most strange results are with DYMO and 5 clients, as it was expected to perform as well as B.A.T.M.A.N., that did not happen and a satisfactory explanation was never found. A speculation could be that, on-demand routing and the various TCP timers for acknowledgements. etc. which had a negative impact on DYMO's performance.

6.3.7 TCP Walking scenarios

In the walking scenarios, the recorded difference is due to how many clients were connected and how they were moving. Like the UDP scenarios, with moderate mobility the routing protocols and the TCP congestion control algorithm perform better, for similar reasons. Despite that TCP-Illinois seemed superior in most scenarios, the performance of TCP-FIT cannot be neglected and is it would be good to note, that it seems to be able to perform better than TCP-Newreno.

Regarding the packet delivery ratio in the walking scenarios, it would be safe to say that according to the number of clients connected**,** both TCP-Illinois and TCP-FIT record very similar results, and should the clients connected were the same amount, we would probably have same results. We could only say that the TCP-Illinois is a better algorithm, due to the number

of clients connected and that it outperformed TCP-Newreno.

Overall in the walking scenarios, the results show that TCP-Illinois for both routing protocols performs much better than TCP-Newreno, which is a congestion control mechanism designed for wired networks. Moreover, although TCP-Illinois and TCP-FIT are giving very similar results, the former is winning the battle mainly due to that most of the clients were connected with the gateway node (TCP Sess.), with the latter being very close behind. From the results, is it worth noting that with TCP-FIT there is a reduction on the overall routing overhead for both protocols.

6.3.8 TCP Driving scenarios

From the data collected from the driving scenarios, it can be observed that, like UDP, on-demand routing protocols can perform better under frequent routing changes. All three TCP variants with DYMO give similar results with the only difference the increase in Goodput in both TCP-Illinois and TCP-FIT, which is as expected, since the developers of the algorithms had in mind to provide better throughput. Better throughput, means an increase in Goodput as well, although slightly less than throughput due to TCP retransmissions.

In the driving scenario both TCP-Illinois and TCP-FIT reduce the overall routing overhead. To conclude, all TCP variants seem to perform well and give similar packet delivery ratios in higher mobility scenarios, which might be due to the client movement the congestion is not reaching critical levels. Further, DYMO is clearly outperforms B.A.T.M.A.N. as expected.

6.3.9 Protocol comparison conclusion

Overall, it cannot be said with certainty that one routing protocol is the best or the worst for a specific case. It would be up to the network designer to take the right decisions regarding a Wireless Mesh Network. For instance, if there is moderate mobility and the extra traffic generated by a proactive protocol is not causing any problems, one can choose that instead of another.

From the UDP results it can be observed that on-demand protocols in our simulation seem to be more stable, and to perform better in the majority of the considered scenarios and environments, and it would be safe to say that they can come first in the order of preference. This is especially true when the network requires low resources, where on-demand protocols with their low routing overhead, should be the first choice or among the first.

Regarding B.A.T.M.A.N., it seems to perform better in static and moderate mobility scenarios, like walking, with very low one-way delays, except

the case when the number of the traffic sources increases, and their one-way delay is competing with that of DYMO. To conclude, some of the facts that one needs considering before choosing a proactive protocol, are:

1. The higher routing overhead.
2. Under frequent route changes, their performance may worsen and,
3. The routing tables as the network grows might grow fairly large, and when combined with all the problems in wireless networks, the network quality will degrade.

The TCP scenarios revealed, that congestion control algorithms should not be neglected and a proper one must be chosen, as they can offer better performance in conjunction with the chosen routing protocol.

7 Conclusion

In this final chapter the report conclusion will be drawn.

In this thesis two routing protocols for Wireless Mesh Networks were compared in a set of scenarios that utilised two transport protocols, namely UDP and TCP.
When TCP is being used there are many congestion control variances that can improve the performance, especially in a wireless communication, where packet collisions and interference can have a significant impact on the user experience.

