ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

« Presentation, study and comparison of simulation software for wireless local area networks »

«Εικόνα»

Του φοιτητή
Κουτλή Μιχαήλ
Αρ. Μητρώου: 144219

Επιβλέπων
Χατζημίσιος Περικλής
Καθηγητής

06/06/2021

Presentation, study and comparison of simulation software for wireless local area networks

Κωδικός Δ.Ε. 21250

Κουτλής Μιχαήλ

Χατζημίσιος Περικλής

Ημερομηνία ανάληψης Δ.Ε. 02/04/2021

Ημερομηνία περάτωσης Δ.Ε. 06/06/2021

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

«Αφιέρωση»

Περίληψη

Στόχος της πτυχιακής εργασίας είναι η εισαγωγή του αναγνώστη στα ασύρματα τοπικά δίκτυα (WLANs), εστιάζοντας στο IEEE 802.11 (και, πιο συγκεκριμένα, στις διάφορες τροπολογίες και στα καινούρια χαρακτηριστικά τους) και στις διάφορες εναλλακτικές λύσεις που υπάρχουν. Στη συνέχεια, θα εστιάσουμε στους διάφορους προσομοιωτές δικτύων που υποστηρίζουν WLANs, όπως ο Riverbed Modeler, το OMNeT++, το J-Sim και ο ns-3 (Network Simulator 3). Πιο συγκεκριμένα, θα παρουσιαστούν τα διάφορα χαρακτηριστικά και οι διάφορες λειτουργίες των παραπάνω προσομοιωτών, καθώς επίσης και τα πλεονεκτήματα και μειονεκτήματα τους. Μετά από αυτό, θα μετρηθούν η απόδοση τους, η κατανάλωση των πόρων συστήματος και η δυνατότητα κλιμάκωσης τους και θα παρουσιαστούν σε μορφή γραφημάτων για ευκολότερη κατανόηση.

Στην επόμενη ενότητα, θα γίνει σύγκριση μεταξύ μερικών από τα πιο δημοφιλή πρωτόκολλα δρομολόγησης (AODV, DSDV και DSR) στο OMNeT++ ως προς το packet error rate, το queueing time και το ελάχιστο signal to interface ratio, για να εντοπιστούν τα δυνατά και τα αδύναμα σημεία τους. Αυτό θα επιτευχθεί με την εκτέλεση προσομοιώσεων για κάθε πρωτόκολλο δρομολόγησης και την εξαγωγή των αποτελεσμάτων των προσομοιώσεων.

Έπειτα, μετά την παρουσίαση και την ανάλυση δύο συνθετικών μοντέλων κίνησης που μιμούνται μοτίβα κίνησης της πραγματικής ζωής, το SOLAR και το CMM, θα γίνουν μετρήσεις για να πραγματοποιηθεί σύγκριση μεταξύ τους. Αυτές οι μετρήσεις περιλαμβάνουν το ελάχιστο Signal-to-Interference-plus-Noise-Ratio, καθώς επίσης και το packet error rate.

Στην τελευταία ενότητα αυτής της πτυχιακής εργασίας, θα παρουσιαστεί ένα custom σενάριο ad-hoc δικτύου που καλύπτει ολόκληρη την έκταση του campus του ΔΙΠΑΕ και χρησιμοποιεί το CMM, και μεταβάλλοντας παραμέτρους όπως ο αριθμός των κόμβων, η ταχύτητα του δικτύου και το μέγεθος του πακέτου θα αποφασιστεί ποια είναι η ιδανική διαρύθμιση για τον συγκεκριμένο μας στόχο.

Presentation, study and comparison of simulation software for wireless local area networks

Michael Koutlis

Abstract

The goal of this thesis is to introduce the reader to the Wireless Local Area Networks (WLANs), focusing on IEEE's 802.11 (and, more specifically its various amendments along with their newest offerings) as well as the alternative solutions that exist. Afterwards, attention will be brought to the various network simulators that support WLANs, like Riverbed Modeler, OMNeT++, J-Sim and ns-3 (Network Simulator 3). More specifically, the various characteristics and features of the aforementioned simulators will be presented, as well as their advantages and disadvantages. After that, their performance, system resource utilization and scalability will be benchmarked and put into graphs for easier understanding.

In the next section, a comparison will take place between some of the most popular routing protocols (AODV, DSDV and DSR) in OMNeT++ in regards to the packet error rate, the queuing time and the minimum signal to interference ratio, to determine their strengths and weaknesses. This will be accomplished by running simulations utilizing each routing protocol and extracting the simulation results.

Next, after presenting and analyzing two synthetic mobility models that mimic real-life movement patterns, SOLAR and CMM, benchmarks will take place to draw comparisons between them. These benchmarks include the minimum Signal-to-Interference-plus-Noise-Ratio, as well as the packet error rate.

In the final section of this thesis, a custom ad-hoc network scenario that extends over the whole IHU's campus and utilizes CMM will be presented, and by modifying parameters such as the number of nodes, the network speed and the size of the packets will be determined which is the most ideal configuration for our particular goal.

Contents

Figure Index

Table index

Screenshot index

Chapter 1: Communication Technologies

1.1 Introduction

Since mankind's dawn, people have had the inherent need to communicate with other people that were further apart. That is how the first forms of communication and the first networks were formed. Since computers and the Internet had not yet been invented, people more often than not utilized sound and light (like fire for example) as their primary means to communicate with other people who were not within earshot or viewing distance. At the same time, the first alphabets and languages were starting to make an appearance.

Of course, since that time technology has made great leaps, especially in the communications department. We now have telephones, smartphones, radios, TVs, and computers. Utilizing these devices, the telecommunication networks were created.

Nowadays, communication is faster than ever before. The Internet allows everyone with access to it to instantly send messages to anyone in the world, to exchange files, to make e-payments, to listen to music, to watch videos, to research information and much more. A big chunk of mobile Internet traffic utilizes Wi-Fi (73.8% in Q4 of 2018 [1]), and by extension WLANs, so this is the main focus of this thesis.

New protocols are proposed all the time, but they first have to be tested in order to determine their functionality and their benefits over already-existing ones. This is where network simulators come into play. They offer the option to test these protocols before implementing them in the real world, saving money (since little to no hardware is necessary) and time.

Of course, network simulators can also be used as an effective means for teaching or demonstrating network concepts to students [2], especially the ones that use graphical interfaces and animations, since they can provide real-time feedback as to how a specific network operates.

1.2 Networks

1.2.1 Network classification

Networks are classified by their scale. In other words, by the area they provide coverage for:

Personal Area Networks (PANs)

A Personal Area Network is a network for interconnecting devices centred on an individual person's workspace. These networks are usually wireless, they have a small range and use the Bluetooth protocol. A typical use case scenario would be various peripherals which are connected to a PC. Another technology utilized by PANs is Radio Frequency Identification (RFID), which uses radio frequencies to identify humans or objects. It could be considered as an evolution of the barcode.

Local Area Networks (LANs)

A Local Area Network is a network for interconnecting computers within a limited area such as a residence, a school, a laboratory, an office campus or an office building, and around them. LANs can either be wired or wireless. The most popular method for wired LAN is Ethernet (IEEE 802.3). Alternative methods include USB and Powerline Networking (PLN). As far as wireless LANs (WLANs) are concerned, Wi-Fi (IEEE 802.11) is the prevalent method.

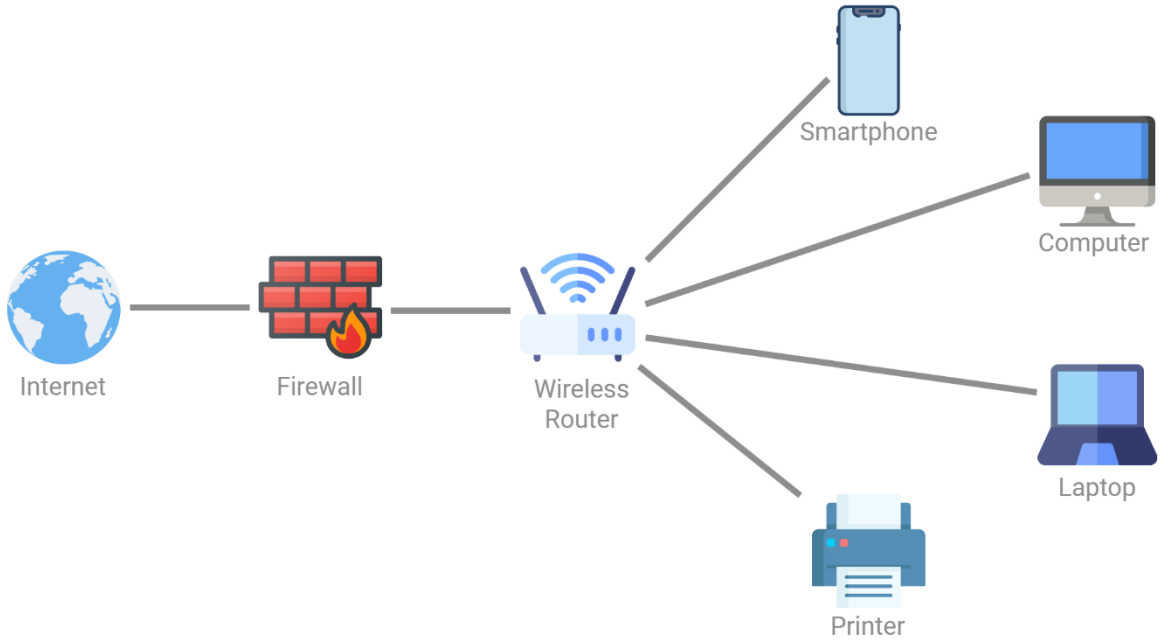


Figure 1.1: A diagram of a LAN

Metropolitan Area Networks (MANs)

A Metropolitan Area Network is a network for interconnecting users with computer resources in a geographic region of the size of a metropolitan area. The term is applied to the interconnection of LANs into a single, larger network which in turn may then also offer efficient connection to a Wide Area Network. The most common communication standard in this category is WiMAX [3] (IEEE 802.16), which is wireless.

Wide Area Network (WANs)

A Wide Area Network is a telecommunications network that extends over a large geographical distance, like a country or a continent, for the primary purpose of computer networking. Businesses, educational institutes and government entities utilize WANs to relay traffic to clients, employees, buyers, suppliers, and students from all over the world. In other words, a business, for example, can conduct its business regardless of its location. In essence, the whole Internet can be considered a WAN.

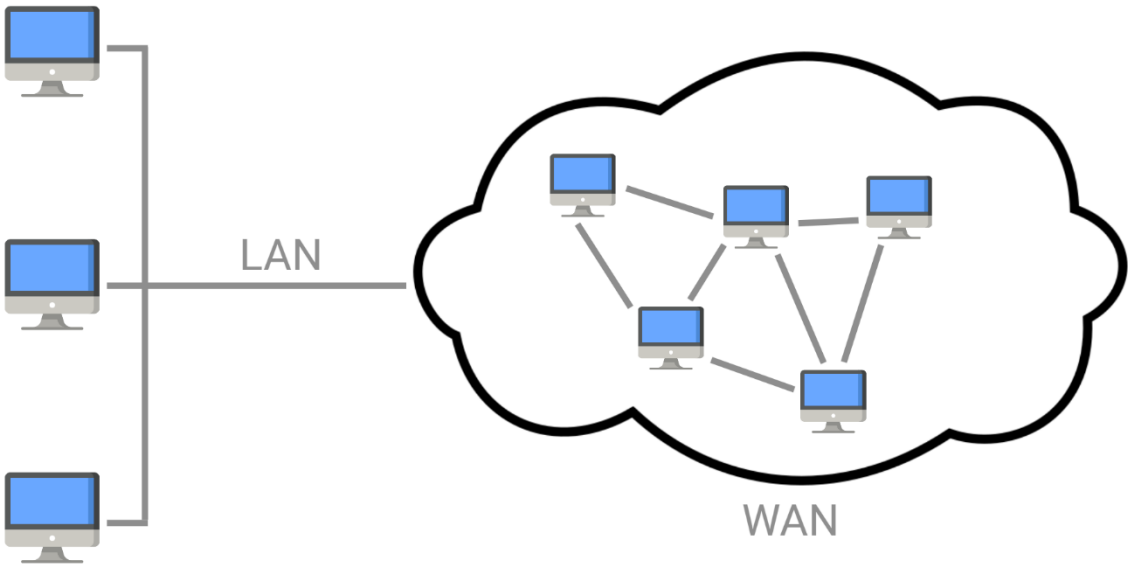


Figure 1.2: A LAN and WAN scheme

1.2.2 Switching

The term “switching” in communication networks describes a technique to forward the information from the sender to the receiver. There are two types: circuit switching and packet switching.

Circuit switching requires the establishment of a dedicated circuit between the two nodes before they can start communicating. The circuit provides guarantee that the full bandwidth of the channel is available and remains connected for the whole duration of the communication session. It basically functions as if the two nodes were physically connected to one another with an electrical circuit.

Packet switching is a method of grouping data that is transmitted over a digital network into packets. These packets consist of a header and a payload. The header is used by networking hardware to direct the packet to its destination, where its payload can be extracted. Unlike circuit switching, the sender and the receiver do not have to be connected to one another. This means that two messages originating from station X, and being sent to station Y could travel through a completely different route.

1.2.3 Multiplexing

Multiplexing (sometimes referred to as "muxing") is a way of sending multiple signals or streams of information over a communications link at the same time in the form of a single, complex signal. Then, the receiver recovers the separate signals with a process called demultiplexing (or demuxing).

Networks use multiplexing for two reasons:

- To enable any network device to talk to other network devices without the need to dedicate a connection for each pair
- To make an expensive resource stretch further (for example, sending multiple signals down each cable/fiber running between major metropolitan areas)

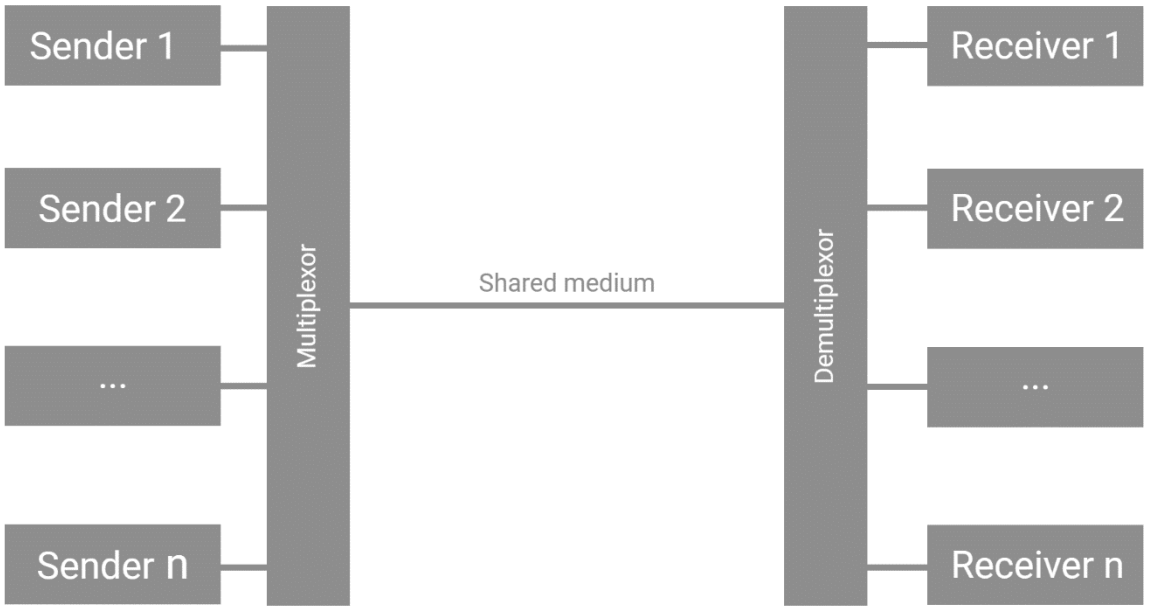


Figure 1.3: How multiplexing and demultiplexing works

1.3 Current status of telecommunications

Especially since the COVID-19 outbreak, the role of telecommunications is more integral to our society that ever before. Online messages, video calls and file sharing are a daily routine for a large percentage of the working force, since many people had, and continue to have, to work from home. This has also led to the development of Internet of Things (IoT) products that can remotely monitor systems, enabling safer pandemic work practices while at the same time allowing production to continue. This means that millions of people around the world rely on their Internet connection to go to work and produce results. People with slow Internet connections suddenly felt left behind, and as a result the demand for high-speed, fiber optics connections increased, despite the heftier price tag associated with them [4]. At the same time, companies rallied to increase the reliability of their services, as more and more people started depending on them, so a few of them invested in submarine infrastructures, generally regarded as the most reliable solution. The list includes companies like Google, Facebook, Amazon and Microsoft, which, collectively owned or leased at least half of the submarine bandwidth in 2018 alone [5]. The lifetime of a submarine cable is expected to be around 25 years and more and more are being deployed every year.

It is fair to say that while the digital transition was already in effect way before the outbreak, the last year or so has accelerated the efforts by a long way. One noteworthy example is Telecom Italia, which transitioned all of its agents (7000 of them) from working in the company's call center to working from their homes in just 72 hours [6].

Demand has also increased for automation tools as well as cloud solutions. This means that, theoretically at least, companies now can be more efficient as well as more flexible, since their need for a physical space where all their machines and files are stored locally is not as strong as it once was.

Apart from the changes COVID-19 brough on, some changes where due to happen anyway. In 2020, the number of mobile internet users was 4.28 billion, meaning that over 90 percent of the global internet population use a mobile device in order to go online. Mobile ownership is expected to increase even more in the future, since mobile technologies are becoming more affordable and available than ever before. This upward trend in mobile internet adoption is particularly visible in developing digital markets where mobile networks are the primary means of internet access. Today, mobile internet traffic accounts for more than 55% of the total web traffic [7].

1.3.1 5G

Another technological advancement that has been happening the last few years and is expected to bring big changes is 5G, the fifth-generation technology standard for broadband cellular networks. The first deployments of 5G by cellular phone companies started in 2019 (with the protocol being introduced in July 2016 [8]) and haven't stopped since. The increased bandwidth they provide has a lot of people expecting 5G to be able to compete with traditional Internet Service Providers (ISPs) and their cable Internet. Apart from the download speeds, the latency is also significantly improved. Results show an average of 10-30 ms, depending on the distance from the antenna. 1 ms – 5 ms is theoretically possible, but highly unlikely for these numbers to be achieved in the real world. Still, by 2023, the total number of devices that will be utilizing 5G is expected to be 1.01 billion [9]. This translates to an annual growth rate of more than 217%.

By 2024, experts predict that there will be around 1.9 billion 5G subscriptions worldwide, with Developed Asia and North America expected to embrace the 5G technology faster than all the other regions: by 2025, around half of overall mobile connections are projected to be 5G-enabled in these two regions. Also, 5G is expected to be embraced by the public quicker than 4G was, with experts predicting that by 2023, more than half of all the new smartphones sold will be 5G-enabled [10].

Of course, another area that is expected to benefit from the advance of 5G is the IoT and machine-to-machine communications, like autonomous vehicles. The total number of 5G-enabled devices is expected to skyrocket as the months and years go by, but, according to Gartner, the biggest slice of the pie is going to be occupied by surveillance cameras, which will account for 1/3 of all of the installed 5G endpoints at the end of 2023 [11].

The increased bandwidth of 5G is achieved thanks to using additional higher-frequency radio waves in addition to the low and medium band frequencies used previously [12]. However, as it is known, higher frequency radio waves have a shorter physical range, so the geographic cells are smaller than they were with 4G. This means that the device has to be quite close to the antenna in order to fully benefit from 5G speeds, with as little in the way of obstacles between them as possible. To combat this, to a degree, 5G can operate on three different frequency bands: low, medium and high.

- Low-band 5G uses the frequency range 600 MHz – 850 MHz, similar to the one on 4G cellphones, with download speeds of 30 Mbit/s – 250 Mbit/s, a little higher than 4G [13]. They extend to great distances, but there aren't very wide channels available, and many of those channels are already being used for 4G.
- Mid-band 5G uses the frequency range 2.5 GHz - 3.7 GHz, with download speeds of 100 Mbit/s – 900 Mbit/s. The area coverage can extend to several kilometers and is the one most commonly deployed, making it, although unofficially, the minimum service level.
- High-band 5G (or millimeter-wave) uses the frequency range 25 GHz – 39 GHz, with download speeds that can reach in the Gbit/s range. These airwaves haven't been used for

consumer applications before. As of 2021, the fastest speed achieved with 5G is 5.23 Gbit/s [14]. Due to the low range and the fact that even windows can obstruct its signal, plans for high-band 5G are limited to areas where a large number of people gather, such as large parks and sporting arenas [15].

Devices compatible with 5G automatically connect to the antenna that can provide the highest connection speed, so the end user always benefits from the higher speeds possible.

1.3.2 Internet of Things

As we all know, the Internet of Things describes a network of physical devices that can exchange data with other devices or be controlled over the Internet. For example, a smart bulb that can be controlled with a smartphone, a smart thermostat, a smart speaker, a humidity sensor for bathroom use and a connected traffic light are all considered IoT devices. There wasn't a specific event that triggered the "birth" of the IoT, but it is believed to be around 2008 to 2009, with the devices/people ratio moving from just 0.08 in 2003 to 1.84 in 2010 [16] and to an expected 3.5 in 2021, meaning there will be more than 27 billion networked devices [17]. By 2040, the number of IoT devices is projected to exceed the 1 trillion mark [18].

IoT devices have a plethora of applications, with perhaps the most well-known being home automation (smart home), wearables and connected vehicles. Other applications include the collection of temperature and humidity levels for farming, automations at city-level (smart cities), devices for reconnaissance for the military, monitoring the production process in factories, spotting a faulty piece in a complex machine and warning its users for the potential failure and freight monitoring. The list is truly endless, and only seems to be limited by the human imagination and, in some cases, the cost of implementation, which while is getting smaller each year, is still a bit too expensive to implement in every object.

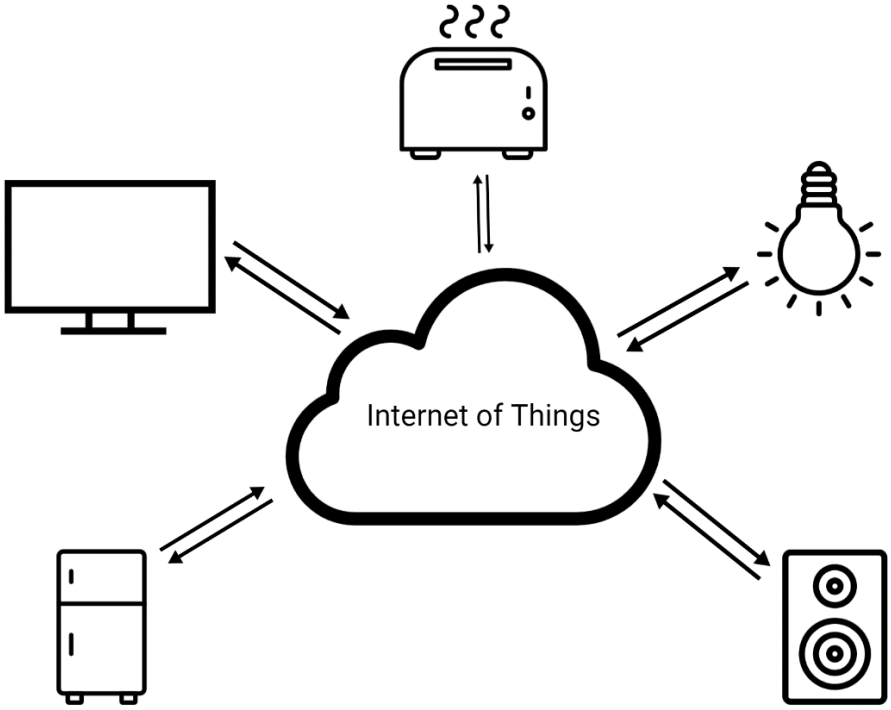


Figure 1.4: A visualization of the Internet of Things

Perhaps the most important problem that IoT faces is the lack of standardization. Most consumer products like smart sensors and security cameras require a hub in order to work properly, which wouldn't be that much of a problem if this hub was compatible with products from other brands as well. Most of the companies seem to follow their own route, using their own protocols, and as a result, these devices can't easily communicate with one another. We can observe a slight shift to this kind of behavior, as more and more devices start to be shipped free from the need to use a dedicated hub, but it is safe to assume that this behavior has already put a dent to the early adoption phase.

Another issue is security. No computer system is 100% secure from threats, and with billions of devices connected to the Internet, security threats need to be addressed. Data integrity issues could topple economies across world, especially since every passing year IoT expands to sectors like construction, city utilities and production. Jamming radio networks in a city or on the battlefield could become disastrous very quickly. These security concerns could potentially delay the progress of IoT.

Finally, the e-waste produced from all of these devices in no laughing matter either. We live in a time where it is easier for us to buy a new product when the current one malfunctions than to try to repair it. We already produce 20 to 50 tons of e-waste every single year [19], so the influx of a variety of new products on the market whose features will be considered perhaps not up to par in a few years, leading to their replacement, can be a problem. Currently, only about 20% of e-waste is recycled [20].

Chapter 2: IEEE 802.11

IEEE 802.11 standards have enabled a mass market, with a huge impact in various aspects of the everyday life, like the work space, the home, in public areas and in commercial establishments. Examples of devices that incorporate WLAN technology include PCs, laptops, TVs, mobile phones, gaming consoles, DVD/Blu-ray players and many, many more. Thanks to its low-cost chipsets and support for high data rates, IEEE 802.11 has become a universal solution for an ever-increasing application space. As a result, several amendments to the basic IEEE 802.11 standard have been developed through the years and/or are still currently under development. These not only fix technology issues, but they also add functionality that is expected to be required by future applications.

Standards in IEEE 802.11 target the Physical Layer (PHY) and Medium Access Control Layer (MAC). During WLAN's conception, it seemed that it would be just another PHY of one of the already available standards. The first candidate that was considered for this was IEEE 802.3 (widely known as Ethernet). Nonetheless, it soon became clear that the radio medium is very different from the well-behaved wire, since the attenuation, even over short distances, is so tremendous that collisions are not feasible to be detected. As a result, IEEE 802.3's carrier sense multiple access with collision detection (CSMA/CD) could not be applied. The attention was then turned towards IEEE 802.4 as the next candidate to be considered. Its coordinated medium access, the token bus concept, was believed to be superior to IEEE 802.3's contention-based scheme. As a result, WLAN began life as IEEE 802.4L.

However, as early as in 1990, it was apparent that token handling in radio networks was difficult. This resulted in the standardization body realizing that a wireless communication standard would need its own, unique MAC. As a direct result, on March 21, 1991, project IEEE 802.11 was approved. It would not be until 1997 that the first standard was published, under the name IEEE 802.11-1997 or IEEE 802.11 (legacy mode), as it came to be known today. Most of the protocols described by this version are rarely used today, which is no surprise, since there has been a plethora of new amendments since then. Despite that, it provides three solutions at the PHY layer:

- A frequency hopping (FHSS)
- A direct sequence spread spectrum (DSSS) PHY in the unlicensed Industrial, Scientific and Medical (ISM) frequency band at 2.4 GHz
- An infrared PHY at 316-353 THz

However, commercial infrared implementations never existed, despite all three solutions providing a basic data rate of 1 Mb/s with an optional 2 Mb/s mode.

Basic IEEE 802.11 MAC operates based on a listen-before-talk scheme, similar to IEEE 802.3, known as the Distributed Coordination Function (DCF). It implements Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) as the medium access method, as opposed to collision detection as in IEEE 802.3. A sizeable percentage of the available raw channel capacity is sacrificed via the CSMA/CA mechanisms in order to improve the reliability of data transmissions under diverse and harsh environmental conditions. Since collisions cannot be detected in the radio environment, IEEE 802.11 waits for a back-off interval before each frame transmission, rather than after collisions. In addition to DCF, the original IEEE 802.11 standard specifies an optional scheme, that depends on a central coordination entity, the point coordination function (PCF). PCF uses the so-called point coordinator (PC) that operates during the so-called contention-free period. This is a periodic interval during which only the PC initiates frame exchanges via polling. On the other hand, PCF failed to provide robustness against hidden nodes, and as a result, the adoption by manufacturers was negligible.

The DSSS version of legacy IEEE 802.11 was rapidly supplemented by the IEEE 802.11b amendment in 1999, which increased the bit rate to 11 Mbit/s, leading to more widespread adoption of IEEE 802.11 networks. Only then did multiple interoperable products become available in the market from multiple vendors. As a direct result, very few networks were implemented on the IEEE 802.11-1997 standard.

After the first IEEE 802.11 standard was published, in 1997, the Working Group (WG) received feedback that many products failed to provide the degree of compatibility that customers expected. For example, the Wired Equivalent Privacy (WEP) encryption scheme would not work between devices that were not from the same vendor. This problem resulted in the formation of the Wireless Ethernet Compatibility Alliance (WECA) in 1999, changing its name in 2003 to the Wi-Fi Alliance (WFA). Since then, IEEE 802.11 has gone through many iterations, each bringing improvements to the PHY layer and to the MAC layer.

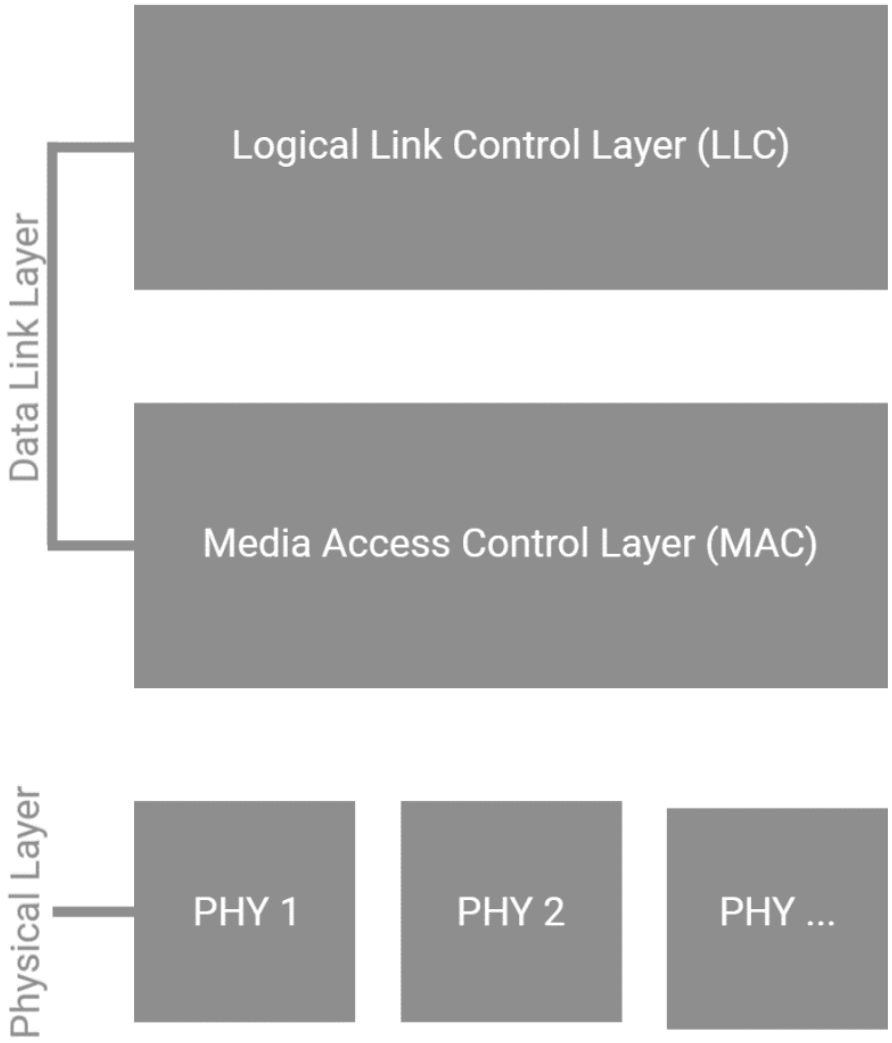


Figure 2.1 IEEE 802.11 Protocol Stack

2.1 IEEE 802.11a

IEEE 802.11a started in September 1997 and was published in 1999, a little after IEEE 802.11b was released. This caused some confusion, as people thought that it was an older amendment, since the letters were in reverse order than normally found. It added a 52-subcarrier orthogonal frequency-division multiplexing (OFDM) PHY that offers support for up to 54 Mb/s data rate. If needed, the data rate can be lowered to 6, 9, 12, 18, 24, 36 or 48 Mb/s.

Of those 52 subcarrier channels, 48 are for data and the remaining 4 are pilot subcarriers. The symbol duration is 4 μs, of which 0.8 μs is the guard interval. In the real world, the achievable data rate is somewhere around 25 Mb/s. IEEE 802.11a originally had 12 or 13 non-overlapping channels, 12 for indoor use and 4 or 5 for outdoor use. Communication with plain IEEE 802.11 devices is impossible, since IEEE 802.11a operates in the unlicensed 5 GHz band. It is also incompatible with IEEE 802.11b devices, unless they are dual band capable, since they operate on different bands. Despite supporting higher rates than IEEE 802.11b, the latter proved to be more popular in the market, as more devices supporting IEEE 802.11b were produced.

Since IEEE 802.11a uses the 5 GHz band, it avoids the 2.4 GHz band and the possible degradation of service that comes with it, thanks to it being heavily used (wireless phones, microwave ovens, baby monitors), often to the point of being considered crowded. This means that IEEE 802.11a boasts an increase in the reliability factor compared to other amendments of IEEE 802.11. On the other hand, using the 5 GHz band means that its range is more limited than that of IEEE 802.11b and IEEE 802.11g, since the signal has less power to pierce through solid obstacles like walls and furniture.

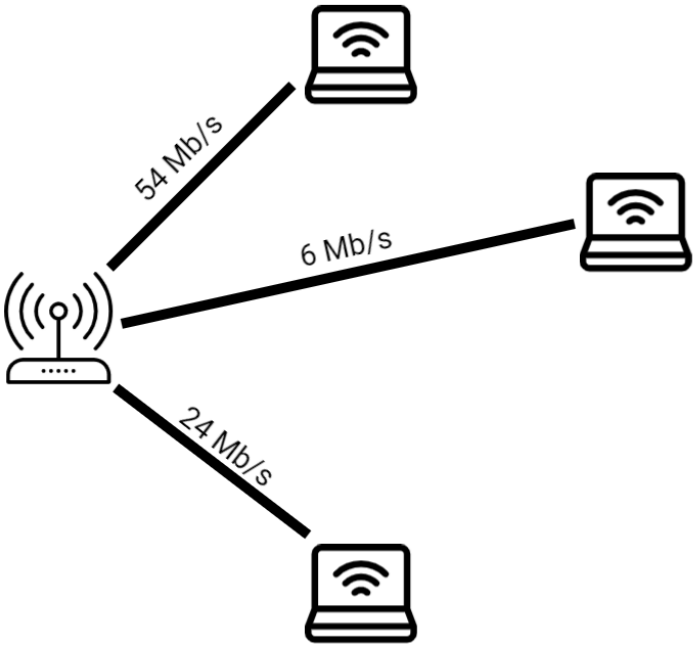


Figure 2.2: An example of an IEEE 802.11a network

2.2 IEEE 802.11b

The IEEE 802.11b standard was published in 1999 and has a maximum raw data rate of 11 Mbit/s, although in real-life conditions, the data rate is about 5.9 Mbit/s is TCP is used and about 7.1 Mbit/s if UDP is used. Products using IEEE 802.11b appeared on the market in early 2000 and the dramatic increase in throughput they offered compared to the original IEEE 802.11 protocol, in conjunction with substantial price reductions led to its rapid acceptance as the definitive wireless LAN technology. This meant that IEEE 802.11b was a true success story, since it led to the widespread adoption of Wi-Fi, with the next amendments building of its success. Unlike IEEE 802.11a, it operates in the 2.4 GHz range.

In order to transmit data, IEEE 802.11b uses the CSMA/CA technique that was defined in the original IEEE 802.11 standard. With this technique, when a node wants to transmit something, it listens for a clear channel and only then transmits. Next, it listens for an acknowledgement and if it doesn't receive one, it backs off for a random amount of time (called back off interval) and then listens for a clear channel and then transmits the data again.

IEEE 802.11b implements another scheme, called Adaptive Rate Selection (ARS). If the signal drops or interference levels rise, the system has the ability to adopt a slower data rate (1, 2 or 5.5 Mb/s) with more error correction. This makes it more resilient.

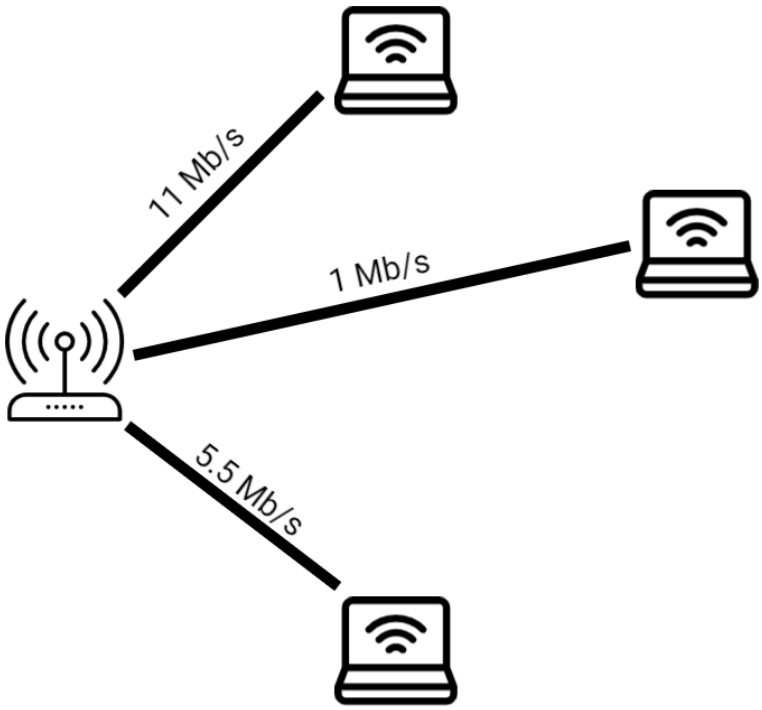


Figure 2.3: An example of an IEEE 802.11b network

2.3 IEEE 802.11g

IEEE 802.11a's lack of interoperability led to the creation of IEEE 802.11g, which was approved in June 2003 and introduced the benefits of OFDM to the 2.4GHz band. In addition, Direct Sequence Spread Spectrum (DSSS) was also used. The DSSS makes the transmitted signal wider in bandwidth than the information bandwidth. After the receiver despreads the signal (which is a relatively easy process since it already knows the spreading sequence produced by the transmitter), the information bandwidth is restored, while the interference is reduced to a great degree [21]. If an "undesirable" transmitter broadcasts on the same channel but using a different spreading sequence, or no sequence at all, the despreading process can and will reduce that signal's power. This provides resistance to intended or unintended signal jamming.

IEEE 802.11g boasts a theoretical maximum data rate of 54 Mbps, although this translates to a real maximum data rate of about 24 Mbps. IEEE 802.11g is fully compatible with IEEE 802.11b, although this was no easy task, with reports stating that a great deal of time was put into making this possible. Furthermore, the presence of an IEEE 802.11b device in an IEEE 802.11g network can have a negative impact to its speed performance.

The migration from IEEE 802.11 to IEEE 802.11g devices was made easy thanks to IEEE 802.11g's extended rate PHY that provided DSSS-compatible signalling. In addition, its proprietary packet binary convolutional code (PBCC) made possible the support for additional data rates of 22 Mb/s and 33 Mb/s. Although PBCC is rarely applied today, it set a de facto standard and became an optional modulation and coding scheme of IEEE 802.11g.

IEEE 802.11g occupies a nominal 22 MHz channel bandwidth, which makes it possible to accommodate up to three non-overlapping signals within the 2.4 GHz band. The space between the channels is 5 MHz.

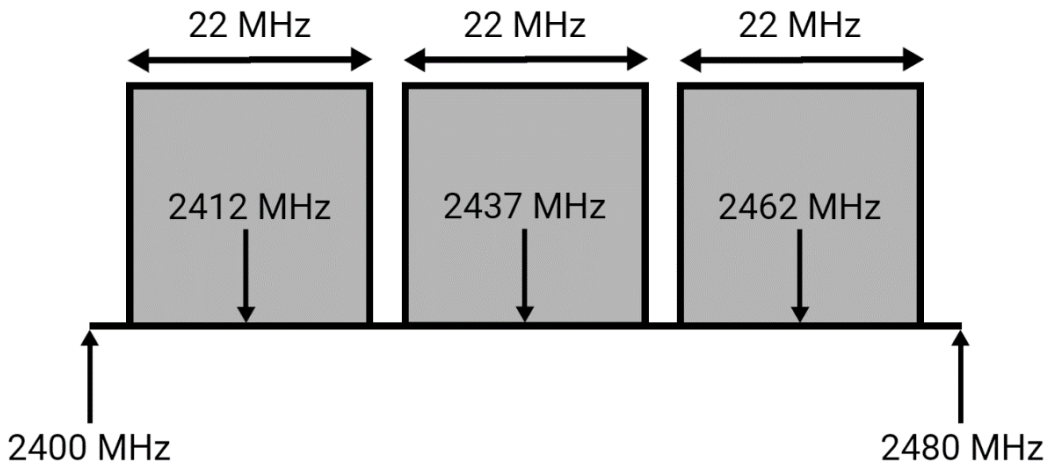


Figure 2.4: Wireless channels of IEEE 802.11g

2.4 IEEE 802.11h

In 2003, in order to comply with the European regulatory requirements for the 5GHz band, IEEE 802.11h was introduced. According to the new regulatory requirements, satellite uplink and radar stations using the same 5 GHz band have to be secured from interference. This is because in the past, the radars used to operate in frequency ranges where they were the only type of device operating there.

As time went on however, regulatory agencies opened those frequencies for other uses, like WLAN. As a direct result, IEEE 802.11h defines MAC mechanisms to enforce dynamic frequency selection (DFS) and transmit power control (TPC). Devices that comply with the DFS protocol are able to detect when a radar is occupying the channel, then stop using that occupied channel for at least 30 minutes, monitor another channel for at least 1 minute and switch to it if it is clear, meaning that there is no radar using that specific channel.

IEEE 802.11h also provides wireless access point (WAP) capabilities to reduce electromagnetic interference between radar and satellite signals existing in a 5 GHz frequency band. Although designed to meet European regulations, IEEE 802.11h is nowadays applicable in many other countries.

2.5 IEEE 802.11i

As discussed in the IEEE 802.11e segment, IEEE 802.11's initial encryption scheme WEP caused a lot of problems in the market. The lack of compatibility between devices from different vendors resulted in a lot of networks operating non-encrypted. Additionally, WEP's pre-shared key concept did not allow it to be implemented into enterprise networks, where every device should have its own, unique, key. As a work-around, companies started using IP-based virtual private networks (VPNs), resulting in IEEE 802.11 becoming synonymous with insecurity.

The establishment of IEEE 802.11i in 2004 attracted many security experts, but the market did not wait for a solution. Thus, WFA started its Wi-Fi Protected Access (WPA) certification program. The upgrades had to be firmware-only in order to not deem the already-existing devices worthless, so WPA does not rely on any new encryption schemes. Its Temporal Key Integrity Protocol uses RC4 for encryption, like WEP, but with a few key changes to key initialization and renewal.

WPA2 denotes the final IEEE 802.11i amendment that includes an additional encryption scheme that was designed from scratch. As the new scheme relies on the Advanced Encryption Standard (AES) defined by the U.S. National Institute of Standards and Technology (NIST), security had been upgraded. However, since the new scheme was not a simple firmware change, older hardware could not be upgraded.

In September 2009, an extension of IEEE 802.11i was published, under the name IEEE 802.11w. It targeted authenticated and encrypted management frames. Although these frames do not carry user data, fraud can cause disconnection and also opens the door for wireless denial of service (DoS) attacks.

2.6 IEEE 802.11j

IEEE 802.11j was finalized in 2004 and it describes the necessary means in order to comply with the Japanese regulatory requirements for the operation of IEEE 802.11 equipment for indoor, outdoor and mobile applications in the 4.9GHz and 5GHz frequency bands. This means that it was designed specifically to be used in the Japanese market. IEEE 802.11j defines uniform methods that let APs move to new frequencies or change channel width in order to achieve better performance or capacity.

It is worth mentioning that in order to use this frequency band, registration is necessary. Currently, only the 4.9-5 GHz is available to use, while the 5.030-5.091 GHz spectrum has been revoked. IEEE 802.11j is also the first amendment that defines PHY operation with 10 MHz bandwidth, in addition to the formerly preferred 20 MHz channelization. It also specifies regulations and recommendations that impose requirements on the power of transmitting nodes, modes of operation, channel alignment and undesired frequency signal levels.

2.7 IEEE 802.11e

In March 2000, IEEE 802.11e was originally approved, having the goal of expanding the WLAN standard. Its key points were improved efficiency, support for QoS, and enhancements in security. However, already in 2001, IEEE 802.11's frame encryption algorithm WEP was broken by an attack, so the security enhancements were displaced to a new TG called IEEE 802.11i. Finally, in 2005, IEEE 802.11e was approved to support QoS. It offers hybrid coordination function (HCF), meaning that it has two MAC protocol versions with centralized and distributed control, respectively.

IEEE 802.11e helped spawn Wi-Fi Multimedia (WMM), which incorporates a subset of functions from IEEE 802.11e Draft 6 (published on November 2003). In May 2007, a Study Group was formed to adapt the IEEE 802.11e to the WMM specification, but a project could not be approved, so the TG was dissolved 6 months later, in November 2007.

It also implements block acknowledgements to improve TXOP efficiency by allowing IEEE 802.11e devices to transmit consecutive frames without intermediate acknowledgement frames (ACKs) being sent to the sender. Instead, the sender receives a single block ACK to indicate success or failure of reception for each frame transmitted. This results in overhead being saved, since gaps for transceiver turnaround can be avoided. However, WMM does not feature this efficiency enhancement.

One common problem with WLANs that use an access point (AP) is that all traffic has to be relayed through the AP. However, thanks to IEEE 802.11's popularity, more and more devices need to exchange traffic locally. For example, the video stream of a Blu-ray player positioned just below an HDTV must be sent not once, but twice in the WLAN:

- From the Blu-ray player to the AP
- From the AP to the HDTV

If a direct frame exchange between adjacent devices was possible, it would save half of the data transmission, and thanks to the shorter communication distance, the devices would also benefit from a higher-rate MCS. This thought led to Direct Link Setup (DLS), which is an optional solution offered in IEEE 802.11e, that requires a DLS-capable AP. Unfortunately, DLS in not part of WFA's WMM certification, which resulted in almost no DLS-capable devices being available in the market.

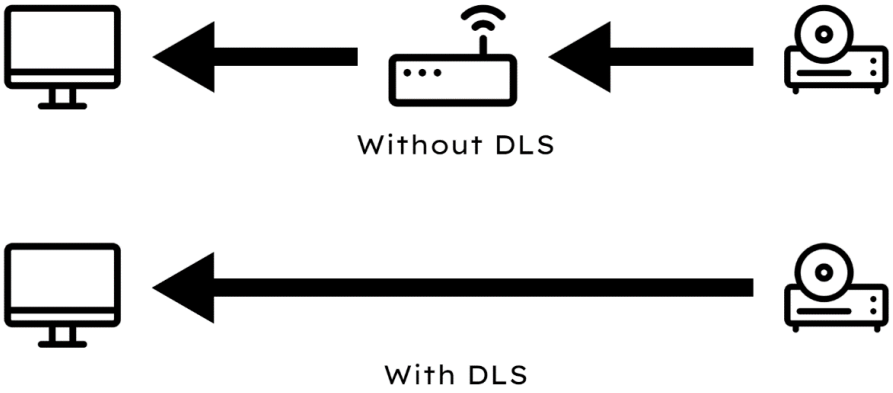


Figure 2.5: An example scenario with and without DLS

2.8 IEEE 802.11r

Since the number of highly mobile devices keeps steadily increasing, roaming support has become increasingly important. Without the features implemented in IEEE 802.11r (published in 2008 [22]), a device in motion can every so often lose connectivity, then start searching for a new AP, and finally re-associate to a new AP. With IEEE 802.11r, devices can register in advance with neighbouring APs. Thus, security and QoS-related settings can be negotiated before a device has to switch to a new AP. This, of course, results in the substantial reduction of the duration of a connectivity loss.

2.9 IEEE 802.11y

In 2005, IEEE 802.11y made its first appearance, but it would not be until 2008 that it would finally be published. It operates in the 3.65-3.7 GHz range and has a maximum power of 20 W. It is used for wide area coverage, on account of it enabling long-range communication. The way it works is rather uncommon. Exclusive medium usage is not guaranteed. Instead, a central database is provided, where users must give the position of their base stations. Due to possible restrictions that are location-dependent, an operator's signalling enables or disables the customers' equipment.

2.10 IEEE 802.11n

In October 2009 IEEE 802.11n (Wi-Fi 4) was published, seven years after development first began, providing a user experience comparable to the well-known Fast Ethernet, also known as IEEE 802.3u. IEEE 802.11n can provide data rates of up to 600 Mb/s (in 40 MHz mode, with four antennas and 400 ns guard interval), depending primarily on the number of wireless radios incorporated in the devices, far beyond the minimum requirements that were derived from its wired paragon's maximum data rate of 100 Mb/s. This is possible thanks to its usage of multiple antennas.

WiFi-4 was a huge step in wireless LAN technology, with many devices still using it to this day. It introduced a number of technologies that helped resolve the rising demands from all these smartphones that now incorporated Wi-Fi, and ended up being a stepping stone for further IEEE 802.11 amendments.

IEEE 802.11n allows up to four transmit antennas, four receive antennas and four data spatial streams (4x4:4). Common configurations of IEEE 802.11n devices include, but are not limited to, 2x2:2, 2x3:2, and 3x2:2. For example, a device that can transmit using two antennas, receive using three and can send/receive two data streams would be described as 2x3:2. All the aforementioned configurations have

the same maximum throughputs and features, with the only differentiating factor being the amount of diversity provided by the antenna systems. One thing to keep in mind though: even if the router features 4 antennas, if it connects with a device that has less than four, like for example a smartphone, which usually only has one antenna, then this is a bottleneck that will affect the maximum performance possible.

Judging from the above, one of its key features is the introduction of multiple-input multiple-output (MIMO) capability, which was seen here for the first time. This concept allows for arrays of up to four antennas that enable spatial multiplexing or beam forming. This technology enables the coherent resolution of more information than previously possible, where only one antenna was used. One way it achieves this is through Spatial Division Multiplexing (SDM), which spatially multiplexes multiple data streams, transferred simultaneously within the same spectral channel of bandwidth. This has the potential to significantly increase the data throughput, since there is an increase in the number of resolved spatial data streams. The “catch” is that each individual special data stream requires their own antenna at the transmitter and the receiver, which makes IEEE 802.11n more expensive to implement, in addition to the fact that each antenna requires their own analog-to-digital converter. This means that MIMO is also more demanding in regards to the power it consumes, so the engineers came up with a solution: the data is transmitted in short bursts, and that means that there are times where the antennas are not in use. During these times, the circuitry is held inactive, which results in no power being consumed.

Moreover, IEEE 802.11n provides backward compatibility for devices utilizing earlier versions of IEEE 802.11, adding a large amount of overhead to the exchanges, thus reducing its data transfer capacity. If the goal is to achieve the maximum data transfer speeds, the backwards compatibility feature can be removed. This is known as Greenfield mode. This of course means that any device not using IEEE 802.11n will stop connecting to the network.