In this report, instead of utilising only the provided TCP congestion mechanisms an effort was made to implement two known mechanisms intended for Wireless networks, namely TCP-Illinois and TCP-FIT and then compare them against each other and against TCP-Newreno, an algorithm intended for wired networks. There were many problems faced during the implementation of the two algorithms, most of which were related to the available documentation.

At first an understanding of how TCP-Illinois was implemented in the Linux kernel was needed, in order to clarify several methods and parameters and then finding their relatives in INET. Secondly, with TCP-FIT which its pseudocode did not mention data types and several other characteristics. Despite all those small problems, the implementation finished successfully and the code is provided, for further refinements and research.

What is more, during the writing of this report it became obvious that OMNET++ and INET, although they are both really powerful frameworks, they are not intented for the researcher that wants to build a network and just simply test it. In other words, a researcher should possess some fundamental coding skills to be able to utilise fully those frameworks.

Lastly, according to the collected data from the scenarios run on UDP, DYMO seem to be a better protocol with B.A.T.M.A.N. to follow closely after. In our case, it can be considered that DYMO performed well in all scenarios. From the scenarios run on TCP, it became apparent that there are a lot of factors involved for the well being of a reliable connection. The data showed that on-demand protocols are more stable and perform better in the majority of the scenarios. However, proactive protocols cannot be neglected as with no mobility or moderate mobility can perform equally well, despite the extra load on the network traffic; which depending on the network requirements, where one could simply not take it into consideration. Also, it was observed that TCP congestion control mechanisms can make a difference and they should be considered and chosen carefully in a network design. The answers to the questions posed in this study are as follows:

1. **Which routing protocol in a Wireless Mesh Network performs better in terms of throughput, routing overhead, packet delivery ratio and end-to-end delay?** On-demand protocols like DYMO are more stable in a variety of scenarios and should come first in order of preference. However, the performance of table-driven or proactive routing protocols in static and moderate mobility scenarios cannot be neglected. Proactive protocols like B.A.T.M.A.N. outperform DYMO in such cases and should the extra routing overhead does not degrade the quality of a network and the network is fairly small (otherwise might result in large routing tables), they can also be chosen.

2. **Which factors might influence their performance?** As seen in the results, factors such as mobility, noise, other transmission impairments and issues like hidden terminals, etc. have a negative impact on the performance of a routing protocol, and consequently on the quality of a network.

3. **Do TCP congestion control algorithms provide better performance in an ad-hoc network in terms of throughput?** TCP congestion control algorithms, do have an impact on the quality of a network and provide better performance over algorithms developed for wired networks.

7.1 Further research

Research in Wireless Mesh Networks is ongoing and many new routing protocols and algorithms are developed, new devices are being build, and standards are being refined. As a result, testing by either simulations or by building a real network is highly required. In this respect, the OMNET++ and INET communities are doing a really good job, maintaining and updating those two good frameworks. At the time of writing INET released a new version with numerous new features that open new horizons for simulations and research. For instance, much of the current research does not address or does not take into consideration the enviroment temperature or the wave propagation through objects, all of which can now be tested with the new version of INET. As a result, future research will be to continue evaluating the performance of existing routing protocols and providing all the interested parties with all the information about the advantages and disadvantages of each protocol.
Further, as seen in this report, TCP congestion control algorithms can offer better performance, and as such further research and evaluation (even to the algorithms implemented for this report) is required. It would also be nice

to take into consideration, when testing TCP congestion control algorithms, the packet loss at the MAC layer and their friendliness. Alternatively, more TCP congestion control algorithms could be implemented or even one could explore the possibility of developing a new algorithm to address issues of existing algorithms.