Another innovation of IEEE 802.11n is the addition of optional 40 MHz channels to the Physical Layer. These channels have double the width of what was implemented so far, where the channels were 20 MHz. This means that it provides twice the PHY data rate over a single 20 MHz channel. This feature was already being used as a proprietary extension of IEEE 802.11a and IEEE 802.11g chipsets, but concerns were raised that it could affect the performance of existing neighbouring devices using IEEE 802.11, IEEE 802.15.1 (Bluetooth), IEEE 802.15.4 (ZigBee), as well as other devices. A compromise had to be developed, leading to the delay of IEEE 802.11n's publication. In the end, the compromise was to disallow 40 MHz channelization for devices that could not detect 20 MHz-only devices. It can be enabled in the 5 GHz mode, or within the 2.4 GHz mode if the user is relatively certain that it will not interfere with any other systems using the same frequencies [23]. One thing to keep in mind is that at 2.4 GHz there is sufficient room for three 20 MHz channels, but only one for a 40 MHz channel. Thus, the choice of whether to use 20 MHz or 40 MHz has to be made dynamically by the devices in the network.

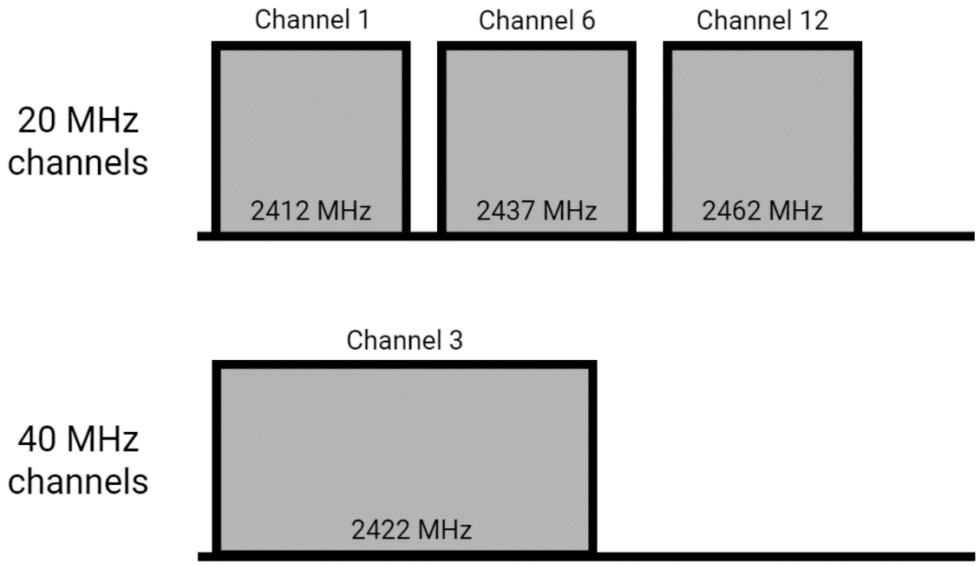


Figure 2.6: Non-overlapping IEEE 802.11n channels

For IEEE 802.11n, even the antenna technology was developed further. Two new additions were made, beamforming and diversity. Typically, wireless signals up to this point were simply thrown out from the router in all directions equally, like ripples created from throwing a stone into a pond. Beamforming can focus the radio signals directly on the receiving antenna to improve the range as well as the overall performance. The higher signal level and the better signal to noise ratio means that the channel can be used more fully. Diversity uses the multiple available antennas and combines the best subset of them to obtain the best possible signal conditions. As IEEE 802.11n supports between one and four antennas, it is possible that one device may have three antennas, while the other one with which it is communicating may have two. As a result, the “extra” antenna can be used to provide diversity reception or transmission.

A further key feature is the debut of frame aggregation in the MAC layer, which means that two or more data frames can be sent in a single transmission, increasing throughput. It also features reduced interframe spacing that eliminates, or at least helps reduce, the transmission idle periods between consecutive frames. The maximum transmission size was also increased to 7955 B, which means that several frames can be included in a single packet.

Thanks to its 20/40 MHz operation and various antenna configurations, IEEE 802.11n defines a total of 76 different MCSs. However, the WFA's certification program decides which ones will in the end be used in the market, since several of them provide similar data rates. Since fast MCSs lead to short frame transmission durations, devices that support IEEE 802.11n may use the Reverse Direction protocol, which grants part of their TXOP to be used for frame reception. This means that a previously receiving device has the ability to send in the reverse direction without the need for a backoff interval. This proves useful in supporting higher-layer protocols like TCP, sending back ACKs to the sender or in VoIP conversations that have bidirectional traffic characteristics.

Table 2.1: IEEE 802.11n features

Maximum data rate	600 Mbps
Frequency band	2.4 GHz, 5 GHz
Channel width	20 MHz, 40MHz
Spatial streams	1, 2, 3 or 4
Modulation	CCK, DSSS, OFDM

When IEEE 802.11g was released, since it shared the band with already-existing IEEE 802.11b devices, it provided ways to ensure coexistence between already-existing and future devices. IEEE 802.11n extends the coexistence management to protect its transmissions from legacy devices, which include IEEE 802.11g, IEEE 802.11b and IEEE 802.11a. MAC and PHY level protection mechanisms were put in place as listed below:

- MAC level: An RTS/CTS frame exchange or CTS frame transmission at legacy rates can be used to protect subsequent IEEE 802.11n transmissions.
- PHY level: Using a 40 MHz channel for transmission when IEEE 802.11a or IEEE 802.11g clients are present requires the use of CTS protection on both 20 MHz halves of the 40 MHz channel, to prevent interference with legacy devices.
- PHY level: Mixed Mode Format protection (also known as L-SIG TXOP Protection): In mixed mode, each 802.11n transmission is always embedded in an 802.11a or 802.11g transmission. For 20 MHz transmissions, this embedding takes care of the protection with 802.11a and 802.11g. However, 802.11b devices still need CTS protection.
- PHY level: Mixed Mode Format protection (L-SIG TXOP Protection): In mixed mode, each IEEE 802.11n transmission is embedded in an IEEE 802.11a or IEEE 802.11g transmission. For 20 MHz transmissions, this embedding is enough. However, IEEE 802.11b devices still require CTS protection.

Ever since the first draft of the IEEE 802.11n standard was published all the way back in 2006, manufacturers around the world had been producing “Draft-N” products claiming to comply with the standard draft, before the standard was even finalized. This potentially meant that these devices might not be inter-compatible with products produced according to the published IEEE 802.11n standard, or even among themselves [24]. As a result, at the end of June 2007, WFA started a certification program based on Draft 2 of IEEE 802.11. Several vendors announced that their devices were compatible with this IEEE 802.11n standard, receiving the respective WFA certification. The certification covered both 20 MHz and 40 MHz channels and up to two spatial streams, for maximum throughputs of 144.4 Mbit/s for 20 MHz and 300 Mbit/s for 40 MHz.

The final draft (Draft 11.0) was approved on July 2009, with 53 votes approved, 1 against and 6 abstained [25]. This enabled IEEE 802.11n to be published on October 2009, as previously mentioned. One noteworthy thing to mention is that even before Draft-N, vendors were offering "Pre-N" products, showing the market demand for new IEEE 802.11 networking hardware.

2.11 IEEE 802.11p

Accepted in September 2003 (but published in 2010), IEEE 802.11p enables communication between devices moving at vehicular speeds of up to 200 km/h. It is part of the Wireless Access in Vehicular

Environments (WAVE) framework. In the United States it operates in the 5.85-5.925 GHz ITS band and in Europe in the newly allocated 5.855-5.905 GHz band. Its PHY is identical to OFDM-based IEEE 802.11a, but with one key difference: in addition to the traditional 20 MHz, IEEE 802.11p can optionally operate with reduced 10 MHz channel spacing (at the same time halving the maximum PHY data rate to 27 Mb/s) to compensate for the increased delay spread in outdoor vehicular environments. Besides the PHY, IEEE 802.11p defines the lower part of the MAC layer, while the IEEE 1609 family of standards address the upper part of the MAC (1609.4), networking (1609.3), security (1609.2), resource management (1609.1), communication management (1609.5), and overall architecture (1609.0). The maximum radio output power may be increased up to 760 mW, to enable longer communication distances.

Since IEEE 802.11p is used outdoors, often in harsh conditions, radios need to be able to operate in the -40oC to 85oC temperature range. The maximum communication range is 1 km, and that means that the time for data exchange between two moving devices before connectivity is lost is limited to just a few seconds. Since the connectivity time is so short, IEEE 802.11p forgoes the association and the authentication before the data exchange takes place. Instead, devices join WAVE networks utilizing an internal device procedure, without any kind of frame exchange on the wireless medium. This lack of association/authentication is a security risk for the data communication, so it is handled by the dedicated security entity specified by 1609.2.

According to the aforementioned specification, the WAVE system works utilizing multi-frequency channels, one of which is the control channel (for safety applications) and the others being service channels (for non-safety applications). This means that global synchronization for vehicular ad hoc networks is of the utmost importance for coordinating multichannel accesses. So, IEEE 802.11p enhances the timing synchronization function to support global timing synchronization based on an external source, like GPS. In addition, to alleviate privacy concerns (for example, to avoid tracking a device along its journey), a device's randomly selected MAC address changes repeatedly.

2.12 IEEE 802.11z

2010 saw the publication of another IEEE 802.11 amendment, IEEE 802.11z. This protocol is also known as Tunnelled Direct Link Setup (TDLS). To initiate a direct link connection, IEEE 802.11z-capable devices use a special Ethertype frame to tunnel setup messages through a legacy AP. Furthermore, the IEEE 802.11z TG allows for the possibility of frequency offloading, where devices use a power save indication to the AP to switch to an empty frequency channel for DLS frame exchanges. Google’s cast protocol (found in Chromecast) uses this technology to initiate screen mirroring [26].

2.13 IEEE 802.11u

Due to the conjunction of WLAN and cellular in mobile phones, IEEE 802.11 needed a framework for interworking with external networks. This is where IEEE 802.11u comes into play. Published in 2011, It offers emergency call handling (911 over VoIP), QoS adaptation, and support for handover and virtual networks. With GSM, operators have shared long base stations. Without IEEE 802.11u, an AP needs one MAC address and broadcast frame per operator that is to be announced. The IEEE 802.11u amendment allows the announcement of multiple network operators by an AP in one single broadcast frame. Since user databases exist in other mobile radio networks, IEEE 802.11u integrates with their authentication and authorizing framework. Based on that, instead of proprietary solutions, network operators could finally offer a standardized solution for seamless global roaming.

2.14 IEEE 802.11aa

In March 2012, after numerous changes to its scope, IEEE 802.11aa was published. It develops the general principles for time-synchronized, low-latency streaming services in IEEE 802.11 networks, while maintaining compatibility with other types of traffic. This set of mechanisms is called Group Addressed Transmission Service (GATS). According to it, the stations have to form a group, so that they can agree on which address to listen to and the specific mechanism to use. One advantage of GATS is that it doesn't alter the IEEE 802.11 MAC to a great degree, which means that implementing it to already-existing hardware is not that hard. This of course was done to ensure backwards compatibility with all the other amendments.

It supports frame differentiation, meaning that, for example, I-frames of an MPEG2 steam may receive higher precedence that their depending B-frames. This means that IEEE 802.11aa can gracefully degrade the stream quality in the event that it detects insufficient channel capacity [27].

The need for this amendment resulted from the fact that, up to this point, IEEE 802.11 was inefficient and unreliable at video streaming over WLANs. There was no acknowledgment for multiple receivers (multicast), which is crucial for improving reliability when packet loss occurs [28].

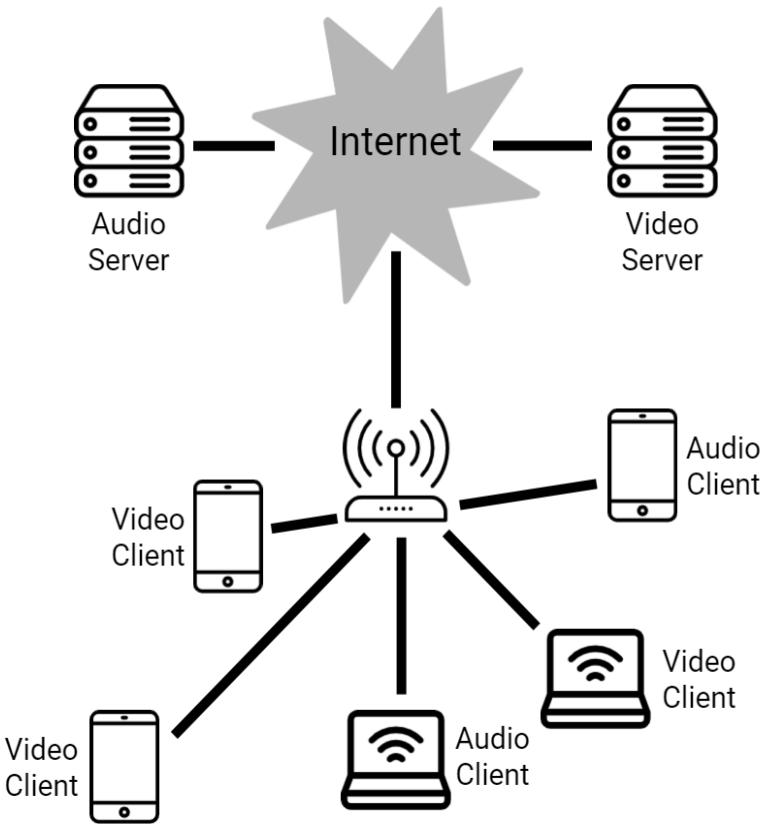


Figure 2.7: An IEEE 802.11aa topology

2.15 IEEE 802.11ad

Version 1.0 of IEEE 802.11ad was released in December 2009, boasting a maximum data rate of up to 8 Gbps. It is designed to be a wireless alternative to fiber optics, being able to achieve speeds 50 times faster than IEEE 802.11n. As an amendment to the overall IEEE 802.11, IEEE 802.11ad was published in 2012 with the title "Enhancements for Very High Throughput in the 60 GHz band". It is different from other IEEE 802.11 amendments, since, as mentioned in its title, it operates on the unlicensed 60 GHz band. This enables it to support very high data rates, but at the cost of distance, since the communication range is about 1-10 meters. As a result, IEEE 802.11ad is ideal for short range, line-of-sight (LOS) connections, although non-LOS is possible with the usage of multiple antennas.

IEEE 802.11ad is also called Directional Multi-Gigabit (DMG). However, commercially, the most prevalent term is Wireless Gigabit (WiGig). The most common use case scenarios involve homes and office spaces, where the devices are in near proximity to one another. For example, using IEEE 802.11ad, the user could get rid of the wires connecting to their TV, Blu-ray player, virtual headset, home theatre speakers and so on. It can also be used for docking purposes, for instant file transfers and HD media streaming (even raw).

Nowadays, the 60 GHz frequency range has been expanded to 57-70 GHz. Six different channels are available (previously only four), each one having a nominal channel bandwidth of 2.16 GHz, which is a lot of spectrum, thus allowing IEEE 802.11ad to offer multi-gigabit speeds. It also provides an optional low-power Single Carrier mode for mobile phones. Channel 2 (59.40-61.56 GHz) is available in all regions, and as such is considered the default channel. Compared to IEEE 802.11ac, it has five times lower power consumption per bit, making it very power efficient.

2.16 IEEE 802.11ac

IEEE 802.11ac was published in December 2013, having the goal of matching the performance of Gigabit Ethernet. Indeed, it has a maximum throughput of at least 1 Gb/s using multiple stations, or 500 Mbit/s using a single link. This was made possible by the extension of the concepts embraced by IEEE 802.11n:

- Wider RF bandwidth (up to 160 MHz, compared to 40 MHz in IEEE 802.11n)
- More MIMO spatial streams (up to eight, compared to four in IEEE 802.11n)
- Downlink multi-user MIMO (MU-MIMO) (up to four simultaneous clients), with support for beamforming
- High-density modulation

The Wi-Fi Alliance separated the launch of IEEE 802.11ac products into two phases, Wave 1 and Wave 2. Products up to 2016 were marked as Wave 1, because the standard was not finalized right up to this year. Products launched after 2016 were Wave 2 products, having new features like MU-MIMO, 160 MHz channel width support, support for more 5 GHz channels, and four spatial streams (compared to three in Wave 1). These all mean that Wave 2 products have a higher bandwidth and capacity compared to those from Wave 1. IEEE 802.11ac operates on the 5 GHz signal range, unlike most previous generations of Wi-Fi that also utilized 2.4 GHz channels. This choice was made for two reasons:

- In order to avoid issues associated with wireless interference, common to 2.4 GHz as there is a huge number of devices that operate in this range
- To implement wider signalling channels that the 2.4 GHz space comfortably allows

Of course, to avoid breaking backwards compatibility with older wireless devices, IEEE 802.11ac also utilized a IEEE 802.11n-style 2.4 GHz protocol support.

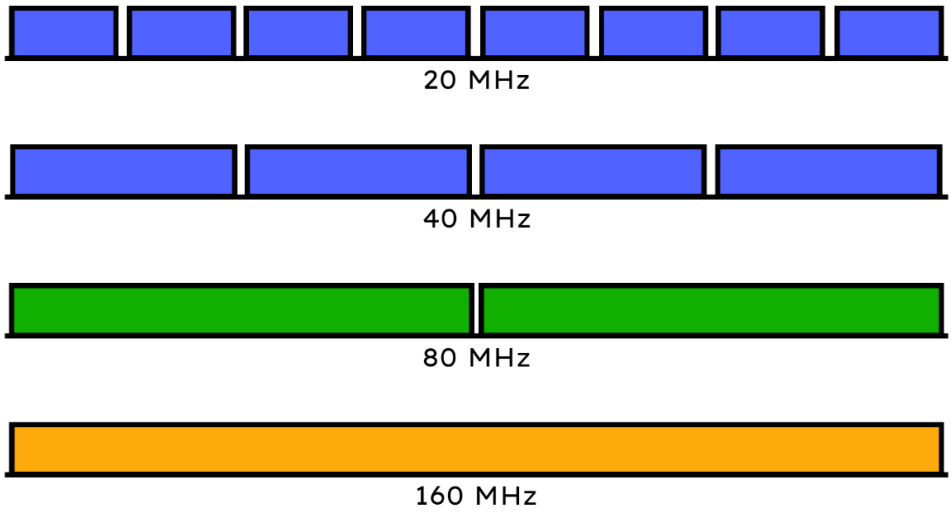


Figure 2.8: IEEE 802.11ac 5 GHz channelization

Table 2.2: IEEE 802.11ac Physical Layer [29]

FEATURE	MANDATORY	OPTIONAL
Channel bandwidth	20MHz, 40MHz, 80MHz	160MHz, 80MHz+80MHz
FFT size	64, 128, 256	512
Data subcarriers / Pilots	52 / 4, 108 / 6, 234 / 8	468 / 16
Modulation types	BPSK, QPSK, 16-QAM, 64-QAM	256-QAM
Spatial streams & MIMO	1	2-8

2.17 IEEE 802.11ax

IEEE 802.11ax (also called Wi-Fi 6) is the successor to IEEE 802.11ac, and aims to increase the efficiency of WLAN networks. The goal is to provide 4 times the throughput of IEEE 802.11ac, while having just 37% higher nominal data rates at the PHY layer. The IEEE 802.11ax standard is still not an official IEEE specification. While IEEE 802.11ac introduced Multi-User MIMO, increasing the overall throughput, IEEE 802.11ax introduces a similar multiplexing technique, named OFDMA [30]. OFDMA enables multiple clients to be assigned with different Resource Units in the available spectrum. Doing so, an 80 MHz channel can be split into multiple Resource Units, so that multiple clients can receive different type of data over the same spectrum, at the same time. In order to have enough subcarriers to support these requirements, four times as many subcarriers are needed than IEEE 802.11ac. In other words, since the available bandwidths have not changed and the subcarriers are four times as many, the OFDM symbols are four times longer. For IEEE 802.11ac the duration of an OFDM symbol is 3.2 microseconds, while for IEEE 802.11ax it is 12.8 microseconds (without guard intervals in both cases).

Table 2.3: Changes between IEEE 802.11ac and IEEE 802.11ax in the Physical Layer [31]

	802.11ac	802.11ax
BANDS	5 GHz	2.4 GHz and 5 GHz
CHANNEL BANDWIDTH	20MHz, 40MHz, 80MHz, 80MHz+80MHz, 160MHz	20MHz, 40MHz, 80MHz, 80MHz+80MHz, 160MHz
FFT SIZES	64, 128, 256, 512	256, 512, 1024, 2048
SUBCARRIER SPACING	312.5 kHz	78.125 kHz
HIGHEST MODULATION	256-QAM	1024-QAM
DATA RATES (80 MHz, 1 SS / 160 MHz, 8 SS)	433 Mbps / 6933 Mbps	600.4 Mbps / 9607.8 Mbps

2.18 Summary of IEEE 802.11 protocols

The IEEE 802.11-2007 standard and its amendments provide a rich feature set for wireless communication. 5, 10, 20, and 40 MHz channel bandwidth in the 2.4, 3.65, and 4.9–5 GHz frequency bands support a wide range of regulatory domains. Furthermore, the IEEE 802.11 MAC has proven to be flexible enough to expand from its ancestral market segments. While IEEE 802.11n and mesh networks extend well-known applications, wide range (IEEE 802.11y) and vehicular communication (IEEE 802.11p) have opened new scenarios for WLAN. But the increasing number of amendments also makes it more and more difficult to maintain a cohesive standard. Work on the latest amendments tends to take longer than those in the past. However, no alternative to the popular, cheap, and flexible IEEE 802.11 technology is visible just yet. Quite the contrary, driven by market demands, the IEEE 802.11 universe keeps on expanding.

A key contributor to IEEE 802.11's success is its simple MAC operation based on the DCF protocol. This scheme has proven to be not only robust and adaptive to varying conditions, but also able to cover most needs to, at the very least, a satisfactory degree. One key element of IEEE 802.11 is the over-provisioning of its capacity, which meant that complex resource scheduling and management algorithms were not necessary to be implemented. However, because IEEE 802.11 has become so popular, WLANs are expected to reach their capacity limits. In addition, the rise of applications for voice and video streaming poses different demands for the Quality of Service (QoS). This means that network management might become a necessity.

2.19 Alternatives to IEEE 802.11

2.19.1 Bluetooth

Bluetooth is a networking technology aimed at low-powered, short-range applications. It is used to connect and exchange information, as well as data, between devices. It was initially developed by Ericsson, but is now an open specification. Especially in its first years, its most commonly described application was that of a “cordless computer”, where peripheral devices like the keyboard, mouse, printer, and scanner are wirelessly connected to the computer.

Today, of course, Bluetooth has infiltrated nearly every aspect of our daily lives since it is incorporated into every smartphone and laptop, and has many more applications than simply being a wire replacement technology. It is used to connect wireless headphones, headsets and speakers to the computer/smartphone, to connect gamepads to the gaming console, to transfer files between devices, etc. It is also the technology (along with Zigbee) of choice for ad hoc networks. This is mostly due to the fact that Bluetooth has a low power consumption and low cost, making an ideal solution for the typical mobile devices that are used in ad hoc networks. Bluetooth can also be used to connect to a network or the Internet from a laptop by connecting wirelessly to a smartphone.

Thanks to its low operating range, which does not exceed the 10-meter mark, it is classified as a Wireless Personal Area specification.

2.19.1.1 Bluetooth operation

Nodes are organized in small groups called piconets. Each piconet has a leading node, known as the "master", while the other nodes are referred to as "slaves". A node can also belong to multiple piconets. In that case, it is referred to as a "bridge". A piconet can have at most 8 members, with the "master" included.

Each and every communication in a piconet relies on the "master", since the "slaves" do not directly communicate with each other, instead relying on the "master" as a transit node. This means that the communication Bluetooth provides is half-duplex. If two nodes that belong to different piconets wish to communicate, then the bridge nodes have to be involved. However, by design, a bridge node cannot be active in more than one piconet. So, while it is active on one piconet, it is "parked" in others. In general, a node can be in an active, idle, parked, or sniffing state.

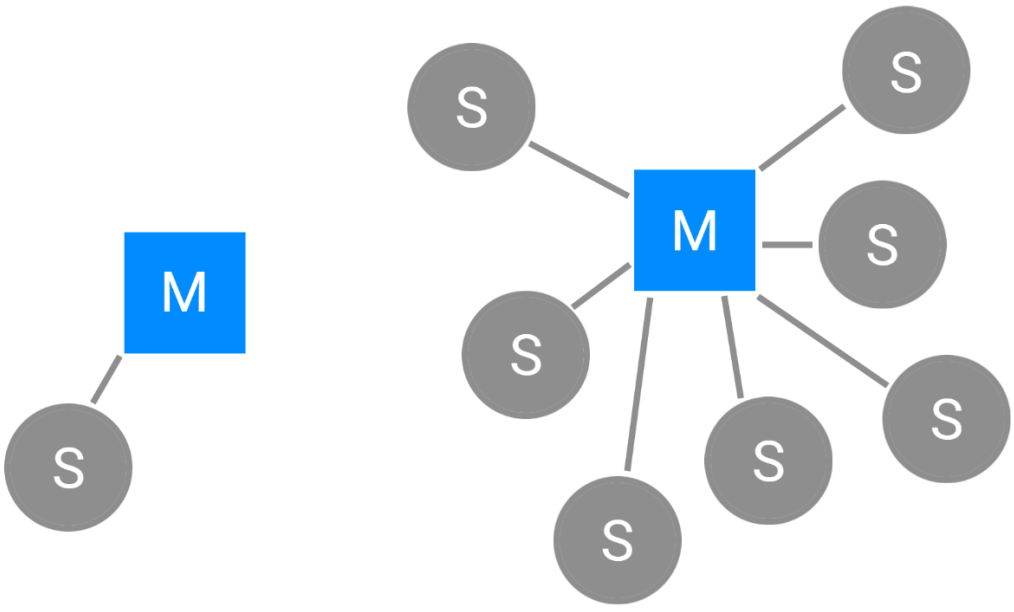


Figure 2.9: An example of a Bluetooth topology

Connecting two Bluetooth devices involves the following steps:

- Inquiry – The sender broadcasts inquiry packets, which do not contain the identity of the sender (or any information for that matter)
- Inquiry scan – While in this state, receiver devices listen for inquiry packets, and when they detect one, they broadcast an inquiry response packet containing the identity of the device and its native clock
- Page – During this state, the sender attempts to establish a connection with a device whose identity and clock are known by sending page packets which contain the sender's address and clock
- Page scan – While in this state, receiver devices listen for page packets. When they do receive them, they send an acknowledgement and synchronization between sender and receiver is established.

2.19.2 IEEE 802.15.7

IEEE 802.15.7 is another standard proposed by IEEE, having received board approval in December 2018. It defines a physical layer (PHY) and medium access control (MAC) sublayer for short-range optical wireless communications (OWC) in optically transparent media using wavelengths of light from 10.000 nm to 190 nm (the visible light is between 700 nm and 400 nm). IEEE 802.15.7 has the capability to deliver data rates that are sufficient for audio and video multimedia services (up to 500 Mbit/s), while at the same time taking into account the mobility of the optical link, compatibility with various light infrastructures, interference and noise from sources like ambient light [32]. Additionally, it implements a MAC sublayer that accommodates the unique needs of visible links, in addition to the other targeted wavelengths.

Talks about this kind of data transmission method had already been taking place for years, but the event that really kicked this kind of feat into action is the appearance of the LED technology, which provides not only the necessary brightness, but also the energy efficiency required for VLC to be a sensible, viable and practical choice.

The proposed use case scenario is the transmission of data to devices through optical communication, where transmitting devices feature sources that emit light (these devices can be as simple as a single light bulb) and receiving devices are digital cameras that incorporate a lens and image sensor (in most cases a simple smartphone camera is sufficient) [33].

VLC can also be used to position and track items in a given indoor space. Real-world tests have proven this technology to be very accurate, reporting results that claim average position errors of less than 3 cm.

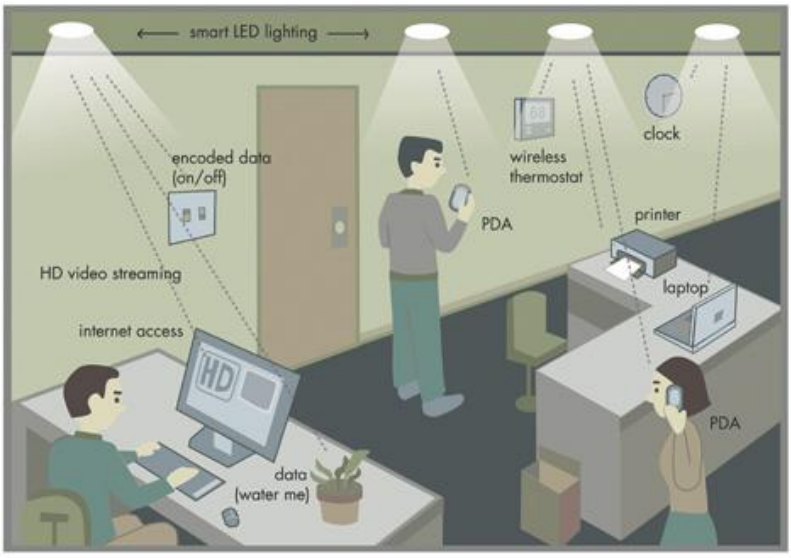


Figure 2.10: An example of an office utilizing VLC https://www.theengineer.co.uk/light-reading-visible-light-communications/

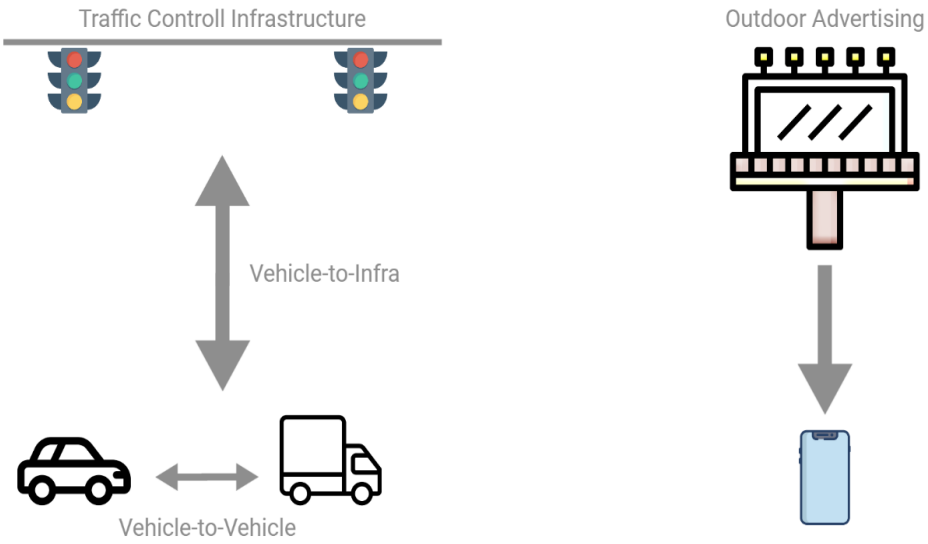


Figure 2.11: VLC can also be used outdoors

2.19.2.1 Advantages of VLC

- It is visible, which makes it aesthetically pleasing
- It is secure, since the user can see what is being sent and when
- It is harmless to the human body, since it adheres to applicable eye safety regulations
- There are no regulations in optical frequencies, like in radio communication systems, making the use of VLC an easy job
- Since it produces no interference, it can be used in restricted and sensitive spaces like hospitals, aircrafts and spaceships

- Utilizing LED technology, its power consumption is much smaller compared, for example, to RF
- Which also means that it is more eco-friendly

2.19.2.2 Disadvantages of VLC

- Since it transmits data through the use of visible light wavelengths, it always requires a line-of-sight between the transmitting and the receiving device
- This also means that the coverage it can provide is narrow and focused
- Which in turn also means that it can provide limited support for moving devices

2.19.3 RFID

Radio-frequency identification (RFID) utilizes electromagnetic fields in order to identify and track tags in a given space that are attached to objects. These tags contain electronically stored information and are divided into two categories. The active ones feature an active power source (a battery for example), which enables them to operate long distances from an RFID reader. On the other hand, the passive ones operate by collecting energy from a nearby RFID reader's radio waves.

The cost of an RFID tag varies greatly, since they typically start from just 0.50€ and can go as high as 100€, or at times even more. Their cost is determined by their type (active or passive), their resistance to the elements (for outdoor use), their range of operation and so on. Some also incorporate temperature and humidity sensors, driving their cost even further.

The RFID technology is not a direct competitor to IEEE's 802.11. That was never the point to begin with. However, it can be used as an alternative to it in a number of applications. Examples include the tracking of goods, people and/or animals, billing processes, and logistics in a warehouse environment. Active RFID tags can also be used as remote sensors that broadcast telemetry back to a base station. Typical use case scenarios for with would be extracting weather reports or monitoring the noise levels in an urban area.

Chapter 3: Simulation

3.1 Introduction

A simulation is, according to Cambridge Dictionary, a model of a set of problems or events that can be used to teach someone how to do something, or the process of making such a model.

Wireless technology advances rapidly as the years go by, and new enhancements are proposed all the time. These enhancements cannot be implemented on a large scale before they are tested, due to the uncertainty of its successful outcome. So, analytical modelling and/or simulation tools are utilized. After the results are extracted and processed, it can then be decided if these enhancements will be implemented in the real world, or not. This process saves researchers not only time, but also money, since it would require not just expensive hardware, but also human resources to test these hypotheses in the real world.

However, analytical processing also has some drawbacks. For example, the deduced results are not decisive as far as the consumed energy, memory and processing power are concerned [34].

Simulations can be divided into three categories:

- Discrete Event Simulation (DES), where the state of the system changes whenever an event takes place.
- Continuous Simulation, where the system undergoes changes continuously as the time goes on and differential equations are being used.
- Three-phased simulation, where the simulation is split into three phases. In Phase 1, the events take place according to their chronological order. Events in Phase 2 are called B-events and events in Phase 3 are called C-events.

Key elements of any simulation include, but are not limited to: a clock (which is a variable that gives the current value of simulated time), an event list and the statistics (the end results).

In order to study a computer network and their behaviour, computer network simulators are utilized, which are pieces of software that enable the construction of network models using stations, servers, links, routers, protocols, and much more. After the simulation ends, the extracted results can help in making future decisions regarding the computer network in question.

3.2 OPNET – Riverbed Modeler

OPNET (Optimized Network Engineering Tool) is a network simulator. It was proposed in 1986 and initially developed by Massachusetts Institute of Technology (MIT) in 1987 using C++, before OPNET Technologies, Inc. took the reins of the project. That was until October 2012, which was the year their acquisition by Riverbed Technologies took place. Since then, it has changed its name from OPNET Modeler to Riverbed Modeler.

It uses a graphical user interface (GUI), making the design and creation of a network using the various network elements it provides an easy task. Some of these network elements are: servers, routers, switches, hubs, stations (for example PCs or smartphones), and links (for example Ethernet).

In its regular edition it provides four distinct editors: a Project Editor, a Node Editor, a Process Editor and a Parameter Editor. It also offers the ability to modify or add new elements written in C++. Of course, it also provides the user with the ability to filter which statistics will be shown after the simulation ends. All these features have contributed to making Riverbed Modeler the most widely used network simulator, commercial or not.

There is also an academic version of Riverbed Modeler, called Riverbed Modeler Academic Edition, with some of its functionality limited. It is targeted to academics and students who wish to learn to use the software, as well as to potential buyers who would like to test its features before committing to purchase a license. Riverbed Modeler, in its Academic Edition, provides a free six-month license upon registering through the official website, with the ability to renew it when it expires.

However, it is worth pointing out that according to their official website, after September 01, 2023, Riverbed Modeler Academic Edition will no longer function. As a result, September 01, 2020 is the last day Riverbed is going to accept any new customers to sign up for Riverbed Modeler Academic Edition, since the company has plans for the current release (17.5) to be the last one.

3.2.1 System requirements

Riverbed Modeler supports Windows XP, Vista, 7 and 10 (both 32 and 64 bit, but only the Professional and Enterprise editions), Windows Server 2003, 2003 R2 and 2008 (both 32 and 64 bit), Red Hat Enterprise Linux 4, 5 and 6, and Fedora Linux 6.

It requires a 2.0 GHz processor for Windows and a 1.0 GHz one for Linux, although it recommends a 3.0 GHz one in both cases. It also requires 512 MB of RAM, although the recommended amount is 1-2 GB.

As for the storage space, it requires at least 3 GB during normal use and at least 5 GB during the installation phase. Finally, the screen resolution has to be at least 1024x768.

3.2.2 Installing Riverbed Modeler Academic Edition

The first step for the user is to navigate to the official website of Riverbed Modeler Academic Edition (https://cms-api.riverbed.com/portal/community_home) and register. An email will then be sent to the email address the user provided during registration, containing a username, a password and a link to download the software. The installer is about 600 MB.

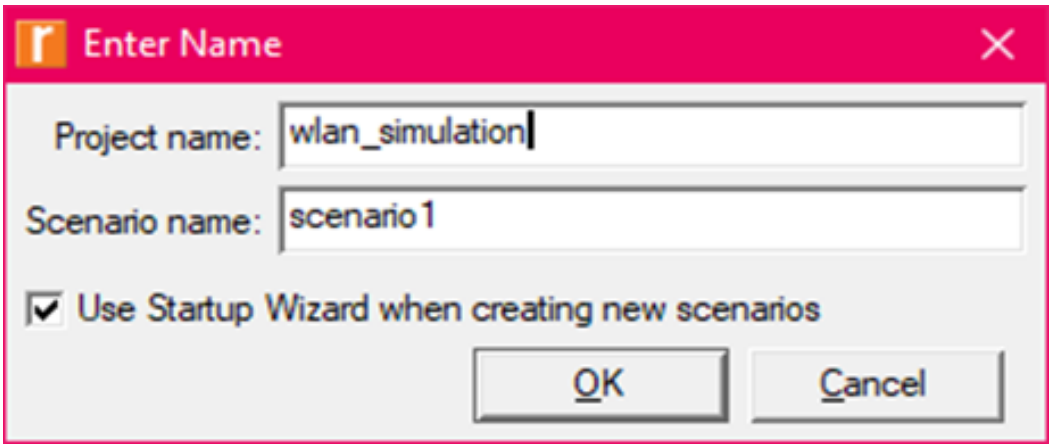
After the download is complete, running the installer and following the instructions it provides results in the program being installed.

3.2.3 Projects and Scenarios

Riverbed Modeler allows the conduction of studies for a plethora of network technologies in a wide range of simulations. The individual simulation studies are referred to within the software as projects. Every project can be expanded upon by creating one or more scenarios. A project can be defined as a collection of simulation studies of a particular system under different configuration parameters. A scenario is a simulation study of a system under a particular parameter configuration.

3.2.4 Creating a new project

In order to create a new project in Riverbed Modeler Academic Edition (version 17.5), the user has to go to **File → New... → Project → OK**. Alternatively, **Ctrl + N** works just as well. The following window will pop up:



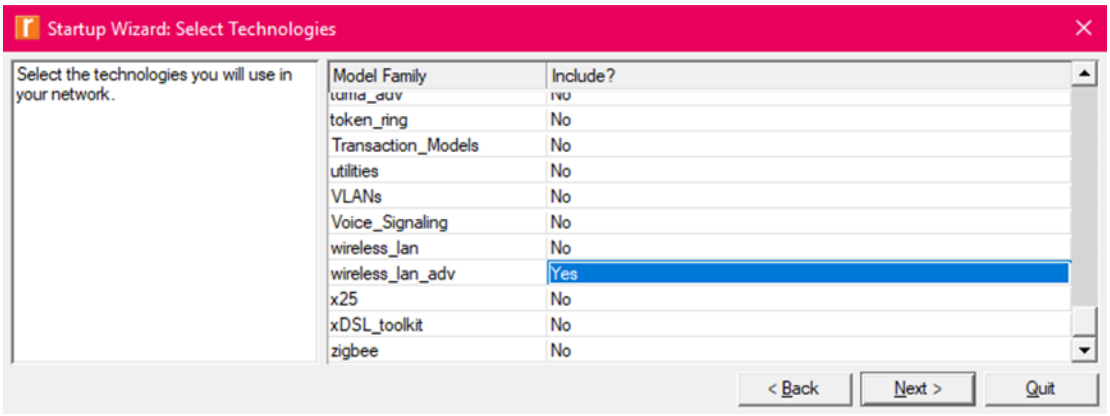
Screenshot 3.1: Setting the name of the project

After the user enters the desired project and scenario name, they click OK and are then transported to the next window, called Initial Topology, where they select **Create empty scenario → Next**.

In the Choose Network Scale window, the user can pick between World, Enterprise, Campus, Office, and Logical, depending on the scale. They can also pick Choose from maps.

The next windows depend on the choice made in the previous window. If World, Enterprise, or Choose from maps was selected, the user is asked to define the geographical scale based on a chosen map through the Choose Map window. Alternatively, if Enterprise, Campus, or Office was selected, the user has to define the width and length of the simulation area through the Specify Size window.

After the dimensions are done being configured, Riverbed Modeler opens the following window:

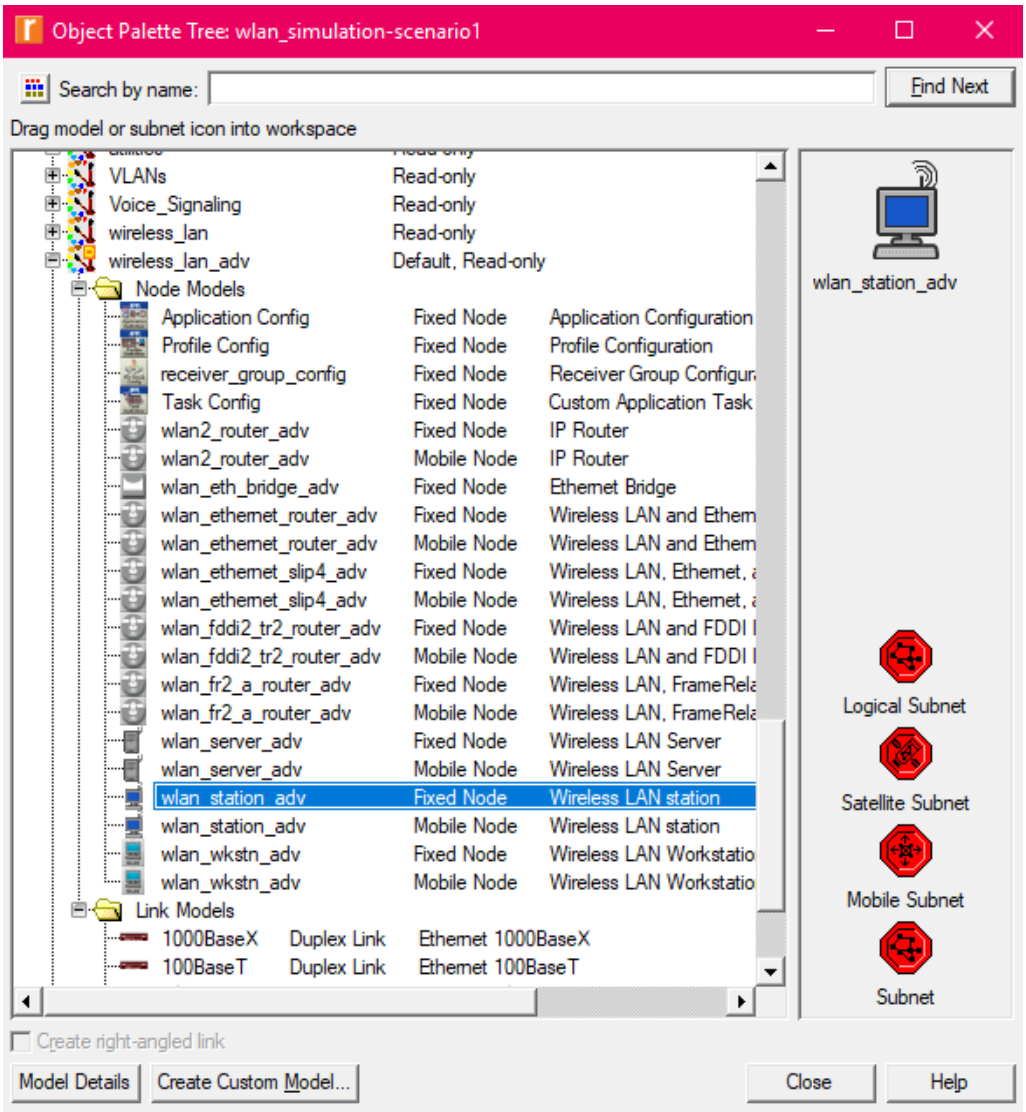


Screenshot 3.2: Setting the Model Family to be used

In the Select Technologies window, the user specifies the Model Family that will be used in the simulation. A Model Family is a collection of models that belong in a particular set of technologies, like internet_toolbox, Ethernet, wireless_lan, Cisco, zigbee, and so on. After the **Next** button is pressed, the user is transported to the Review window, where pressing the **Finish** button creates the project.

3.2.5 Adding network elements

All the network elements are located in the Object Palette, which can be found in the menu **Topology → Open Object Palette**. The user can also opt to import their own models, if they so desire.



Screenshot 3.3: The Object Palette, a core element of Riverbed Modeler

Using simple drag-and-drops, the user can add any element they see fit from the Object Palette to their simulation area.

3.2.6 Creating network topologies

In general, network topology is the arrangement of the elements (like for example the links and nodes) of a communication network. There are seven basic topologies:

- Ring topology
- Mesh topology
- Star topology
- Fully connected topology
- Line topology
- Tree topology
- Bus topology

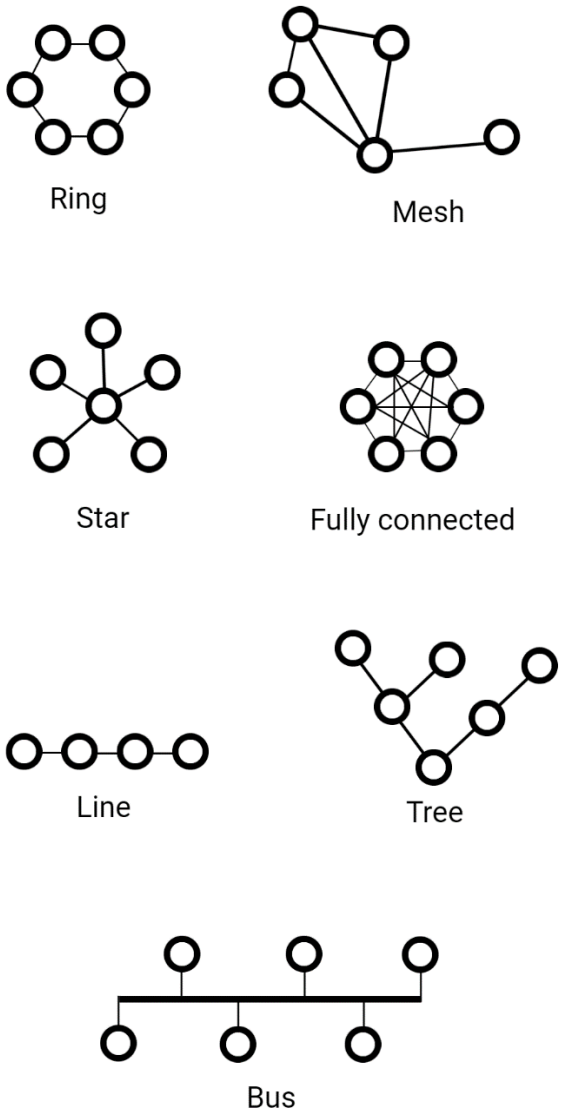


Figure 3.1: Network topologies

In order to create a network topology using Riverbed Modeler, the user first needs to open the Object Pallete Tree through the menu **Topology → Open Object Pallete**. One useful feature that Riverbed Modeler offers is the search bar, which can save time if the user knows exactly what they want to find in the Object Pallete. Of course, they can also scroll through the list manually. After they have located the object they want to insert to their simulation, they drag-and-drop it to their workspace. After all the objects have been inserted, they need to be connected to one another, so it is time to insert the links between them. The links are located in the Link models section of the Object Pallete Tree. Like the objects, they can be inserted through drag-and-drop actions as well. Finally, all that remains is to use these links to connect the objects according to the topology in mind.

However, Riverbed Modeler offers another method to do this, especially if the topology is going to feature a large number of nodes. This method is using the Rapid Configuration Tool, which is located in the menu **Topology → Rapid Configuration...**. A drop-down selector appears, where the user can pick between Bus, Mesh Full, Randomized Mesh, Ring, Star, Tree or Unconnected Net as their topology.

There is also the button **Seed...**, which opens up a new window where the user can set the seed parameter (or click Generate to automatically set a value) that is important for the topologies that place their nodes in a randomized manner. After the parameter has been set, the user clicks **OK** to close the window and then **Next** to be transported to the next one. Here, the parameters the user has to set depend of the topology selected in earlier steps.

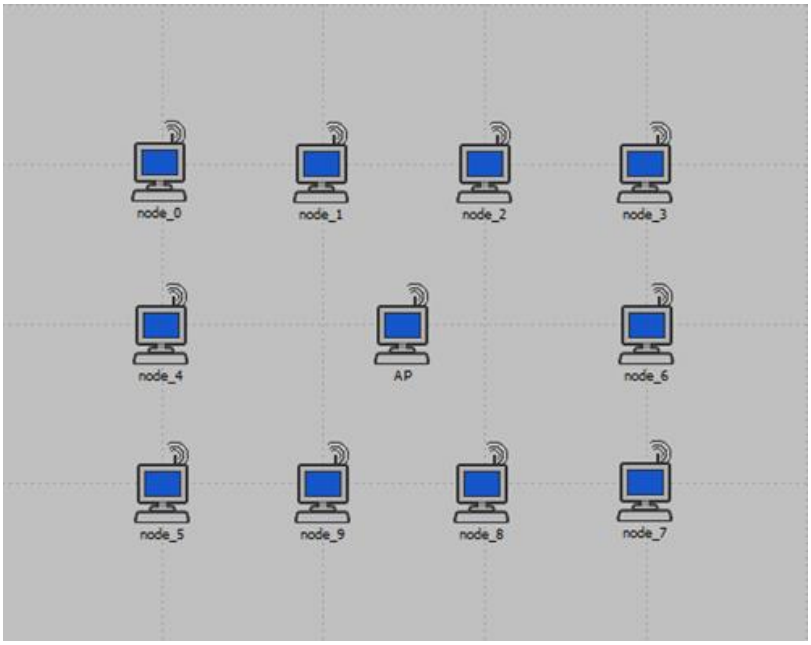
For example, is the user chose a star topology, they are asked to set the central node model, the periphery node model and their number, the link model, the placement of the whole topology in the X and Y axis in relation to the centre, and its radius. If the models the user desires are not available, they can click on the **Select Models...** button and pick the technology family from the list that includes these models. Finally, after everything has been configured, all that needs to be done is to press the **OK** button and the topology is automatically created using the provided parameters.

3.2.7 Configuring the links

All link types contain characteristics like the transmission rate and the propagation seed. Some of these characteristics are configurable, while some others are not. That depends on the link type and the characteristic itself. For example, say that the user desires to set the propagation delay of a point-to-point link. All they have to do is right click on the link → **Edit Attributes** → check the **Advanced** checkbox on the bottom right part of the window → **delay**: Distance Based (the most common setting, means that the propagation delay is relative to the distance the link has to cover). Of course, they can also click on the **Edit** button and set a fixed value.

3.2.8 Complete scenario

Let's assume that we have a computer network with ten terminals and one AP in the middle. In Riverbed Modeler, it looks something like this:



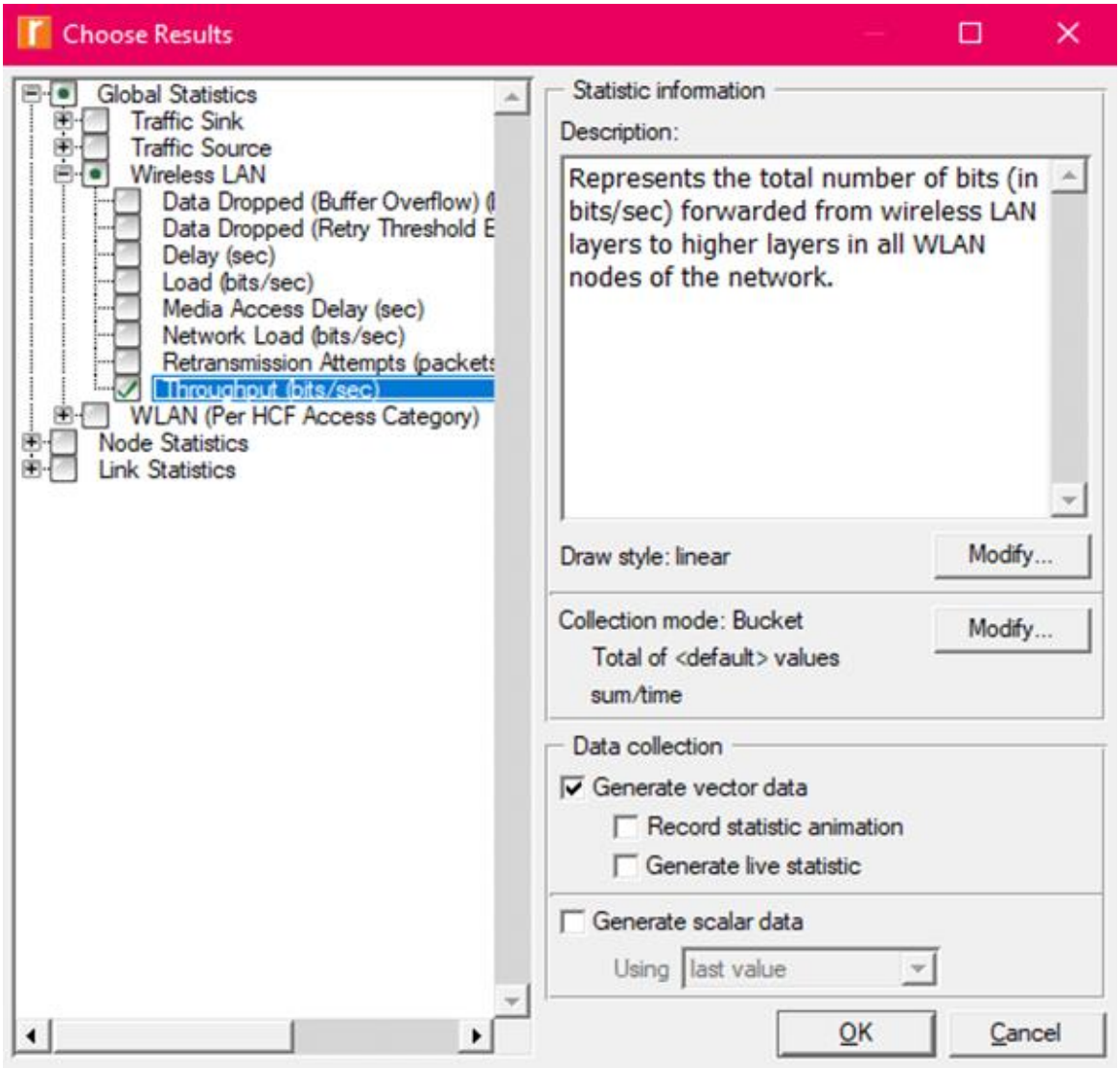
Screenshot 3.4: How the scenario looks in the software

In order to configure the various parameters, all the terminals are selected (node_0-node_9) → Right click → **Edit Attributes → Traffic Generation Parameters → Start Time (seconds)**: constant (5) → **Apply Changes to Selected Objects → OK**

Then, in order to set the functionality of the AP node, after it has been selected → Right click → **Edit Attributes → Wireless LAN → Wireless LAN Parameters → Access Point Functionality**: Enabled → **OK**

Before the simulation can begin, one more step has to be taken. Through the menu **Protocols → Wireless LAN → Configure PHY and Data Rate...** the **Technology** parameter is set to IEEE 802.11 (Frequency Hopping) and the **Data Rate** to 2 Mbps.

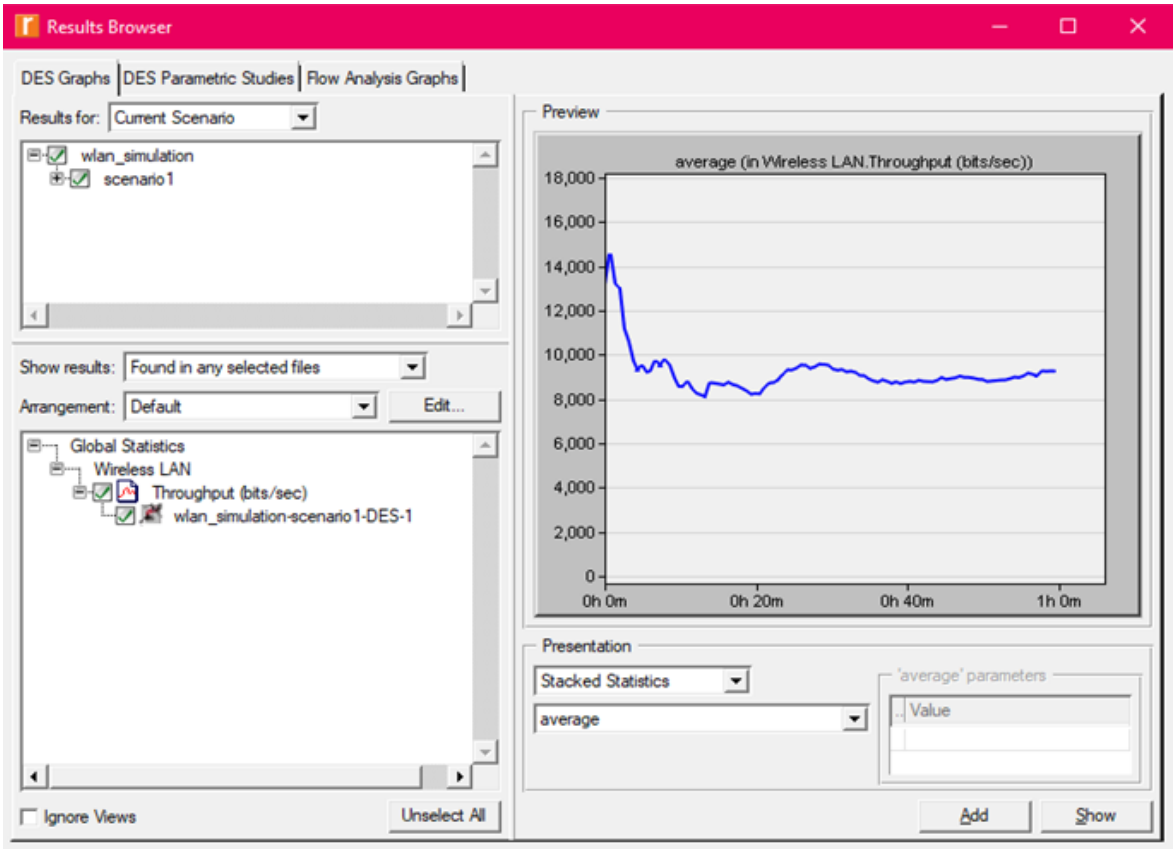
The next thing that needs to be done is to choose the results to be calculated and displayed. This is accomplished through the menu **DES → Choose Individual Statistics...**



Screenshot 3.4: Choosing the results to be displayed

All that needs to be done now is run the simulation, which is done through the menu **DES → Configure/Run Discrete Event Simulation...**, while at the same time setting the amount of time the simulation is going to run for.

In order to view the simulation results, the user must navigate to the menu **DES → Results → View Results...**. The following window will pop up:



Screenshot 3.5: Displaying the chosen results

Another feature of Riverbed Modeler is the fact that a single project can include more than one scenario. In order to create a new scenario, all the user has to do is navigate to the menu **Scenarios → New scenario...**.

Additionally, if the user were to create a new scenario where, for example, only a single parameter differs, they do not have to create a new one from scratch. Through the menu **Scenarios → Duplicate Scenario...** they can duplicate the current scenario and make the changes they desire in no time.

3.2.9 What about wireless networks?

Riverbed Modeler provides modelling, simulation and analysis for wireless networks too. It offers full protocol stack modelling capability, with the ability to model all aspects of wireless transmissions, like RF propagation, the amount of interference, various transmitter/receiver characteristics, node mobility (including handover), and the interconnection with wired networks. It also supports a wide range of wireless networking technologies like IEEE 802.11, MANET (Mobile Ad hoc network), 3G, GSM, LTE, IEEE 802.16, Bluetooth, ZigBee, Satellite, and Ultra Wideband.

3.2.10 Advantages of Riverbed Modeler

- Riverbed Modeler is by far the most widely used network simulator, having an impressive number of users. This makes it easy for the end user to receive support not only from the official source, but also through third-party sites.

- It also has the tendency to receive periodic updates, so it never ceased to receive support and fall behind the times, like most of the smaller-scale simulators ended up doing.

- In addition, Riverbed Modeler boasts an easy learning curve, mostly thanks to its fairly simple and to-the-point UI. Of course, should the user ever require assistance, a handy user guide is available at Riverbed's support website (https://support.riverbed.com/content/support/software/steelcentral-npm/modeler-index.html#docDocumentation).

- Finally, it has a very large library of simulation models

3.2.11 Disadvantages of Riverbed Modeler

- One point of criticism has been Riverbed Modeler's use of a pseudorandom number generator which is based on the random() function found in Unix BSD source code. This random number generator suffers from its low cycle length, which is not enough to avoid repetition of randomness. This can potentially be a problem with producing accurate simulation estimates with large variance (for example, when a large number of events are involved) and is a handicap for large-scale network simulations.

- Furthermore, since the kernel is not open source, no external tools are supported. Creating a custom component is also quite complex, so it is fortunate that Riverbed Modeler provides such a large model library.

- On top of that, as previously mentioned in its introduction, the Academic Edition of Riverbed Modeler will cease functioning by September 01, 2023, probably meaning that the software is going to be commercial-only (payware).

3.2.12 Benchmarking Riverbed Modeler

The computer that the tests (for this and all the other simulators) were performed on features an Intel Core i5-4690 (Haswell architecture) at 3,5 GHz base clock and 3,9 GHz boost clock and 8 GB of DDR3 RAM clocked at 1600 MHz. The software version was 17.5, which was the latest at the time of writing. Each benchmark was run three times and the final results are the average values between all three of them.

The simulation parameters are presented in the Table 3.1.

AODV was chosen as the routing protocol (for all the network simulators, not just for Riverbed Modeler) because of its popularity (it is the routing protocol ZigBee uses) [35], [36] and because it is offered as a ready-made protocol in all of the tested simulators. The host mobility was set to static, to eliminate any randomness to the end results due to the random mobility of the hosts.

The measurements were taken using Task Manager on Windows.

Table 3.1: Simulation parameters

Routing protocol	AODV
Simulation area	650 m x 650 m
Packet size	512 KB
Simulation time	500 seconds
Number of hosts	400-2000
Mobility	static

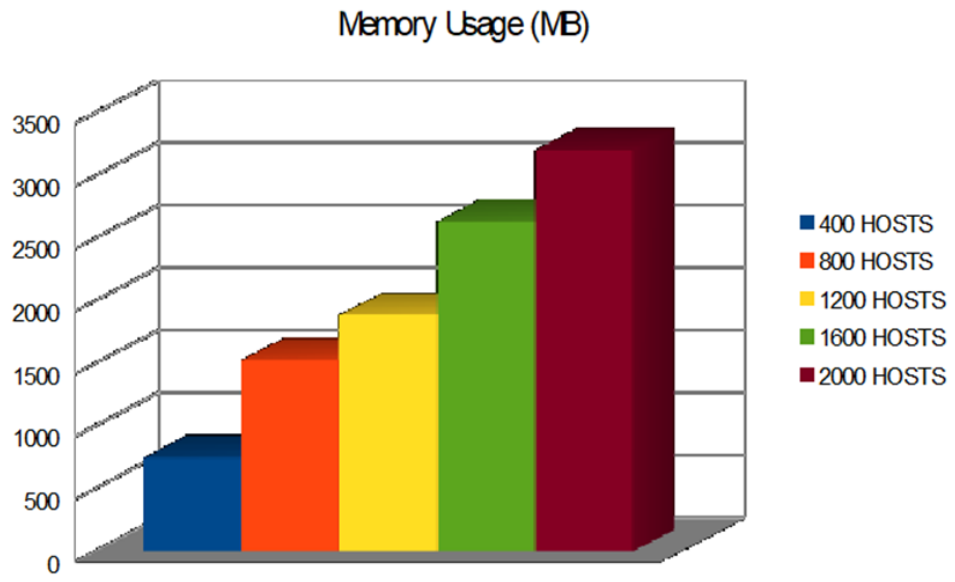


Figure 3.2: RAM consumption as the number of hosts increases

Table 3.2: Measurements results

Number of hosts	RAM consumption
400	478 MB
800	1527 MB
1200	1884 MB
1600	2633 MB
2000	3202 MB

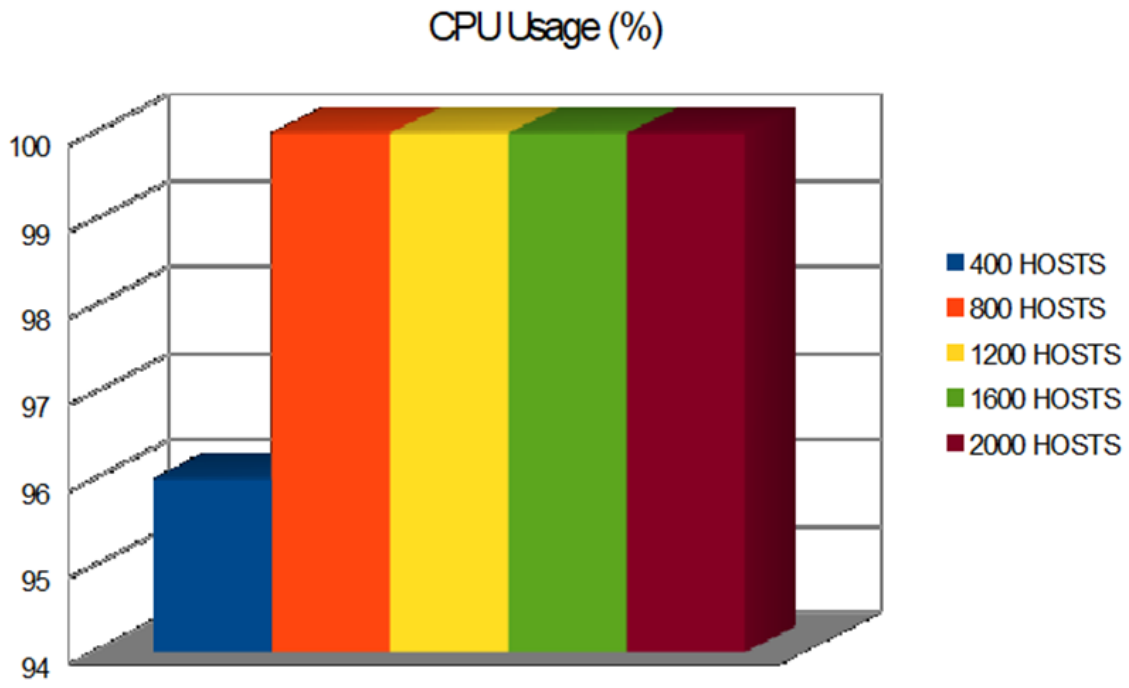


Figure 3.3: CPU consumption as the number of hosts increases

The results make evident that the RAM consumption increases in a linear fashion as the number of hosts increases, which is to be expected since an increase in the number of hosts results in more network events being generated, which in turn have to be recorded and/or used in the simulation itself.

Measurements were also taken for the CPU usage.

Table 3.3: Measurements results

Number of hosts	CPU usage
400	96 %
800	100 %
1200	100 %
1600	100 %
2000	100 %

The above result makes it clear that Riverbed Modeler tries to always max out the CPU's potential, to deliver the simulation results as fast as possible, which is the ideal behaviour.

3.3 ns-3

ns-3 (Network Simulator 3) is a free, open-source, discrete event network simulator. Due to its nature, it is not an officially supported software product of any company. It is used to simulate network protocols with different network topologies. It can simulate both wired and wireless networks and is written in C++ and Python.

It was initiated in 2006 and is still considered to be under development. Contrary to what some people might believe, it is not a backwards-compatible extension of ns-2. It is a completely new simulator. However, some models from ns-2 have already been ported from ns-2 to ns-3. Its latest version is ns-3.30, which was released in August 2019.

ns-3 is designed as a set of libraries that can be combined together, as well as with other external software libraries. Although several external animators, data analysis and visualization tools can be used with ns-3, users, for the most part, are expected to work using the command line and C++/Python software development tools [37]. Additionally, ns-3 network simulations can be implemented in pure C++, while some parts of the simulation can also be written in Python. It supports simulation, as well as emulation, using sockets. In order to analyse network traffic, standard tools, like for example Wireshark, can be utilized to read trace files. It requires a Linux operating system in order to work properly, but there are some, although limited, options if the user runs Windows or MacOS on their computer or laptop.

If the user runs Windows, slight support for Microsoft Visual Studio has been developed. However, it is worth mentioning that only Microsoft Visual Studio 2012 is supported. In addition, some elements of ns-3, like the emulation and TapBridge features, will not work since they depend on Linux support.

If the user runs MacOS, the Apple LLVM version must be 8.0.0 or above. Real-time simulation is also not available.

The first “universal” option, no matter the operating system, is through the use of virtual machines. All the user has to do is install a virtual machine that suits their preferences and download a Linux image (.iso file) of their choosing (Ubuntu is recommended) from the official website. Then, they can import the image directly to the virtual machine and start using the simulator that way.

The second option is to forgo the use of virtual machines and just install the Linux distribution of their choosing (again, Ubuntu is recommended) parallel to their main operating system. That way, their desktop computer or laptop can run Linux, and by extension ns-3, natively.

3.3.1 System requirements

ns-3 is primarily developed on (and pretty much requires) GNU/Linux platforms, and the absolute minimal requirements are:

- a GCC (at least 4.9) or Clang (any recent version) compiler
- a Python interpreter (at least 2.7 for version 2 and at least 3.4 for version 3)

Running ns-3 on Windows is technically feasible, but it requires either the use of a virtual machine, or NS-3 on Visual Studio, which is an experimental project using Microsoft's Visual Studio, although some of its core code has been altered to avoid compiler warnings and errors. Some results can also differ in comparison to the ones on Linux, since some of the utilized methods are different, or the OS operates in a different way.

As far as macOS is concerned, ns-3 depends on the Xcode development app that Apple provides and its clang/llvm compiler. The latest macOS version it supports, although with some problems, is Mojave (10.14).

3.3.2 Installing ns-3

Since the recommended OS is Linux, these steps are for the Linux platform. There are also guides in the official page of ns-3 (www.nsnam.org) for making it run on Windows or macOS, if the user so desires.

The first step is to install the required compilers, additional packages and libraries. To do so, the user needs to open a terminal and execute all the following commands:

- sudo apt-get install gcc g++ python
- sudo apt-get install python-dev
- sudo apt-get install qt4-dev-tools
- sudo apt-get install mercurial
- sudo apt-get install
- sudo apt-get install cmake libc6-de libc-6-dev-i386 g++-multilib
- sudo apt-get install gdb valgrind
- sudo apt-get install gsl-bin libgs10-dev libgs101db1
- sudo apt-get install flex bison libfl-dev
- sudo apt-get install tcpdump
- sudo apt-get install sqlite sqlite3 libsqlite3-dev
- sudo apt-get install libxml2 libxml2-dev
- sudo apt-get install libgtk2.0-0 libgtk2.0-dev
- sudo apt-get install vtun 1xc
- sudo apt-get install uncrustify
- sudo apt-get install doxygen graphviz imagemagick
- sudo apt-get install texlive texlive-extra-utils texlive-latex-extra texlive-font-utils dvipng
- sudo apt-get install python-sphinx dia
- sudo apt-get install python-pygraphviz python-kiwi python-pygoocanvas libgoocanvas-dev
- sudo apt-get install libboost-signals-dev libboost-filesystem-dev
- sudo apt-get install openmpi-common openmpi-doc libopenmpi-dev

Then, the user has to visit the official website and download the latest version of ns-3. At the time of writing the latest version is 3.30, therefore the file name is ns-allinone-3.30.tar.bz2. After it is done being downloaded, it is recommended to place it in the home folder (/home/) and extract it there.

After that, again through the terminal, the following commands have to be executed from the home folder in order to build the ns-3 package:

- cd ns-3-allinone
- ./build.py

This will take some time, depending on the memory and the CPU installed on the computer.

Finally, the last step is to configure it. This is done through the following commands:

- CXXFLAGS=”-03” ./waf configure
- ./waf -d optimized configure; ./waf
- ./waf --enable-examples configure
- ./waf --enable-tests configure

After all the above steps are done, the user can check that the installation went smoothly by running this command:

- ./test.py

3.3.3 Creating a new project and running a simulation

The first step is the programming of the actual network with its topology, the protocols it uses and its applications. The MAC and IP addresses are also set up in this step, as well as adding and writing the application classes, where most of the logic is implemented.

Using commands such as *#include “ns3/core-module.h”* and *#include “ns3/network-module.h”*, the files needed for the particular network in question are loaded.

Afterwards, the ns-3 namespace is used (*using namespace ns3;*) and a component for logging purposes is defined (*NS_LOG_COMPONENT_DEFINE (“ScriptExample”);*).

Since every ns-3 script is essentially just a C++ program, a main function needs to be declared. This is accomplished with the following code:

int

*main (int argc, char *argv[]) {*

Time::SetResolution (Time::NS);

This command sets the time resolution to 1 nanosecond, which is the default value. In other words, the smallest time value that can be represented is 1 ns.

Now, in order to represent the computers in a network, these two commands are used:

NodeContainer nodes;

nodes.Create (2);

At this point, the nodes do nothing, as they are not connected to a network. This can be addressed using the following code, which creates a simple point-to-point connection between them:

PointToPointHelper pointToPoint;

pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("10Mbps"));

pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer devices;

devices = pointToPoint.Install (nodes);

The above lines of code finish the configuration of the devices and the channel. In essence, the second line in particular “installs” the devices and makes the connection between them.

The next step is installing an Internet protocol stack. After the execution of the following commands, an Internet Stack will be installed on every node in the node container.

InternetStackHelper stack;

stack.Install (nodes);

Ipv4AddressHelper address;

address.SetBase ("10.1.1.0", "255.255.255.0");

Ipv4InterfaceContainer interfaces = address.Assign (devices);

This set of commands instructs the simulator to start allocating IP addresses from the network 10.1.1.0, using the mask 255.255.255.0 to the devices in the simulation.

UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));

serverApps.Start (Seconds (2.0));

serverApps.Stop (Seconds (60.0));

The above code snippet is an example as to how to install an application in ns-3. In this particular case, a UdpEchoServerApplication. The start parameter dictates at which point it time the application will start. The stop parameter is optional and dictates at which point the application will stop. So, according to the above code, the simulation in general is going to last at least 60 seconds.

UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);

echoClient.SetAttribute ("MaxPackets", UintegerValue (1));

echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));

echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));

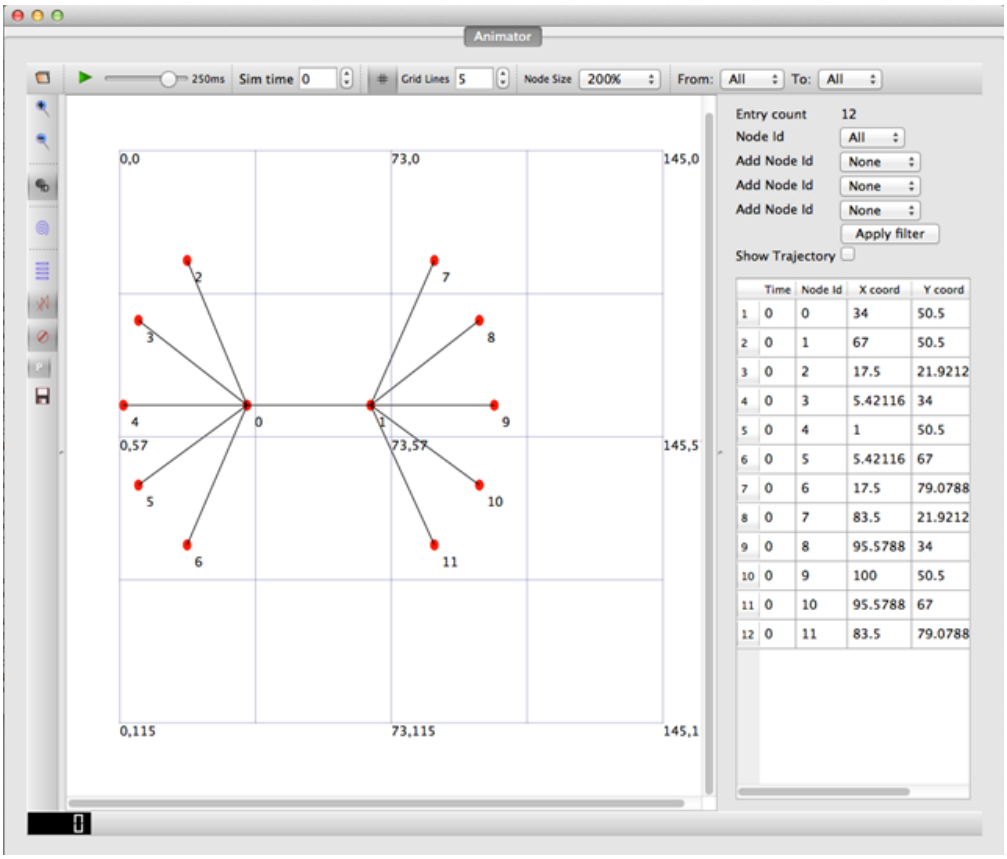
clientApps.Start (Seconds (5.0));

clientApps.Stop (Seconds (60.0));

This code is for the configuration of the UdpEchoClientApplication, which requires more parameters to be set.

The second step is describing the simulation behaviour. In addition, methods for initiating and stopping the simulation, as well as other control (like debug logging) and tracing methods have to be included. In general, the simulator offers trace sources and the user implements the trace sinks, which specify which information to keep and which to ignore. This is a key step, as too much or too little information can have a significantly negative impact on the simulation results.

The third step is the execution of the simulation, while the fourth and final one is the analysis of the simulation output. Unfortunately, ns-3 does not offer any tools to analyse, visualize or process the output data, so free tools like NetAnim are more often than not utilized.



Screenshot 3.7: The NetAnim GUI

The command to build the project is *$ ./waf*, while the one to run the simulation is *Simulator::Run ();*.

After the 60 second mark (which was defined at the stop parameters of both scheduled events), the simulation is going to stop. All that remains is to clean up by using this code:

Simulator::Destroy ();

return 0;

}

3.3.4 What about wireless networks?

ns-3 does provide a set of IEEE 802.11 models that attempt to provide an accurate MAC-level implementation of the IEEE 802.11 specification, as well as a “not-so-slow” PHY-level model of the IEEE 802.11a specification. All in all, ns-3 supports the following wireless models: IEEE 802.11a, IEEE 802.11b, IEEE 802.11g, IEEE 802.11e and IEEE 802.11s. Node mobility and propagation loss and delay models are also supported.

3.3.5 Advantages of ns-3

- Despite being an open-source project, its code is well organized
- Standard tools like Wireshark can be used to read trace files, helping in analysing network traffic

- It has a large user community, and as a result, a lot of ready-made models are available to download and use. The large number of users also help when searching for help on various online forums.
- Unlike some of the other network simulators, ns-3 is still under development and continues to receive periodic updates to this date

3.3.6 Disadvantages of ns-3

- It offers no graphical user interface, making it intimidating for users who are not comfortable using the command line and programming in general
- There are no first-party tools to analyse, visualize or process the output data of the simulations

3.3.7 Benchmarking ns-3

As with Riverbed Modeler, the tests started at 400 hosts, with each run adding 400 more until hitting the 2000 hosts mark. The software version was 3.30, which was the latest at the time of writing.

The command used for the measurements was top, which not only displays in real-time the percentage of CPU and RAM usage, but can also sort the processes by RAM or CPU usage. So, in the end, the commands used were *top -o %MEM* and *top -o %CPU*.

The results of these benchmarks were put in the following bar graphs and tables.

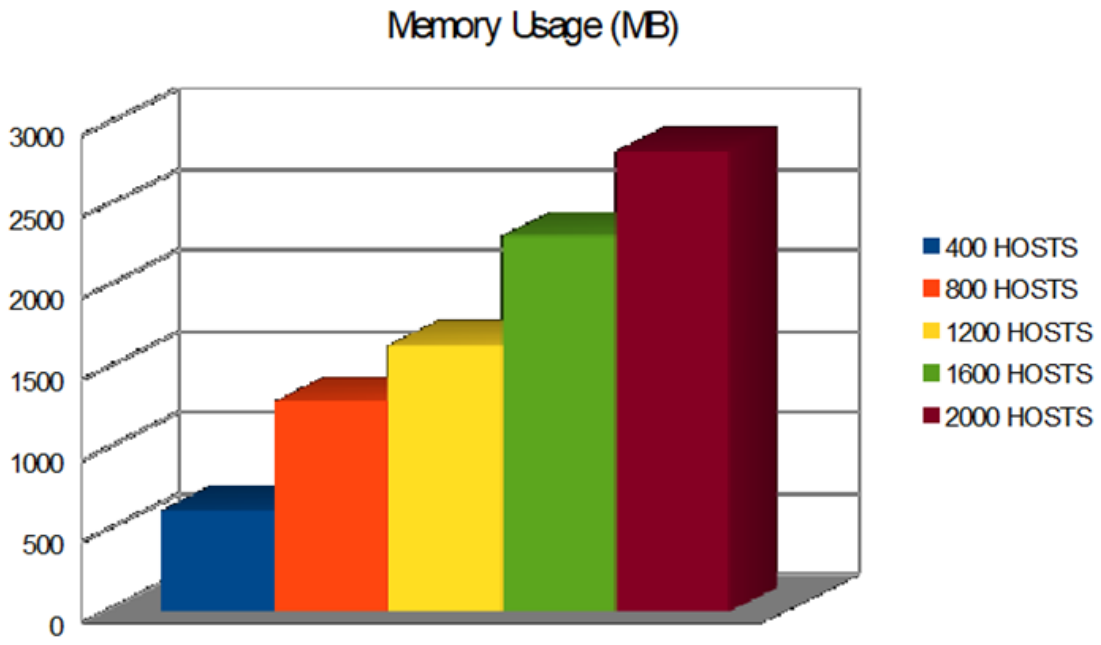


Figure 3.4: RAM consumption as the number of hosts increases

Table 3.4: Simulation results

Number of hosts	RAM usage
400	623 MB
800	1294 MB
1200	1638 MB
1600	2310 MB
2000	2834 MB

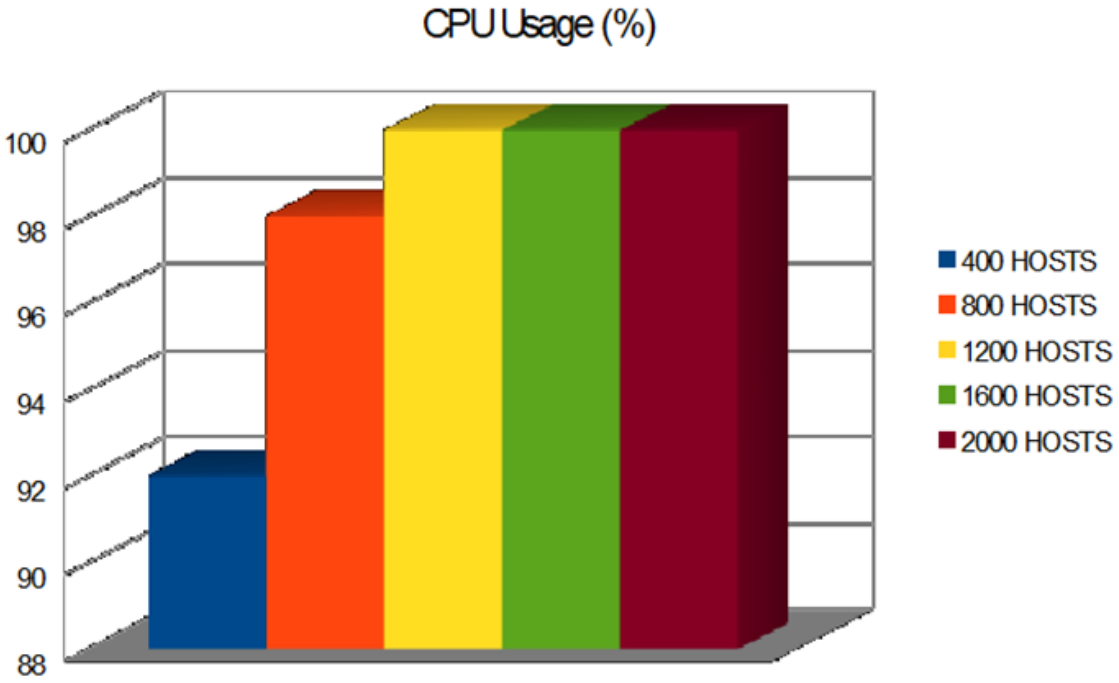


Figure 3.5: CPU consumption as the number of hosts increases

As seen in the same section for Riverbed Modeler, ns-3's RAM usage keeps increasing in parallel to the number of hosts to cope with the increased computational demand thanks to the increased number of events being generated.

Table 3.5: Simulation results

Number of hosts	CPU usage
400	92 %
800	98 %
1200	100 %
1600	100 %
2000	100 %

After the 1200 hosts mark, the CPU usage maxed out, meaning that ns-3 started taking full advantage of the available CPU power.

3.4 OMNeT++

OMNeT++ (Objective Modular Network Testbed in C++) is a discrete event, component-based (meaning modular), open-architecture, simulation framework, making it an appropriate simulation environment for network simulations. Its goal is to bridge the gap between open-source, research-oriented simulation software (like ns-2 and ns-3 for example) and expensive commercial alternatives like Riverbed Modeler. It features a strong graphical user interface (GUI) and intelligence support, as well as an embedded simulation kernel.

It was first launched in September 1997, and has since seen many iterations, while also having a large number of users. It is fairly easy to master because it makes it easy for the user to establish models, to modularize the software and to expand it.

Unlike ns-3, OMNeT++ was not designed just for network simulations. Because of its generic and flexible architecture, it can also be used in other areas, like simulating complex IT systems, designing multiprocessors and distributed hardware systems, and evaluating the performance of complex software systems [38].

Since it takes a component-based approach, it encourages structured and reusable models, which can be atomic (which capture the actual behaviour), or they can be composed of sub-models. These models are programmed in C++ and new ones are developed using the C++ class library which consists of the simulation kernel and utility classes for random number generation, statistics collection, topology discovery, etc. Modules communicate with each other by sending and receiving messages through their gates, which are linked to each other via connections. Modules that are connected are called compound modules. Each module has the ability to generate, read or react to messages, also known as events.

A high-level language, called Network Description (NED) is used to assemble individual components into larger components and models. In other words, it is a network description language used to create network topologies. Aside from the simulation kernel library, OMNeT++ also provides a Graphical Network Editor (GNED), a NED compiler, graphical and command line interfaces for simulation executions, graphical tools to analyse the simulation results, a model documentation tool, documentation, a user manual and example simulation scenarios.

Since it is a discrete event simulator, it generates predictions at a low level, which means that they are accurate, but at the cost of computational time. Also, apart from deterministic modelling, it also supports stochastic variables (both discrete and continuous) that give randomness to models. Other features

include failure modelling for nodes as well as links (crucial for accurate predictions), built-in traffic scenarios, multiple open-source random number generators, and parallel simulation functionality, which not only makes the simulation quicker, but it also enables OMNeT++ to model large-scale networks made up of tens of thousands of nodes [39].

It also has a strong GUI support, thanks to GNED, which not only enables the user to build network topologies graphically, but it can also generate these topologies as NED files. NED files are not used directly but are translated into C++ codes by the NEDC compiler, then compiled by the C++ compiler and linked into the simulation executable [40]. Tkenv supports interactive execution of the simulation (the user can start, pause or restart the simulation and change variable values or objects inside the simulation), tracing and debugging. Contrary to some other simulators, OMNeT++ can display messages informing the user of the state of the simulation, change the appearance of nodes to reflect an inner state, and even display simulation results while the simulation in still running.

Cmdenv is primarily designed for executing simulations in batches. Plove turns the output vector files of the network simulations into visual representation, while having the ability to plot multiple output vectors into a single graph. Scalar does the same thing, but it visualizes the output scalar files by creating XY plots or bar charts.

Finally, a feature that many users find useful is that the simulator executables that OMNeT++ creates can be run on other computers without the simulator, since they are standalone programs.

3.4.1 System requirements

There are no official system requirements available, other than the Installation Guide (version 5.6.1 at the time of writing) which only states the operating systems OMNeT++ has successfully been tested to be working on. The list includes Linux (Ubuntu 16.04 LTS, Fedora Core 25, Red Hat Enterprise Linux Desktop Workstation 7.x, OpenSUSE 42), Windows (7 and 10, 64-bit only) and macOS Sierra (10.12).

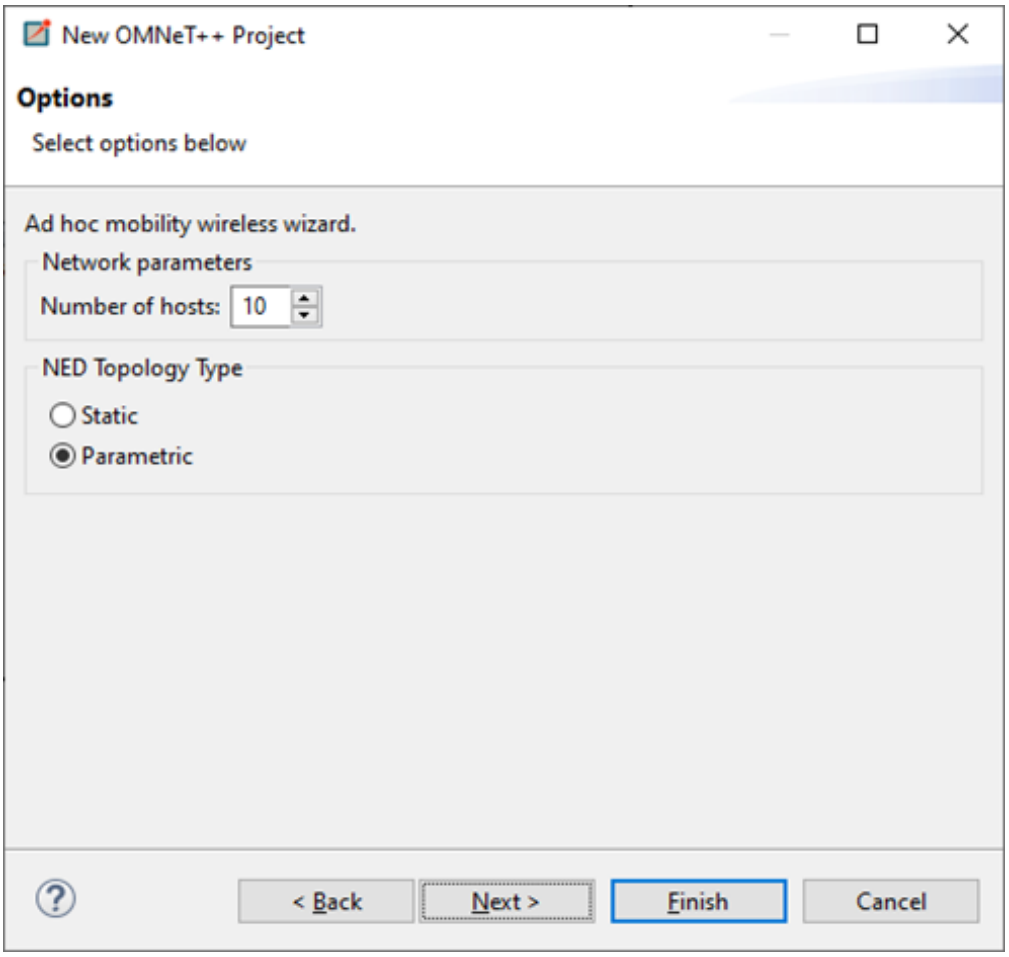
3.4.2 Installing OMNeT++

In order to download OMNeT++, you visit the official website (omnetpp.org), select the operating system you are going to use the simulator in, and download the program. There is also the option to download older versions of the software. Another way to download OMNeT++ is through their official GitHub repo, which can be found here: https://github.com/omnetpp/omnetpp. One advantage with this method is that the user has the option to download “preview” editions, meaning versions of the program still in their beta stage, which are yet to be publicly available.

After the download is completed, the user uncompresses the download which contains a single folder, where the whole program resides. Double-clicking on mingwenv opens a command prompt, where pressing any button will initiate the extraction process for the files needed. After that's done, typing *./configure* and then *make* will build the simulation libraries. When the process is done, typing *omnetpp* will open the OMNeT++ IDE.

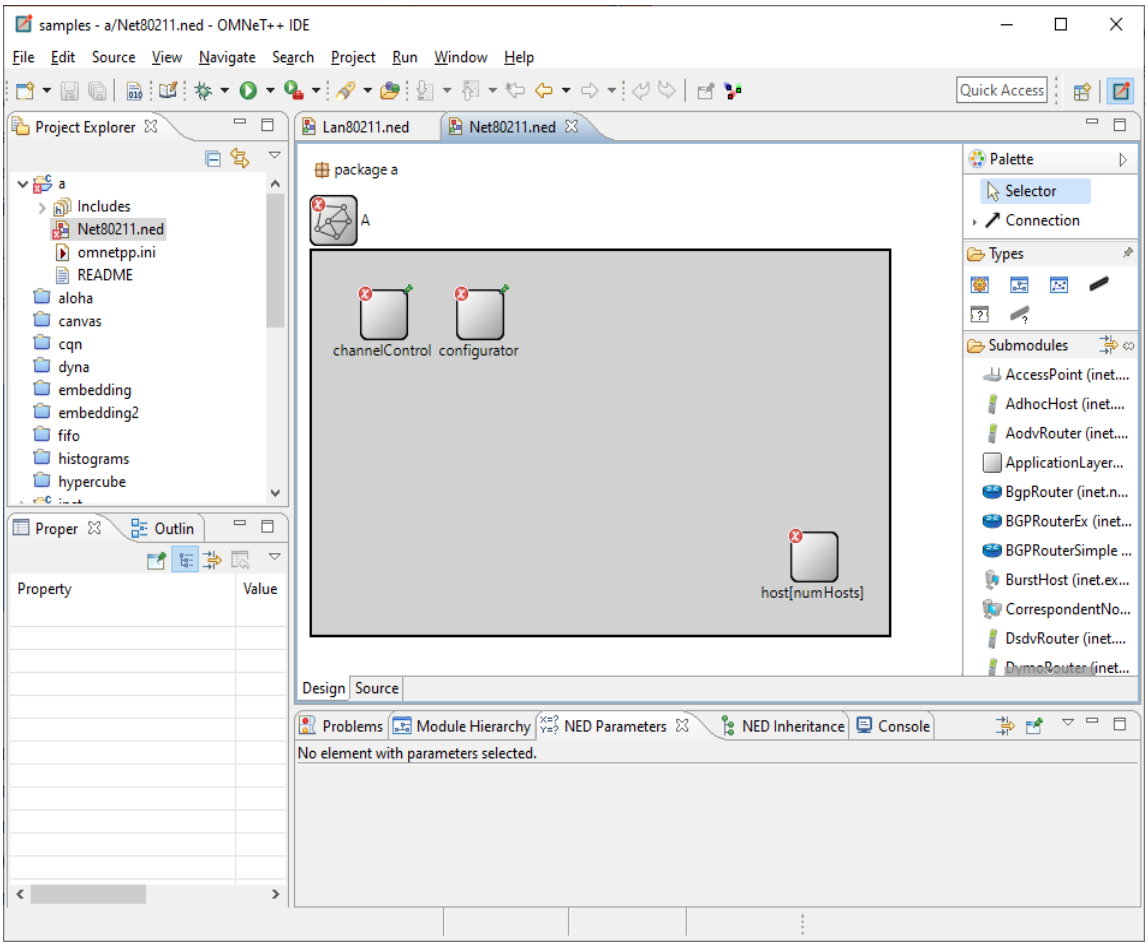
3.4.3 Creating a new project and adding network elements

In order for the user to create a new project, all they have to do is navigate to the **File** menu, then to **New** and finally select **OMNeT++ project...**. After defining the project name and its location, they are given the option to start the project from scratch or select from some pre-made ones which can parametrically set the number of nodes. The menu looks like this:



Screenshot 3.8: The configuration wizard for creating a new project

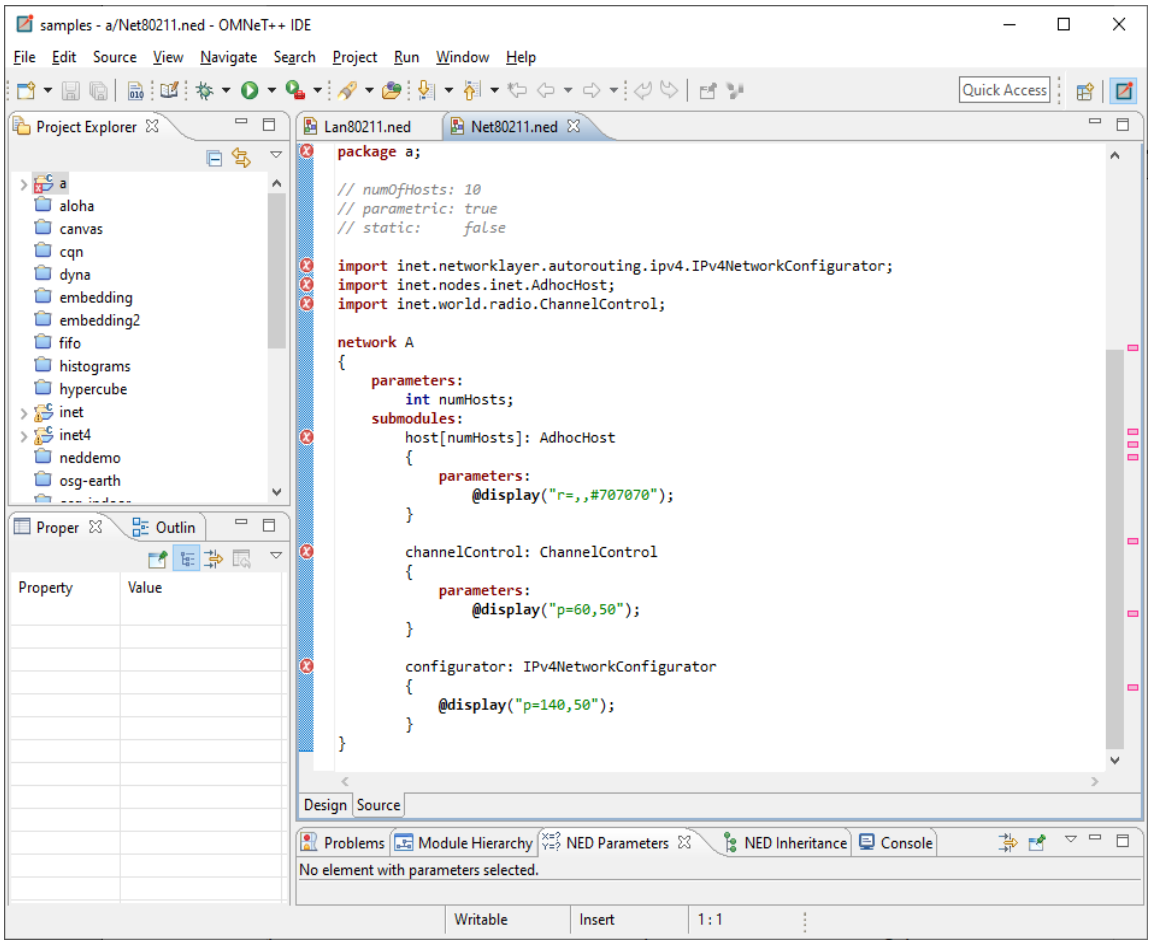
After pressing **Next** a couple more times, the user can finally access the workspace. From there, adding new network elements is as simple as drag-and-dropping them from the Submodules panel on the right to the workspace.



Screenshot 3.9: The workspace on OMNeT++

3.4.4 Adding the NED file

The NED file is created automatically when starting a new project, if for some reason the user wants to create a new one, all they have to do is right-click the project directory from the panel on the left and choose **New → Network Description File (NED)**. The same result can be achieved by going to **File → New → Network Description File (NED)**.This file has two modes: Design and Source. Changes made in one mode immediately reflect on the other one, so the user has the option to work on whichever is easier for them.



Screenshot 3.10: The Source mode in OMNeT++

3.4.5 Adding the C++ files

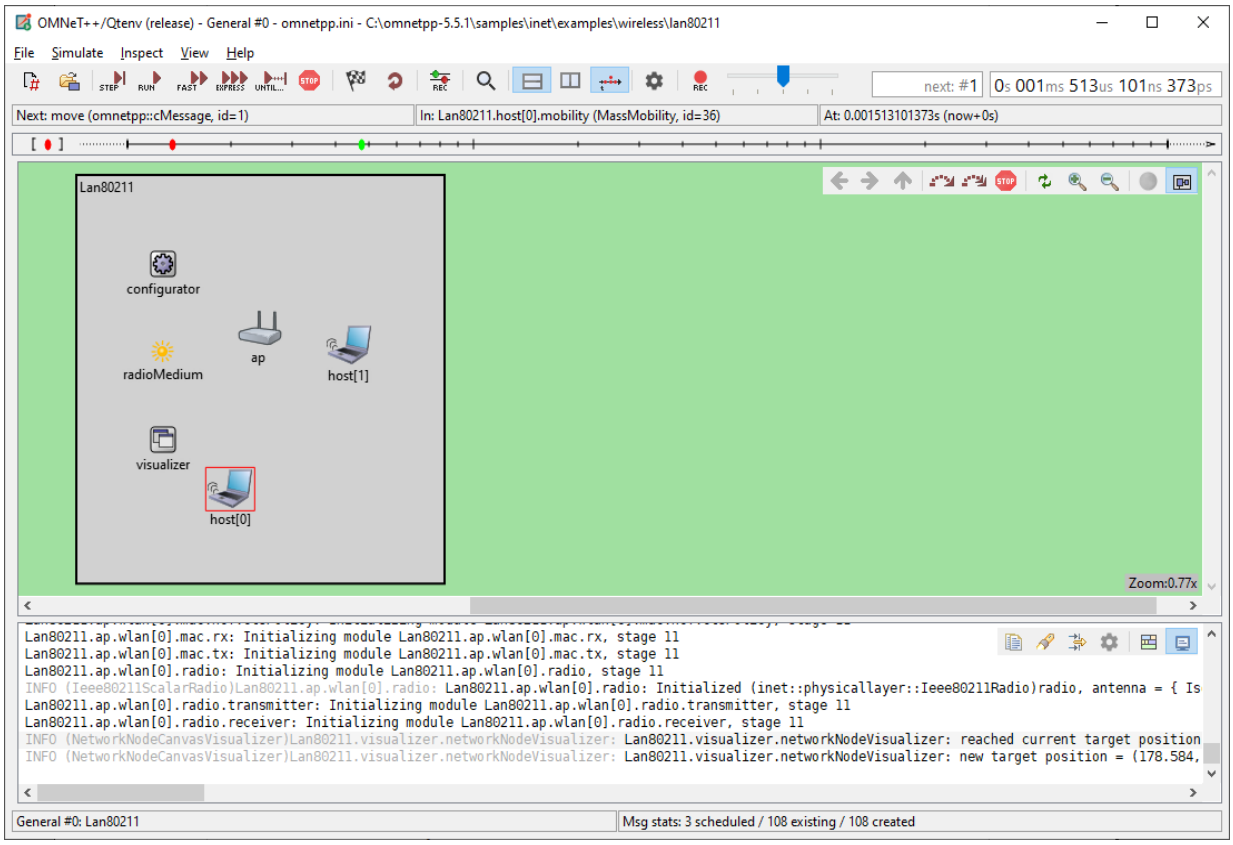
The C++ files are needed to implement the functionality of the modules included in the NED file. The process to create a new one is right-click on the project directory from the left panel, then pick **New → Source File**. The same result can be achieved by going to **File → New → Source File**.