References

[1] M. A. hoc Networks Working Group, "Ad hoc on-demand distance vector routing version 2 (aodvv2) draft-ietf-manet-aodvv2-12," RFC Editor, RFC draft-ietf-manet-aodvv2-12, oct 2015. [Online]. Available: https://tools.ietf.org/html/draft-ietf-manet-aodvv2-12

[2] Freifunk. (2015) B.a.t.m.a.n. protocol. [Online]. Available: http://www.open-mesh.org/projects/open-mesh/wiki

[3] P. Rajankumar, P. Nimisha, and P. Kamboj, "A comparative study and simulation of AODV MANET routing protocol in NS2 amp; NS3," in *2014 International Conference on Computing for Sustainable Global Development (INDIACom)*, Mar. 2014, pp. 889–894.

[4] M. Zabin and R. R. Mannam, "Comparative Performance Analysis of MANET Routing Protocols in Internet Based Mobile Ad-hoc Networks," Växjö, 2012.

[5] B. Nyirenda, "Performance evaluation of routing protocols in mobile ad hoc networks (manets)," 2009.

[6] A. K. G. Sandeep Kaur, "A Comparative analysis of AODV, DSR & DYMO reactive routing protocols for MANETs," *M.Tech. Thesis*, 2012.

[7] N. Kumar, K. Vashishtha, and K. Babu, "A Comparative Study of AODV, DSR, and DYMO routing protocols using OMNeT+," *International Journal on Recent and Innovation Trends in Computing and Communication, ISSN*, pp. 2321–8169.

[8] O. team. (2015) Omnet++ simulator. [Online]. Available: https://omnetpp.org/

[9] I. Community, "Inet framework," 2015. [Online]. Available: https://inet.omnetpp.org/

[10] S. Marinis-Artelaris, "Tcp-fit and tcp-illinois implementation source code," 2015, last visited on 2015-07-23. [Online]. Available: https://github.com/SpyrosMArtel/TCP-Fit-Illinois

[11] T. Clausen, C. Dearlove, and J. Dean, "Mobile Ad Hoc Network (MANET) Neighborhood Discovery Protocol (NHDP)," RFC Editor, Tech. Rep. RFC6130, Apr. 2011. [Online]. Available: https://www.rfc-editor.org/info/rfc6130

[12] "IEEE Standard for Information Technology–Telecommunications and information exchange between systems–Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications Amendment 10: Mesh Networking," *IEEE Std 802.11s-2011 (Amendment to IEEE Std 802.11-2007 as amended by IEEE 802.11k-2008, IEEE 802.11r-2008, IEEE 802.11y-2008, IEEE 802.11w-2009, IEEE 802.11n-2009, IEEE 802.11p-2010, IEEE 802.11z-2010, IEEE 802.11v-2011, and IEEE 802.11u-2011)*, pp. 1–372, Sep. 2011.

[13] "IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pp. 1–2793, Mar. 2012.

[14] G. Held, "Introduction to Wireless Mesh Networking," in *Wireless Mesh Networks*, 1st ed. Boca Raton, FL: Auerbach Publications, Jun. 2005, pp. 1–20.

[15] M. J. Lee, J. Zheng, X. Hu, H.-h. Juan, C. Zhu, Y. Liu, J. S. Yoon, and T. Saadawi, "A new taxonomy of routing algorithms for wireless mobile ad hoc networks: the component approach," *IEEE Communications Magazine*, vol. 44, no. 11, pp. 116–123, Nov. 2006.

[16] S. Ratliff, J. Dowdell, C. Perkins, V. Mercieca, and L. Steenbrink, "Ad Hoc On-demand Distance Vector (AODVv2) Routing." [Online]. Available: https://tools.ietf.org/html/draft-ietf-manet-aodvv2-09

[17] C. Aichele, S. Wunderlich, A. Neumann, and M. Lindner, "Better Approach To Mobile Ad-hoc Networking (B.A.T.M.A.N.)." [Online]. Available: http://tools.ietf.org/html/draft-wunderlich-openmesh-manet-routing-00

[18] Freifunk, "Originator message (ogm)," May 2011, visited on 2015-08-20. [Online]. Available: https://www.open-mesh.org/projects/batman-adv/wiki/OGM

[19] C. M. Kozierok, *The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference*, 1st ed. San Francisco: No Starch Press, Oct. 2005.