3.4.6 Adding the omnetpp.ini file

The final step before running a simulation is creating the omnetpp.ini file, which defines which network is going to be simulated. This is necessary since NED files can contain more than one networks. This is done through right-clicking on the project directory from the left panel, then picking **New → Initialization File (ini)** or through **File → New → Initialization File (ini)**.

This file has two modes: Form and Source. The Form mode is much more user-friendly and requires no programming. The Source mode is basically just a text editor, but enables the user to enter module parameters.

After all the files have been created and configured, the simulations can begin. This is accomplished by clicking on the **Run** button. A new window opens, which has all the buttons and sliders needed to control the simulation, like a start, pause, cancel, fast-forward buttons and a slider to control the playback speed.



Screenshot 3.11: The simulation control environment

3.4.7 What about wireless networks?

Thanks to its plethora of external extensions, OMNeT++ can provide support for simulation of wireless networks. The most popular extensions are INET Framework and Mobility Framework for mobile ad-hoc networks. INET Framework is maintained by the OMNeT++ team for the community, utilizing patches and new models contributed by members of the community. It focuses on modelling TCP, UDP and IP networks, providing the modelling of IPv4, IPv6, IEEE's 802.11 based protocols, Ethernet and PPP. It also provides MANET routing protocols [41]. Thanks to their popularity, both INET Framework and Mobility Framework have API documentation.

3.4.8 Advantages of OMNeT++

- As previously mentioned, it takes the module approach, so it is highly modular
- As opposed to most network simulators, OMNeT++ is not limited to just network protocol simulation. It can, for example, be used to design multiprocessors.
- Being open-source, its source code is publicly available to anyone interested in studying it or even contributing to it
- It can also model mobility, which is a must-have feature for wireless network simulation that unfortunately a lot of network simulators fail to incorporate
- Thanks to its large and active user base, these are also tutorials and a user manual available
- The simulator executables that the simulator creates are actually standalone programs
- Unlike ns-3, it provides a GUI, which is also easy to understand and use

3.4.9 Disadvantages of OMNeT++

- It has been mentioned more than once that OMNeT++ performs not that great in reporting the simulation results, so it is advised for the users to code a lot of useful measurements themselves
- Some have also found the model design GUI to not be detailed enough, to the point that it fails to be useful
- The mobility extension has also received criticism for being somewhat incomplete

3.4.10 Benchmarking OMNeT++

As previously stated, the tests started at 400 hosts, with each run adding 400 more until hitting the 2000 hosts mark. The software version was 5.5.1, the latest at the time of writing.

The measurements were taken using Task Manager on Windows.

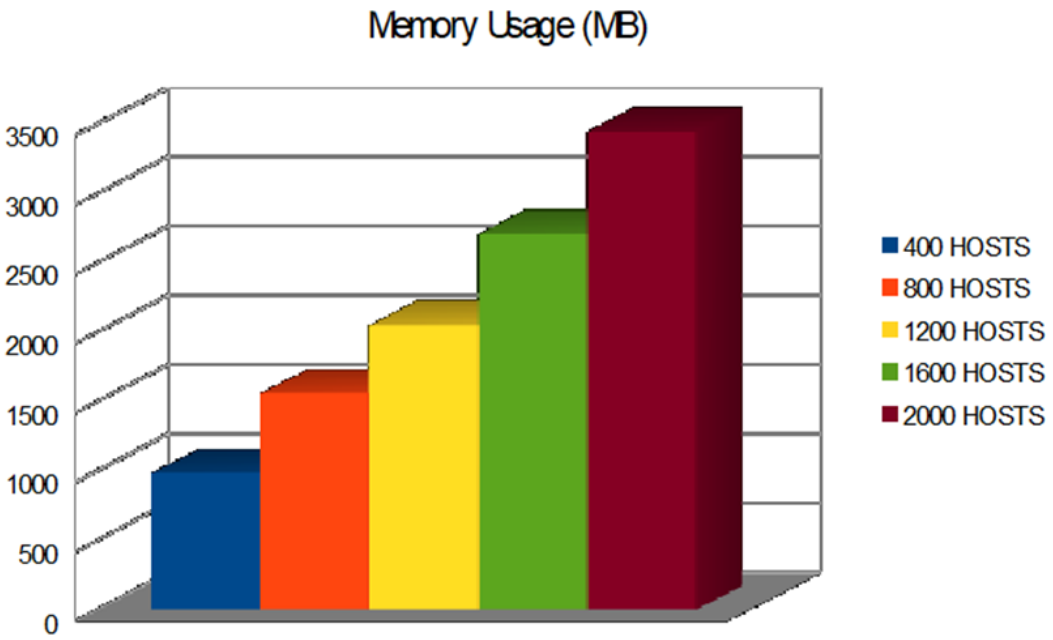


Figure 3.6: RAM consumption as the number of hosts increases

Table 3.6: Simulation results

Number of hosts	RAM usage
400	983 MB
800	1556 MB
1200	2048 MB
1600	2703 MB
2000	3441 MB

The results show the same pattern as Riverbed Modeler's and ns-3's, meaning that the RAM usage increases as the number of hosts do too. As previously stated, this is so that the system can cope with the increase in the network events being generated from an ever-increasing number of hosts.

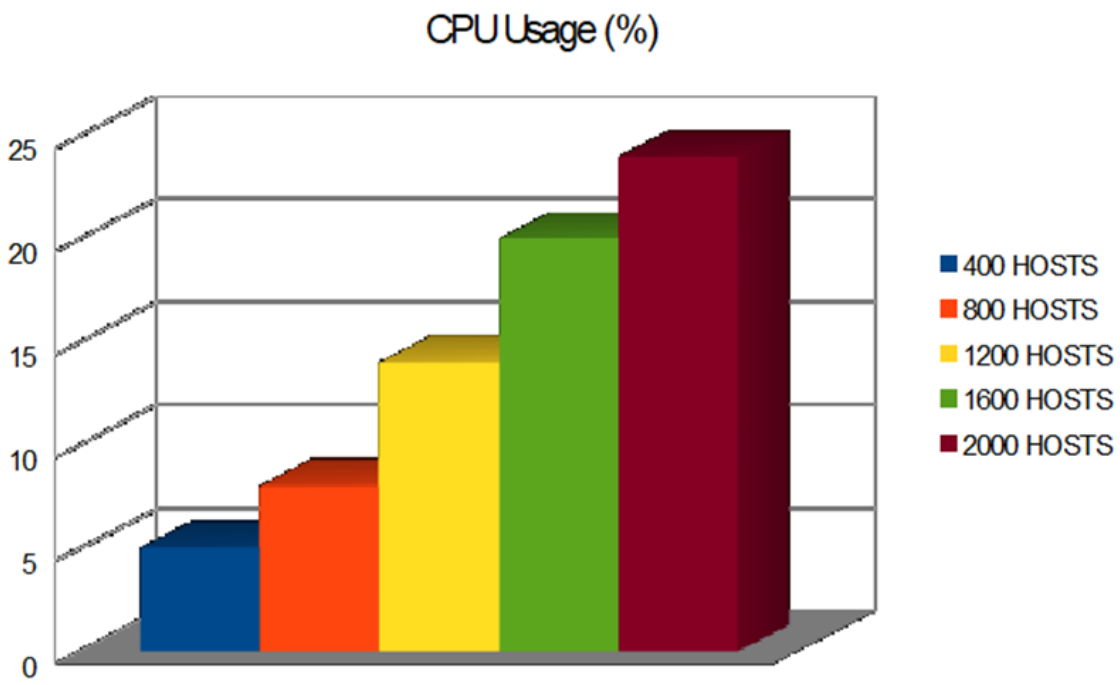


Figure 3.7: CPU consumption as the number of hosts increases

Table 3.7: Simulation results

Number of hosts	CPU usage
400	5 %
800	8 %
1200	14 %
1600	20 %
2000	24 %

The above numbers show that the CPU utilization grows in an almost linear fashion as the number of hosts keeps increasing, behaviour comparable to the one observed by the RAM usage in all the benchmarks performed. However, it never passed the 24% utilization mark, meaning that OMNeT++ did not max out the processor's capabilities, resulting in wasted potential processing time being saved.

3.5 J-Sim

J-Sim, initially known as JavaSim, is a Java-based, component-based, compositional simulation software developed by the Distributed Real Time Computing Laboratory of the Ohio State University and by the Illinois University. It has been designed based on the autonomous component architecture. This means that every network entity in J-Sim is a component. Of course, a component can also be composed of other, inner, components. They communicate through their ports, and 1-1, 1-many, many-many connection types are supported. How each component behaves is described with a contract. For example, a port contract describes the communication pattern of a component with the other components that are connected to an individual port.

It was modelled after ns-2, but with components in mind, since the goal was to scale better. It also features support not only for sensors, but also for physical phenomena. Additionally, it also supports the connection of real hardware.

The last patch J-Sim received was in June 2006, while the last "sign of life" in general was in May 2013, when an updated version of one of its components was released. All the above point to the fact that J-Sim is not in development any more, especially if one were to consider the fact that J-Sim's website has been left incomplete for many years. For example, a lot of links on the site still redirect to the old, original site (j-sim.cs.uiuc.edu), which is no longer up. In addition, most of the site images fail to load and are no longer available, since the domain that hosted them has been bought by another company and it currently being used as a manufacturing industry news outlet.

3.5.1 System requirements

In order for J-Sim to run, it requires a Java Virtual Machine (JVM) and a Java 2 SDK (J2SDK)-compliant class library. Its version should be at least 1.4. For stability reasons, the recommended one is the one Sun provides in their official website. Using a multi-processor machine with a large amount of memory is also recommended, although J-Sim has been tested to work on some very low-specification hardware, especially in today's time.

For the ones using Linux, the people behind J-Sim recommend IBM's Developer Kit or OpenJDK as viable alternative options.

3.5.2 Installing and running J-Sim

Since release 1.3, binary class files are not included in the download package. After downloading the package, found in the official website (sites.google.com/site/jsimofficial), the user needs to compile it using either "make" or "Apache Ant". The development team recommends Apache Ant, since it is easier to maintain and runs, to quote them, "much faster" than the alternative option.

Instructions for Windows (from the development team):

- Unpack the downloaded archive to any directory (as long as it already exists)
- In "setcpath.bat", modify the J_SIM and JAVA_HOME directories

- Run the “setcpath.bat” script to set the J_SIM, JAVA_HOME and CLASSPATH environment variables before using the package
- Compile the source using “make” or “ant compile”, if Apache Ant is installed
- Instructions for Linux:
- Unpack the downloaded archive to any directory (as long as it already exists)
- Set up the J_SIM, JAVA_HOME, CLASSPATH and IS_UNIX environment variables
- Compile the source using “make” or “ant compile”, if Apache Ant is installed

The command to run J-Sim is: java drcl.ruv.System <script> <argument>. In this command, <script> is the initial script to run with.

For example, java drcl.ruv.system test.tcl 10 “arg2” runs J-Sim with the test.tcl script, which in turn takes 10 and arg2 as its input arguments.

3.5.3 Creating a new project and adding network elements

After opening J-Sim, the user is greeted with a window that has two sections. The main one, which is where the command results are displayed, and the bottom one, which is where the commands are written.

So, if the user were to create a new component, they would have to execute something in the lines of the following:

```
public class MyComponent extends drcl.comp.Component {
    drcl.comp.Port myPort;
    public MyComponent() {
        super();
        myPort = addPort("ping-pong");
    }
    public void process(Object data_, drcl.comp.Port inPort_) {
        if (inPort_ == myPort && "ping".equals(data_))
            inPort_.doSending("pong\n");
    }
}
```

The code above creates a component that responds to “ping” messages with “pong”.

Now, in order to use the component, all that needs to be done is run J-Sim in the same directory the component is located and run these commands:

```
set c [java::new MyComponent]
[$c getPort "ping-pong"] connectTo [! /.term/tcl0/result@]
java::call drcl.comp.Util setRuntime $c [java::new drcl.comp.ARuntime]
java::call drcl.comp.Util inject "ping" [$c getPort "ping-pong"]
```

Since J-Sim follows a parent/child hierarchy, it is possible to set a component as a child of another component, thus exposing it to the parent. For example,

set c [java::new MyComponent]

set parent [java::new drcl.comp.Component]

$parent addComponent $c

java::call drcl.comp.Util setRuntime $parent [java::new drcl.comp.ARuntime]

$parent expose $c

[$parent getPort "ping-pong"] connectTo [! /.term/tcl0/result@]

java::call drcl.comp.Util inject "ping" [$parent getPort "ping-pong"]

will yield the same results as the previous code snippet.

3.5.4 What about wireless networks?

J-Sim does provide a wireless extension for the simulation of wireless networks. However, the only available MAC layer in this extension is IEEE's 802.11 [42].

3.5.5 Advantages of J-Sim

- One nice feature for which J-Sim receives praise for is the ability to set “enable”, “disable” and “display” flags for the components. This means that the user can set failures for nodes or protocols by simply disabling them. Moreover, J-Sim offers the ability for its components to be plugged into a software system, even during execution.
- Additionally, since it has been developed entirely in Java, in conjunction with the autonomous component architecture it incorporates, it is a truly platform-neutral, extensible and reusable environment. This however does not mean that the user is forced to use Java. J-Sim provides a script interface to allow the integration with script languages like Python, Perl, or Tcl. For Tcl in particular, Scriptics Corporation has developed Jacl, which is a Tcl interpreter, making J-Sim a dual-language environment. So, the user can use Java to create the components and Tcl to integrate them at run time and to provide dynamic control
- Diving deeper to the way J-Sim works, throughout the duration of a simulation, it maintains a globally-observed, virtual system time that is proportional to the real-world time. When no WorkerThread is currently active, the simulation engine makes adjustments to the virtual system time to the nearest future where at least one execution can be activated. In other words, it basically performs a “fast-forward”. So, at any given time, at least one WorkerThread is active. This mechanism ensures that a simulation is able to run in the same manner a real system would, in the sense that executions are taking place in real time and not at fixed time points in discrete event simulation. This of course makes the simulation results more reliable and closer to the ones in the real world.

3.5.6 Disadvantages of J-Sim

- One point of criticism is that all the work on J-Sim is done voluntarily, so no one in particular is responsible for bugs. This makes a lot of people avoid using it in research projects, as the validity of its models is questionable.
- In addition, J-Sim uses Java's own utility to generate random numbers, which has an insufficient cycle length to avoid repetition of randomness in large-scale networks. This means that the software is best suited for small to medium scale networks.

- There is also the fact that J-Sim's documentation is, according to many, inadequate and its result reporting poor, meaning that the user has to code many measurements themselves
- J-Sim was not originally meant to simulate wireless sensor networks, so its design prevents most users from developing (and by extension adding) new protocols or node components [43].
- Of course, and perhaps more importantly, J-Sim has not been updated since 2006, meaning that it remains unchanged for at least 14 years.

3.5.7 Benchmarking J-Sim

As with all the other simulators, the tests started at 400 hosts, with each run adding 400 more until hitting the 2000 hosts mark. The software version was 1.3 with patch4, which was the latest at the time of writing. The measurements were taken using Task Manager on Windows.

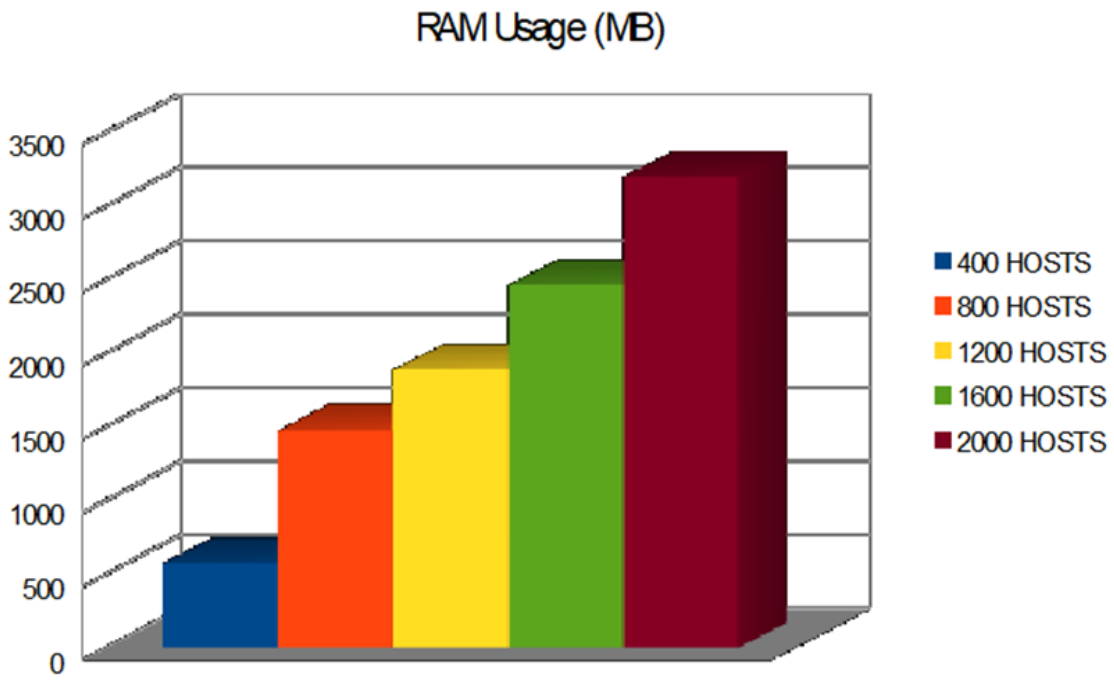


Figure 3.8: RAM consumption as the number of hosts increases

Table 3.8: Simulation results

Number of hosts	RAM usage
400	573 MB
800	1475 MB
1200	1884 MB
1600	2458 MB
2000	3195 MB

The results at this point were not surprising, showing an increase in RAM as the number of hosts increases. However, they are still interesting as they confirm that neither simulator has a “baked-in” cap for the amount of RAM it consumes (at least not one below 3 GB).

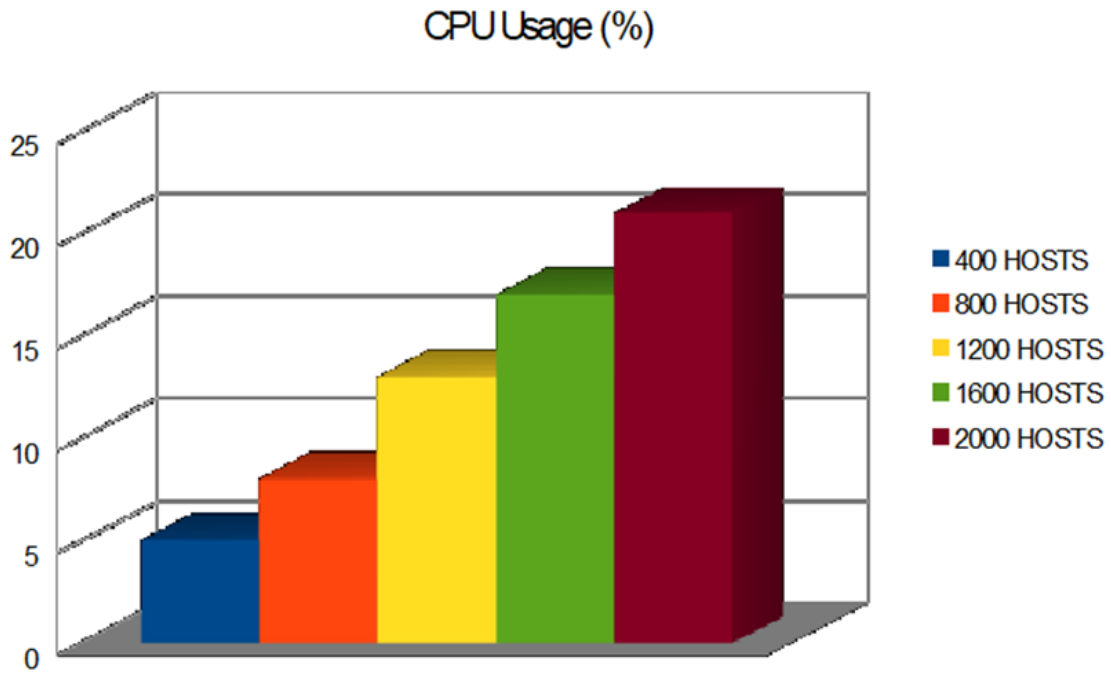


Figure 3.9: CPU consumption as the number of hosts increases

Table 3.9: Simulation results

Number of hosts	CPU usage
400	5 %
800	8 %
1200	13 %
1600	17 %
2000	21 %

As the numbers show, the CPU utilization grew in an almost linear fashion as the number of hosts kept increasing. However, it never passed the 21% utilization mark, meaning that J-Sim could potentially deliver its results faster, if it could max out the processor. The same behaviour was previously observed with OMNeT++ as well.

3.6 Comparing Riverbed Modeler, ns-3, OMNeT++ and J-Sim

Each network simulator has its own strengths and weaknesses, and that means that each one is more suitable for specific purposes that the others. Starting the comparison, among the aforementioned simulators, ns-3 is, according to many research papers, the fastest in terms of computation time. In addition, it has the ability to reduce its CPU utilization when other applications are executed in parallel, which is a common occurrence since simulations usually take a long time, so researchers use other programs in the meantime.

On the other hand, it is the most difficult simulator to learn how to use by a significant margin, since it requires TCL language [44] in order to design the network and AWK scripting to manually extract data from trace files. It also offers an integrated debugging environment, so the user has to write print commands in order to understand how the simulator works. From the simulator with the biggest learning curve to the one with the smallest, the list would probably be: ns-3, J-Sim, OMNeT++ and finally Riverbed Modeler.

This is in part thanks to the fact that Riverbed Modeler and OMNeT++ offer a GUI, which makes them much more user-friendly and easier to understand. In contrast, ns-3 and J-Sim do not offer one, so the user needs to have some programming experience. There are some third-party projects attempting to remedy this, but they have to be installed manually and offer limited features.

3.6.1 System resources usage

According to the benchmarks performed, ns-3 is the most RAM-friendly simulator in all the various scenarios, while OMNeT++ is the one which consumes the most memory. The second least memory hungry one is J-Sim, and the third spot is occupied by Riverbed Modeler.

In terms of CPU usage, Riverbed Modeler is the first one to reach 100% utilization (with 800 hosts), with ns-3 following close behind (with 1200 hosts). OMNeT++ and J-Sim performed a bit differently, since they never reached more than 24% and 21% utilization respectively, resulting in most of the potential processing power remaining unused.

Table 3.10: RAM usage for each simulator

SIMULATOR	400 NODES	800 NODES	1200 NODES	1600 NODES	2000 NODES
Riverbed Modeler	478 MB	1527 MB	1884 MB	2633 MB	3202 MB
ns-3	623 MB	1294 MB	1638 MB	2310 MB	2834 MB
OMNeT++	983 MB	1556 MB	2048 MB	2703 MB	3441 MB
J-Sim	573 MB	1475 MB	1884 MB	2458 MB	3195 MB

Table 3.11: CPU usage for each simulator

SIMULATOR	400 NODES	800 NODES	1200 NODES	1600 NODES	2000 NODES
Riverbed Modeler	96 %	100 %	100%	100%	100%
ns-3	92%	98%	100%	100%	100%
OMNeT++	5%	8%	14%	20%	24%
J-Sim	5%	8%	13%	17%	21%

3.6.2 Scalability

The term refers to “the capacity to be changed in size and/or scale”. This means that, in this case, a network simulator offers great scalability if the association between the increase in the number of nodes and the increase in RAM consumption (or the increase in the time it takes to complete the simulation) follows a linear path or is even better.

Measuring the percentage increase of the RAM usage each time the number of nodes was doubled, gave the following numbers, which can help in determining the scalability potential of each software individually.

The results, as presented in Table 3.12, show that OMNeT++ is the most scalable one, since the increase in the RAM usage was not only fairly consistent, but also under 200% in both cases. Riverbed Modeler and ns-3 also did quite well, staying very close to each other's numbers. J-Sim had the worst performance out of them all by a pretty significant margin.

Table 3.12: Simulation results

SIMULATOR	INCREASE FROM 400 TO 800 NODES	INCREASE FROM 800 TO 1600 NODES
Riverbed Modeler	204%	172%
ns-3	208%	179%
OMNeT++	158%	174%
J-Sim	257%	167%

Chapter 4: Comparing AODV, DSDV and DSR in OMNeT++

4.1 AODV

Ad hoc On-Demand Distance Vector routing (AODV) is a routing protocol for wireless ad hoc networks, more often than not Mobile Ad hoc Networks (MANETs). It was presented to the public on July 2003. AODV was developed by C. Perkins (Nokia Research Center), E. Belding-Royer (University of California, Santa Barbara) and S. Das (University of Cincinnati) [45].

AODV is being used by ZigBee as its routing protocol, which is a wireless ad hoc network that features low power and low data rates. Implementations of AODV include MAD-HOC, Kernel-AODV, AODV-UU, AODV-UCSB and AODV-UIUC.

AODV was designed to handle from tens up to thousands of mobile nodes with low, medium and even relatively high mobility rates. It was also designed to reduce the dissemination of control traffic and also to eliminate overhead on data traffic, so as to improve the scalability and the performance.

The algorithm AODV uses features dynamic, self-starting and multihop routing between the participating mobile nodes who want to establish, as well as maintain, an ad hoc network. It enables mobile network nodes to quickly obtain routes for new destinations and does not require from nodes to maintain routes to destinations that are not part of an active communication. In addition, it allows mobile nodes to respond in a timely fashion to link changes/breakages in the network topology. AODV operates loop-free, and by avoiding the "counting to infinity" problem, it offers quick convergence when the ad hoc network topology changes (more often than not when a node or multiple nodes at once move). When links break, AODV notifies the affected set of nodes in order for them to be able to invalidate the routes using the lost links.

A feature that differentiates AODV is that is uses a destination sequence number for each and every route entry. The destination sequence number is created by the destination in order to be included with any route information being sent to requesting nodes. Using destination sequence numbers ensures loop freedom and as a bonus, it is simple to implement programmatically. When there are two different routes to a destination, a requesting node is required by AODV to select the one that has the greatest sequence number.

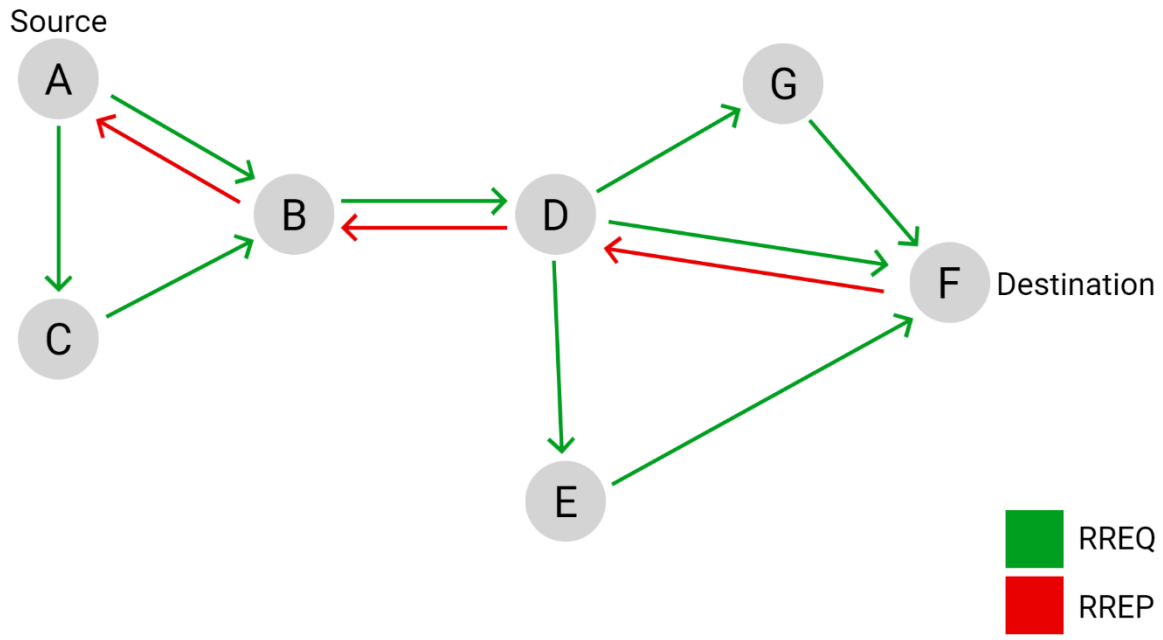


Figure 4.1: Example of an AODV routing protocol

4.2 DSDV

Destination-Sequenced Distance-Vector Routing (DSDV) is a routing scheme for ad hoc mobile networks that is table-driven and is based on the Bellman–Ford algorithm, like AODV, in order to solve the routing loop problem. This is no coincidence, as it was developed by C. Perkins and P.Bhagwat in 1994. C. Perkins also contributed to AODV, which was rolled out nine years later.

Each entry in the routing table contains a sequence number. This number is even if a link is present and odd if not. This number is being generated by the destination, and the source has to send out the next update with this number. Routing information is distributed between the nodes by infrequently sending out full dumps and more frequently smaller, incremental, updates.

When a router receives some new information, it uses the latest sequence number. If the sequence number is the same as the one already existing in the table, the route with the better metric is picked to be used. Entries that haven't been updated for a while (called stale entries), as well as the routes that use these nodes as next hops are deleted.

Since there are always paths available to all destinations in the network, the path setup process requires less delay than it would traditionally require. In addition, the method of incrementally updating with the sequence number labels makes the existing wired network protocols adaptable to ad hoc wireless networks. This means that all the available wired network protocols can be used in ad hoc wireless networks with little modification required.

As for DSDV's disadvantages, one is the fact that it uses a small amount of energy and bandwidth even when the network is sitting idle, because it requires regular updates to its routing tables. Another disadvantage is the fact that DSDV is not suitable for networks that are large-scale or highly dynamic. This is because whenever the network topology changes, a new sequence number is necessary before

the network can reconverge. Of course, this does not affect network traffic in the regions of the network that are not affected by the change in the topology.

Nowadays, DSDV has mostly been phased out, with other protocols that feature similar techniques taking its place. The best-known sequenced distance vector protocol is AODV, which, since it is a reactive protocol, can use simpler sequencing heuristics. Another one is Babel, which is an attempt to make DSDV more robust, more efficient and more widely applicable while at the same time staying within the framework of proactive protocols.

4.3 DSR

Dynamic Source Routing (DSR) is a routing protocol used in wireless mesh networks. It is an on-demand routing protocol designed to restrict the bandwidth consumed by control packets in wireless ad hoc networks by eliminating the periodic table-update messages required in the table-driven approach. It is also beaconless and as a result does not require periodic beacon (hello) transmissions to inform the neighboring nodes of its presence. As was the case with DSDV, it is similar to AODV in the sense that it forms a route on-demand when a transmitting node requests one. However, unlike AODV, it uses source routing (alternatively called path addressing) instead of relying on the routing table at each intermediate node. This means that the transmitting node can specify a partial or the complete route that the packet will take through the network. This way of routing makes troubleshooting easier, but, on the other hand, the source can't force packets to travel through a specific path in order to prevent congestion. In other words, direct management of the network performance is not possible. In addition, the connection setup delay is higher than in table-driven protocols, because each connection has to be created on the fly when needed. Performance also suffers when there is high mobility involved, although DSR performs well when there is no mobility involved, or when there is low mobility.

During route discovery, in order to determine the source route, the address of each device between the source and the destination is required. This information is cached by nodes processing the route discovery packets. In order to achieve source routing, each packet contains the addresses of every node it will have to travel through. This can result in high overhead, especially in scenarios where the path is very long, or the addresses themselves are long (like in IPv6). To combat this, there is also the option to allow the packet to be forwarded on a hop-by-hop basis.

There are two phases in DSR, Route Discovery and Route Maintenance. There is also Route Reply, which only occurs when the packet has successfully reached its destination, and Route Error, when a node has "failed". This flags this specific node for avoidance and the Route Discovery phase begins once again, to avoid any paths using this problematic node.

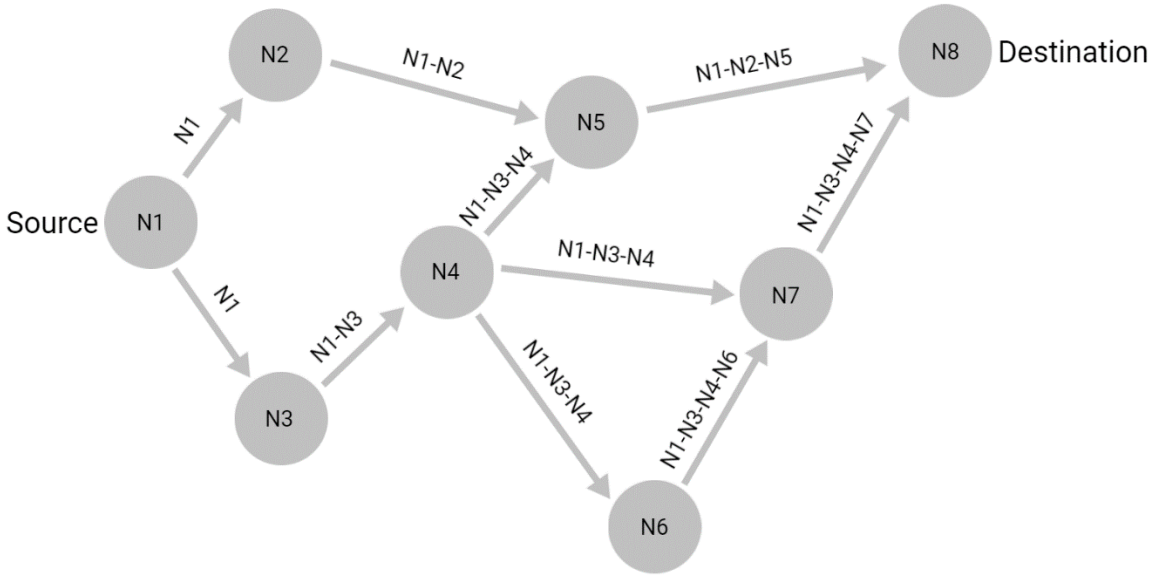


Figure 4.2: DSR Route Discovery

4.4 Packet error rate

In this second round of benchmarks, for each of the aforementioned routing protocols, 5 benchmark runs were run. Starting with 10 nodes in the network, each run added 10 more, stopping at 50 nodes. The parameters that will be compared are the packet error rate, the queueing time and the minimum signal to interference ratio.

Table 4.1: Simulation results

Number of nodes	AODV	DSDV	DSR
10	0,036	0,01	0
20	0,117	0,113	2,412
30	0,044	0,221	0,495
40	0,062	0,317	3,037
50	0,636	0,419	0,587

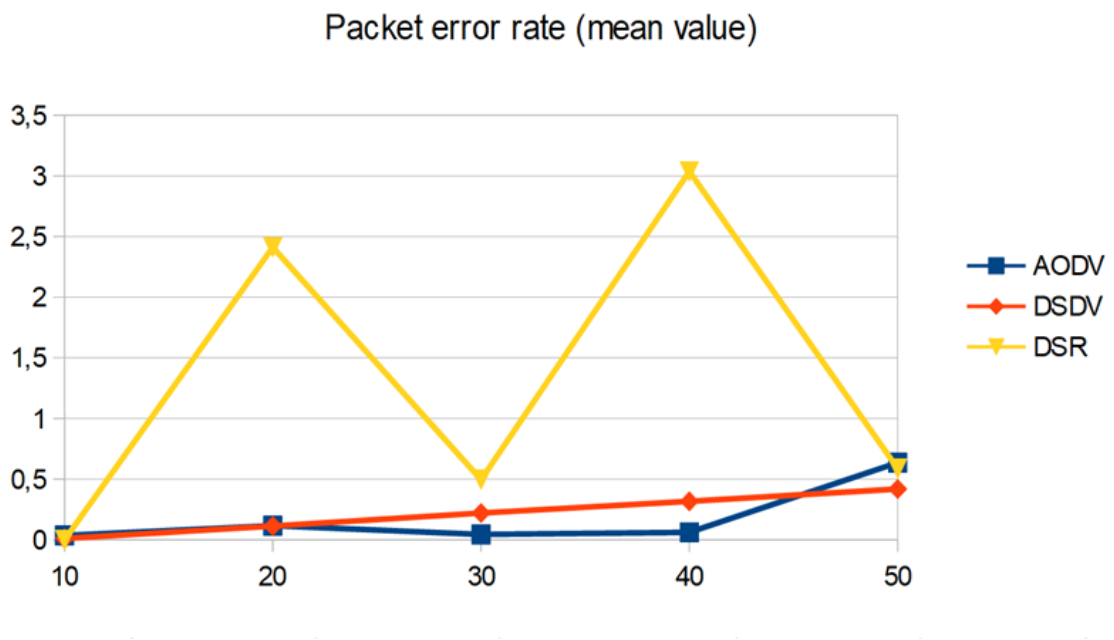


Figure 4.3: The packet error rate as the number of nodes increases

Judging from the results above, it is safe to assume that DSR is the worst routing protocol out of the three in regards to the packet error rate, displaying consistently worse results across all the benchmark runs. It was also the protocol with the most fluctuations in its measurements. DSDV measurements followed an almost linear growth path, while AODV's were very consistent until the run with the 50 nodes.

4.5 Queueing time

Table 4.2: Simulation results

Number of nodes	AODV	DSDV	DSR
10	3,326	4,078	1,42
20	4,412	4,677	3,296
30	0,207	3,850	2,471
40	0,867	2,508	3,028
50	0,201	3,031	2,083

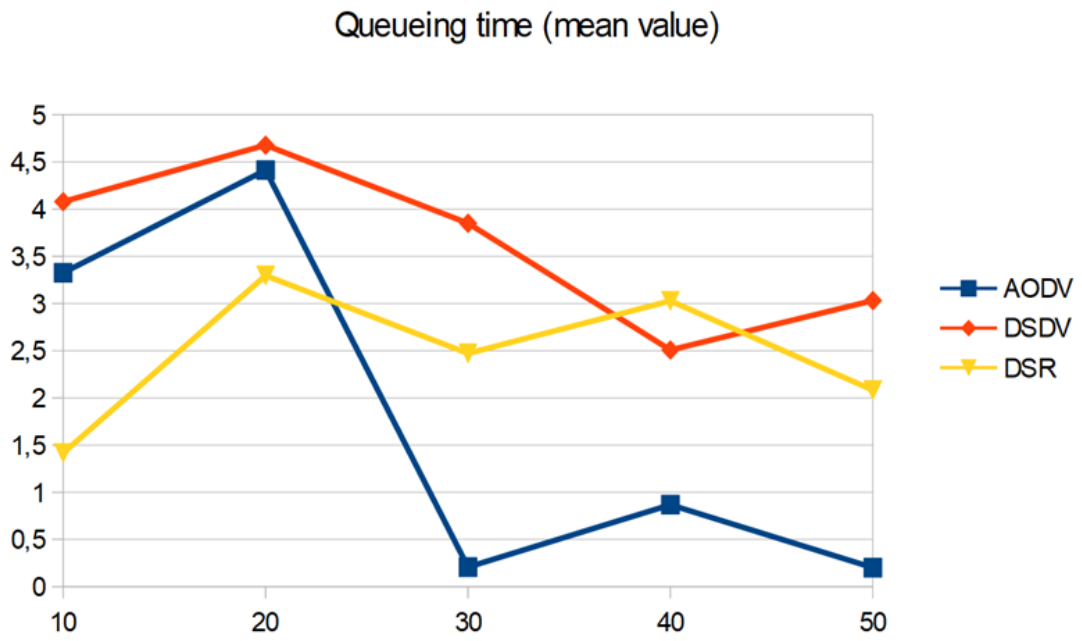


Figure 4.4: The queueing time as the number of nodes increases

Judging from the results above, it is safe to assume that DSDV is the worst routing protocol out of the three in regards to the queuing time, displaying consistently worse results across all the benchmark runs, apart from the run with the 40 nodes, which came a close second to DSR. It is also worth noting that DSR (and DSDV to a lesser extent) were fairly consistent, while the measurements for AODV varied considerably.

4.6 Minimum SINR

In telecommunication engineering, the Signal-to-Interference-plus-Noise-Ratio (SINR) is a means to express the theoretical upper bounds on the capacity of a given channel. In other words, it describes the maximum rate of information transfer. It is defined as the signal power divided by the sum of the interference power, plus the power of background noise. As a result, SINR is used as a means to quantify the quality of a certain wireless connection.

AODV produced null readings across all the simulation runs, so this comparison includes only DSDV and DSR.

Table 4.3: Simulation results

Number of nodes	DSDV	DSR
10	1394,78	654,79
20	1789,634	1874,022
30	5490,644	2198,435
40	3054,015	1503,705
50	1859,160	3955,679

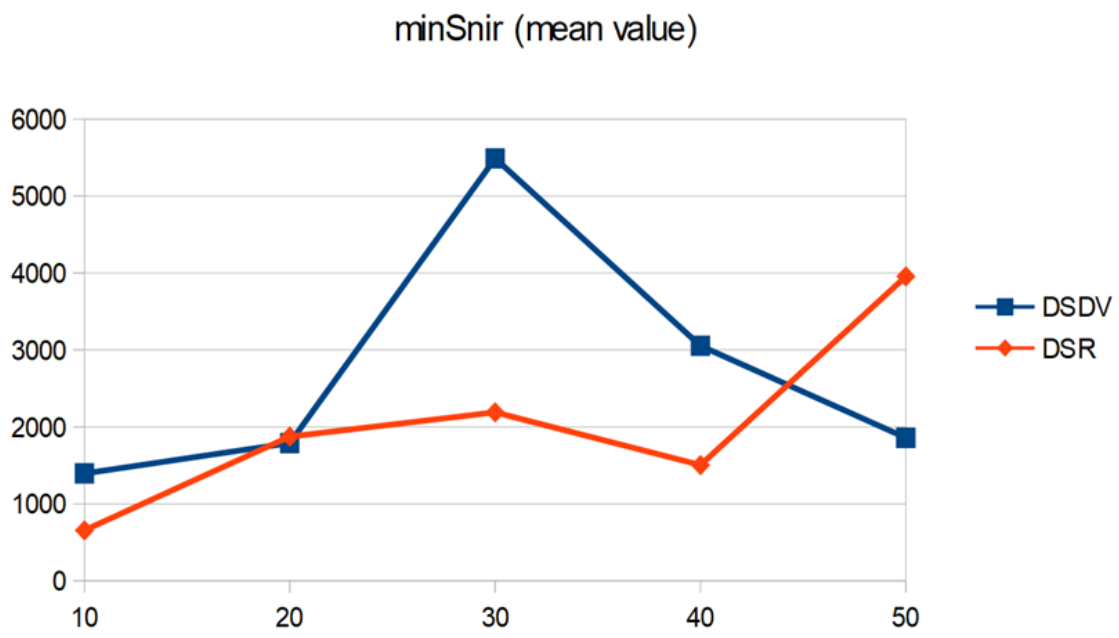


Figure 4.5: The minimum SINR as the number of nodes increases

Judging from the results above, it is safe to assume that DSDV displays a better performance than DSR in regards to the minimum SINR, displaying for the most part better results across the benchmark runs, apart from the run with the 50 nodes, which came second to DSR. However, when DSDV was ahead, it was by a significant margin, like in the run with the 30 nodes for example.

Chapter 5: CMM and SOLAR

Both the Community-based Mobility Model (CMM) and the ORBIT Mobility Model (SOLAR) are synthetic mobility models that simulate the mobility of the nodes in an ad-hoc network. Their basic principle is the same, but they differentiate themselves from one another in the details.

5.1 Presenting CMM

CMM uses as an input an Interaction Matrix, which is the social network that describes the relationship between every node in the network. This is accomplished by setting a value between 0 and 1 in each cell to represent how strong the social ties of each node are with all the others. For example, the element m1,4 represents how strong the social tie between node 1 and node 4 is.

$$\mathbf{M} = \begin{bmatrix} 1 & 0.76 & 0.64 & 0.11 & 0.05 & 0 & 0 & 0.12 & 0.15 & 0 \\ 0.76 & 1 & 0.32 & 0 & 0.67 & 0.13 & 0.23 & 0.45 & 0 & 0.05 \\ 0.64 & 0.32 & 1 & 0.13 & 0.24 & 0 & 0 & 0.15 & 0 & 0 \\ 0.11 & 0 & 0.13 & 1 & 0.54 & 0.83 & 0.57 & 0 & 0 & 0 \\ 0.05 & 0.67 & 0.24 & 0.54 & 1 & 0.2 & 0.41 & 0.2 & 0.23 & 0 \\ 0 & 0.13 & 0 & 0.83 & 0.2 & 1 & 0.69 & 0.15 & 0 & 0 \\ 0 & 0.23 & 0 & 0.57 & 0.41 & 0.69 & 1 & 0.18 & 0 & 0.12 \\ 0.12 & 0.45 & 0.15 & 0 & 0.2 & 0.15 & 0.18 & 1 & 0.84 & 0.61 \\ 0.15 & 0 & 0 & 0 & 0.23 & 0 & 0 & 0.84 & 1 & 0.65 \\ 0 & 0.05 & 0 & 0 & 0 & 0 & 0.12 & 0.61 & 0.65 & 1 \end{bmatrix}$$

Figure 5.1: Example of an Interaction Matrix of 10 nodes

$$\mathbf{C} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Figure 5.2: Example of a Connectivity Matrix of 10 nodes

Later, a threshold value is set which dictates how the Connectivity Matrix will be populated. The process is quite simple. If the value of a cell in the Interaction Matrix is lower than the threshold value, then the corresponding value of the Connectivity Matrix cell will be 0, and 1 otherwise. The final result looks something like Figure 5.2.

After the Connectivity Matrix is populated, the highly connected set of nodes are isolated. This is accomplished by a series of rounds, where in each round, one of the edges of the node with the highest centrality is removed. The number of runs is determined by Q, which measures the proportion of the edges in the network minus the expected value in a network with the same community division but with random connections between them. Its minimum value is 0 (meaning that it is no better than random) and its maximum is 1. The rounds end when Q becomes less than the one in the previous round. The final result is a network which consists of distinguishable groups.

After the communities are defined, they are associated to a specific square location in the simulation area. At this point, the model is established and the social-based attraction laws define how the nodes move. The first final destination is a random point in the same square.

Each square exerts a certain social attractivity for a certain host. This is calculated by the strength of the social ties with the nodes that move towards that particular square. The new final destination is then chosen from inside this square.

As people in real life tend to do, the nodes are periodically assigned to new communities, depending on the time that has passed, similar to how someone could spend for example 8 hours at work and another 8 hours at home with their family. These new communities are then randomly associated to squares in the simulation area. Then, the nodes gradually start moving towards these squares.

5.2 Presenting SOLAR

SOLAR utilizes a partially deterministic orbital movement pattern around specific points in the simulation area, known as hubs. These hubs stem from observing real life, where people tend to routinely spend their time at a few select places, with their movements usually confined within and between these hubs. However, while SOLAR defines these hubs (at random points in the simulation area), it doesn't define a specific schedule or route for the nodes (that's why it is partially deterministic).

It is also efficient to implement, since it doesn't require constant location updates to track the mobile nodes. Even if the exact location of a specific node is unknown, a list of possible places where it might be can be identified. The hub list of each node is also periodically changed, and new ones are given.

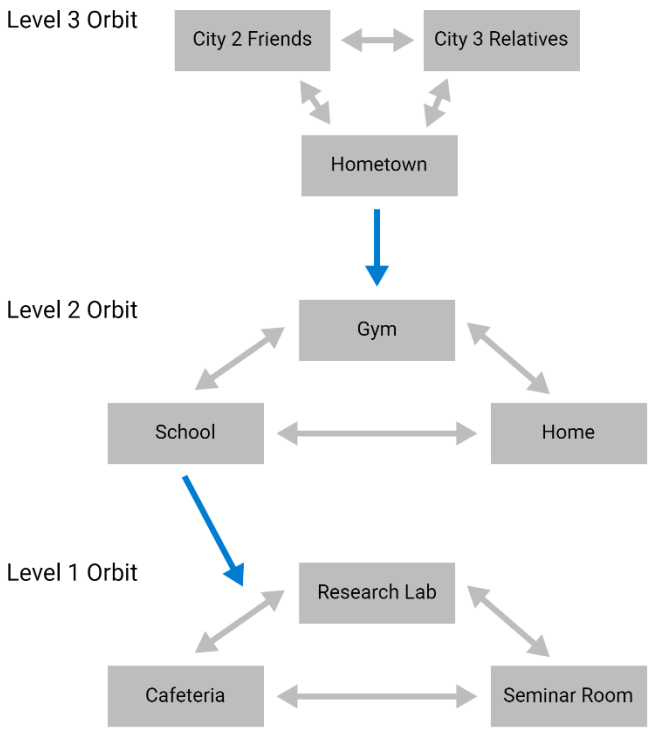


Figure 5.3: Examples of sociological orbits

Generally, the movements can be placed into two categories: movements inside the hubs and movements between the hubs. For the first category, SOLAR uses the Random Waypoint mobility model, which is perhaps the most widely used one. As every node moves around, devices can record the most visited hubs for each node and share this hub-based mobility profile with other trusted nodes, often called acquaintances. As for the second category, SOLAR uses a Point-to-Point Linear movement model, where a point in another hub is chosen and the node starts moving towards it linearly.

Nodes in SOLAR only need to know their own location, in addition to the location of the hubs. Periodically they broadcast this information to facilitate neighbour discovery and the sharing of the hub lists. Routing is accomplished by each intermediate node choosing as the next hop the node from its acquaintances that is the closest to the destination. If the destination's hub list is not known, then a query is sent to the acquaintances of the destination, where they respond with the destination's hub list, if they have it cached. If not, then the query is instead sent to the acquaintances of the source. If the maximum number of hops is reached and the answer is still not available, then either

- The data packet is dropped
- Another query is sent to a different set of hubs
- Flooding of query packets takes place

During the transmission of data, for each packet, both the source and the destination insert into each packet their current hub and hub list, so they can keep track of one another.

An optimization technique that SOLAR uses is the reduction of the hub list to the absolute minimum. During their lifetime, the nodes will make a lot of acquaintances, so it is not wise to store all of them. So, each node only stores all the Prime Acquaintances (the ones where their hub list is not a subset of

any of the other nodes' hub list) and the Essential Prime Acquaintances (the ones where their hub lists cover at least one node not covered by the Prime Acquaintances) that cover the maximum number of "uncovered" nodes until they are all covered.

5.3 Comparing CMM and SOLAR

Since CMM and SOLAR are just node mobility models, they have to be matched to a routing protocol in order to be put to use. The protocol chosen was IEEE's 802.11, which is widely used in ad-hoc networks. The simulator used was OMNeT++. The simulation parameters are presented in the following table:

Table 5.1: Simulation parameters

Simulation duration	86400 seconds (24 hours)
Number of nodes	25
Simulation area	1000m x 1000m
Number of hubs	5
Radio range	200m (default)

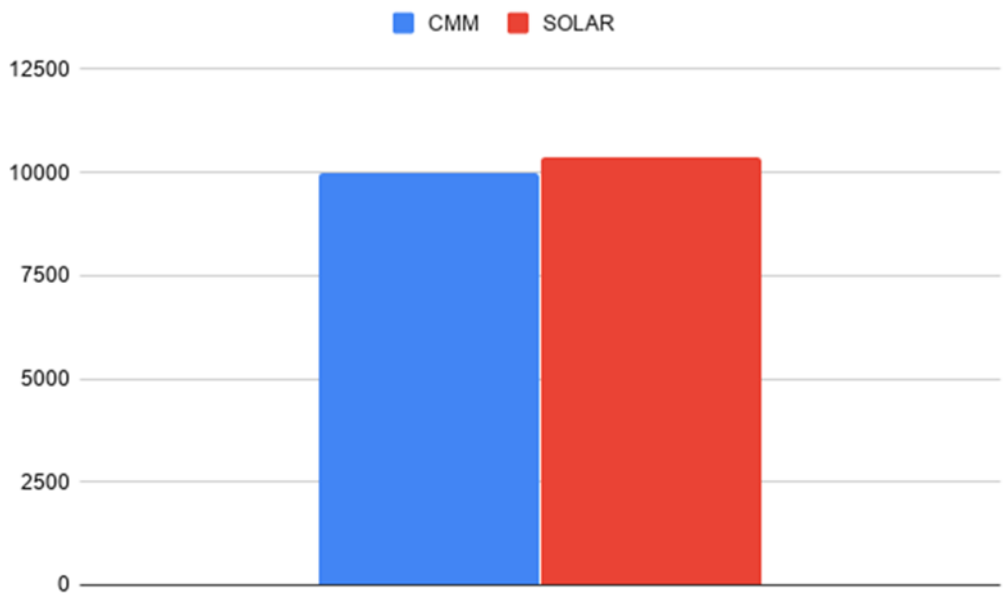


Figure 5.4: The minimum SINR values for CMM and SOLAR

The first measurement to be examined was the minimum Signal-to-Interference-plus-Noise-Ratio. As previously stated, SINR is a means to express the theoretical upper bounds on the capacity of a given channel.

As the graph reveals, the results were very close, with CMM reporting 10000 and SOLAR 10377. The difference between them is 3.7%, a little over what most people would consider margin of error. This means that in the real world, in regards to this aspect at least, both CMM and SOLAR would perform basically identically, with any difference between their minimum SINR barely noticeable.

From the simulation results were also extracted values for their packet error rate. These values were, as before, put in a graph for easier understanding. The results show that CMM reported 1.77%, while SOLAR reported 2.33%. This means that, in this specific scenario at least, SOLAR is 27.3% more likely to deliver a package with errors in comparison to CMM. Having that in mind, we decided to use CMM as the mobility protocol for the custom ad-hoc scenario in the following chapter.

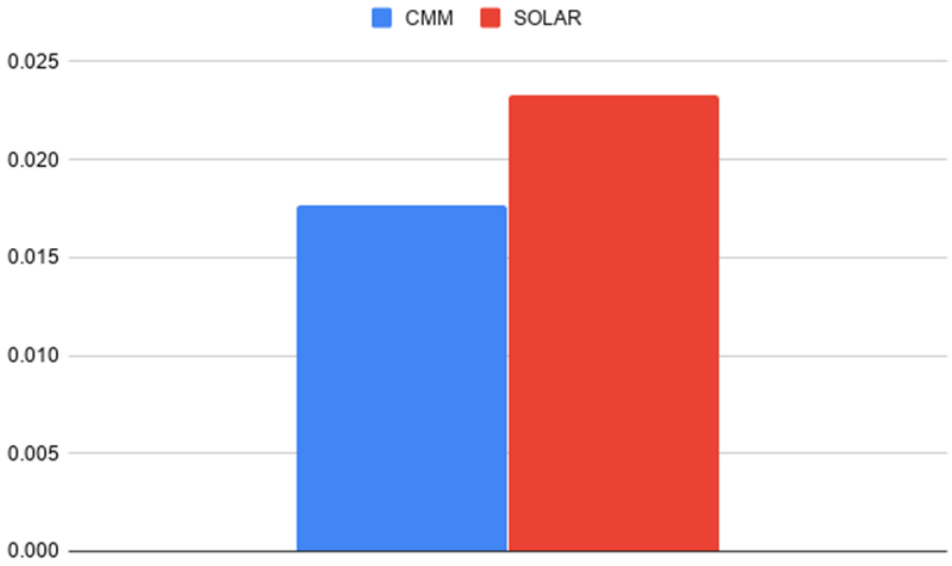


Figure 5.5: The packet error rate values for CMM and SOLAR

Chapter 6: Custom ad-hoc network scenario using CMM

The scenario is the following: We wanted to cover an area of about the same size as the International Hellenic University's campus (850m x 850m, we turned it into a square area for simplicity's sake) using an ad-hoc network, where the nodes in the network are represented by real people, who go about their daily lives. This means that not only do they have to be mobile, but also move in a way that is human-like. Would it be possible for such a network to work, and, if so, what kind of results are to be expected?

Since we are talking about real people, the mobility model had to mimic real-life movement patterns of people, so the standard mobility models included in OMNeT++ were out of the question. This is why we turned our attention to CMM, since it was exactly what we were looking for. The movement patterns generated by CMM mimic to an incredible degree real-life movement pattern. CMM's creators have proved this by comparing their results with IBM's real-life measurements, and they found out that their synthetic results were indeed very close to the real ones, proving that CMM is a viable option to generate realistic movement patterns.

To measure the performance of our network in various scenarios, the simulation was run a total of 36 times, in order to cover every possible combination of the various values for the bitrate (2mbps, 11mbps, 54mbps), the number of nodes (10, 15, 25, 50) and the packet size (128bytes, 768bytes, 1408 bytes). For the bitrate and the packet size, we chose a really low value, the maximum supported value and a value somewhere in the middle. As for the number of nodes, going any higher that 50 (say, 100) resulted in a very dense network, with very high amounts of traffic that was very time-consuming to simulate and at the same time produced so many errors that it was deemed to be not worth implementing.

In order to present the results in an orderly fashion, since there is a lot of data to cover, we will start by studying the packet error rate when the bitrate is 2mbps, 11mbps and 54mbps. At the same time, we will study how the number of nodes affects the packet error rate. After that, we will do the same will the RTT and the throughput.

Table 6.1: Simulation parameters

Area size	850m x 850m
Number of nodes	10, 15, 25, 50
Packet size	124 bytes, 768 bytes, 1408 bytes
Bitrate	2mbps, 11mbps, 54mbps

6.1 The effect of bitrate and number of nodes on the packet error rate

6.1.1 2mbps

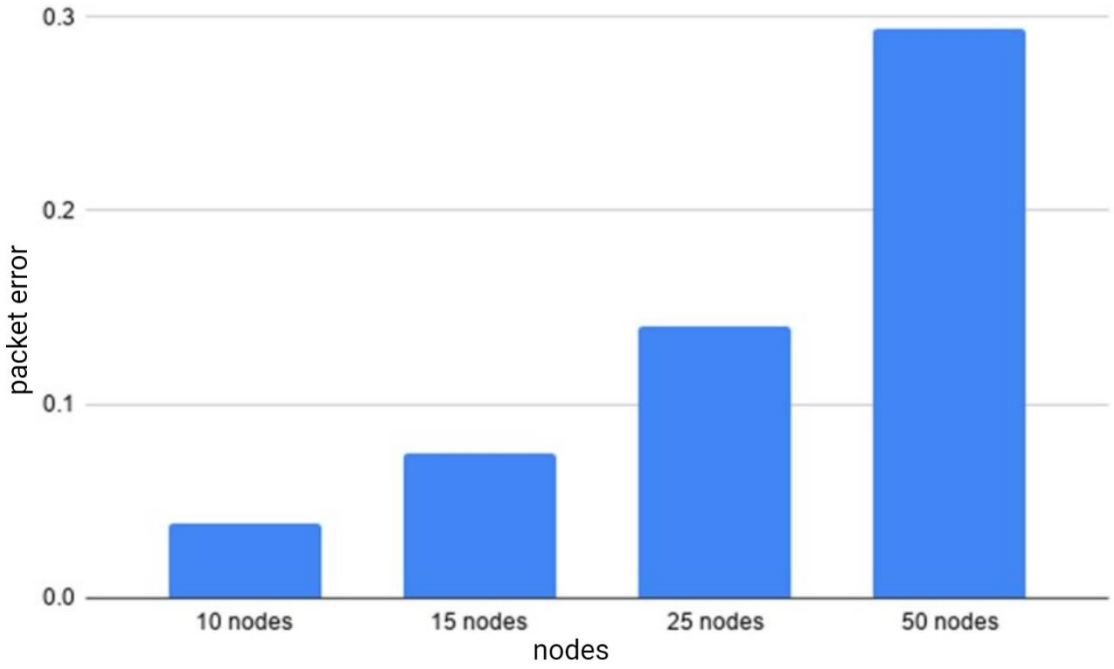


Figure 6.1: 2mbps – 128 bytes – packet error

The first scenario to cover had a bitrate of 2mbps and the packet size was set to 128 bytes. The parameter that changes is the number of nodes. Looking at the above graph which resulted from our simulation, we can very easily see that as the number of nodes increased, so did the packet error rate. This is an expected result, since we know that more nodes in a network can result in more network traffic being generated, and more network traffic being generated equals an increase in the possibility of network collisions, which can lead to an increased packet error rate.

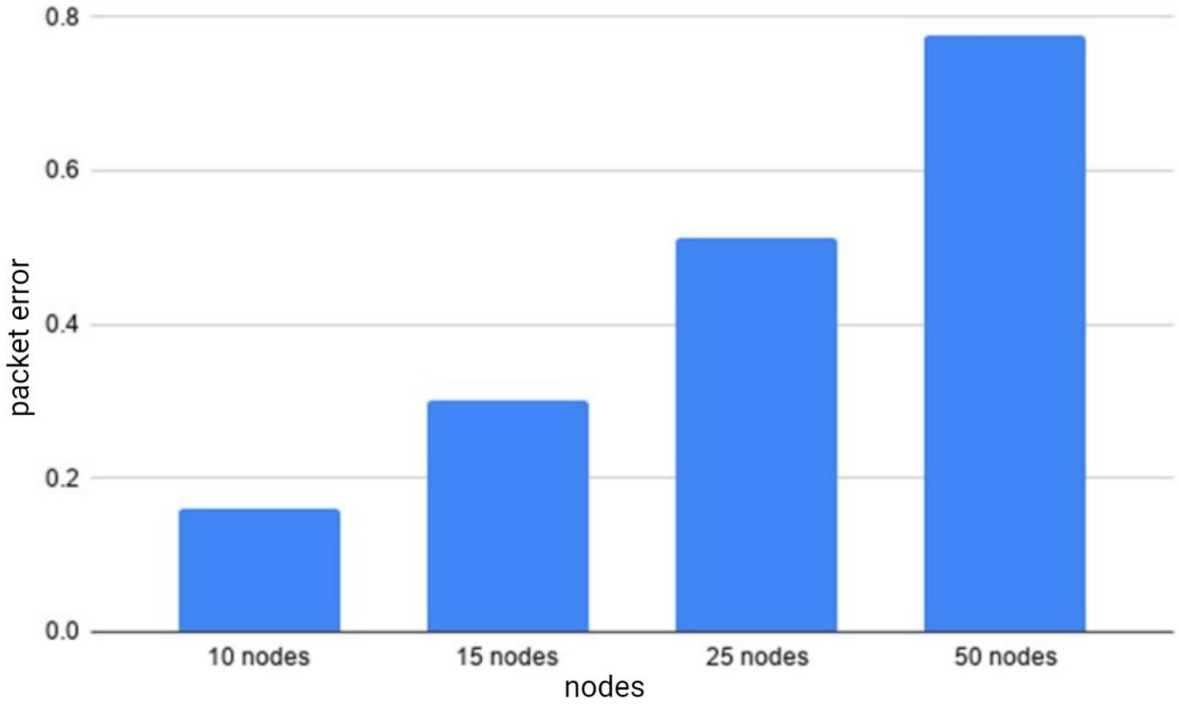


Figure 6.2: 2mbps – 768 bytes – packet error

Increasing the packet size to 768 bytes, we can once again see that more nodes mean a higher packet error rate, which is not surprising. What is perhaps a bit more however, is the fact that the packet error rate is higher that it was when the packet size was 128 bytes. In more detail, in the scenario with the 10 nodes we observed an increase of about 319%, in the scenario with the 15 nodes we observed an increase of about 305%, in the scenario with the 25 nodes we observed an increase of about 264% and in the scenario with the 50 nodes we observed an increase of about 164%. This tells us that the network now has a harder time delivering the packets correctly, no doubt thanks to the fact that the increased packet size puts more of a strain to our network.

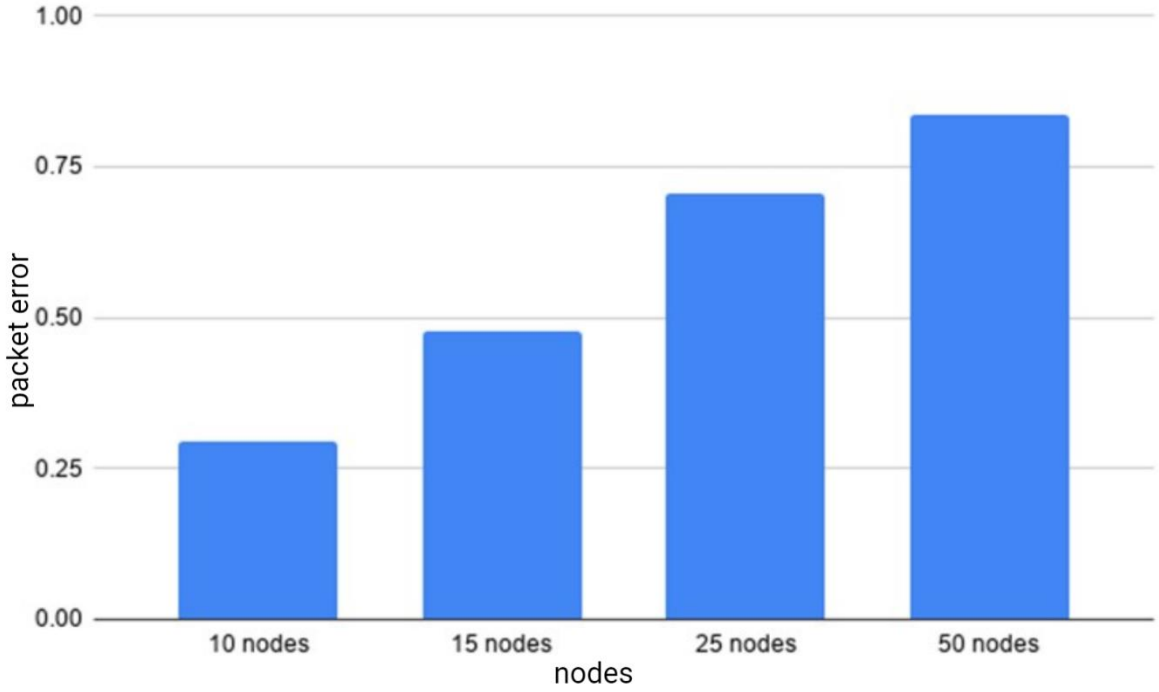


Figure 6.3: 2mbps – 1408 bytes – packet error

Increasing the packet size one final time to 1408 bytes resulted in the above graph. Once again, the higher number of nodes results in a higher packet error rate. The packet error rate was also, once again, higher that it was in the previous scenario where the packet size was 768 bytes. More specifically, we saw an increase of about 84% for the packet size in the scenario with the 10 nodes, an increase of about 59% in the scenario with the 15 nodes, an increase of about 38% in the scenario with the 25 nodes and an increase of about 8% in the scenario with the 50 nodes.

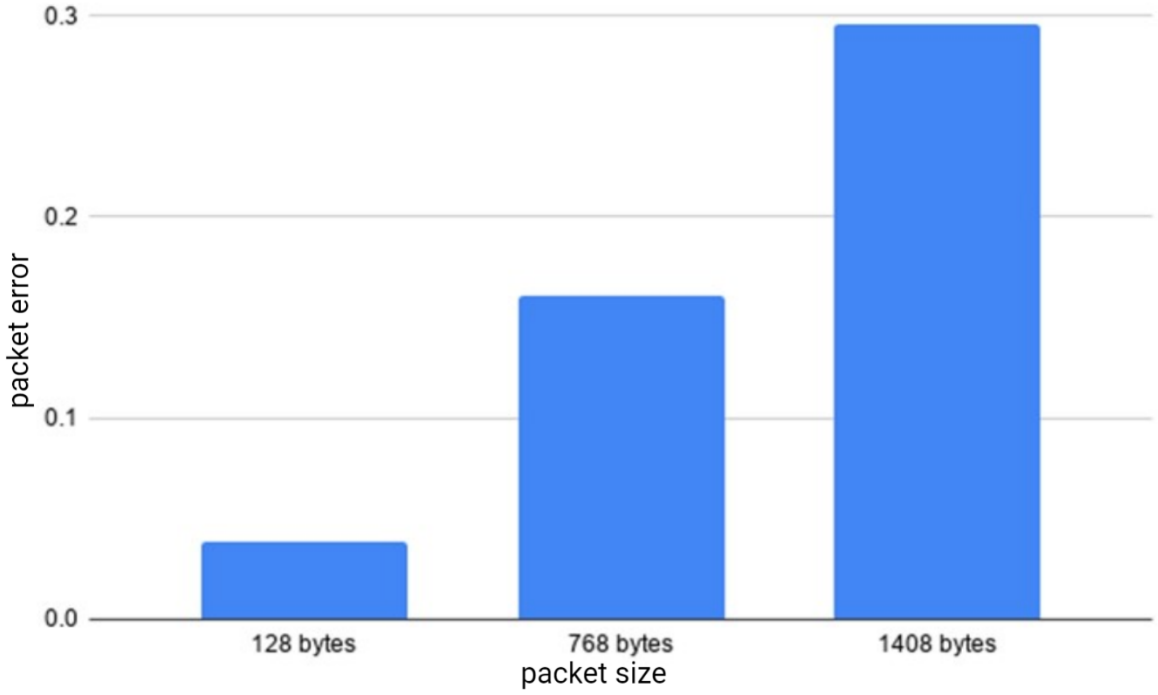


Figure 6.4: 2mbps – 10 nodes – packet error

For the second phase of the simulations, we kept the number of nodes the same and instead changed the values of the packet size. This first graph represents the results we received when we set the number of nodes to 10. As it is pretty obvious, the higher packet sizes result in higher values for the packet error rate. This is normal, since a higher packet size means the network is more congested, and in turn packet collisions are more common.

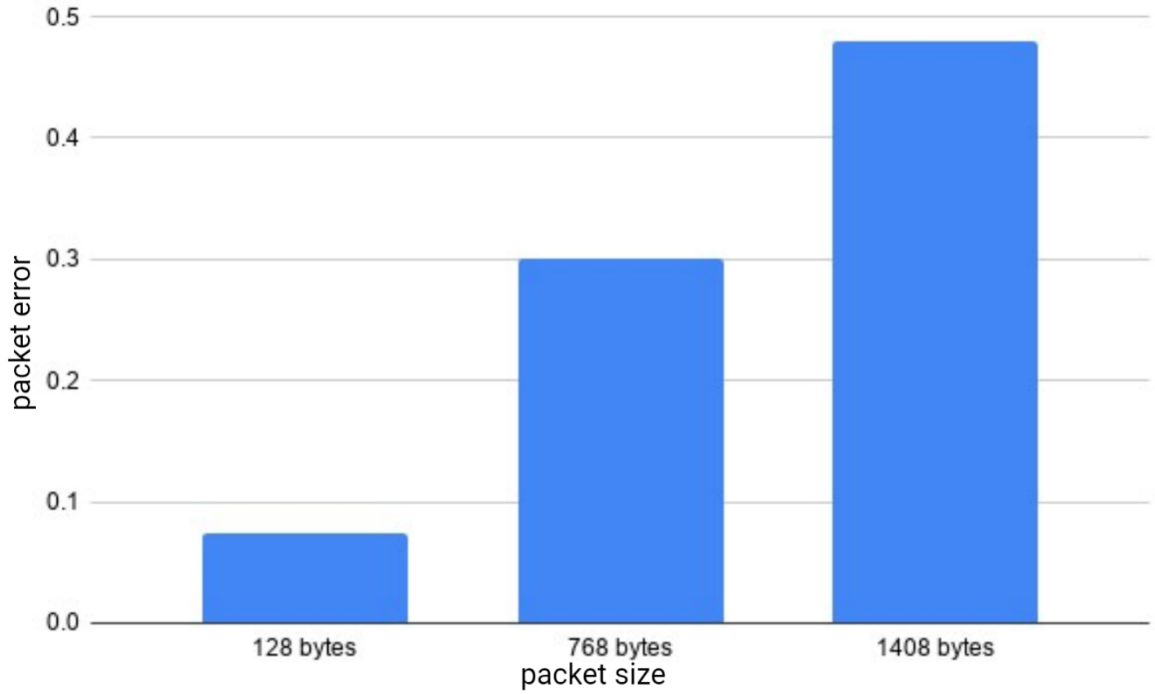


Figure 6.5: 2mbps – 15 nodes – packet error

Increasing the number of nodes to 15, we observed an increase of about 94% in the scenario where the packet size was set to 128 bytes, an increase of about 87% where the packet size was set to 768 bytes and an increase of about 62% where the packet size was set to 1408 bytes. This increase is to be expected, since more nodes in the network means more network traffic is being generated, which can cause congestion, packet collisions, and, as a result, a higher packet error rate.

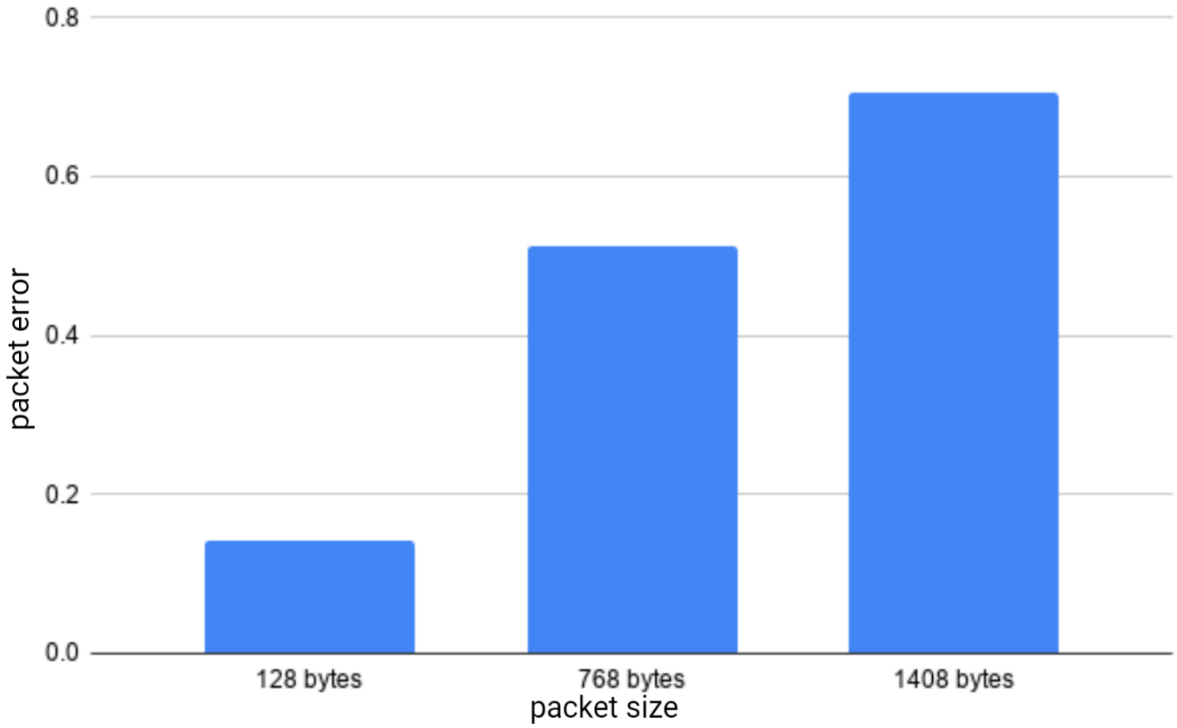


Figure 6.6: 2mbps – 25 nodes – packet error

When the number of nodes was set to 25, we once again saw that the packet error rate had increased. When the packet size was set to 128 bytes the increase was about 89%, when it was set to 768 bytes the increase was about 71% and when it was set to 1408 bytes the increase was about 47%. Again, the scenario with the higher packet error rate is the one with the higher packet size.

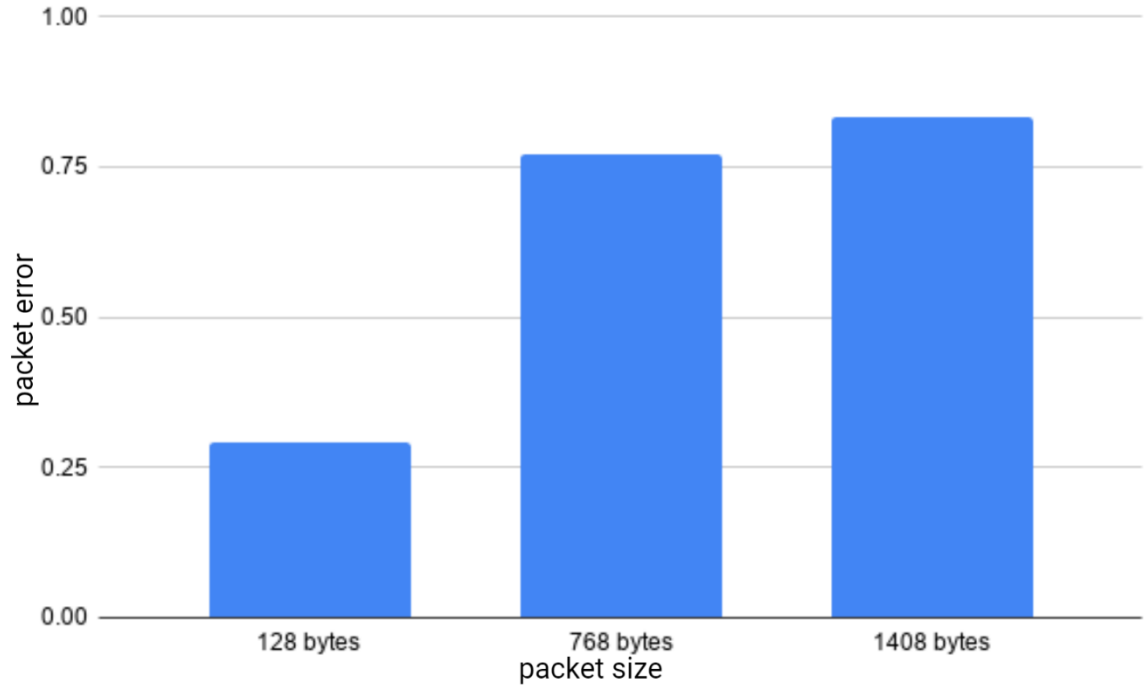


Figure 6.7: 2mbps – 50 nodes – packet error

Finally, setting the number of nodes to 50 netted us the above results. The packet error rate had once again risen, which was expected. In the scenario where the packet size was set to 128 bytes the increase was about 109%, in the scenario where it was set to 768 bytes the increase was about 51% and in the scenario where it was set to 1408 bytes the increase was about 18%.

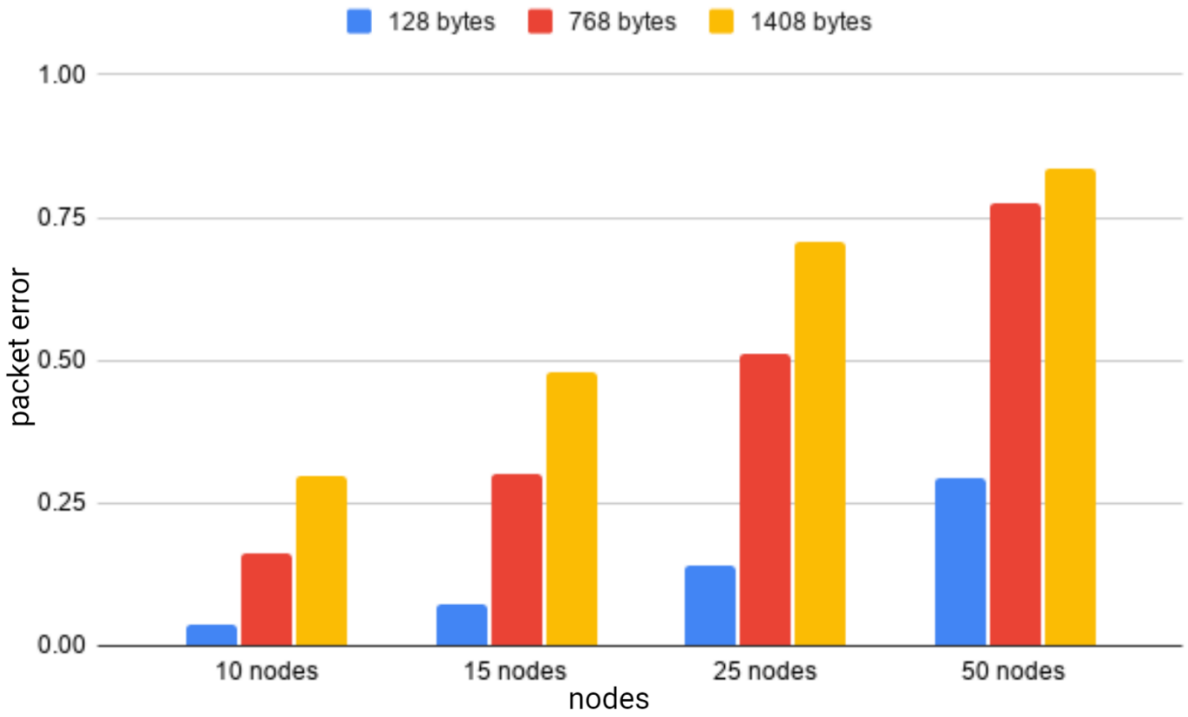


Figure 6.8: Packet error rate - 2mbps

As we can see, the packet error rate increases both when the number of nodes increases, and when the packet size increases. This is an expected behavior, since both the increased number of nodes and the increased packet size led to a greater number of incorrectly received packets, thanks, in part at least, to the increased number of collisions caused by the increase in the network congestion. As expected, the worst possible combination for the packet error rate is the highest number of nodes and the biggest packet size, although the scenario with the highest number of nodes and a packet size of 768 bytes came fairly close.

One thing worth pointing out is that the packet error rate never exceeded or even reached 1%, meaning that it is within the limits we set for usability. To be exact, the highest value we measured was about 0.84%.

6.1.2 11mbps

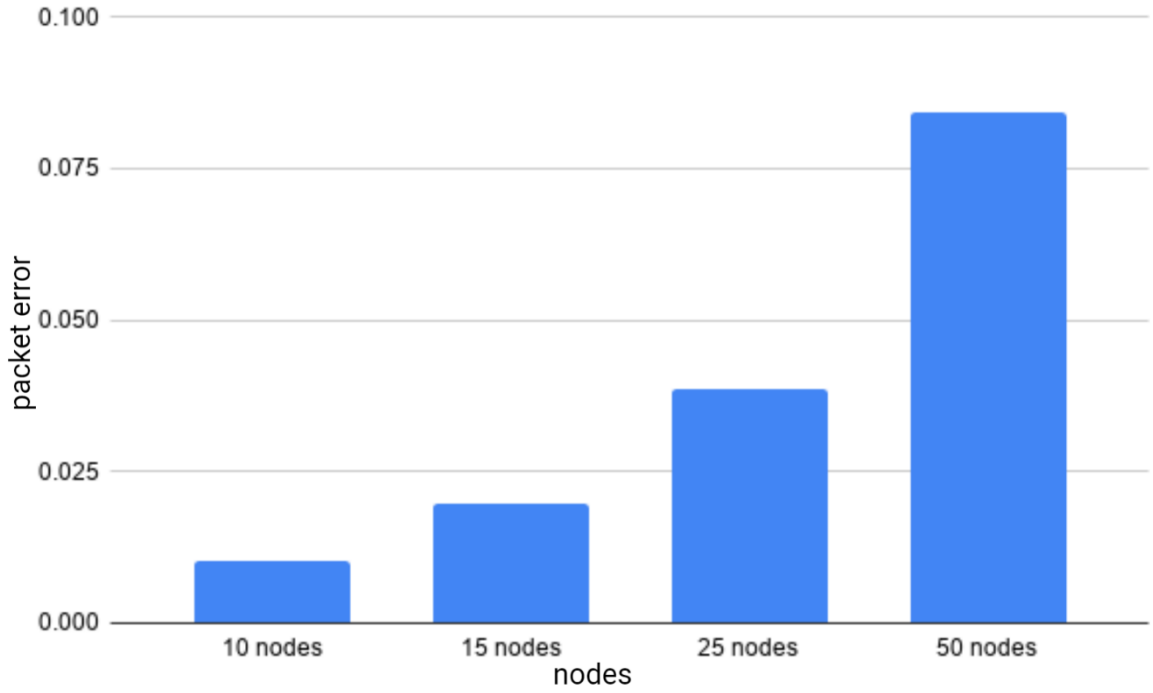


Figure 6.9: 11mbps - 128 bytes - packet error

We have already established that a higher number of nodes can lead to a higher packet error due to the network congestion they can cause, so the results are as expected. What would be interesting however is to study the changes in our simulation results brought on by the increase of the bitrate from 2mbps to 11mbps. In detail, in the scenario with the 10 nodes, the packet error rate decreased by about 74%, in the scenario with the 15 nodes by about 73%, in the scenario with the 25 nodes by about 73% and in the scenario with the 50 nodes by about 71%. This shows us that the increase in the bitrate resulted in less packets being received incorrectly, which makes sense since the network is now less prone to congestion.

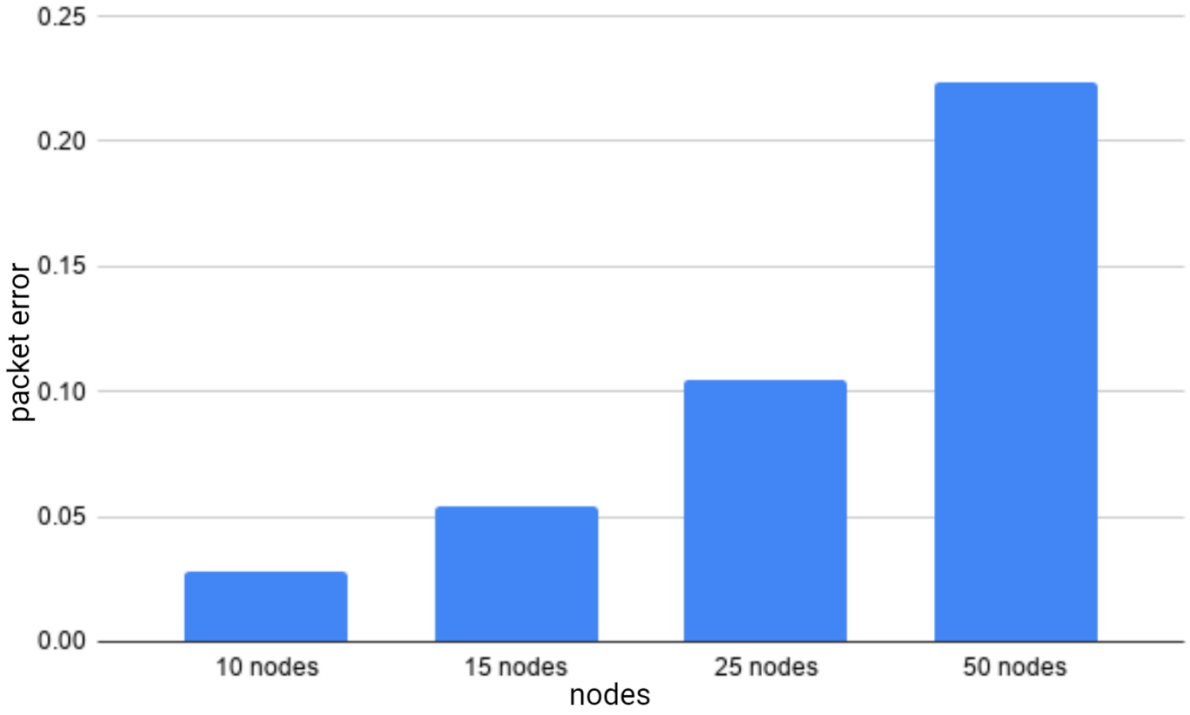


Figure 6.10: 11mbps - 768 bytes - packet error

Increasing the packet size to 768 bytes netted us the following results: In the scenario with the 10 nodes, we observed an increase of about 181%, in the one with the 15 nodes an increase of about 173%, in the one with the 25 nodes an increase of about 170% and in the one with the 50 nodes an increase of about 165%. Comparing the results with the scenario where the bitrate was 2mbps, we saw a decrease of about 82% in the scenario with the 10 nodes, about 82% in the scenario with the 15 nodes, about 80% in the scenario with the 25 nodes and about 71% in the scenario with the 50 nodes.

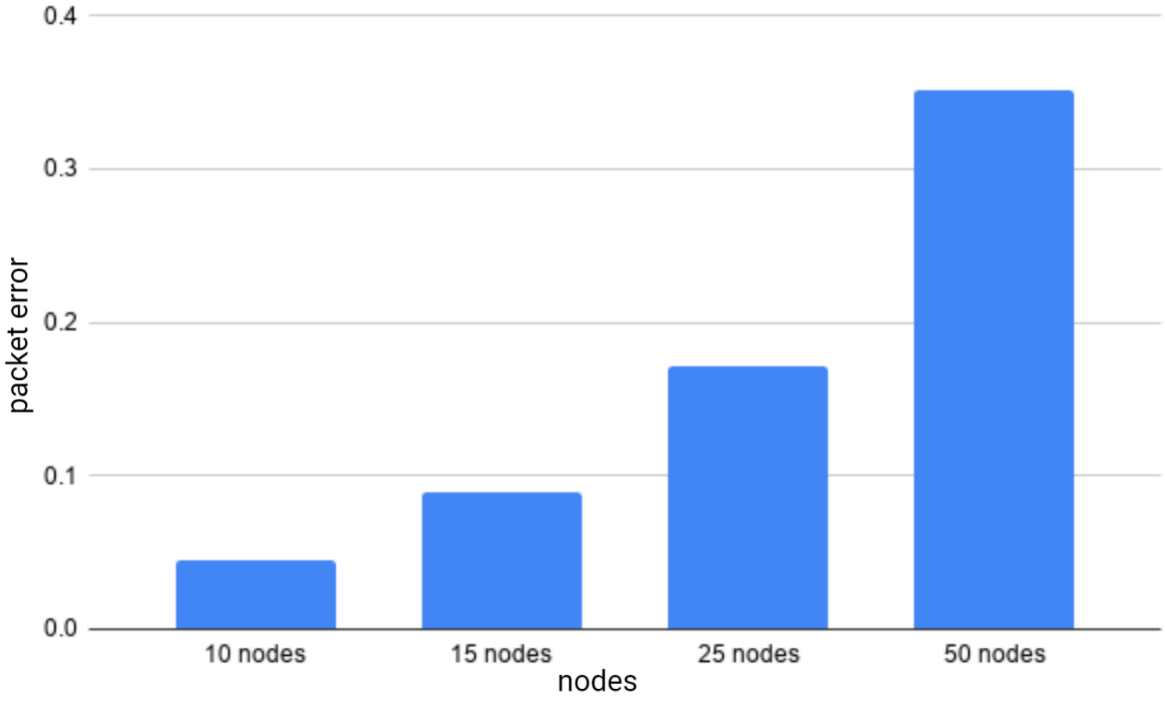


Figure 6.11: 11mbps - 1408 bytes - packet error

Increasing the packet size one last time to 1408 bytes, we observed an increase of about 59% in the scenario with the 10 nodes, about 66% in the scenario with the 15 nodes, about 64% in the scenario with the 25 nodes and about 57% in the scenario with the 50 nodes. Now, comparing with the scenario where the bitrate was 2mbps, we saw a decrease in the packet error rate of about 85%, about 81% in the scenario with the 15 nodes, about 76% in the scenario with the 25 nodes and about 58% in the scenario with the 50 nodes. We can clearly see that the extra bitrate really helps our network cope much better in the scenarios that involve high amounts of traffic due to the increase in the number of nodes and/or the packet size.

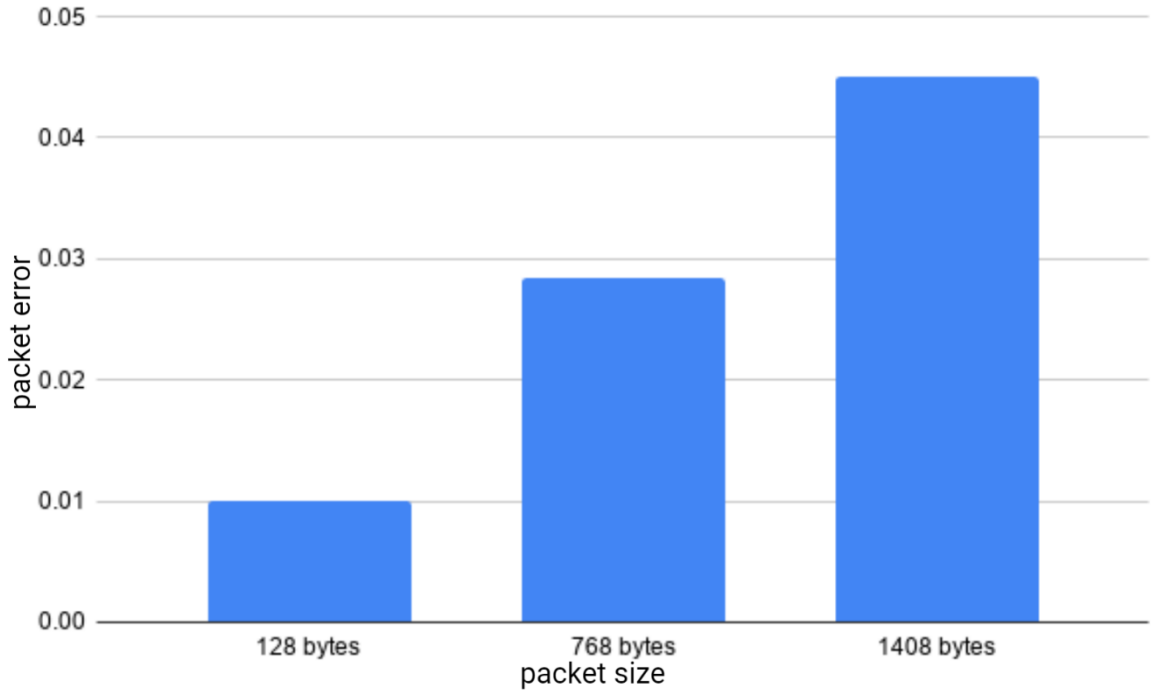


Figure 6.12: 11mbps - 10 nodes - packet error

This time, we are keeping the number of nodes the same and instead change the values of the packet size. This first graph represents the results we received when we set the number of nodes to 10. The higher packet sizes result in higher values for the packet error rate, which is normal, since a higher packet size means the network is more congested, and in turn packet collisions are more common. Comparing these results to the ones where the bitrate was 2mbps, we saw about a 74% decrease in the packet error rate when the packet size was 128 bytes, about 82% when the packet size was 768 bytes and about 85% when the packet size was 1408 bytes. This means that the increase in the bitrate was advantageous.

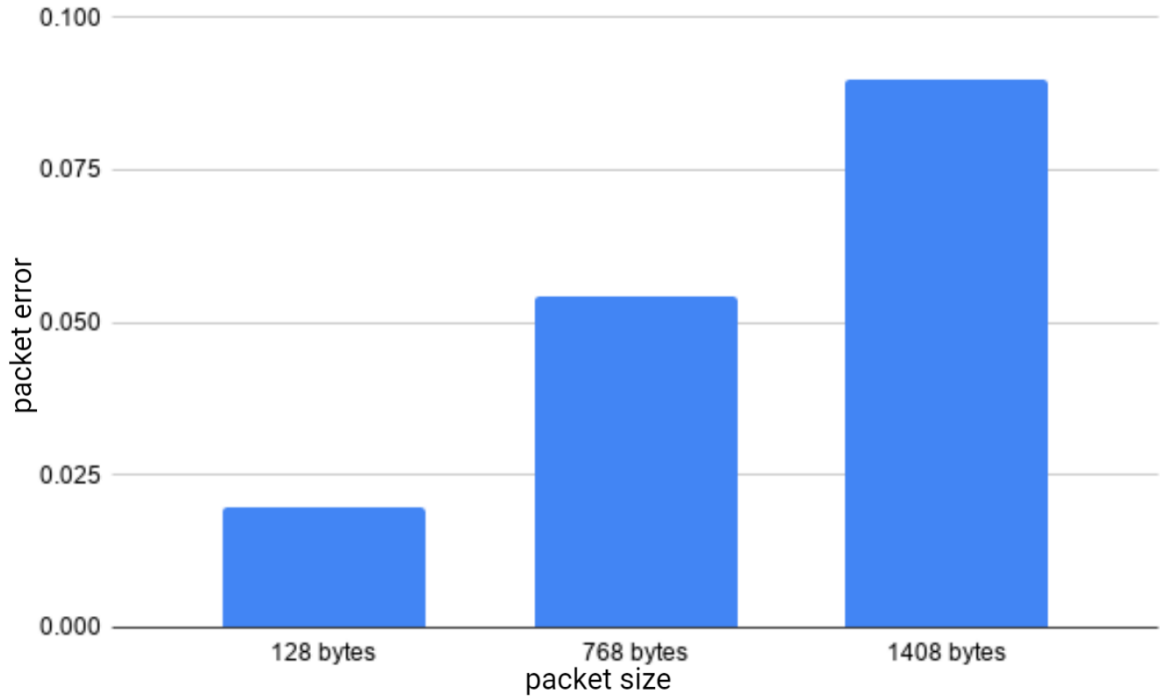


Figure 6.13: 11mbps - 15 nodes - packet error

Increasing the number of nodes from 10 to 15, we documented a 96% increase in the packet error rate in the scenario where the packet size is 128 bytes, a 90% increase when the packet size is 768 bytes and a 99% increase when the packet size is 1408 bytes. The jump from 2mbps to 11mbps also resulted in some big changes, when the packet size is 128 bytes, we saw a 73% decrease, when it is 768, we saw an 82% decrease and when it is 1408 bytes, we saw an 81% decrease.

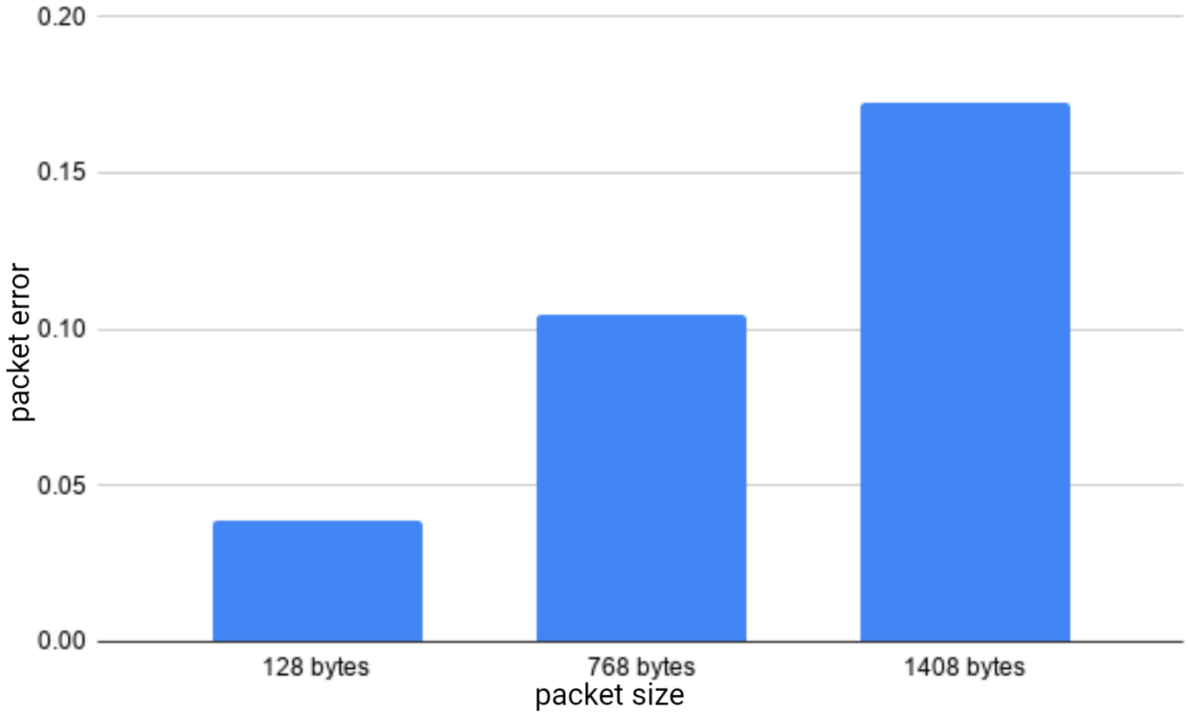


Figure 6.14: 11mbps - 25 nodes - packet error

Increasing the number of nodes from 15 to 25, we documented a 95% increase in the packet error rate in the scenario where the packet size is 128 bytes, a 93% increase when the packet size is 768 bytes and a 91% increase when the packet size is 1408 bytes. The change from 2mbps to 11mbps also brought some changes, when the packet size is 128 bytes, we saw a 73% decrease, when it is 768, we saw an 80% decrease and when it is 1408 bytes, we saw a 76% decrease.

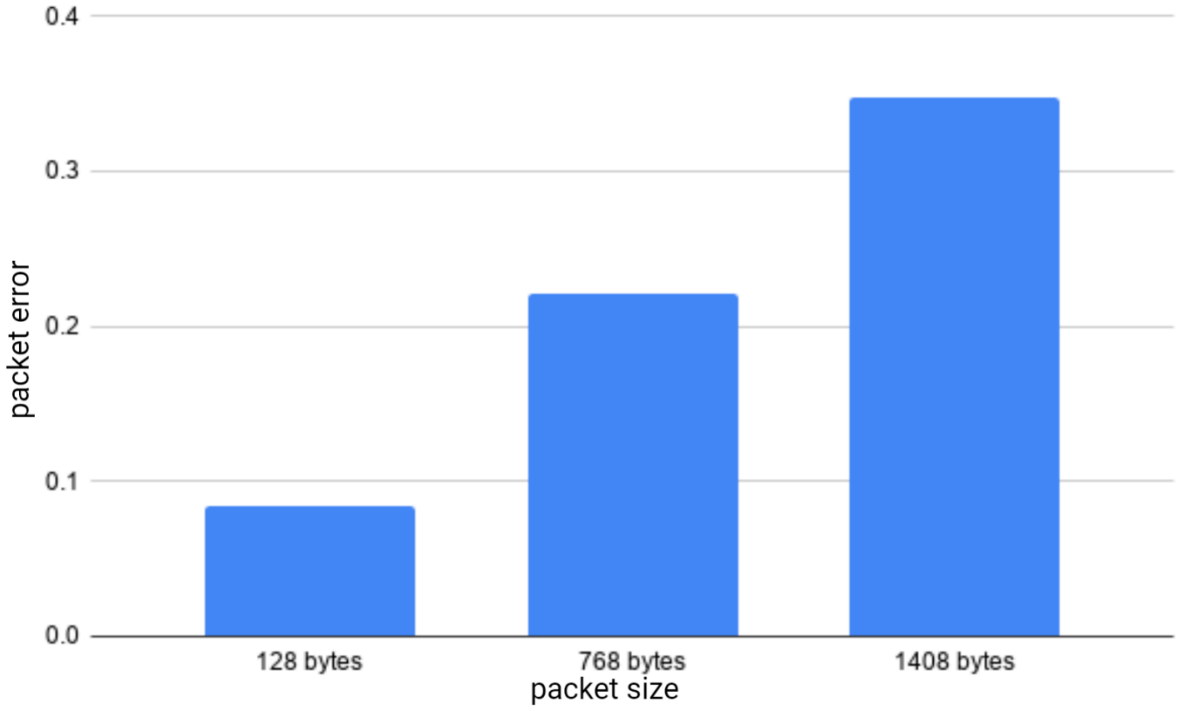


Figure 6.15: 11mbps - 50 nodes - packet error

Increasing the number of nodes one final time from 25 to 50, we documented a 118% increase in the packet error rate in the scenario where the packet size is 128 bytes, a 114% increase when the packet size is 768 bytes and a 104% increase when the packet size is 1408 bytes. The change from 2mbps to 11mbps also brought some changes, when the packet size is 128 bytes, we saw a 71% decrease, when it is 768, we saw again an 71% decrease and when it is 1408 bytes, we saw a 58% decrease.