[20] V. Jacobson, “Congestion Avoidance and Control,” in *Symposium Proceedings on Communications Architectures and Protocols*, ser. SIGCOMM ’88. New York, NY, USA: ACM, 1988, pp. 314–329. [Online]. Available: http://doi.acm.org/10.1145/52324.52356

[21] M. Allman, V. Paxson, and E. Blanton, “TCP Congestion Control,” RFC Editor, Tech. Rep. RFC5681, Sep. 2009. [Online]. Available: https://www.rfc-editor.org/info/rfc5681

[22] ITU-T, “One-way transmission time,” May 2003. [Online]. Available: http://handle.itu.int/11.1002/1000/6254

[23] “OMNeT++ - Manual version 4.6.” [Online]. Available: http://www.omnetpp.org/doc/omnetpp/manual/usman.html

[24] “INET Framework for OMNeT++ Manual,” Jun. 2012. [Online]. Available: http://omnetpp.org/doc/inet/api-current/inet-manual-draft.pdf

[25] W. Jingyuan, W. Jiangtao, Fellow, IEEE, H. Yuxing, Z. Jun, L. Chao, and X. Zhang, “TCP-FIT: An Improved TCP Congestion Avoidance Algorithm for Heterogeneous Networks,” Tech. Rep. [Online]. Available: http://www.tcpengines.com/wp-content/uploads/2013/11/tcp-whitepaper.pdf

[26] S. Liu, T. Basar, and R. Srikant, “TCP-Illinois: A Loss and Delay-based Congestion Control Algorithm for High-speed Networks,” in *Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools*, ser. valuetools ’06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1190095.1190166

[27] S. Hemminger, “Tcp illinois congestion control source code for linux kernel,” 2007. [Online]. Available: lxr.free-electrons.com/source/net/ipv4/tcp_illinois.c

[28] R. Jain, “Wireless local area networks (wlans) part i,” 2010. [Online]. Available: http://www.cse.wustl.edu/ jain/cse574-10/ftp/j_6lan.pdf

A Appendix 1 - Simulation Configuration

This appendix contains all the settings that were used during the simulations.

Channel Physical Settings	
*.channelControl.carrierFrequency	2.4GHz
*.channelControl.pMax	6mW
*.channelControl.sat	-110dBm
*.channelControl.propagationModel	LogNormalShadowingModel
*.channelControl.alpha	2
*.channelControl.numChannels	1

Table 1.1: Radio Medium Settings

WLAN NIC Configuration (All Mesh Nodes)	
wlan[*].opMode	g
wlan[*].mgmtType	Ieee80211MgmtAdhoc
wlan[*].bitrate	54Mbps
wlan[*].mac.basicBitrate	6Mbps
wlan[*].mac.controlBitrate	6Mbps
wlan[*].mgmt.frameCapacity	10
wlan[*].mac.address	auto
wlan[*].mac.maxQueueSize	14
wlan[*].mac.rtsThresholdBytes	3000B
wlan[*].mac.retryLimit	7
wlan[*].mac.cwMinData	31
wlan[*].mac.cwMaxData	1023
wlan[*].mac.cwMinMulticast	31
client[*].wlan[*].radio.transmitterPower	2mW
wlan[*].radio.transmitterPower	5mW
wlan[*].radio.thermalNoise	-110dBm
wlan[*].radio.sensitivity	-85dBm
wlan[*].radio.pathLossAlpha	2
wlan[*].radio.snirThreshold	4dB
wlan[*].radio.berTableFile	per_table_80211g_Trivellato.dat

Table 1.2: WLAN NIC settings for all Mesh Nodes

Other Settings (ARP, etc)	
arp.retryCount	3
arp.cacheTimeout	30s
arp.retryTimeout	3s
arp.globalARP	TRUE
broadcastDelay	uniform(0s,0.005s)

Table 1.3: Other Settings

TCP	
tcpType	TCP
tcp.advertisedWindow	14*1072
tcp.delayedAcksEnabled	FALSE
tcp.nagleEnabled	TRUE
tcp.limitedTransmitEnabled	FALSE
tcp.increasedIWEnabled	FALSE
tcp.sackSupport	FALSE
tcp.windowScalingSupport	FALSE
tcp.timestampSupport	FALSE
tcp.mss	1072
tcp.tcpAlgorithmClass	*variable*
tcp.recordStats	TRUE

Table 1.4: TCP Settings

TCP on Client Nodes	
client[*].numTcpApps	1
client[*].tcpApp[*].typename	TCPSessionApp
client[*].tcpApp[*].active	TRUE
client[*].tcpApp[*].connectAddress	meshnode[19](ipv4)
client[*].tcpApp[0].connectPort	1000
client[*].tcpApp[0].tOpen	uniform(0.5s, 5s)
client[*].tcpApp[0].tSend	uniform(5s, 6s)
client[*].tcpApp[0].sendBytes	300000000B
client[*].tcpApp[0].sendScript	
client[*].tcpApp[0].tClose	3500s

Table 1.5: TCP Settings on Client Nodes

UDP	
meshnode[19].numUdpApps	1
meshnode[19].udpApp[*].typename	UDPSink
meshnode[19].udpApp[0].localPort	100
client[*].numUdpApps	1
client[*].udpApp[*].typename	UDPBasicBurst
client[*].udpApp[0].localPort	100
client[*].udpApp[0].destPort	100
client[*].udpApp[0].messageLength	1024B
client[*].udpApp[0].sendInterval	0.02s + uniform(-0.01s,0.01s)
client[*].udpApp[*].destAddresses	meshnode[19](ipv4)
client[*].udpApp[0].chooseDestAddrMode	once
client[*].udpApp[0].burstDuration	1s
client[*].udpApp[0].sleepDuration	0s
client[*].udpApp[0].stopTime	-1s
client[*].udpApp[0].startTime	0s
client[*].udpApp[0].delayLimit	100s

Table 1.6: UDP Settings

Mobility (Stationary)	
meshnode[*].mobilityType	StaticGridMobility
meshnode[*].mobility.columns	5
meshnode[*].mobility.rows	4
meshnode[*].mobility.separationX	200m
meshnode[*].mobility.separationY	200m
meshnode[*].mobility.initialZ	6m
client[*].mobilityType	StationaryMobility
client[*].mobility.initialZ	0
mobility.constraintAreaMinZ	0m
mobility.constraintAreMaxZ	200m
mobility.constraintAreaMinX	0m
mobility.constraintAreaMinY	0m
mobility.constraintAreaMaxX	1150m
mobility.constraintAreaMaxY	880m

Table 1.7: Mobility Static

Mobility (Walking)	
client[*].mobilityType	MassMobility
client[*].mobility.speed	3.1mps
client[*].mobility.changeInterval	normal(12s, 0.1s)
client[*].mobility.changeAngleBy	normal(0deg, 40deg)
client[*].mobility.initialZ	0
mobility.constraintAreaMinZ	0m
mobility.constraintAreMaxZ	200m
mobility.constraintAreaMinX	0m
mobility.constraintAreaMinY	0m
mobility.constraintAreaMaxX	1050m
mobility.constraintAreaMaxY	880m
mobility.initFromDisplayString	false