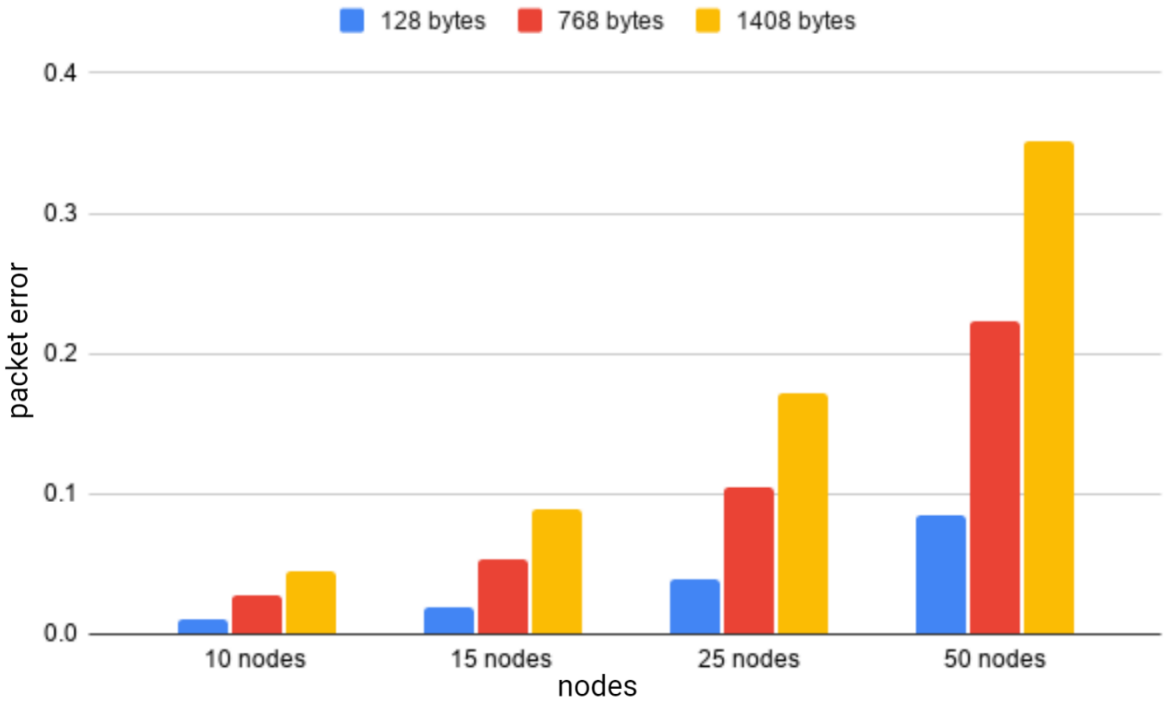


Figure 6.16: Packet error rate - 11mbps

Increasing the bitrate to 11mbps we can see that the packet error rate has decreased to a great extent, across all the various combinations. This is because the higher bitrate has helped deal with network congestion, so packets are more rarely received incorrectly. Still, as expected, the worst value for the packet error rate is in the scenario with the 50 nodes and the highest packet size, while the lowest value is in the scenario with the 10 nodes and the smallest packet size.

6.1.3 54mbps

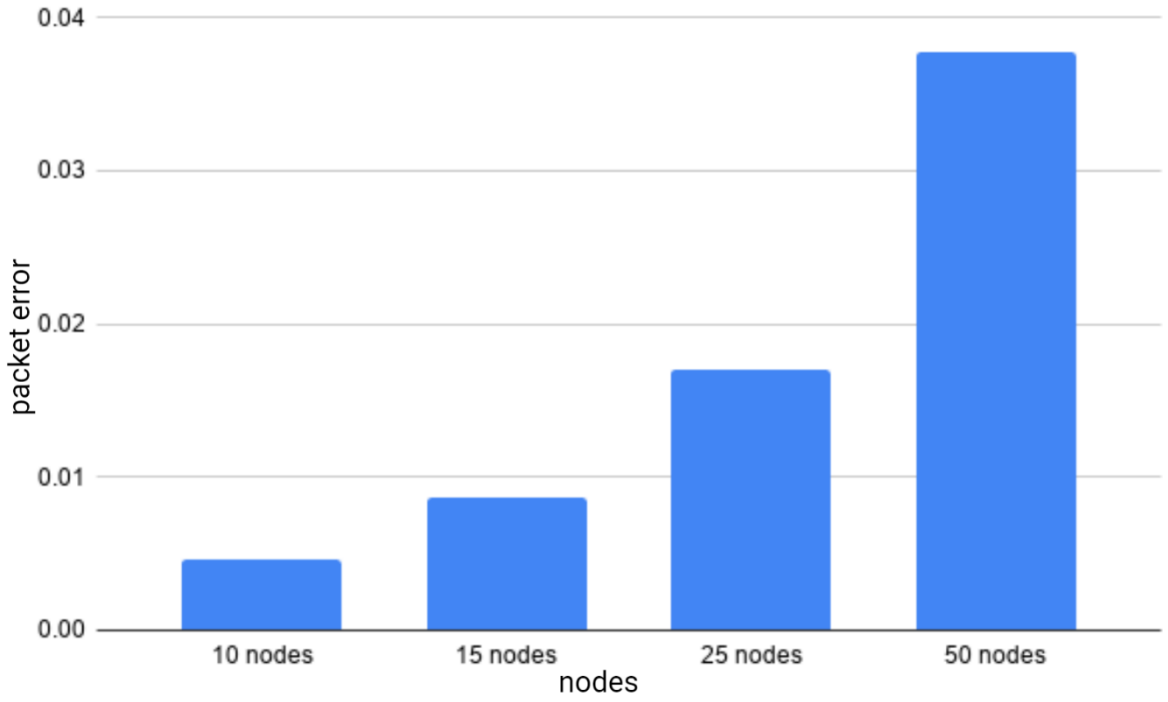


Figure 6.17: 54mbps – 128 bytes - packet error rate

For the final set of simulations, we set the bitrate to 54mbps. The increase from 11mbps to 54mbps resulted in a 54% decrease in the packet error rate when the nodes were set to 10, in a 56% decrease when the nodes were 15, in a 56% decrease when the nodes were 25 and in a 55% decrease when the nodes were 50.

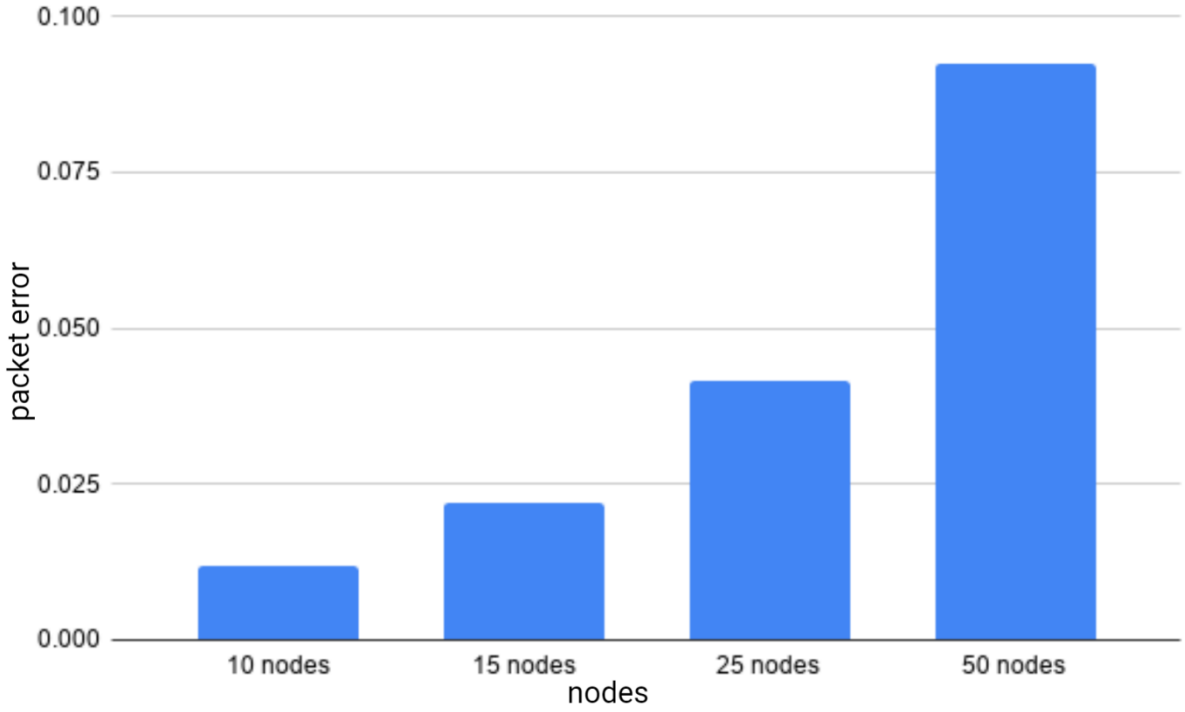


Figure 6.18: 54mbps – 768 bytes - packet error rate

Increasing the packet size to 768 bytes resulted in a 152% increase in the packet error rate when the nodes were set to 10, in a 154% increase when the nodes were 15, in a 145% increase when the nodes were 25 and in a 145% increase when the nodes were 50. As we have already established, the increased packet size negatively affects our packet error rate through increased network traffic resulting in more collisions. The increase in the bitrate however is expected to give the opposite results, since it allows the network to transfer more data without encountering so many problems. Indeed, jumping to 54mbps we observed a 59% decrease in the packet error rate when the nodes were 10, a 59% decrease when the nodes were 15, a 60% decrease when the nodes were 25 and a 59% decrease when they were 50.

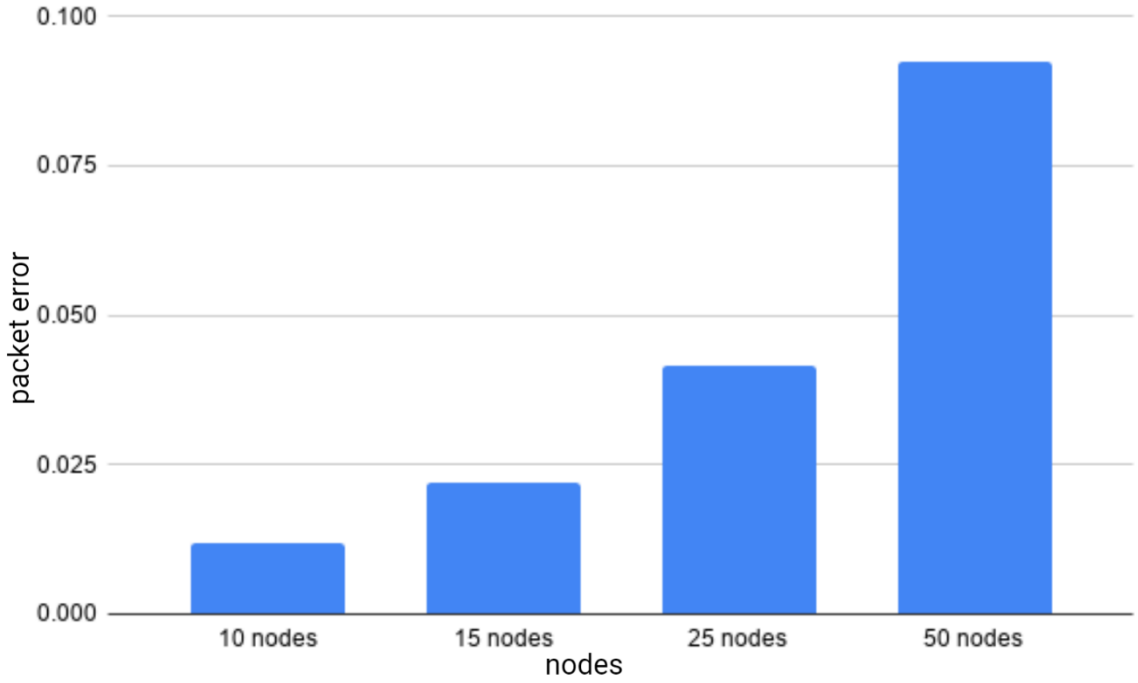


Figure 6.19: 54mbps – 1408 bytes - packet error rate

Increasing the packet size to 1408 bytes resulted in a 53% increase in the packet error rate when the nodes were set to 10, in a 60% increase when the nodes were 15, in a 62% increase when the nodes were 25 and in a 57% increase when the nodes were 50. Jumping to 54mbps we observed a 60% decrease in the packet error rate when the nodes were 10, a 61% decrease when the nodes were 15, a 61% decrease when the nodes were 25 and a 59% decrease when they were 50. In other words, almost identical results with the previous graph.

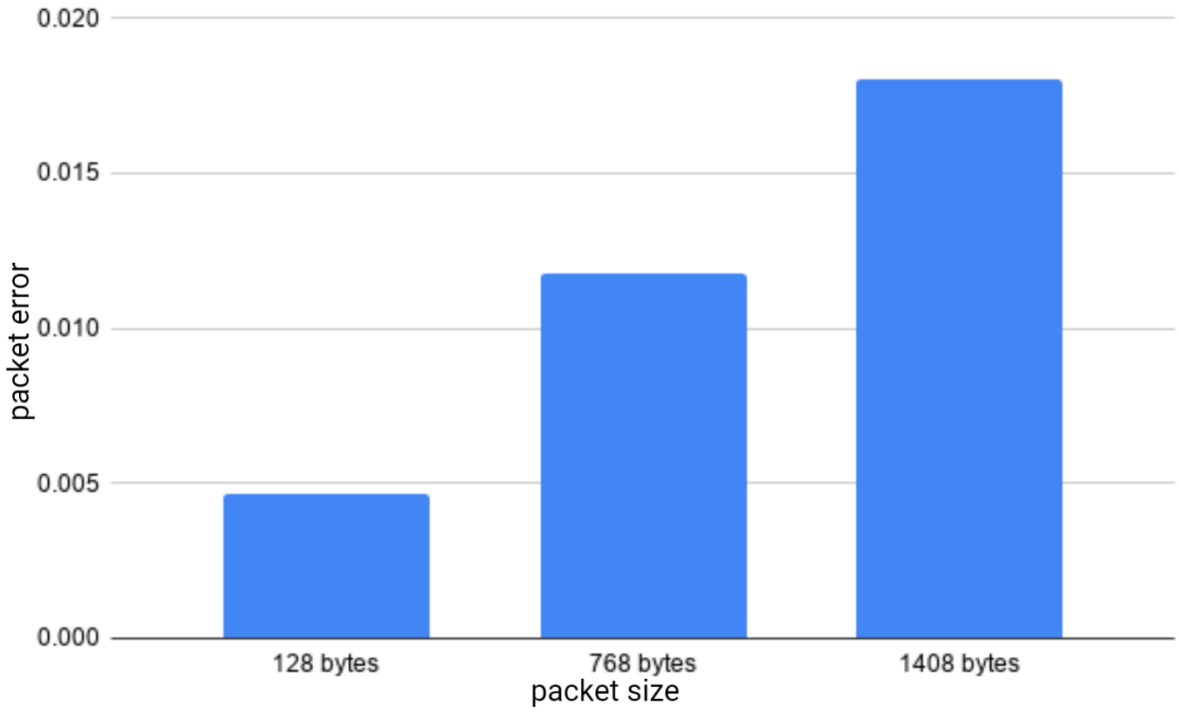


Figure 6.20: 54mbps – 10 nodes - packet error rate

Now changing the packet size instead of the number of nodes and comparing the results between the 11mbps and the 54mbps bitrate, we saw a 54% decrease in the packet error rate when the packet size was set to 128 bytes, a 59% decrease when the packet size was set to 768 bytes and a 60% decrease when the packet size was set to 1408 bytes.

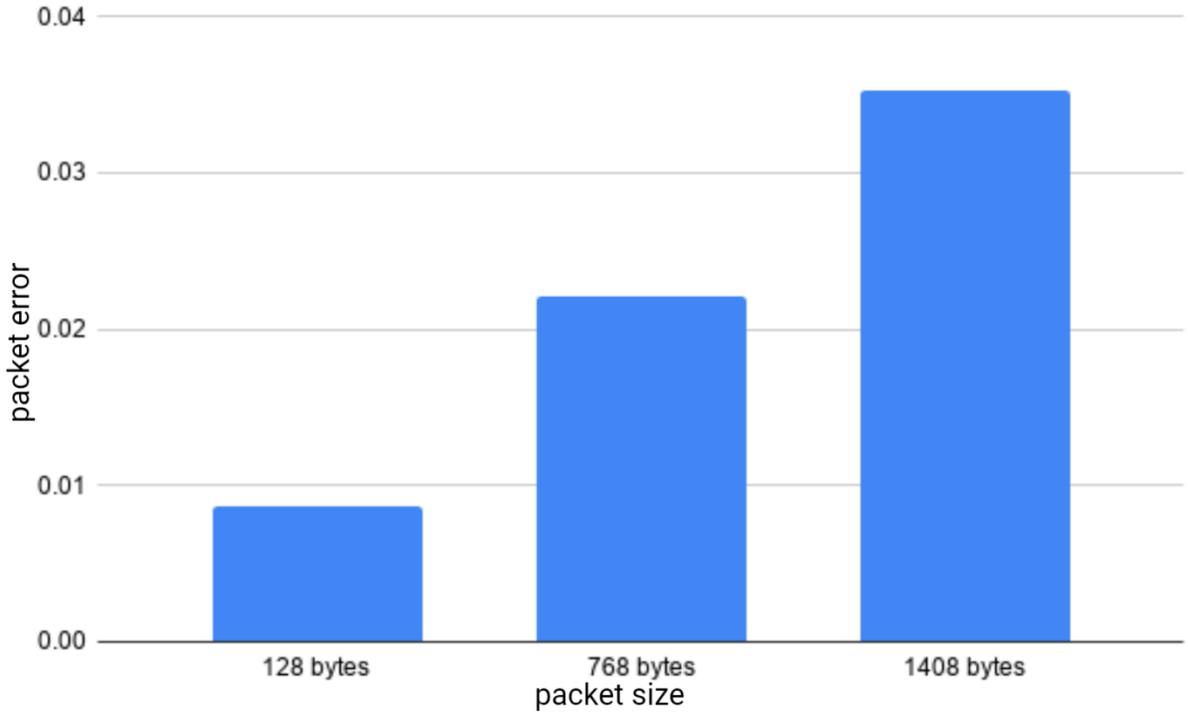


Figure 6.21: 54mbps – 15 nodes - packet error rate

Changing the number of nodes to 15, we saw a 85% increase in the packet error rate when the packet size was set to 128 bytes, an 87% increase when the packet size was set to 768 bytes and a 96% increase when the packet size was set to 1408 bytes. The jump from 11mbps to 54mbps decreased the packet error rate by 56%, 59% and 61% respectively.

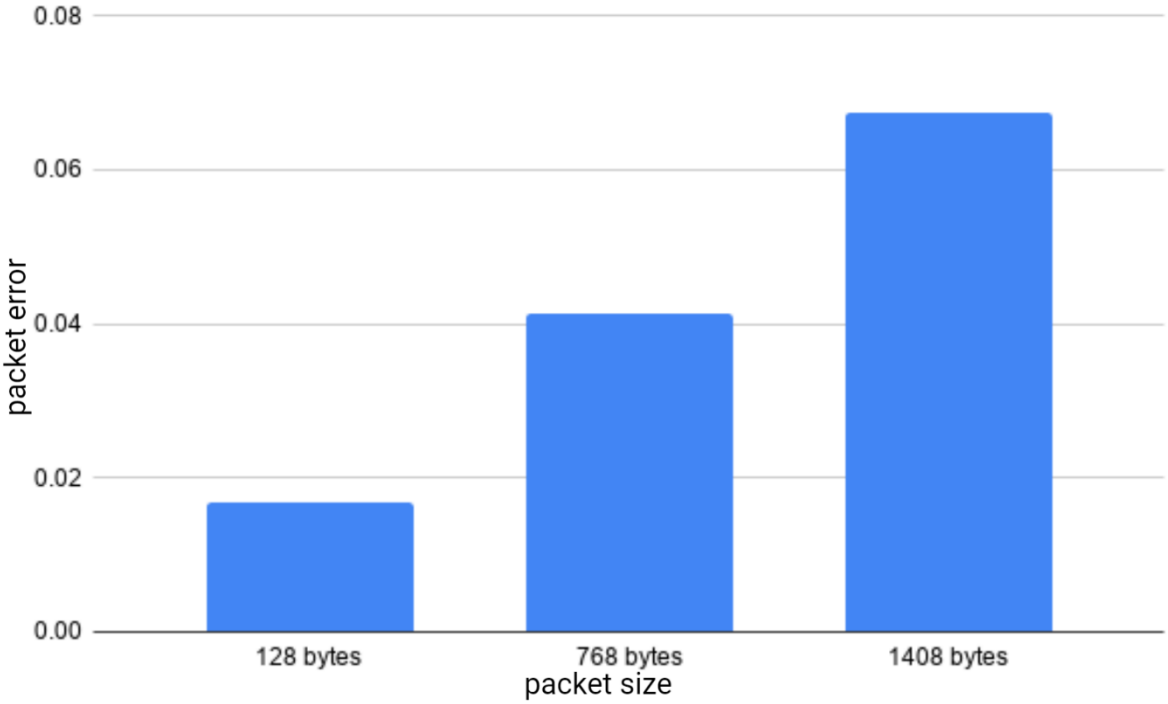


Figure 6.22: 54mbps – 25 nodes - packet error rate

Changing the number of nodes to 25, we saw a 95% increase in the packet error rate when the packet size was set to 128 bytes, an 88% increase when the packet size was set to 768 bytes and a 91% increase when the packet size was set to 1408 bytes. The jump from 11mbps to 54mbps decreased the packet error rate by 56%, 60% and 61% respectively, matching almost exactly the results we gathered from the previous graph.

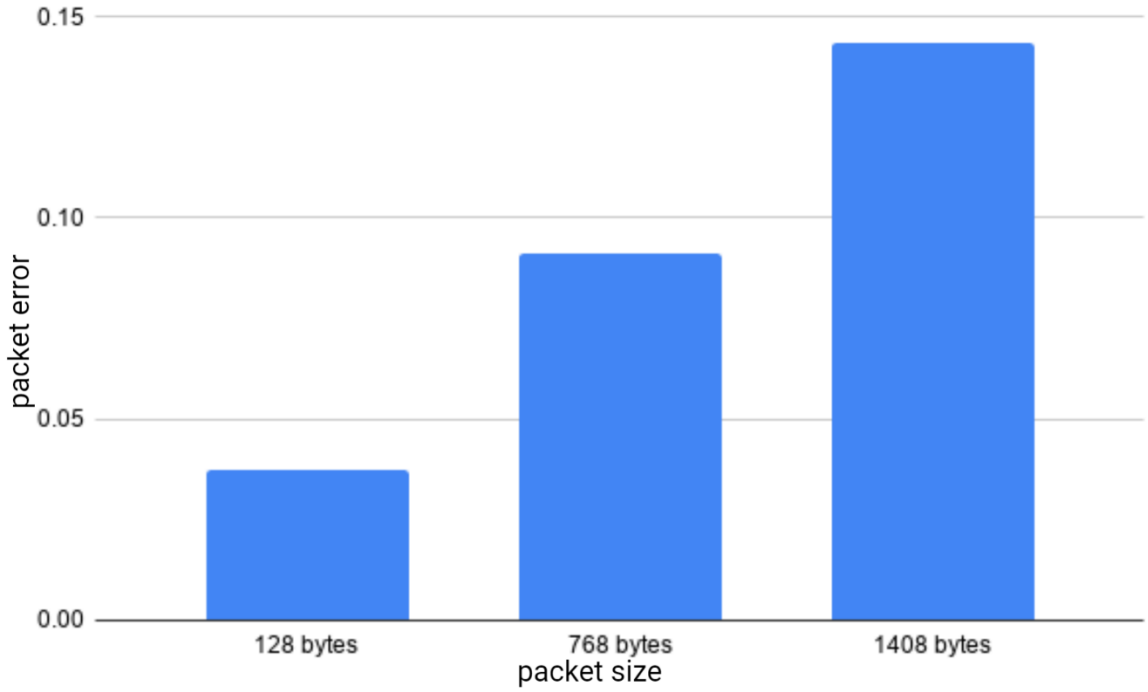


Figure 6.23: 54mbps – 50 nodes - packet error rate

Finally, changing the number of nodes to 50, we saw a 122% increase in the packet error rate when the packet size was set to 128 bytes, a 123% increase when the packet size was set to 768 bytes and a 115% increase when the packet size was set to 1408 bytes. The jump from 11mbps to 54mbps decreased the packet error rate by 55%, 59% and again 59% respectively, matching once again very closely the results we gathered from the previous two graphs.

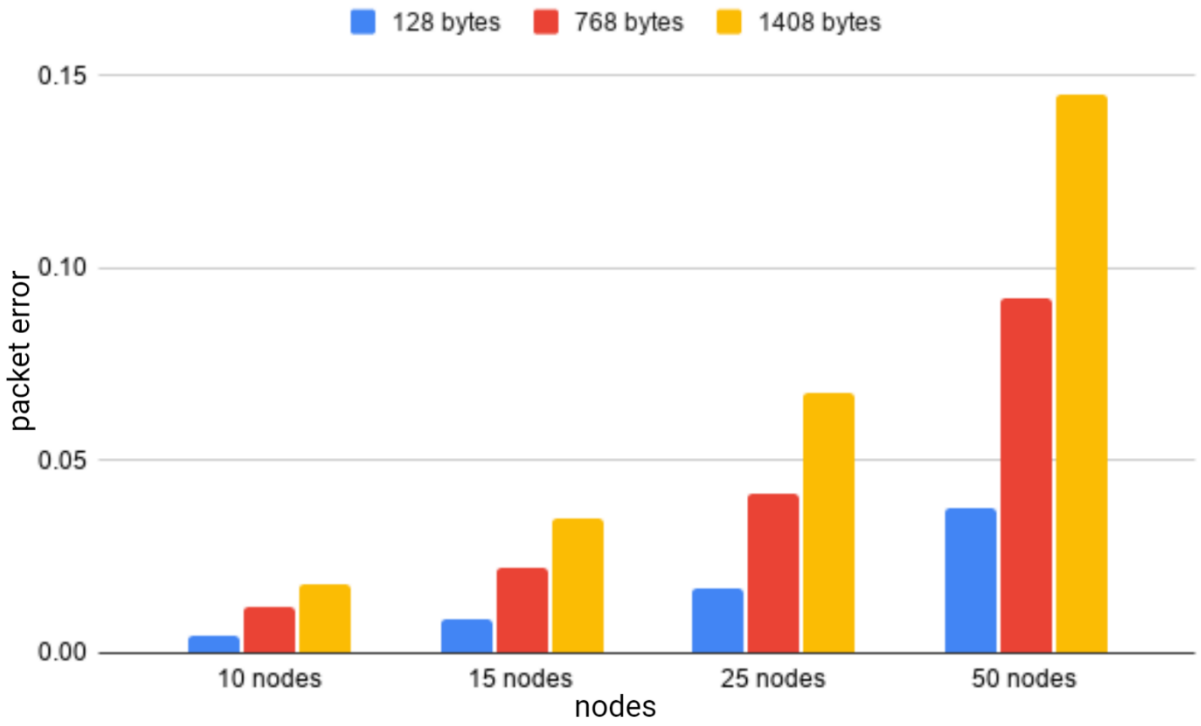


Figure 6.24: Packet error rate - 54mbps

Focusing our attention to the packet error rate, it is quite clear that the higher bitrate helps improve our results compared to when the bitrate was 11mbps. For example, in the worst configuration out of them all for the packet error rate (50 nodes, 1408 bytes packet size), the result decreased from about 0.351 to about 0.145, a decrease of a little less than 58.69%! In all the other, less “extreme” configurations, the packet error rate was about half of what it was when the bitrate was set to 11mbps. We also do not see any spikes in the graph, like we saw when the bitrate was 11mbps, which signals to us that the bitrate is more than enough, and increasing it any further will likely not yield us better results.

6.2 The effect of bitrate and number of nodes on the RTT

6.2.1 2mbps

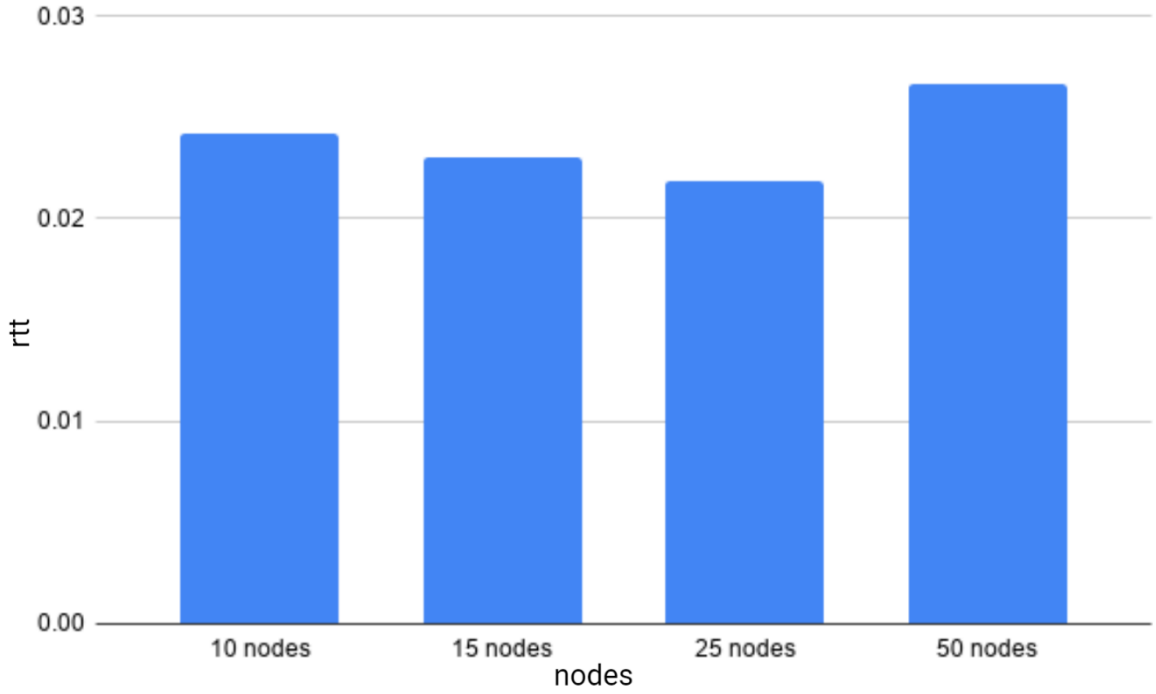


Figure 6.25: 2mbps - 128 bytes – RTT

We once again returned to the scenario where the packet size was set to 128 bytes and the number of nodes varied. This time we studied the RTT (round-trip time) of the network packets. The scenarios with the 10, 15 and 25 nodes where quite close as far as their RTT results go, but still, we can observe a tendency to reduce as the number of nodes increases. This is because the network is denser, so the RTT is slightly lower. In the scenario with the 50 nodes however, we can see that the RTT increases, and we can assume it is because the network is finding it difficult to deal with all this traffic being generated, so congestion takes place and increases the RTT.

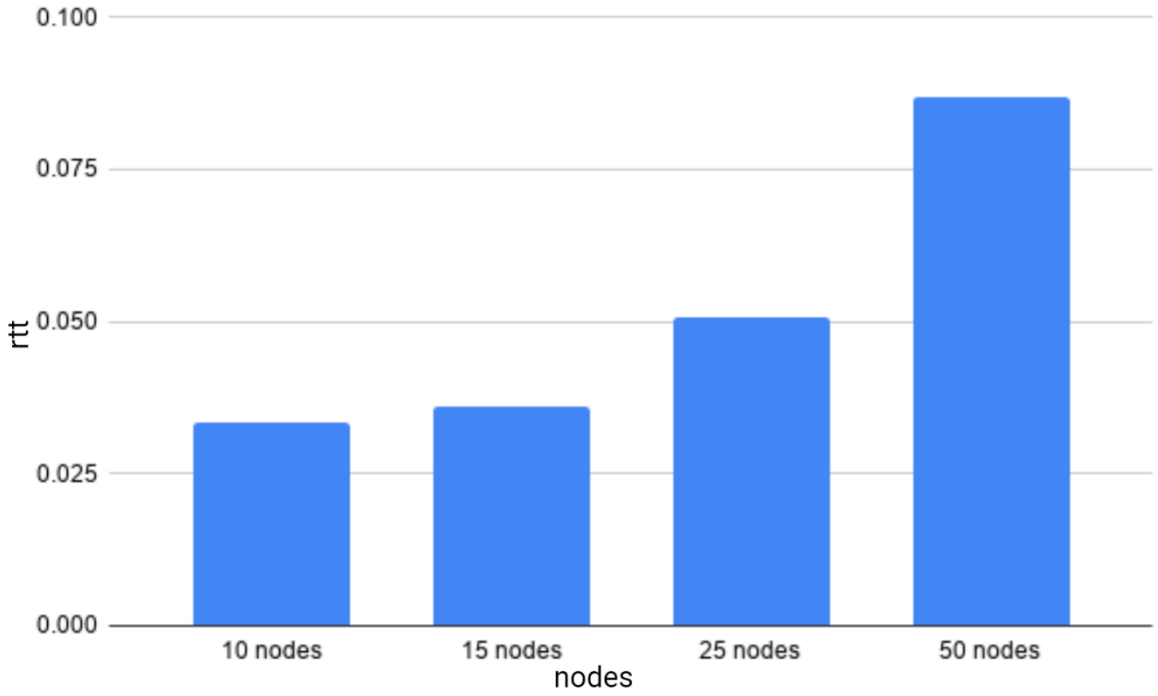


Figure 6.26: 2mbps - 768 bytes – RTT

Increasing the packet size to 768 bytes, we expected the RTT to increase alongside with it, especially in the scenarios with the higher number of nodes, since this is where the network has to deal with a lot of traffic and can struggle. Indeed, the RTT had increased in all of the scenarios, more specifically, in the one with the 10 nodes it increased by about 39%, in the one with the 15 nodes it increased by about 60%, in the one with the 25 nodes it increased by about 138% and in the one with the 50 nodes it increased by about 235%, proving that the most negative effects where when the network has to handle a lot of traffic.

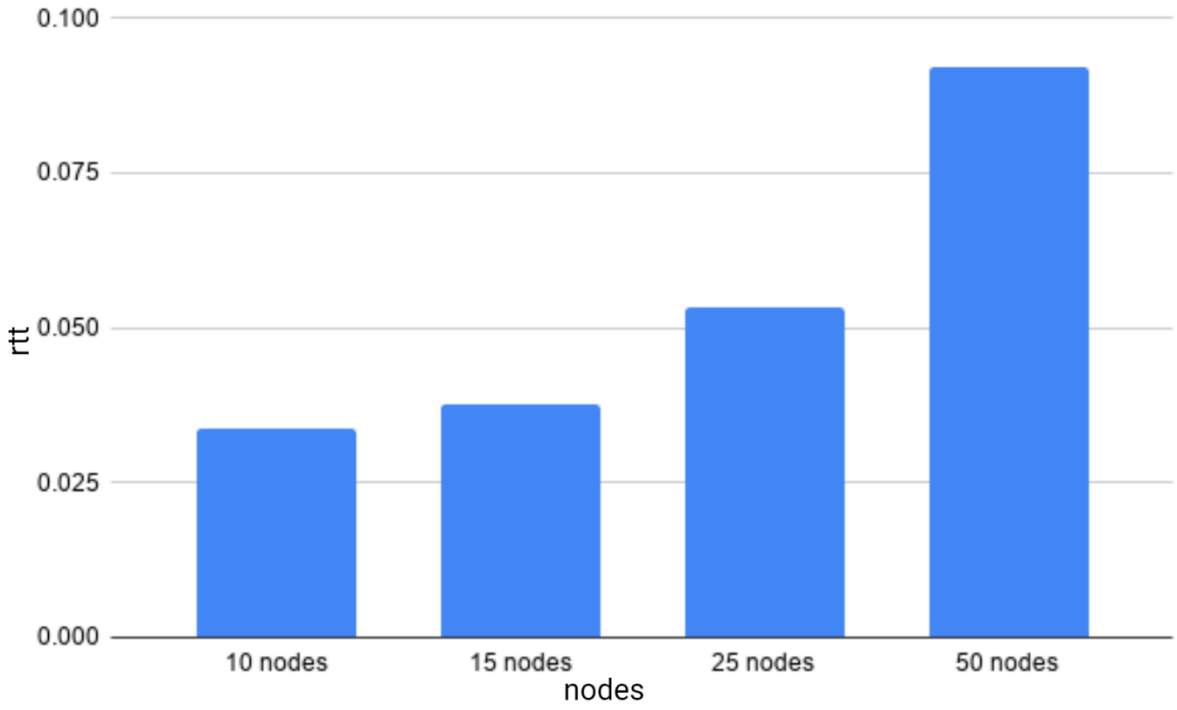


Figure 6.27: 2mbps - 1408 bytes – RTT

Finally, setting the packet size to 1408 bytes, we were met with these numbers. Again, the RTT had increased, as expected. In the scenario with the 10 nodes, the increase was about 62%, in the scenario with the 15 nodes it was about 67%, in the scenario with the 25 nodes it was about 85% and in the scenario with the 50 nodes it was about 1026%! This tells us that the network is really struggling with 50 nodes, at a packet size of 1408 and at such a low bitrate (2mbps).

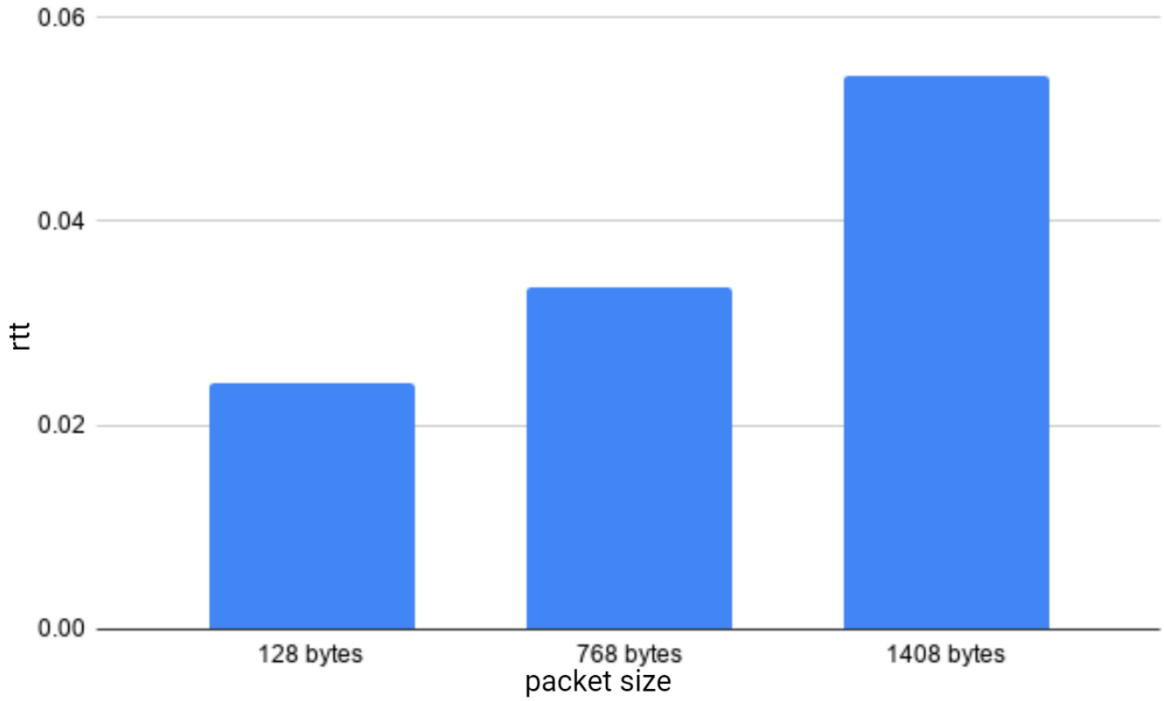


Figure 6.28: 2mbps - 10 nodes - RTT

Keeping the number of nodes steady and varying the packet size, we saw that (in the scenario with the 10 nodes at least) a higher packet size resulted in a higher RTT, and it makes sense since a higher packet size leads to more network load, which in turn leads to more packet collisions, and, by extension, to a higher RTT.

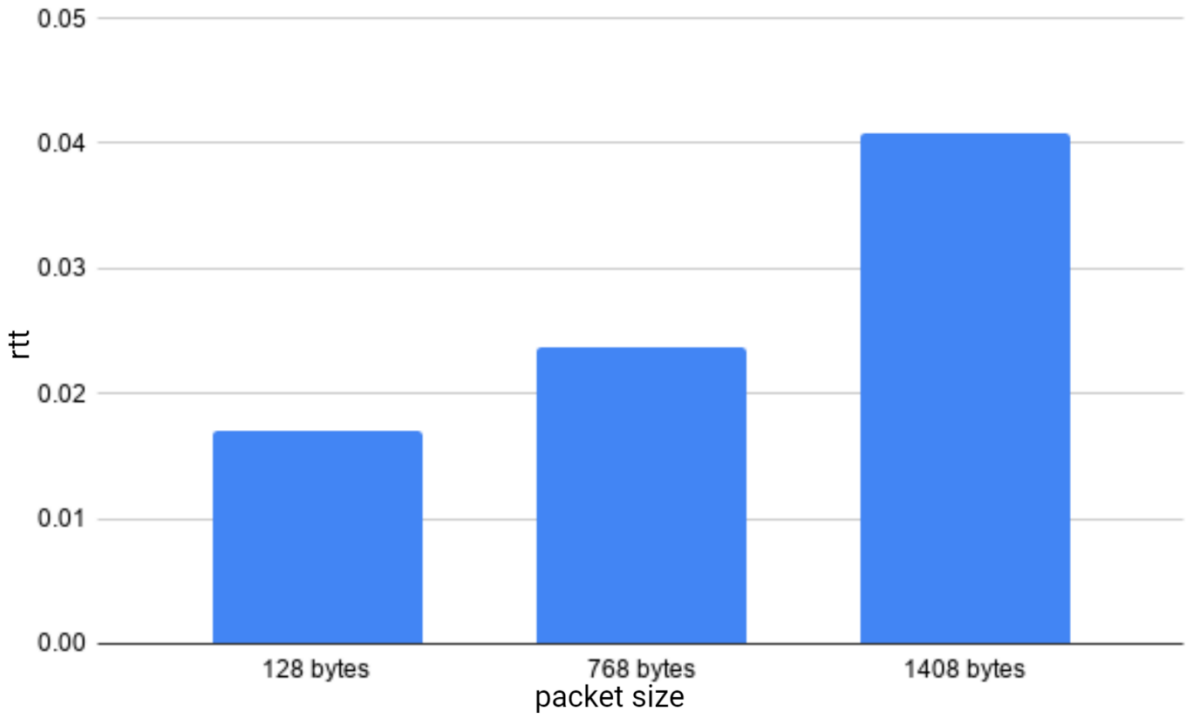


Figure 6.29: 2mbps - 15 nodes – RTT

Increasing the number of nodes by 5, to 15, we saw that the RTT values had decreased. This result was expected, since, as we said before, more network nodes mean that the network is denser, so the distance the packets have to travel shortens. This drives the value of the RTT down. In detail, in the scenario where the packet size was 128 bytes the RTT decreased by about 5%, in the scenario where the packet size was 768 bytes the RTT decreased by about 9% and in the scenario where the packet size was 1408 bytes the RTT decreased by about 13%. The biggest change was observed once again in the scenario with the higher packet size.

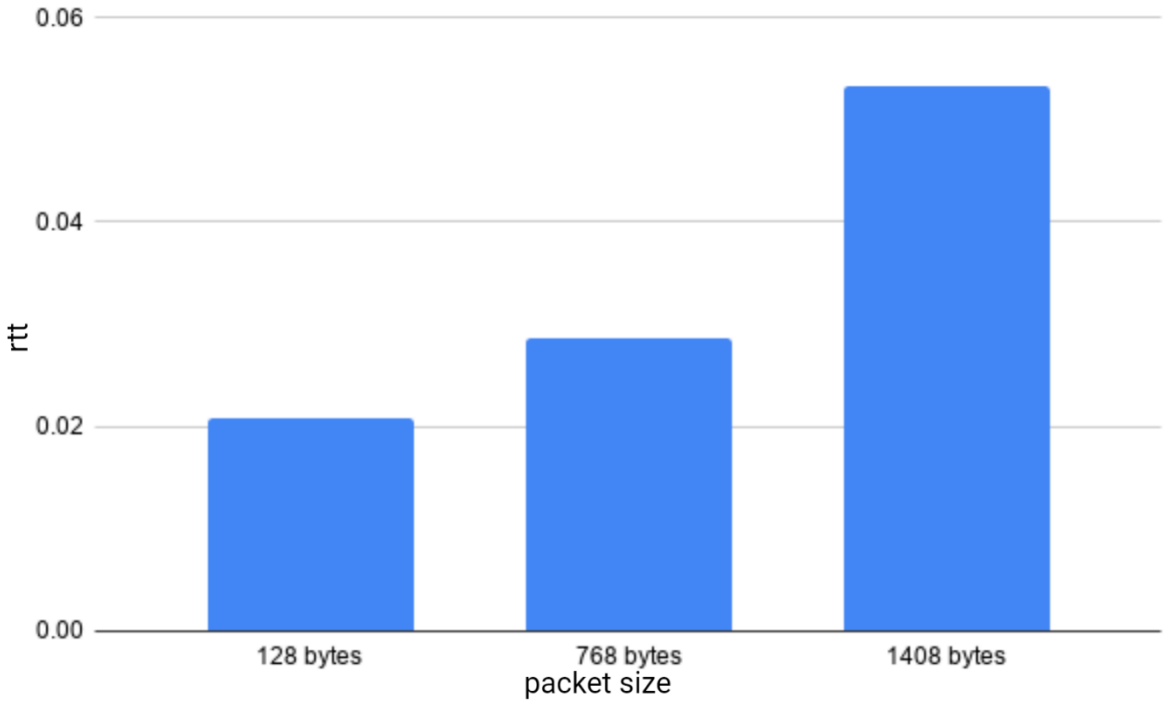


Figure 6.30: 2mbps - 25 nodes – RTT

Setting the nodes to 25, we started to observe something interesting. When the packet size was 128 bytes, the RTT decreases by about 5%, when it is 768 bytes however, it increased by about 42%. It also increased when the packet size was set to 1408 bytes, by about 57%. This showed us that when the packet size was 768 bytes or 1408 bytes and the number of nodes 25, the network struggled to manage all these packets, and congestion lead to a higher RTT. In other words, our changes started to make the network less optimized that it was before.

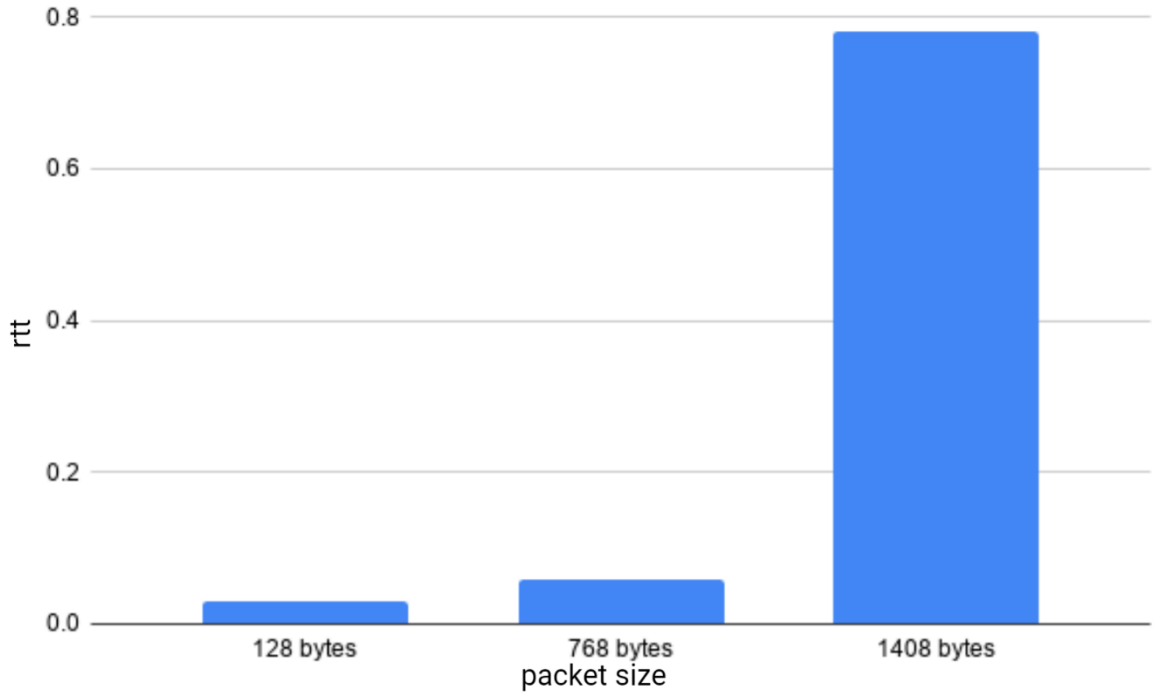


Figure 6.31: 2mbps - 50 nodes – RTT

From the results we saw when the number of nodes was 25, we can expect that now that the nodes were 50, an even higher network congestion would lead to an increase in the RTT. Indeed, about 22% for the scenario with the packet size of 127 bytes, about 72% for the scenario with the packet size of 768 bytes and about 943% for the scenario with the packet size of 1408 bytes! This shows us that a high number of nodes (50) and a big packet size is not good for our network, which finds it difficult to handle all these packets being generated. At this point, increasing the bitrate would be beneficial, which we would do later.

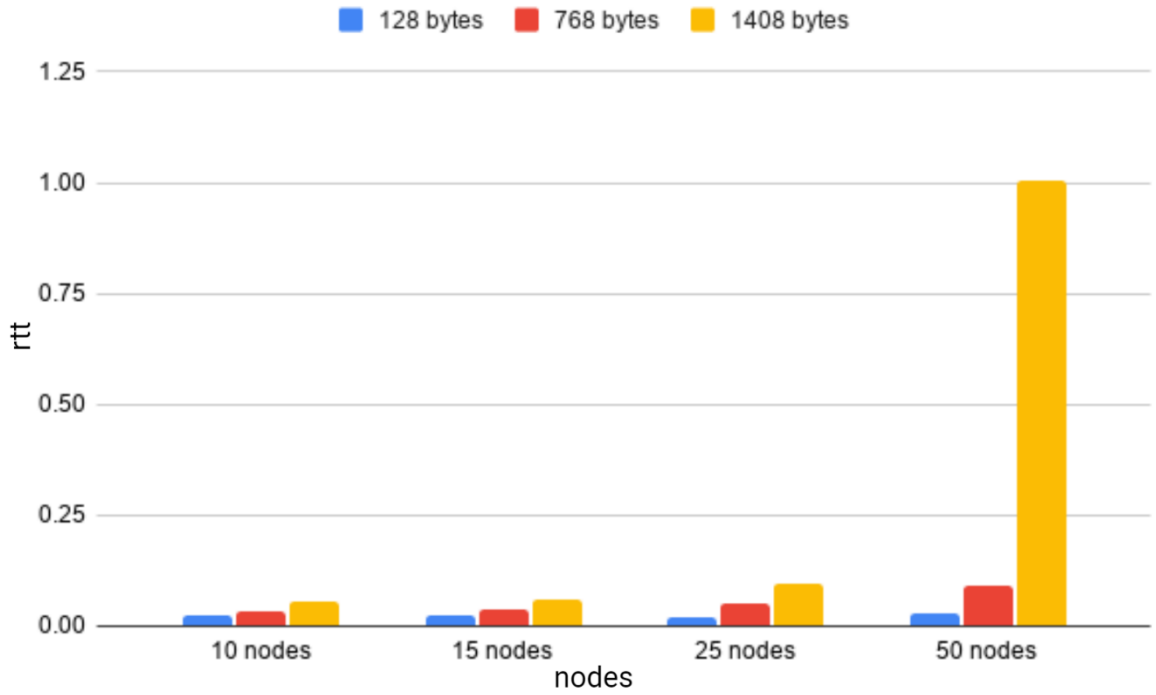


Figure 6.32: RTT - 2mbps

Moving on to the RTT, we can see that when the packet size is 128 bytes, the RTT remains basically unchanged. When the packet size is bumped up to 768 bytes there is a slight increase as the number of nodes keeps increasing, especially at the scenario with the 50 nodes. The interesting results are from the scenario with the 50 nodes and the 1408 bytes packet size, where the RTT practically skyrockets. In the end, this is also to be expected, since increased packet sizes lead to an increase in the RTT, as we previously discussed. In combination with a high number of nodes, the RTT results are expected to be high. The difference with all the other measurements we took signals us that 2mbps is too low of a bitrate for this scenario, and the performance suffers for it.

6.2.2 11mbps

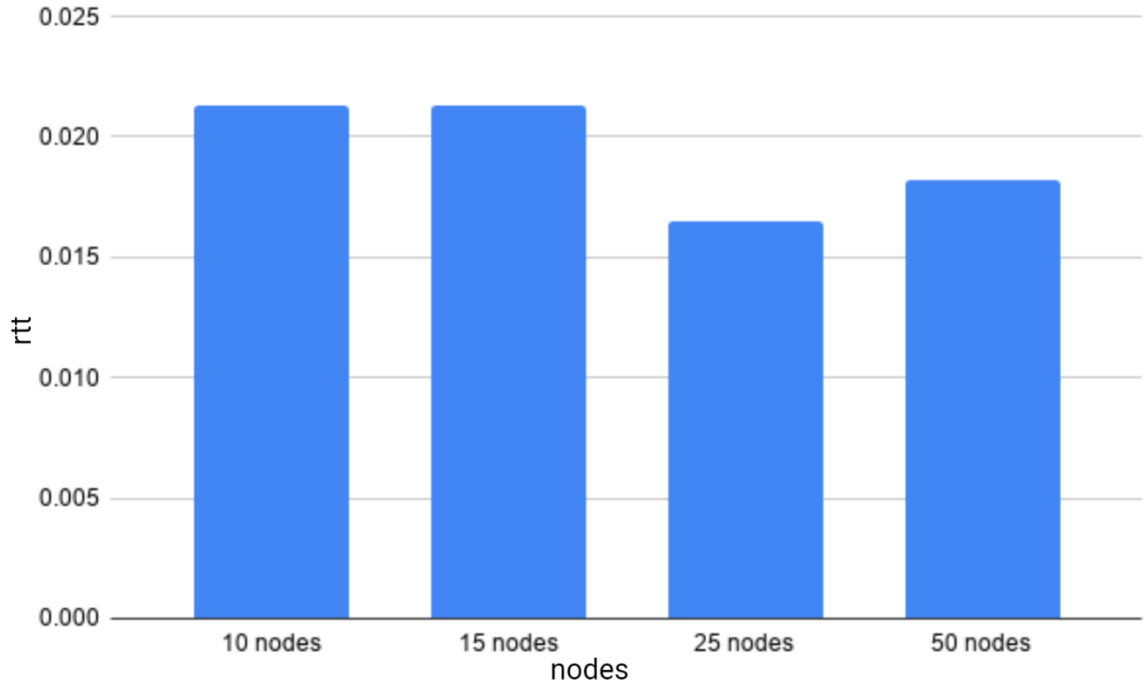


Figure 6.33: 11mbps - 128 bytes – RTT

Turning our attention to the RTT, we kept the number of nodes the same and changed the value of the packet size. As we can see from the above graph, the optimal scenarios for the best possible RTT values when the packet size is 128 bytes is 10 and 15 nodes, which displayed very close results. The change from 2mbps to 11mbps was also quite significant, since we observed a 12% decrease in the scenario with the 10 nodes, a 7% decrease in the scenario with the 15 nodes, a 24% decrease in the scenario with the 25 nodes and a 32% decrease in the scenario with the 50 nodes. As expected, the scenarios with the 25 and 50 nodes saw the biggest change, since they were the ones most affected by the small bitrate.

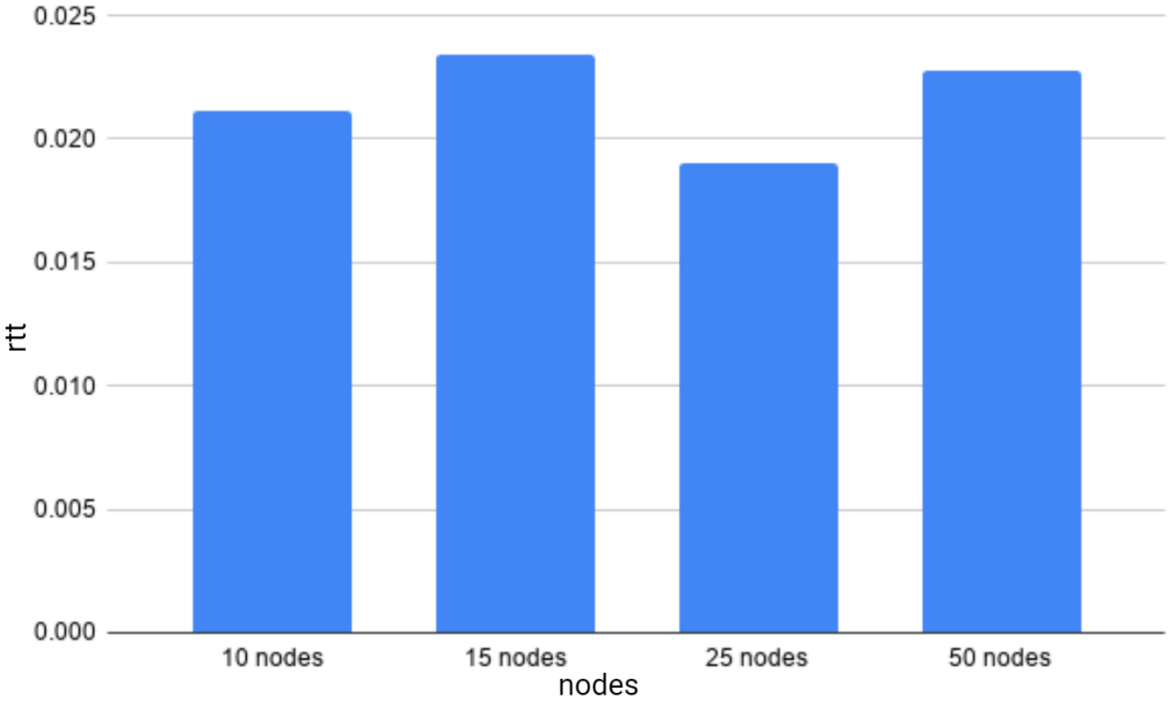


Figure 6.34: 11mbps - 768 bytes – RTT

Increasing the packet size to 768 bytes, we documented a 0.4% decrease in the RTT in the scenario with the 10 nodes, a 14% increase in the one with the 15 nodes, a 17% increase in the one with the 25 nodes and a 26% increase in the one with the 50 nodes. The change from 2mbps to 11mbps also brought about changes, when the nodes were 10, we saw a 37% decrease, when they were 15, we saw a 34% decrease, when they were 25, we saw a 63% decrease and when they were 50, we saw a 74% decrease.

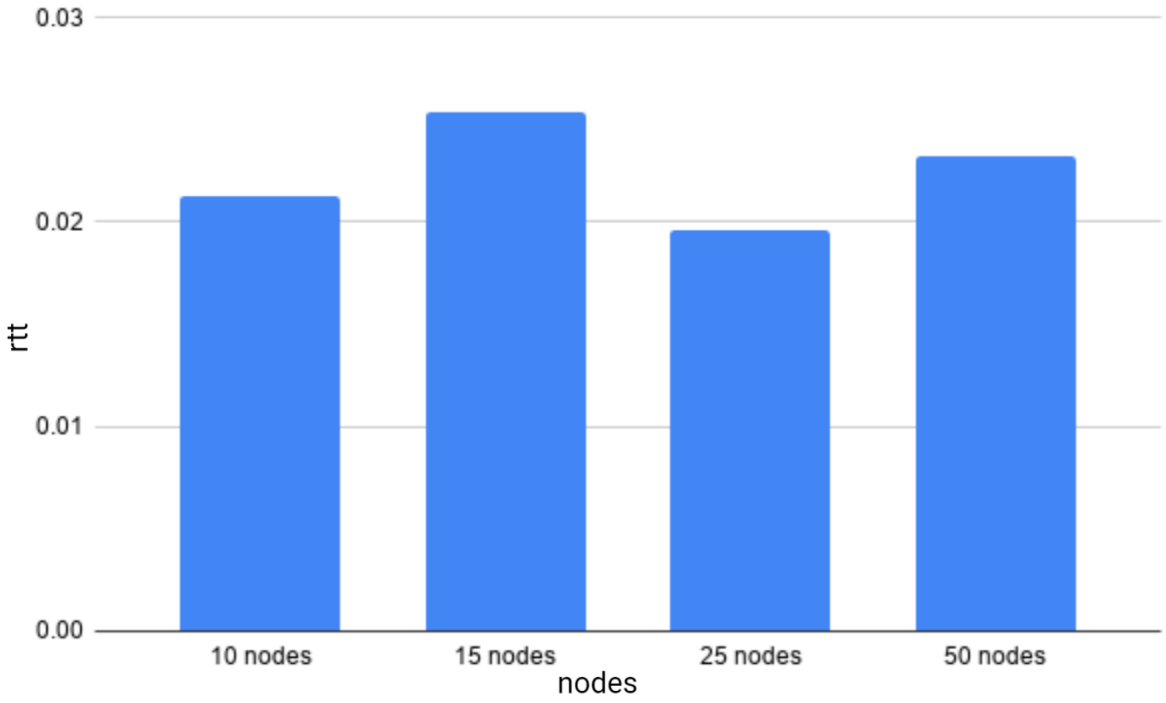


Figure 6.35: 11mbps - 1408 bytes – RTT

Increasing the packet size to 1408 bytes, we documented a 17% increase in the RTT in the scenario with the 10 nodes, a 2.24% increase in the one with the 15 nodes, a 24% increase in the one with the 25 nodes and a 20% increase in the one with the 50 nodes. The change from 2mbps to 11mbps also brought about changes, when the nodes were 10, we saw a 54% decrease, when they were 15, we saw a 59% decrease, when they were 25, we saw a 75% decrease and when they were 50, we saw a 97% decrease.

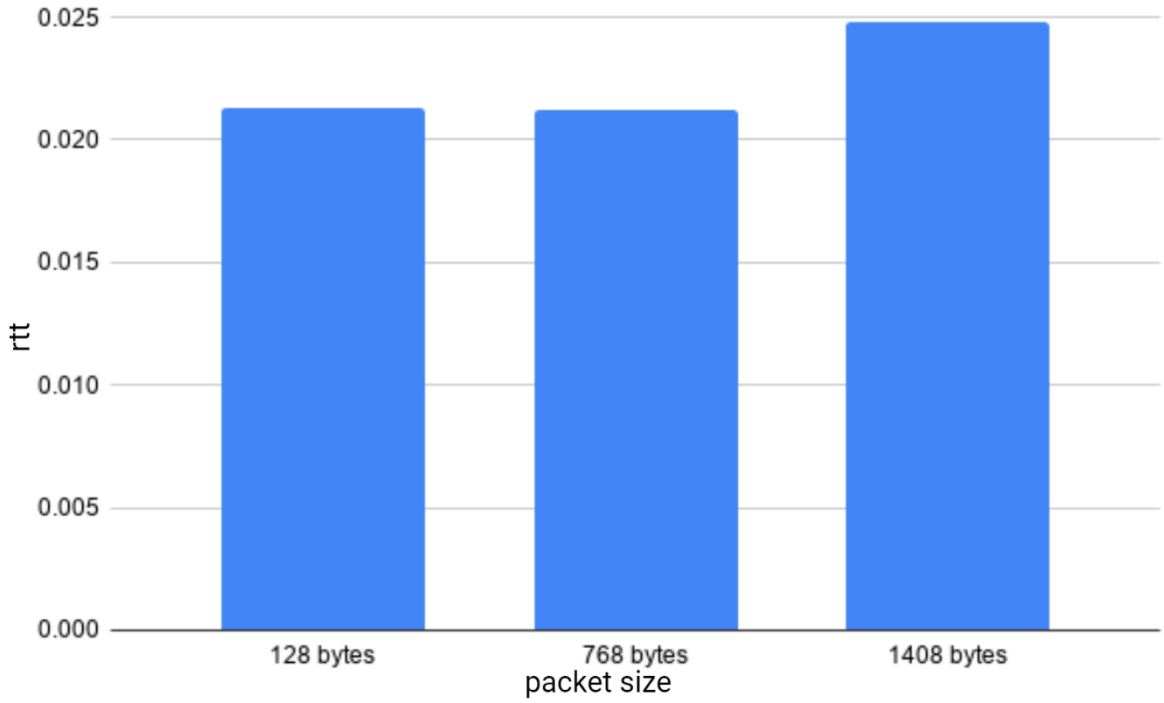


Figure 6.36: 11mbps - 10 nodes – RTT

Keeping the number of nodes the same and varying the value of the packet size, these are the results we gathered. When the packet size is 128 bytes, the jump from 2mbps to 11mbps resulted in a 12% decrease in the RTT, in a 37% decrease when the packet size is 768 bytes and in a 54% decrease when the packet size is 1408 bytes.

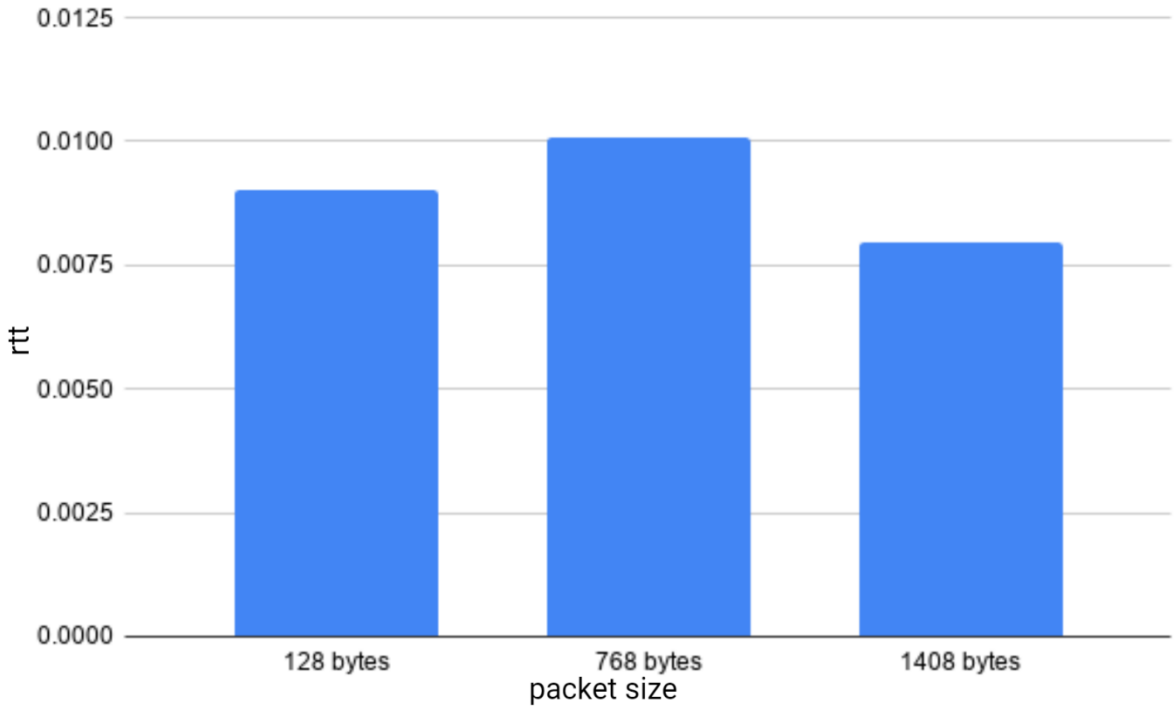


Figure 6.37: 11mbps - 15 nodes – RTT

Increasing the number of nodes by 5 to 15, we observed a 0.2% increase when the packet size is 128 bytes, a 15% increase when it is 768 bytes and a 0.4% increase when it is 1408 bytes. The increase in the bitrate from 2mbps to 11mbps resulted in a 7.2% increase in the RTT, in the scenario with the packet size of 128 bytes, in a 34% decrease in the scenario with the packet size of 768 bytes and in a 59% increase in the scenario with the packet size of 1408 bytes.

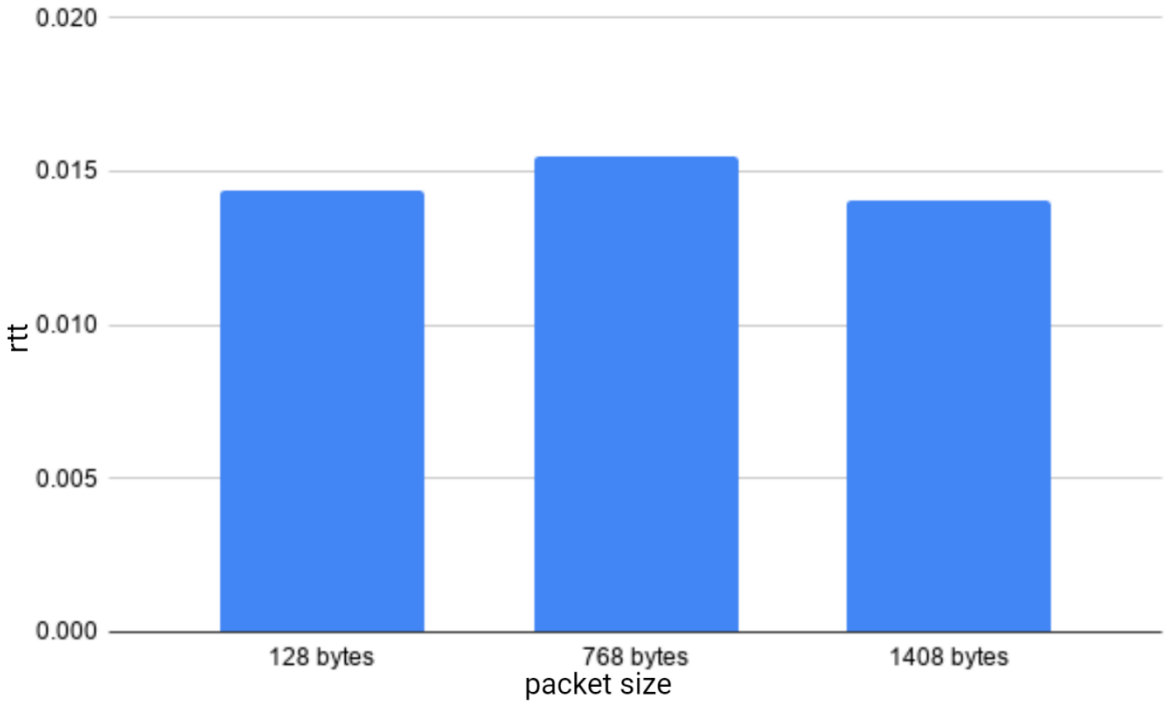


Figure 6.38: 11mbps - 25 nodes – RTT

Increasing the number of nodes to 25, we observed a 23% decrease when the packet size is 128 bytes, a 21% decrease when it is 768 bytes and a 4.1% decrease when it is 1408 bytes. The increase in the bitrate from 2mbps to 11mbps resulted in a 24% decrease in the RTT, in the scenario with the packet size of 128 bytes, in a 63% decrease in the scenario with the packet size of 768 bytes and in a 75% increase in the scenario with the packet size of 1408 bytes.

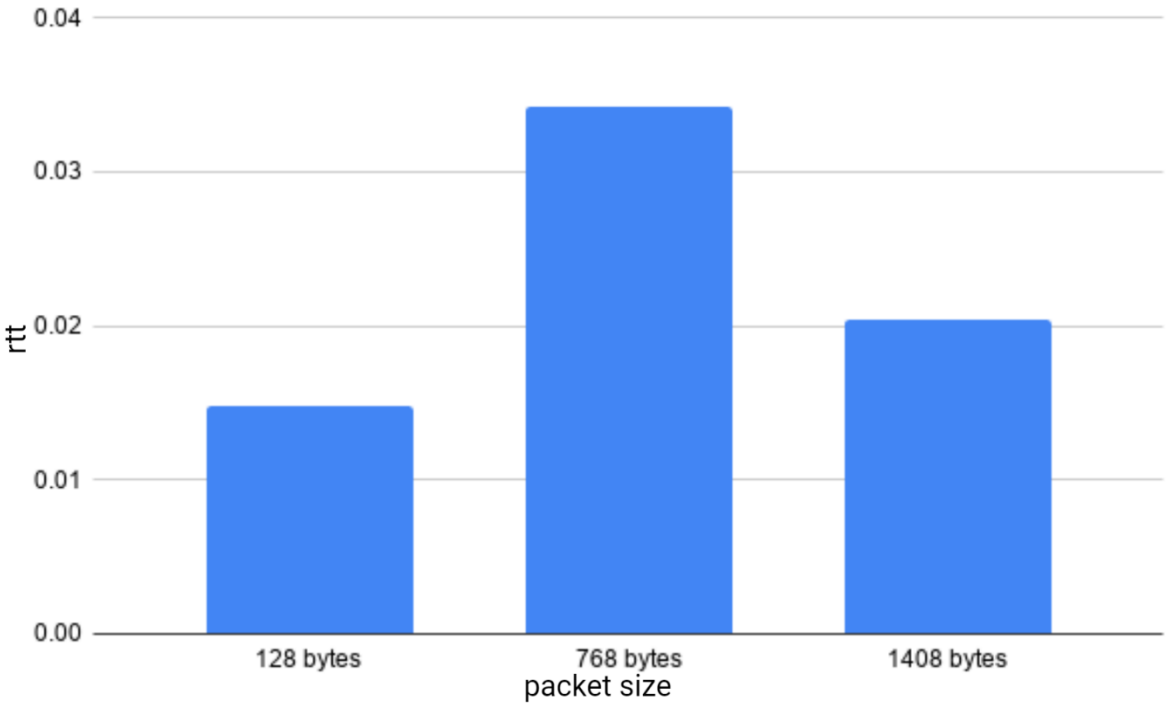


Figure 6.39: 11mbps - 50 nodes – RTT

Increasing the number of nodes to 50, we observed a 11% increase in the RTT when the packet size is 128 bytes, a 19% increase when it is 768 bytes and a 15% increase when it is 1408 bytes. The increase in the bitrate from 2mbps to 11mbps resulted in a 32% decrease in the RTT, in the scenario with the packet size of 128 bytes, in a 74% decrease in the scenario with the packet size of 768 bytes and in a 97% increase in the scenario with the packet size of 1408 bytes.

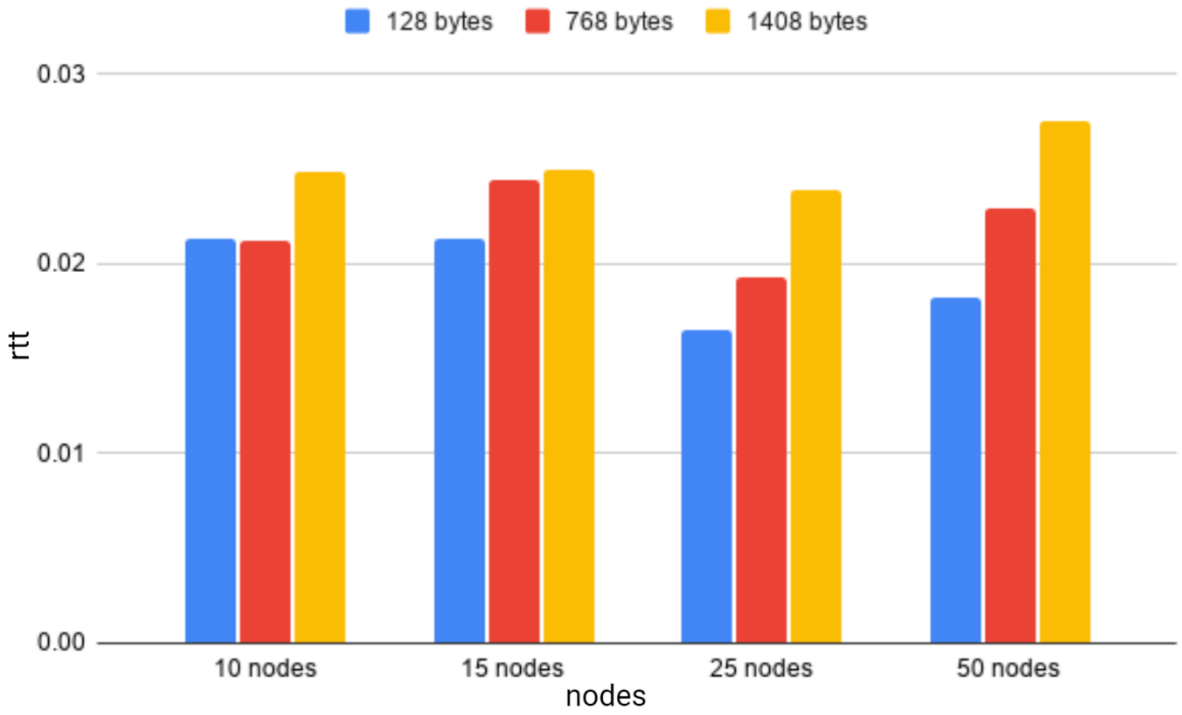


Figure 6.40: RTT - 11mbps

We see a similar situation with the RTT, where the results this time are only a fraction of what they were when the bitrate was 2mbps. This is normal behavior, since 2mbps is not really enough to serve so many packets without delays, especially when the packet size is big, and so are the number of nodes. Apart from the scenario with the 50 nodes and the packet size of 1408 bytes, all the other measurements were fairly consistent, with the best result being achieved with 25 nodes and a packet size of 128 bytes. It was expected that the smallest packet size would lend itself to the lowest RTT, but why 25 nodes and not, say, 10 or 50? Well, 25 nodes seem to be the perfect balance between too few nodes (meaning that the network is less dense, so the packets have to travel greater distances) and too many (meaning that a great number of packets is being produced, which can lead to network congestion).

6.2.3 54mbps

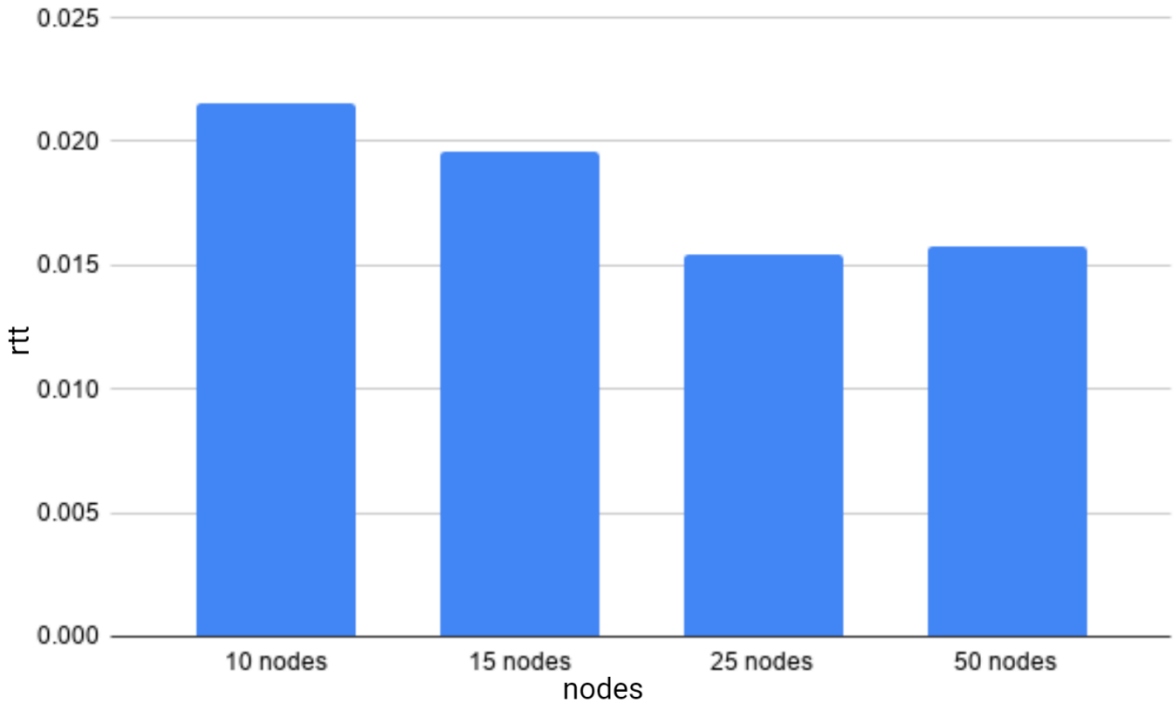


Figure 6.41: 54mbps – 128bytes – RTT

Focusing now on the RTT, when the packet size was set to 128 bytes, in the scenario where the nodes were 10 it increased by 1.3%, when they were 15 it decreased by 8.1%, when they were 25 it decreased by 6.5% and when they were 50 it decreased by 13.5%. It makes sense that the biggest difference would be in the scenario with the most nodes, since this is when the network can benefit from the higher bitrate.

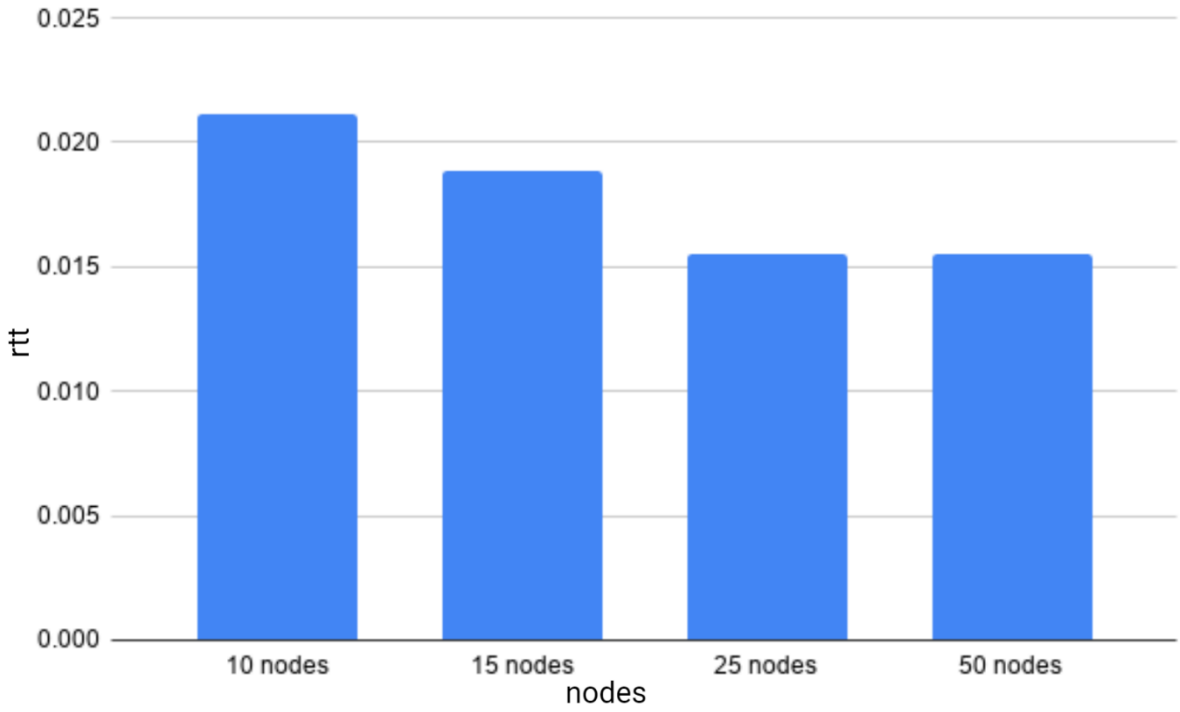


Figure 6.42: 54mbps – 768 bytes – RTT

When we set the packet size to 768 bytes, in the scenario where the nodes were 10 it decreased by 1.6%, when they were 15 it increased by 0.5%, when they were 25 it increased by 3% and when they were 50 it decreased by 0.8%. The increase in the bitrate provided bigger changes to our results, seeing a 0.1% increase in the scenario with the 10 nodes, a 19% decrease in the scenario with the 15 nodes, an 18% decrease in the scenario with the 25 nodes and a 32% decrease in the scenario with the 10 nodes. This proves that a denser network usually equals a lower value for the RTT, provided that the network doesn't produce a lot of errors.

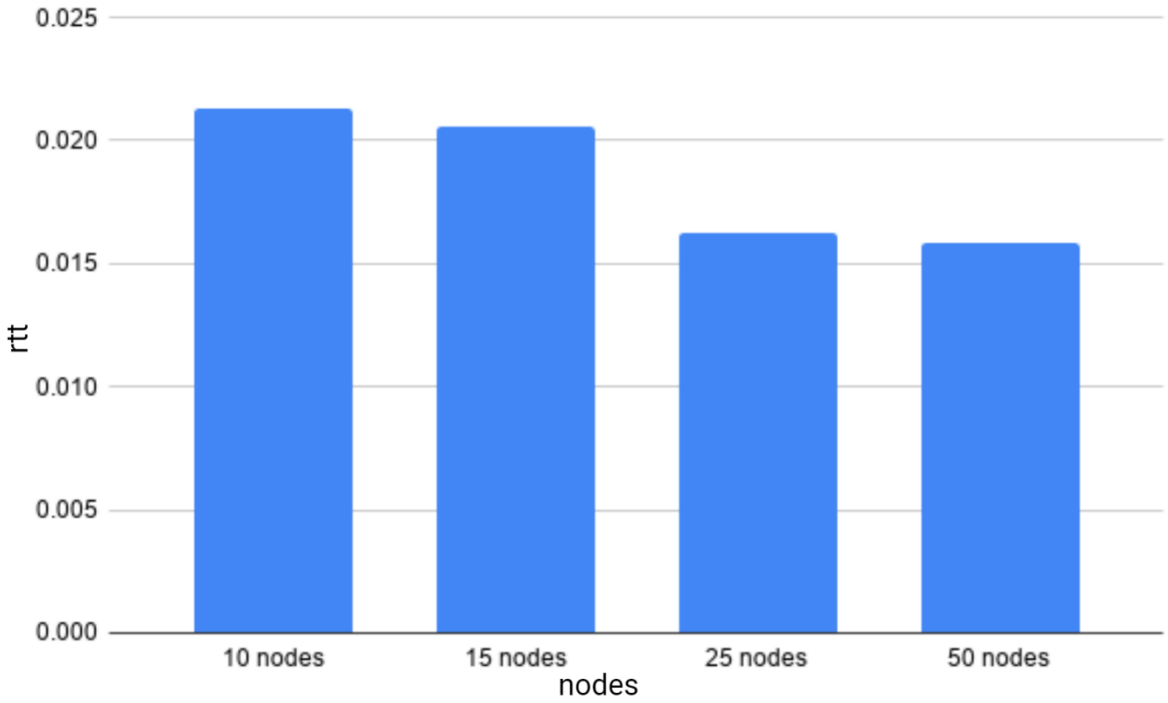


Figure 6.43: 54mbps – 1408 bytes – RTT

When we set the packet size to 1408 bytes, in the scenario where the nodes were 10 it increased by 9.4%, when they were 15 it decreased by 2%, when they were 25 it decreased by 1.1% and when they were 50 it increased by 7.6%. The increase in the bitrate provided once again bigger changes to the RTT, seeing a 6.5% decrease in the scenario with the 10 nodes, a 23% decrease in the scenario with the 15 nodes, an 34% decrease in the scenario with the 25 nodes and a 39% decrease in the scenario with the 10 nodes.

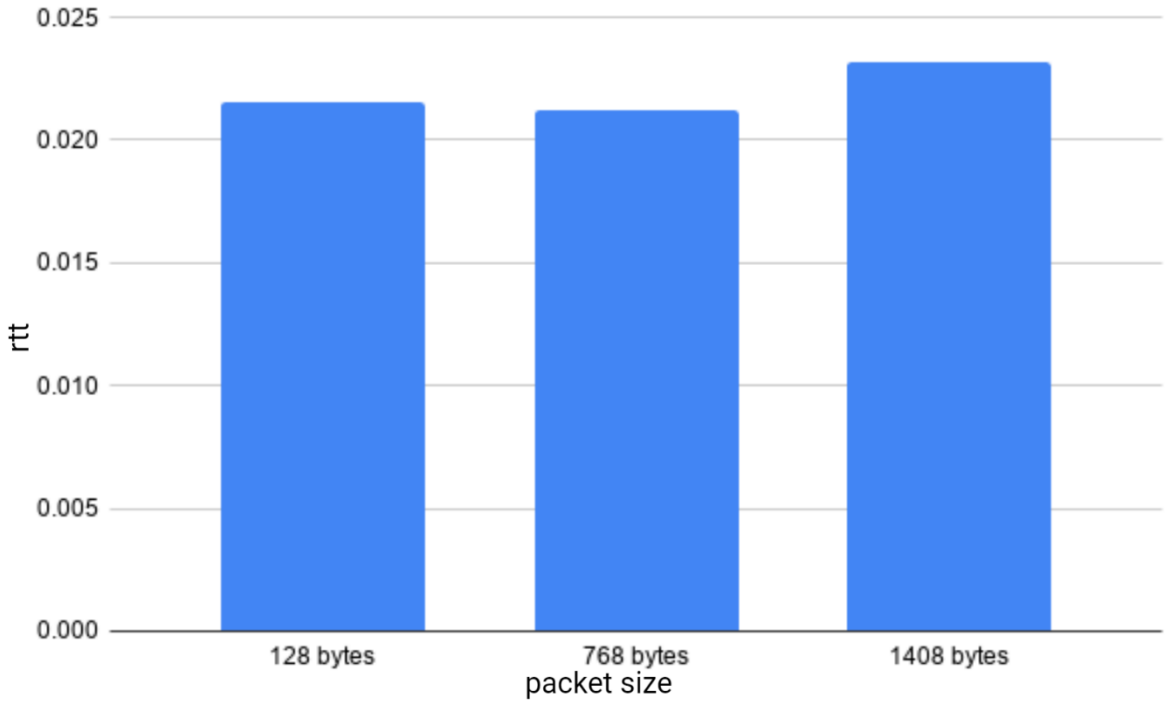


Figure 6.44: 54mbps – 10 nodes – RTT

Now varying the packet size and leaving the number of nodes the same, we observed that the increase in the bitrate resulted in some very minor changes in the first two scenarios. More specifically, when the packet size was set to 128 bytes the increase in the RTT was 1.3%, when the packet size was 768 bytes the increase was just 0.1% and when the packet size was 1408 bytes the decrease was 6.5%. The first two scenarios provided us with little to no change, because the network traffic is not high enough to take advantage of the extra head room. With a high enough packet size however, we can definitely observe some difference for the best.

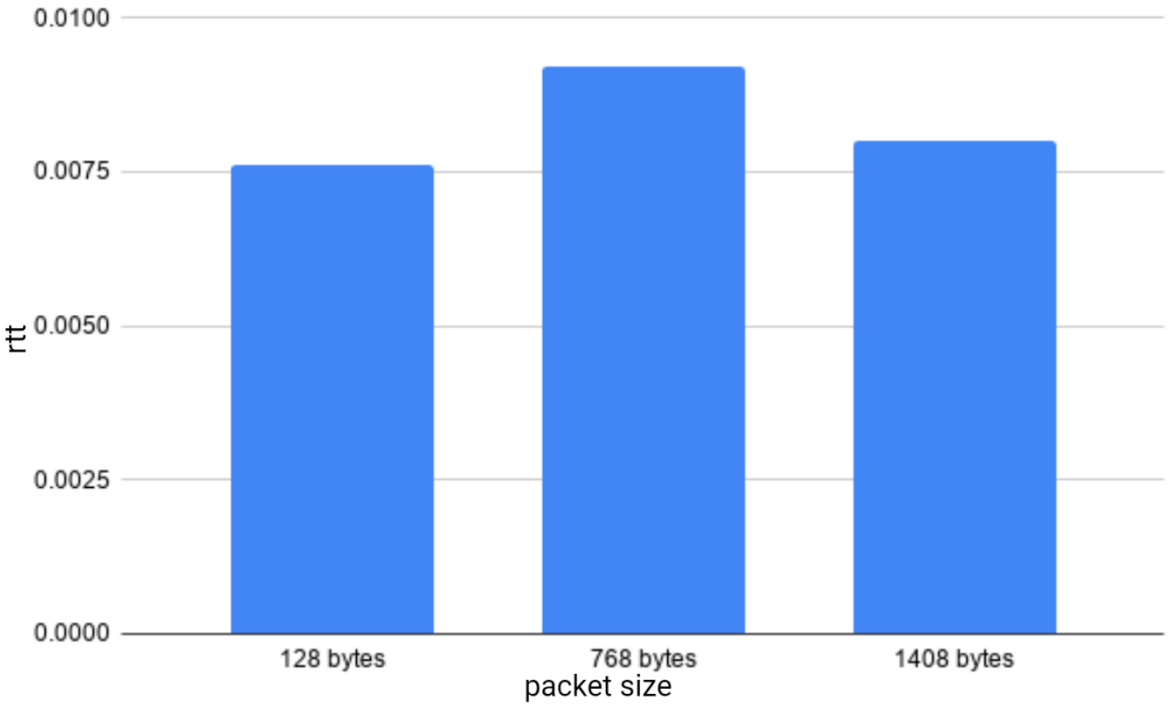


Figure 6.45: 54mbps – 15 nodes – RTT

Increasing the nodes to 15, we observed a 9.1% decrease in the RTT in the scenario where the packet size was 128 bytes, a 7.1% decrease in the scenario where the packet size was 768 bytes and a 17% decrease when the packet size was 1408 bytes. Additionally, the increase in the bitrate resulted in a 8.1%, 19% and 23% drop respectively.

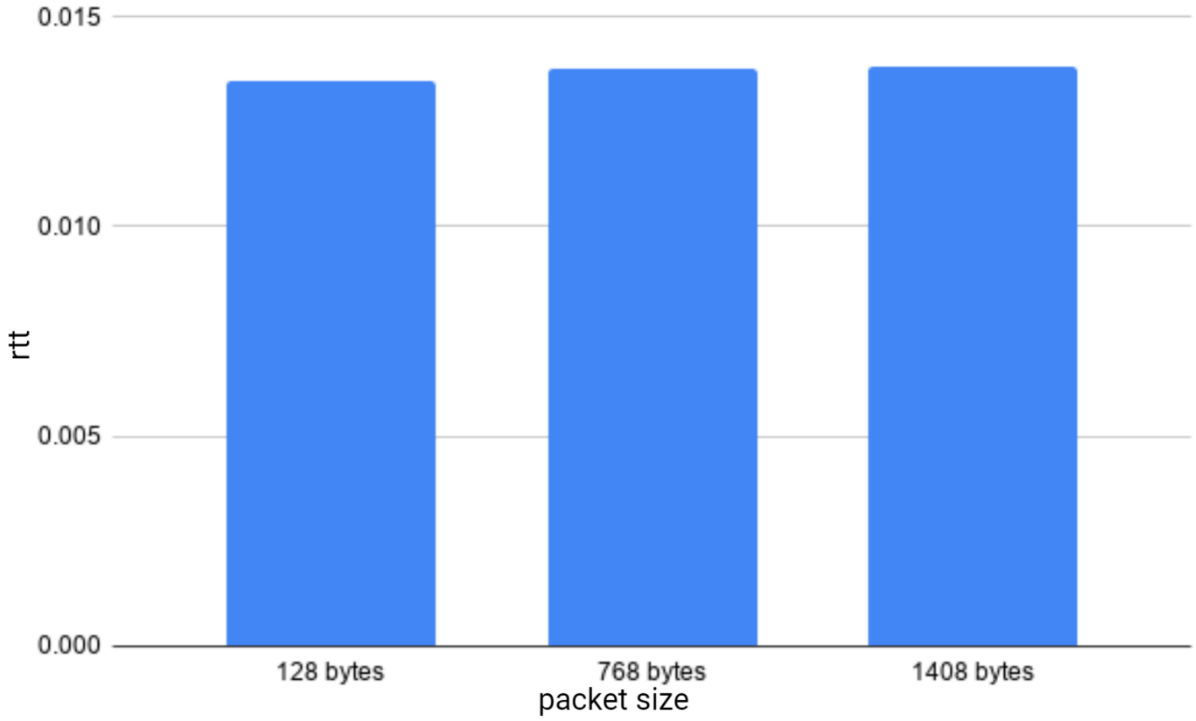


Figure 6.46: 54mbps – 25 nodes – RTT

Increasing the nodes to 25, we observed a 21% decrease in the RTT in the scenario where the packet size was 128 bytes, a 19% decrease in the scenario where the packet size was 768 bytes and again a 19% decrease when the packet size was 1408 bytes. The increase in the bitrate resulted in a 6.5%, 18% and 34% drop respectively.

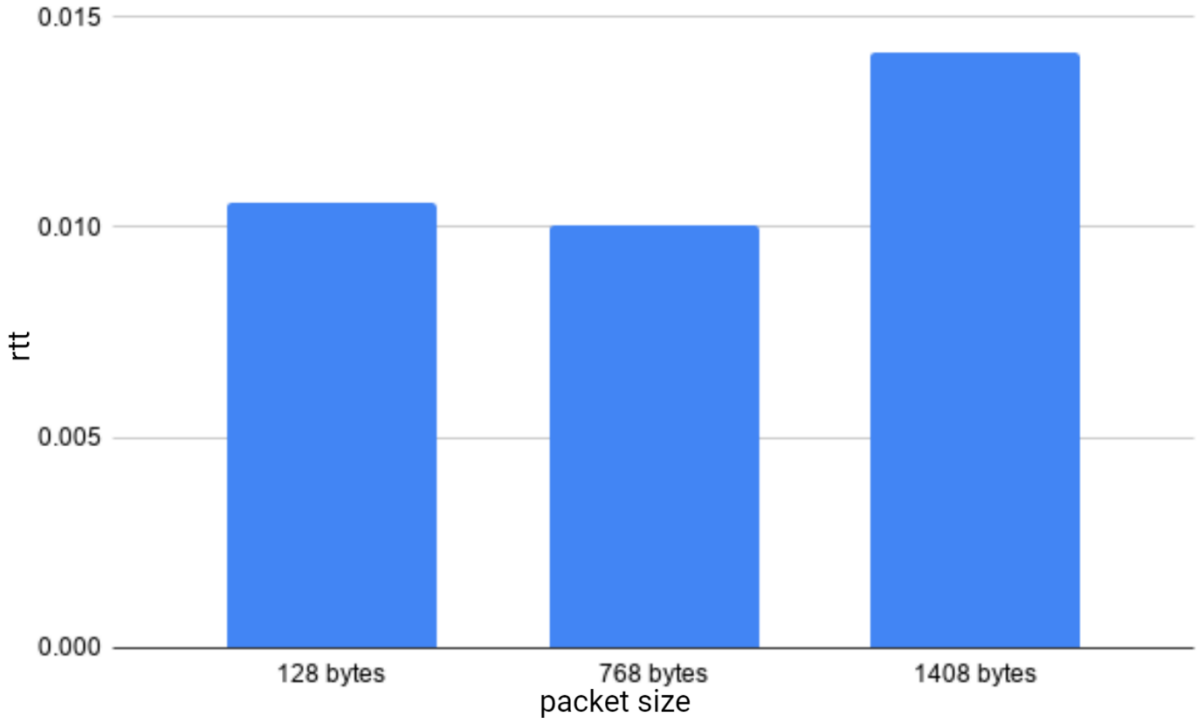


Figure 6.47: 54mbps – 50 nodes – RTT

Increasing the nodes one last time to 50, we observed a 2.5% increase in the RTT in the scenario where the packet size was 128 bytes, a 1.3% decrease in the scenario where the packet size was 768 bytes and a 7.4% increase when the packet size was 1408 bytes. These results show us that we have reached the point where making the network more dense has little to no positive effect, or even affects in negatively. The increase in the bitrate resulted in a 13%, 32% and 39% drop respectively, proving to be a better option to reduce the RTT.

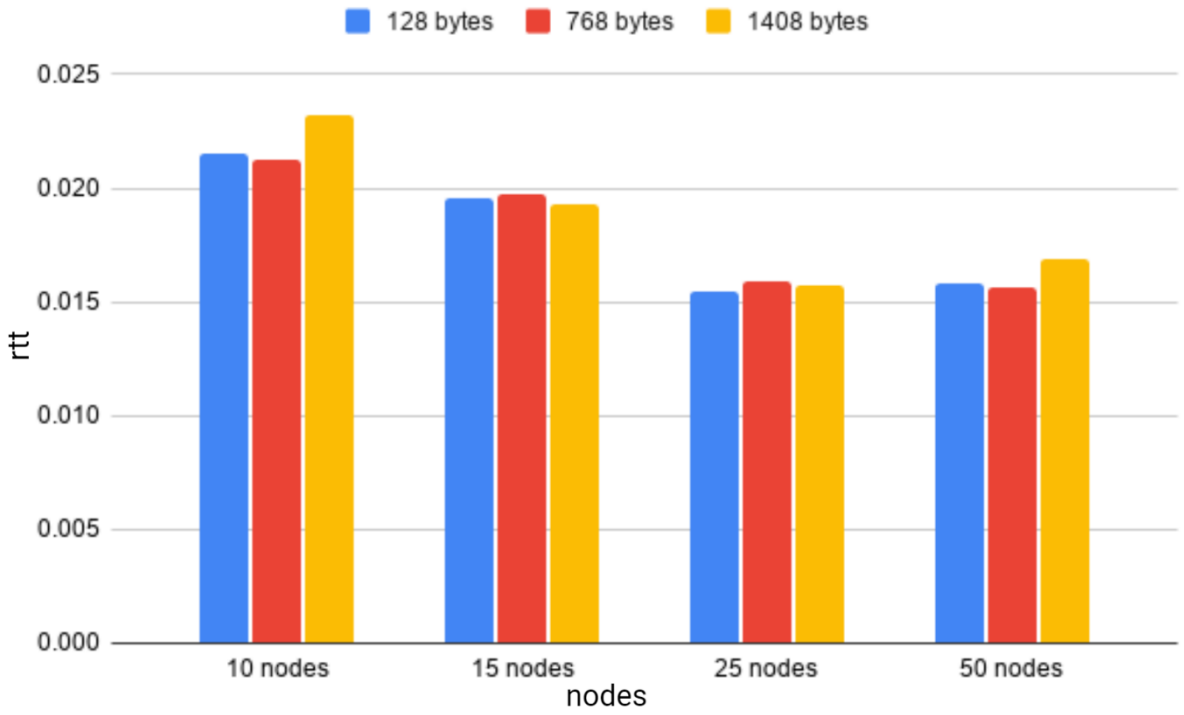


Figure 6.48: RTT - 54mbps

As far as the RTT is concerned, the increase from 11mbps to 54mbps did not change all that much, in some cases the RTT was a bit higher, and in some others, it was a bit lower. The change that is the most obvious is the fact that the results here seem more weighted, the scenarios where the number of nodes is 50 do not show a sudden increase in the RTT, as we can observe in the scenarios where the bitrate was 11mbps. This is expected, since the higher bitrate alleviates any problems caused by a large number of nodes. In other words, the increased bitrate helps the network cope much better to the increased network traffic.