Table 1.8: Mobility Walking

Mobility (Driving)	
client[0].mobilityType	RectangleMobility
client[1].mobilityType	RectangleMobility
client[2].mobilityType	RectangleMobility
client[3].mobilityType	RectangleMobility
client[4].mobilityType	RectangleMobility
client[0].mobility.startPos	0
client[0].mobility.constraintAreaMaxX	950m
client[0].mobility.constraintAreaMaxY	200m
client[1].mobility.startPos	1
client[1].mobility.constraintAreaMaxX	500m
client[1].mobility.constraintAreaMaxY	500m
client[2].mobility.constraintAreaMaxX	960m
client[2].mobility.constraintAreaMaxY	860m
client[3].mobility.startPos	3
client[3].mobility.constraintAreaMaxX	960m
client[3].mobility.constraintAreaMaxY	700m
client[4].mobility.startPos	3
client[4].mobility.constraintAreaMaxX	100m
client[4].mobility.constraintAreaMaxY	860m
client[0].mobility.speed	uniform(18.64mps, 24.85mps)
client[1].mobility.speed	uniform(18.64mps, 24.85mps)
client[2].mobility.speed	uniform(18.64mps, 24.85mps)
client[3].mobility.speed	uniform(18.64mps, 24.85mps)
client[4].mobility.speed	uniform(18.64mps, 24.85mps)
client[5].mobilityType	MassMobility
client[6].mobilityType	MassMobility
client[7].mobilityType	MassMobility
client[8].mobilityType	MassMobility
client[9].mobilityType	MassMobility
mobility.speed	9.6mps
mobility.changeInterval	normal(12s, 0.1s)
mobility.changeAngleBy	normal(0deg, 45deg)
client[10].mobilityType	TractorMobility
client[10].mobility.x1	5m
client[10].mobility.y1	5m
client[10].mobility.x2	960m
client[10].mobility.y2	860m
client[11].mobilityType	TractorMobility
client[11].mobility.x1	250m
client[11].mobility.y1	5m
client[11].mobility.x2 E	960m
client[11].mobility.y2	860m

client[12].mobilityType	TractorMobility
client[12].mobility.x1	500m
client[12].mobility.y1	300m
client[12].mobility.x2	960m
client[12].mobility.y2	860m
client[13].mobilityType	TractorMobility
client[13].mobility.x1	200m
client[13].mobility.y1	250m
client[13].mobility.x2	960m
client[13].mobility.y2	860m
client[14].mobilityType	TractorMobility
client[14].mobility.x1	5m
client[14].mobility.y1	400m
client[14].mobility.x2	960m
client[14].mobility.y2	860m
mobility.speed	24.85mps
mobility.rowCount	4
mobility.initialZ	0
mobility.updateInterval	1s
mobility.constraintAreaMinZ	0m
mobility.constraintAreaMaxZ	200m
mobility.constraintAreaMinX	0m
mobility.constraintAreaMinY	0m
mobility.constraintAreaMaxX	960m
mobility.constraintAreaMaxY	860m

Table 1.9: Mobility Driving

B Appendix 2 - Creating a new project

To create a new simulation project it requires a number of steps:

1. Start OMNeT++ IDE by double clicking on its icon, if using Windows, otherwise run omnetpp at a linux terminal.

2. Select a workspace to host your project. As a rule of thumb try to avoid directories with spaces.

3. From the File menu select New -> OMNeT++ project. Choose a name and click next.

4. On this screen, choose Empty Project and click Finish or in case more options are required choose "Next".

5. A default Network Description File (NED) should have been created automatically for you.

6. Should you desire to create a new NED file, choose File -> New -> Network Description File then select the project folder, choose a file name along with the ".ned" extension, select an Empty NED File and click Finish. By default the new NED file will open in the design mode where dragging and dropping components is possible. Of course, there is the option to switch to source code mode, by clicking "Source" at the bottom of the pane. With source mode one can copy-paste code from existing projects or write his/her own code.

7. The last component that is needed is the omnetpp.ini, which is the heart of any simulation as it contains all the parameters and configurations for all the scenarios. To create the file (if not there), navigate to File -> New -> Initialisation File (ini) and choose the Empty Ini template. In case you are being asked to choose a network click Browse... and OMNeT will detect and propose the network in the NED file you created earlier. When done click Finish to conclude this step.

8. One last step that might be required if you choose to use the INET framework is: right click on your project and select Properties. At the left pane there would be an option named "Project References" select it, and on the right pane, locate and choose "inet".

Now the workspace and the project are ready for deployment. It is only needed to define a network by adding the necessary components in the NED file, and set the right parameters in the omnetpp.ini file.