6.3 The effect of bitrate and number of nodes on the throughput

6.3.1 2mbps

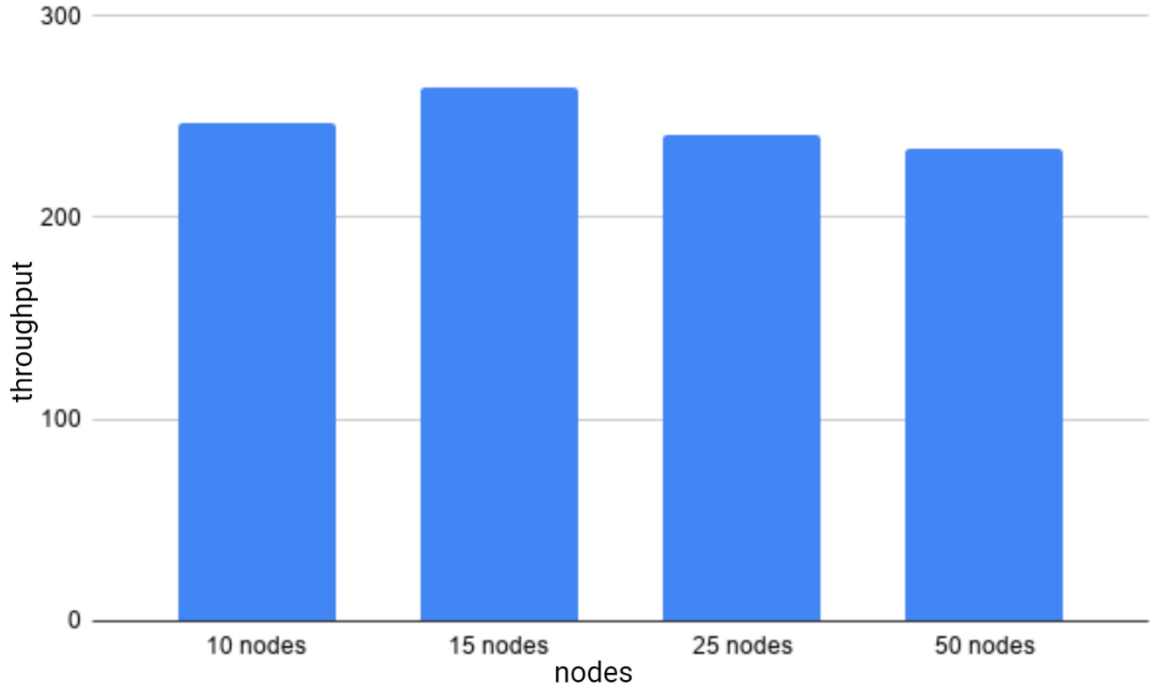


Figure 6.49: 2mbps - 128 bytes – throughput

Going back to the original scenario with the packet size of 128 bytes and varying the number of nodes, we turned out attention to the network throughput. Looking at the graph we can see that the optimal number of nodes (for a packet size of 128 bytes) is 15, followed closely by 10 and then 25. 50 nodes yielded the worst result out of them all, because, as we previously saw, in this scenario the packet drop rate is higher, and so is the RTT. These two parameters have a great effect on the network throughput, so it makes sense for the results to be the worst here.

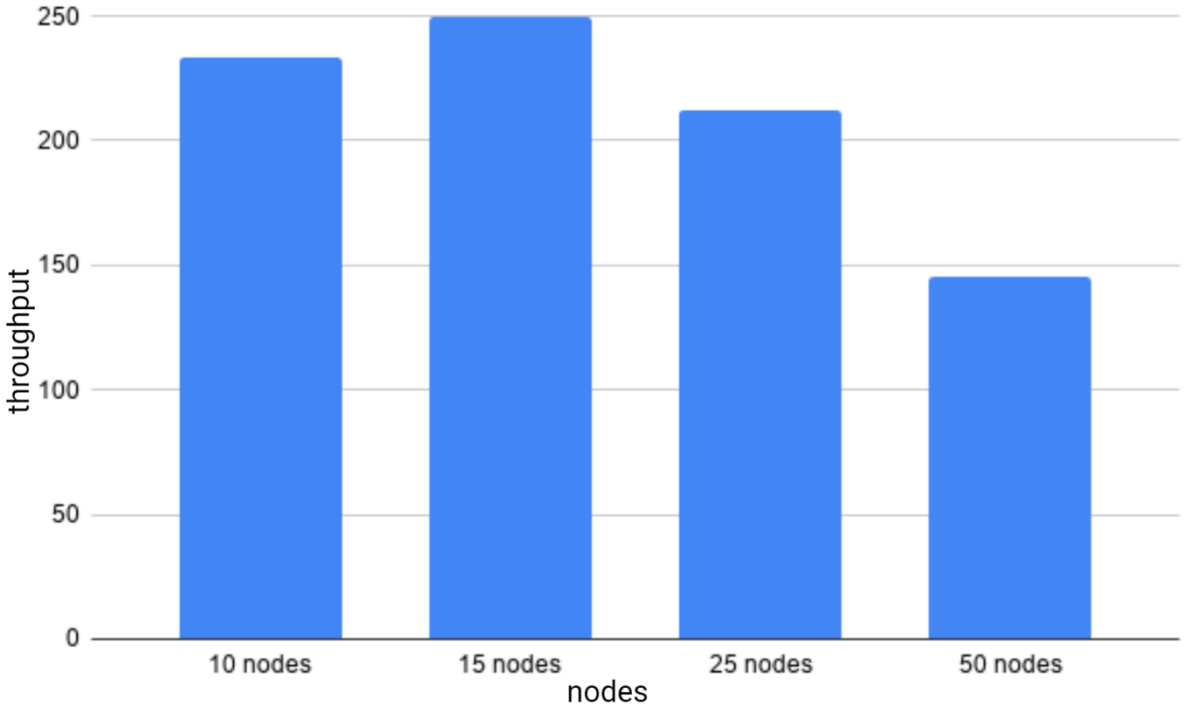


Figure 6.50: 2mbps - 768 bytes – throughput

Increasing the packet size to 768 bytes, we observed that the network throughput decreased, especially in the scenario with the 50 nodes. In more detail, in the scenario with the 10 nodes the throughput decreased by about 6%, in the scenario with the 15 nodes by about 5%, in the scenario with the 25 nodes by about 12% and in the scenario with the 50 nodes by about 38%. As previously stated, a higher packet size can lead to more network congestion and, in turn, in less throughput, so the results are not anything unexpected.

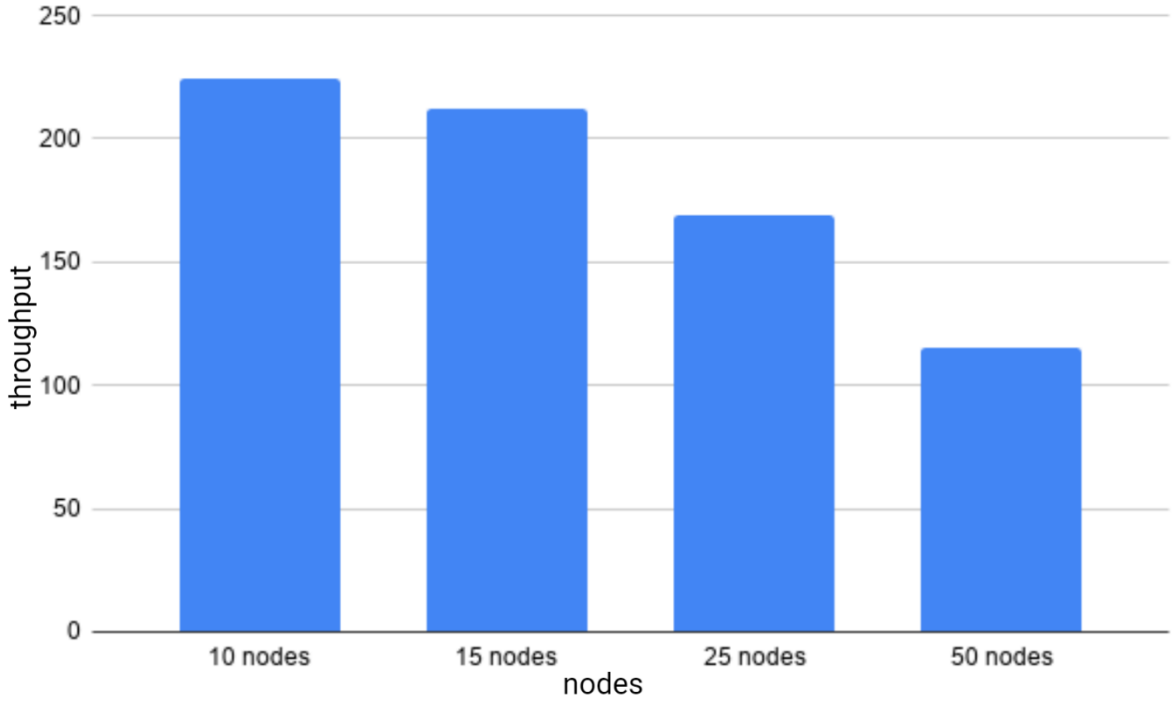


Figure 6.51: 2mbps - 1408 bytes – throughput

Increasing the packet size one last time to 1408 bytes, we, again, saw that the network throughput had taken a hit. By about 4% in the scenario with the 10 nodes, by about 15% in the scenario with the 15 nodes, by about 20% in the scenario with the 25 nodes and by about 21% in the scenario with the 50 nodes.

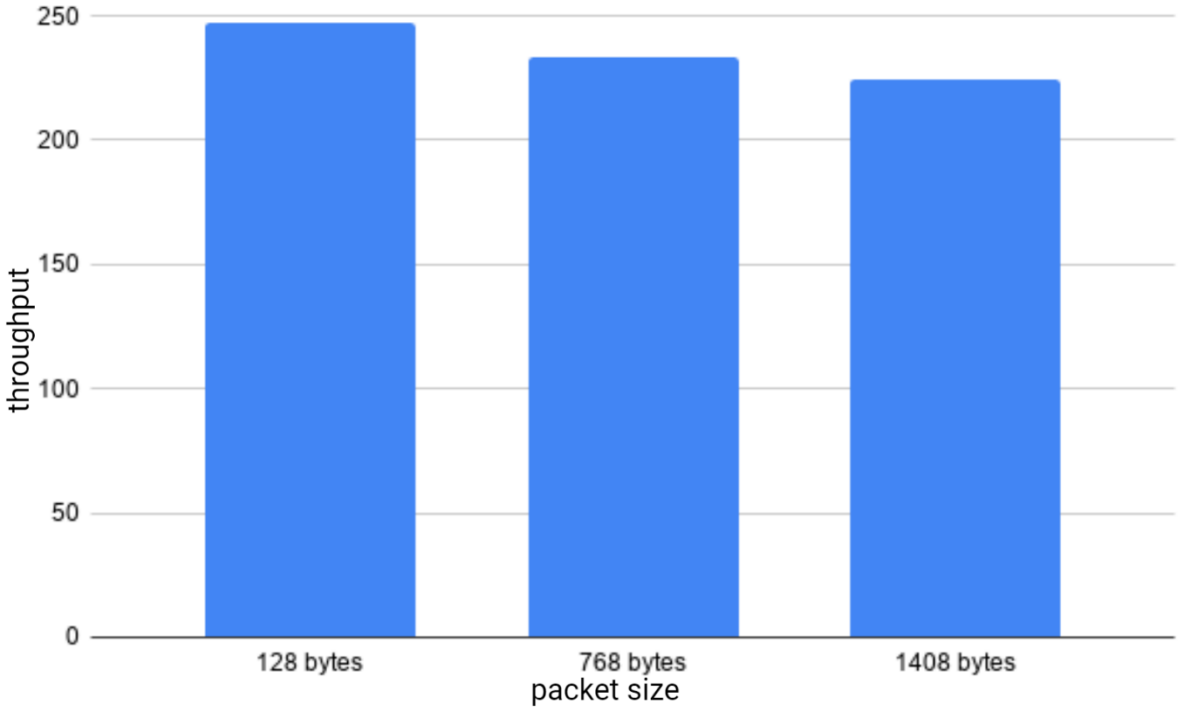


Figure 6.52: 2mbps - 10 nodes – throughput

Varying the packet size and keeping the number of nodes the same (10 in this scenario), we generated the above results. The smallest packet size produced the best throughput, because the largest packet sizes put more strain to the network and as a result its throughput worsens. The second place was taken by the in-the-middle packet size, so it is safe to say that the more the packet size increases, the more the network throughput worsens.

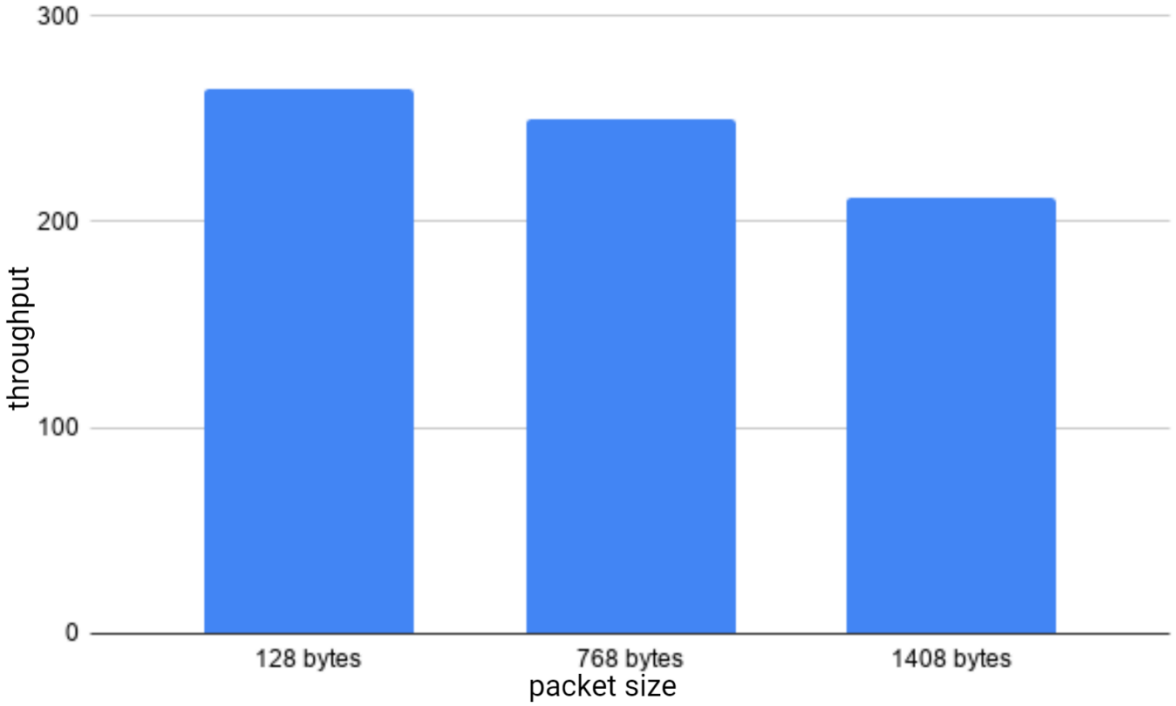


Figure 6.53: 2mbps - 15 nodes – throughput

In the scenario where the nodes were increased from 10 to 15, we saw similar results, although the network throughput was slightly higher across the board. We saw about a 7% increase when the packet size was 128 bytes, same with the packet size of 768 bytes and about a 5% increase when the packet size was 1408 bytes. So, although the number of nodes increased by 50%, the network throughput only saw a slight, but still measurable, increase.

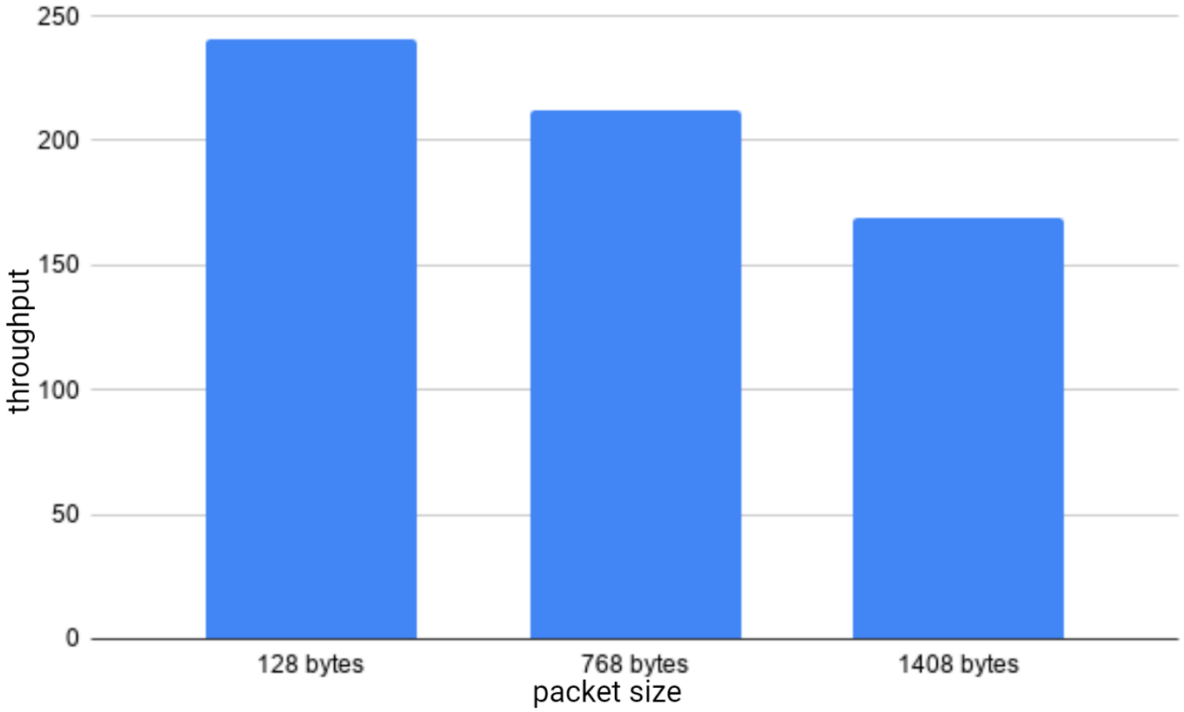


Figure 6.54: 2mbps - 25 nodes – throughput

With 25 nodes, the results were still, for the most part, unchanged. The smallest packet size still provided us with the best throughput. What changed however, were the values we generated. In the scenario where the packet size was 128 bytes, we observed about a 9% decrease going from 15 to 25 nodes, in the scenario where the packet size was 768 bytes, we observed about a 15% decrease and in the scenario where the packet size was 1408 bytes, we observed about a 20% decrease. These results show that 25 nodes are starting to put a strain to our network, reducing its throughput. When the nodes increase to 50 in the next graph, we expected an even bigger drop in the network throughput.

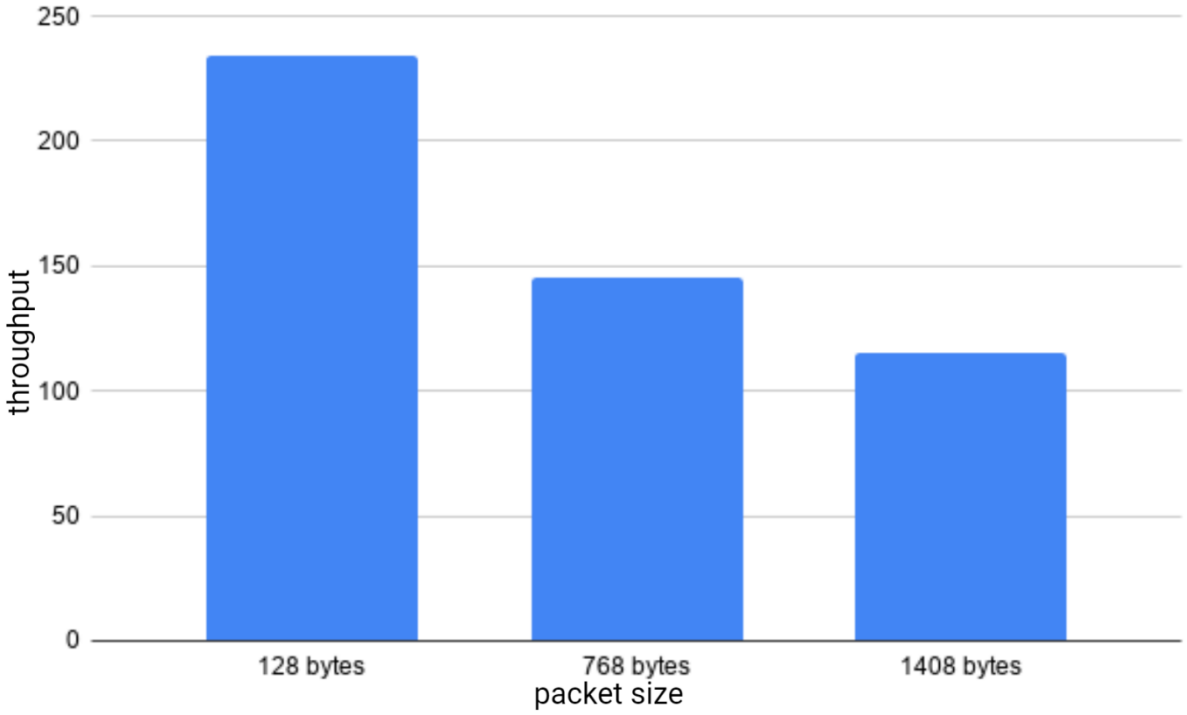


Figure 6.55: 2mbps - 50 nodes – throughput

Indeed, the drop this time was even more significant, thanks to the congestion a high number of nodes can produce. The drop was about 3% when the packet size was set to 128 bytes, about 32% when it was set to 768 bytes and about 32%, once again, when it was set to 1408 bytes. In general, 50 nodes and a packet size of 1408 bytes produced the lower value for our network throughput out of all the network scenarios.

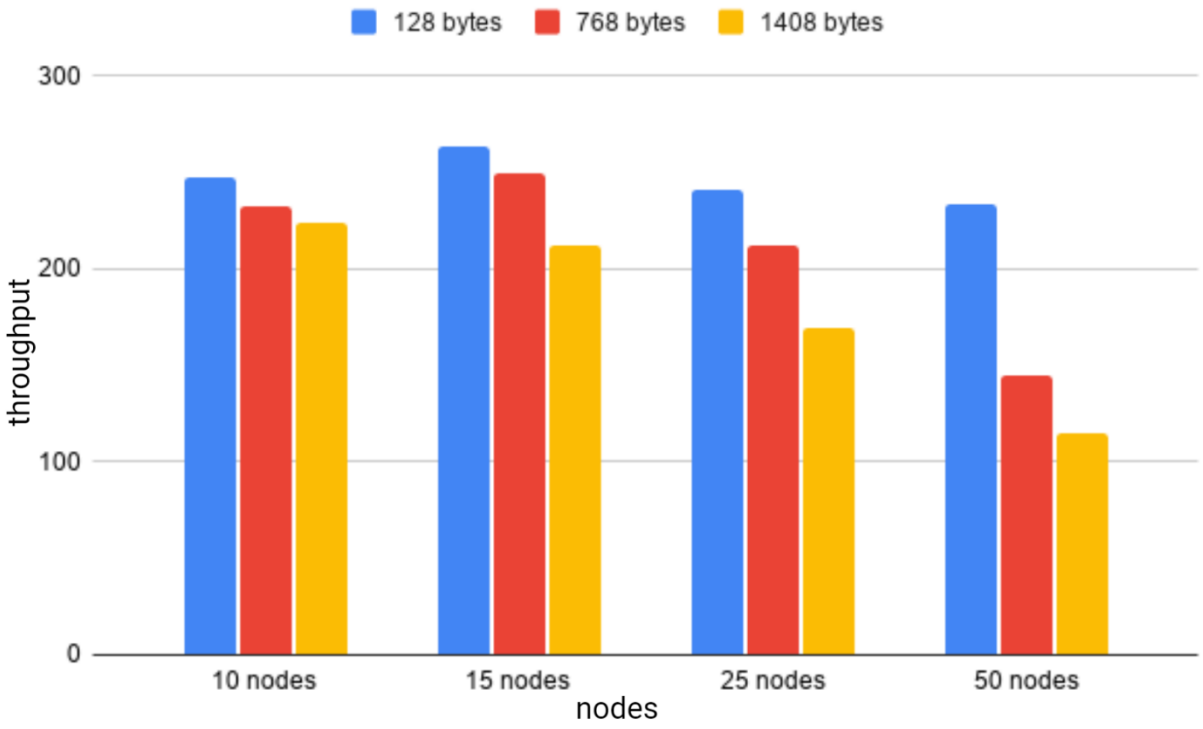


Figure 6.56: Throughput - 2mbps

Finally, another metric worth taking a closer look into is the throughput. Taking a quick look at the graph is all it takes to deduce that the scenario with the packet size of 128 bytes and the 15 nodes has the best throughput, with the scenarios with the 15 nodes and a packet size of 768 and the scenario with the 10 nodes and a packet size of 128 bytes following closely behind. Generally, the smallest packet size (128 bytes) across all the various number of nodes leads to the best throughput results, and as the packet size keeps increasing, the throughout keeps decreasing. The worst possible combination for the throughput results is the highest number of nodes and the highest packet size, as it affects the RTT to a great degree, which in turn affects the throughput.

6.3.2 11mbps

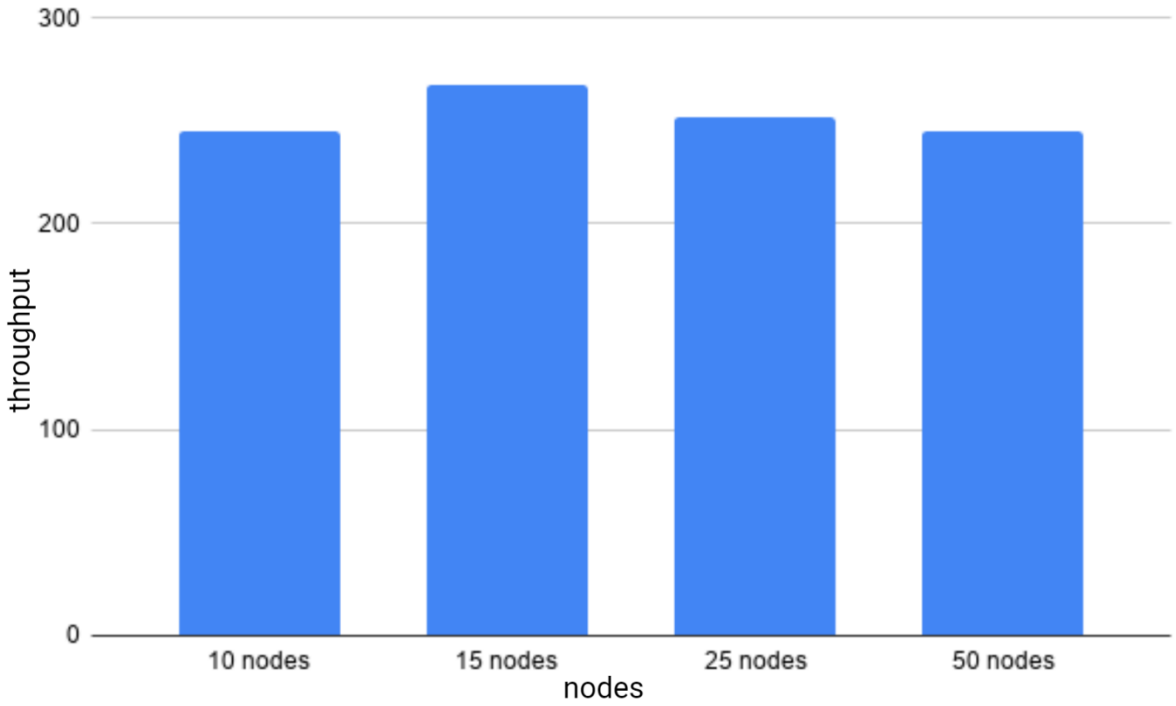


Figure 6.57: 11mbps – 128 bytes – throughput

Focusing now on the network throughput, the increase in the bitrate from 2mbps to 11mbps resulted in a 0.8% decrease in the scenario with the 10 nodes, in a 1.1% increase in the scenario with the 15 nodes, in a 4.6% increase in the scenario with the 25 nodes and in a 4.7% increase in the scenario with the 10 nodes. As we can see, the results in the first two scenarios remained about the same, since signals us that the network is not bottlenecked by the small bitrate, so the increase is of no use. In the scenarios where the number of nodes is higher, we can start to see some difference.

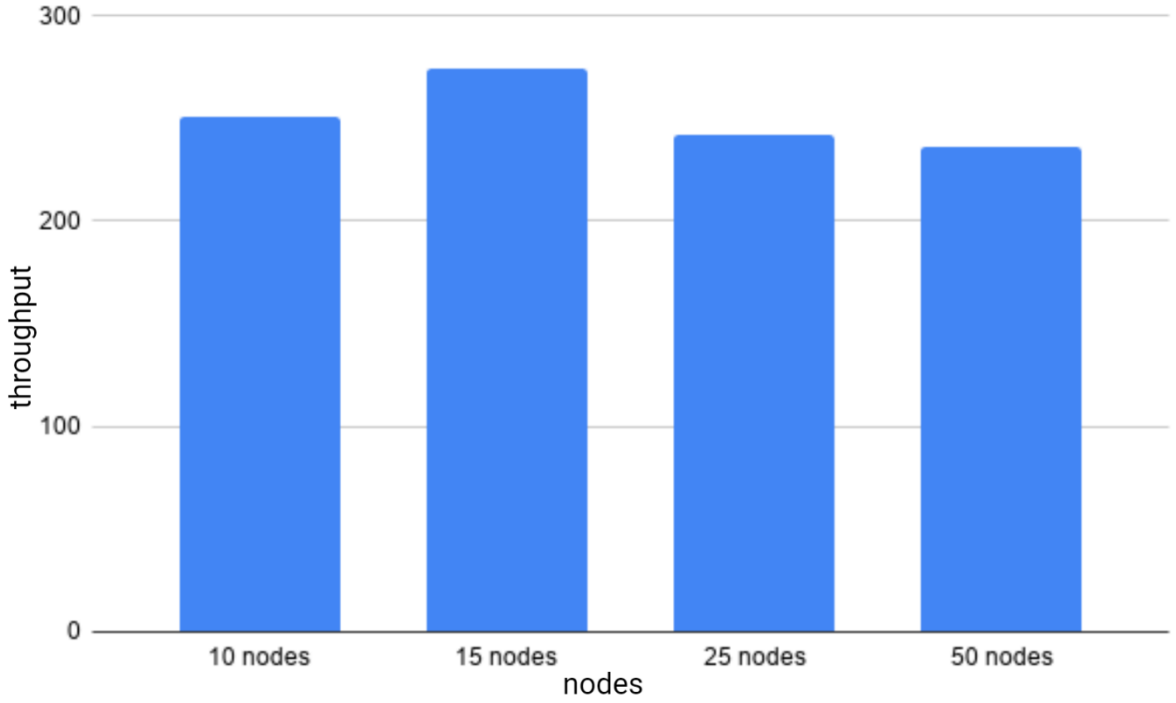


Figure 6.58: 11mbps – 768 bytes – throughput

Changing the packet size to 768 bytes, the throughput increased by 2.5% in the scenario with the 10 nodes, increased by 2.6% in the scenario with the 15 nodes, decreased by 4% in the scenario with the 25 nodes and decreased by 3.6% in the scenario with the 50 nodes. Changing the bitrate to 11mbps, the throughput increased by 7.7% in the scenario with the 10 nodes, it increased by 9.6% in the scenario with the 15 nodes, it increased by 14% in the scenario with the 25 nodes and it increased by 63% in the scenario with the 50 nodes.

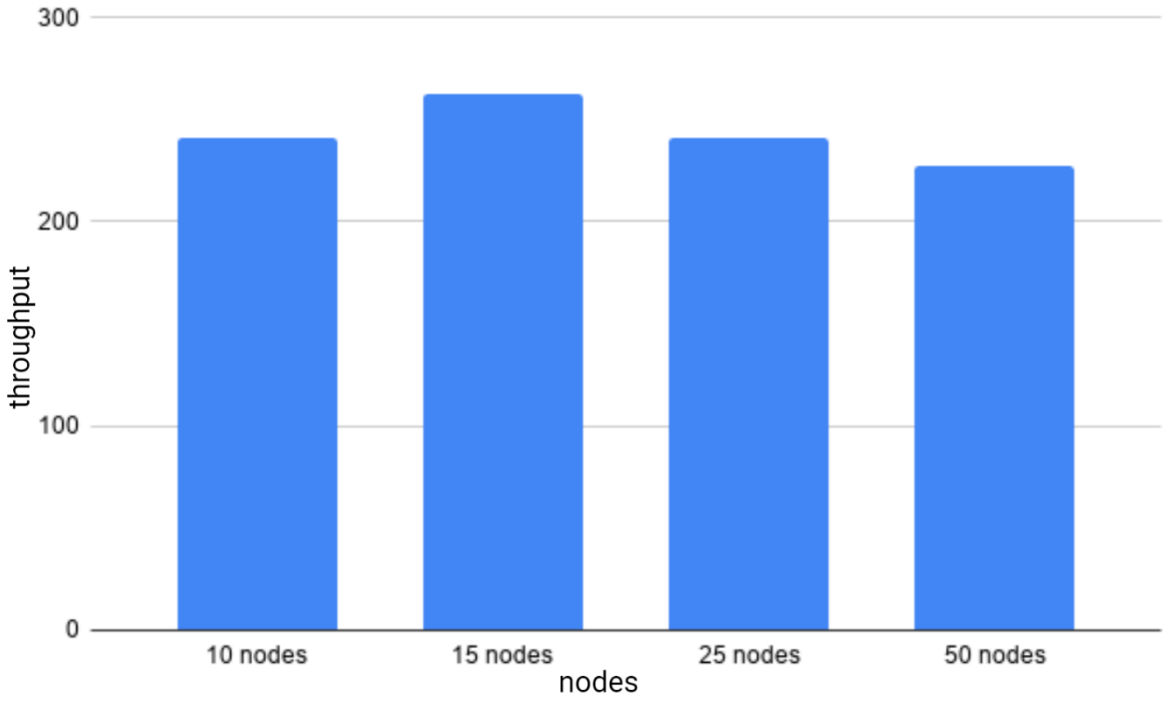


Figure 6.59: 11mbps – 1408 bytes – throughput

Changing the packet size to 1408bytes, the throughput decreased by 4% in the scenario with the 10 nodes, decreased by 4.4% in the scenario with the 15 nodes, decreased by 0.4% in the scenario with the 25 nodes and decreased by 3.8% in the scenario with the 50 nodes. Changing the bitrate to 11mbps, the throughput increased by 7.6% in the scenario with the 10 nodes, it increased by 24% in the scenario with the 15 nodes, it increased by 43% in the scenario with the 25 nodes and it increased by 97% in the scenario with the 50 nodes.

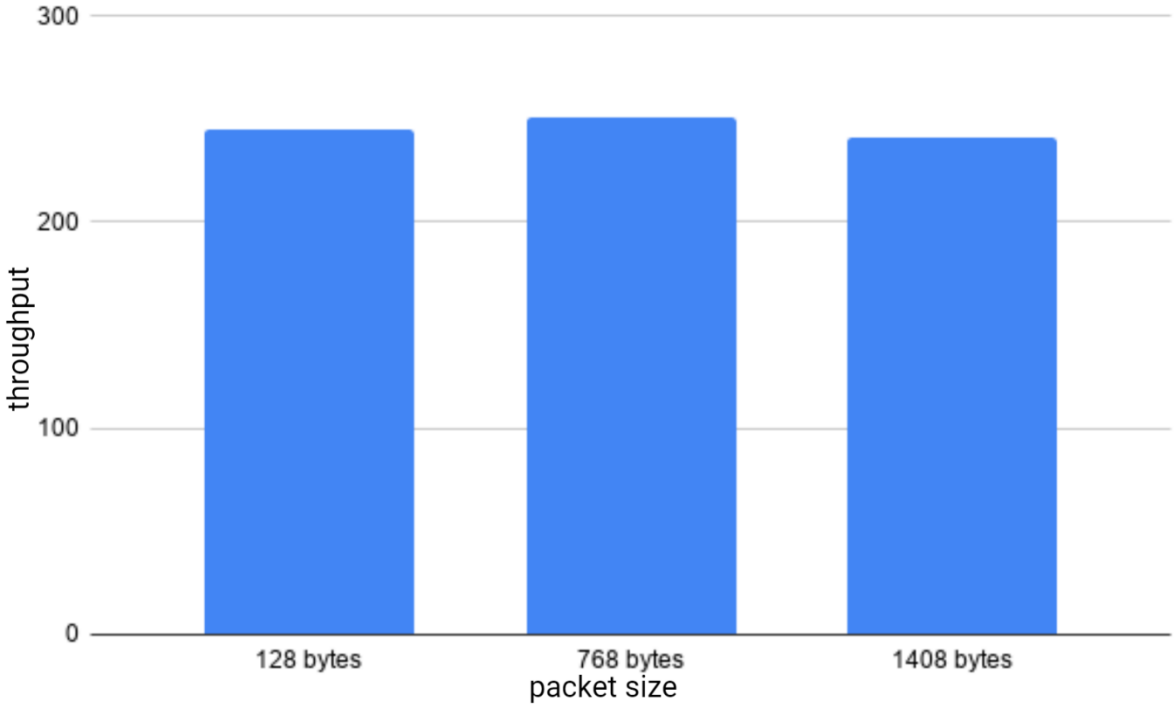


Figure 6.60: 11mbps – 10 nodes – throughput

Changing the bitrate to 11mbps while keeping the number of nodes the same, the throughput decreased by 0.8% in the scenario with the packet size of 128 bytes, it increased by 7.7% in the scenario with the packet size of 768 bytes and it increased by 7.6% in the scenario with the packet size of 1408 bytes.

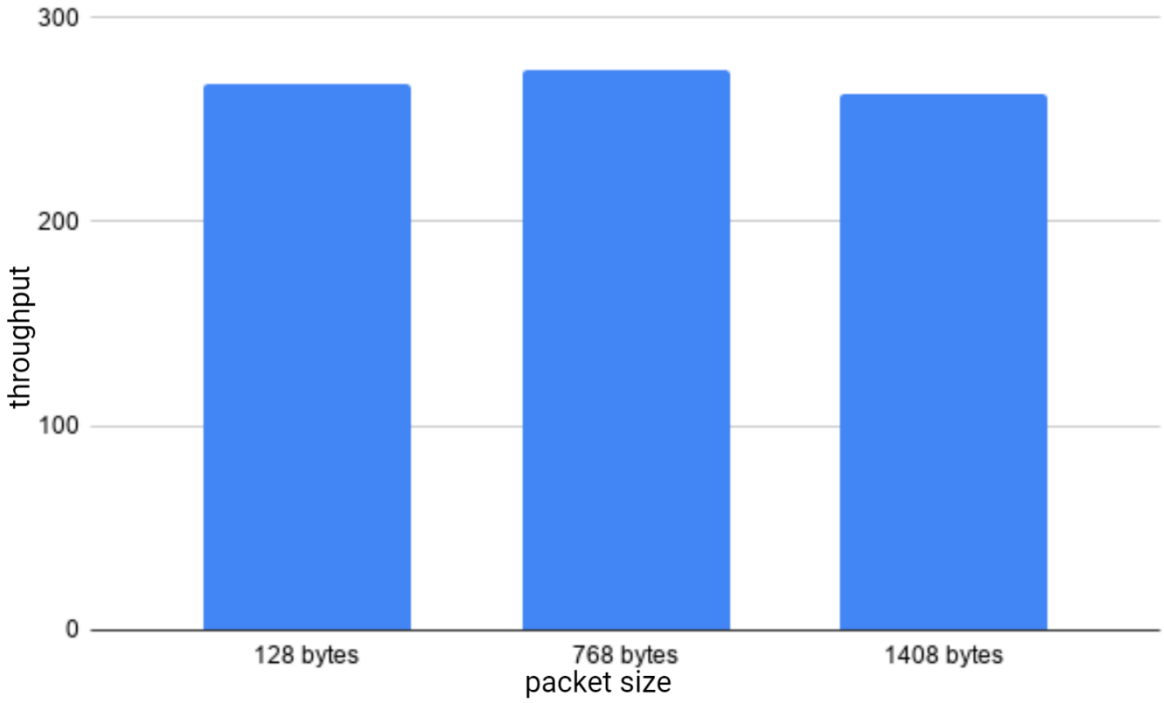


Figure 6.61: 11mbps – 15 nodes – throughput

Increasing the number of nodes to 15, we observed a 9% increase when the packet size is 128 bytes, a 9.2% increase when it is 768 bytes and an 8.7% increase when the packet size is 1408 bytes. The increase in the bitrate to 11mbps resulted in a 1.1% increase when the packet size is 128 bytes, a 9.6% increase when the packet size is 768 bytes and a 24% increase when the packet size is 1408 bytes.

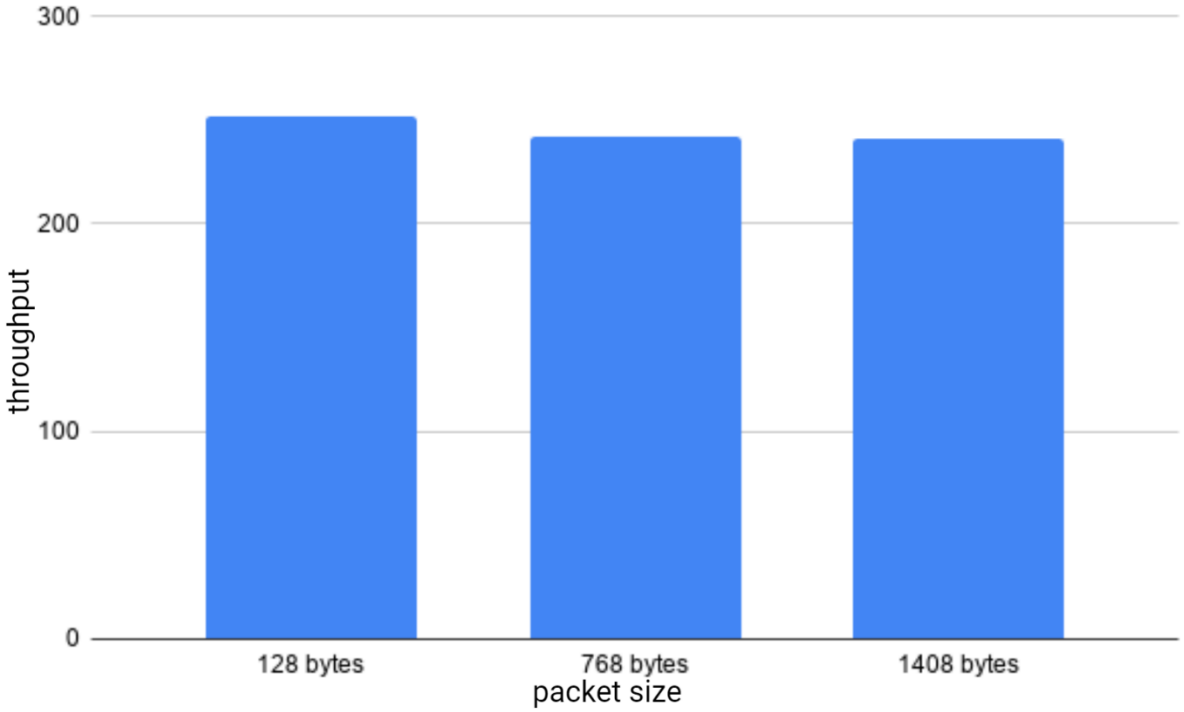


Figure 6.62: 11mbps – 25 nodes – throughput

Increasing the number of nodes to 25, we observed a 5.6% decrease when the packet size is 128 bytes, a 11.7% decrease when it is 768 bytes and an 8% decrease when the packet size is 1408 bytes. The increase in the bitrate to 11mbps resulted in a 4.6% increase when the packet size is 128 bytes, a 14% increase when the packet size is 768 bytes and a 43% increase when the packet size is 1408 bytes.

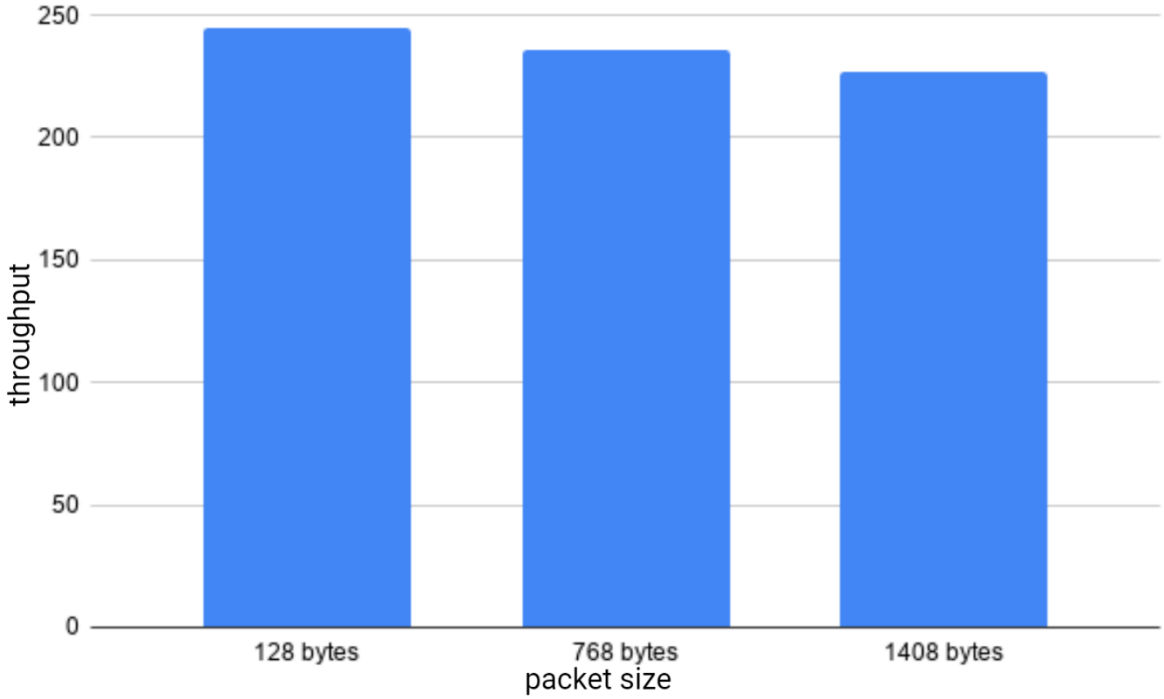


Figure 6.63: 11mbps – 50 nodes – throughput

Increasing the number of nodes to 50, we observed a 2.8% decrease when the packet size is 128 bytes, a 2.5% decrease when it is 768 bytes and a 5.8% decrease when the packet size is 1408 bytes. The increase in the bitrate to 11mbps resulted in a 4.7% increase when the packet size is 128 bytes, a 63% increase when the packet size is 768 bytes and a 97% increase when the packet size is 1408 bytes.

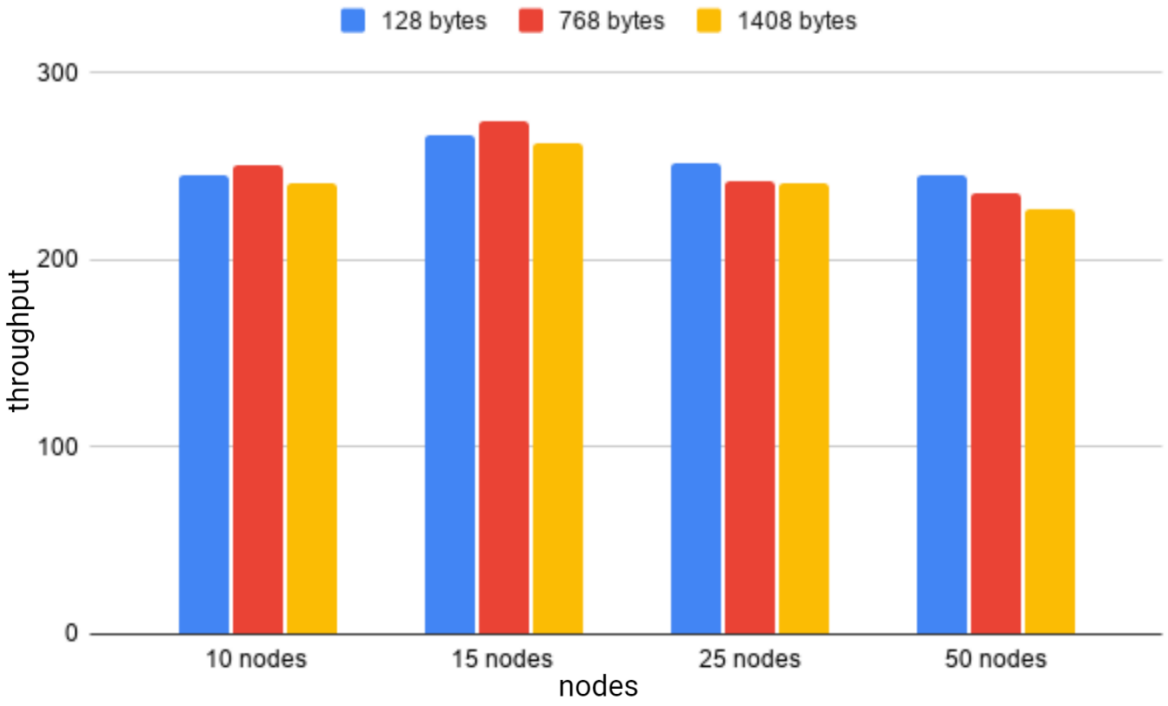


Figure 6.64: Throughput - 11mbps

Turning our attention to the throughput, we can easily see that the optimal configuration, at least as far as the throughput is concerned, is the scenario with the 15 nodes and the packet size of 768 bytes. The other two places to complete the top-3 are occupied by the scenario with the 15 nodes and the 128 bytes packet size and the scenario with the 15 nodes (again) and the 768 bytes packet size. The worst result was produced from the scenario with the 50 nodes and the packet size of 1408 bytes. The same applied just before when we studied the RTT, so it is no surprise to see this result, since the RTT and the throughput are closely related. Comparing the results with the scenarios with the bitrate of 2mbps, when the packet size is 128 bytes, the throughput actually decreased by a little bit, while in the other two scenarios (768 and 1408 bytes) the throughput saw a significant increase, as the network is no longer congested due to the low bitrate.

For the final round of benchmarks, we increased the bitrate once again to 54mbps.

6.3.3 54mbps

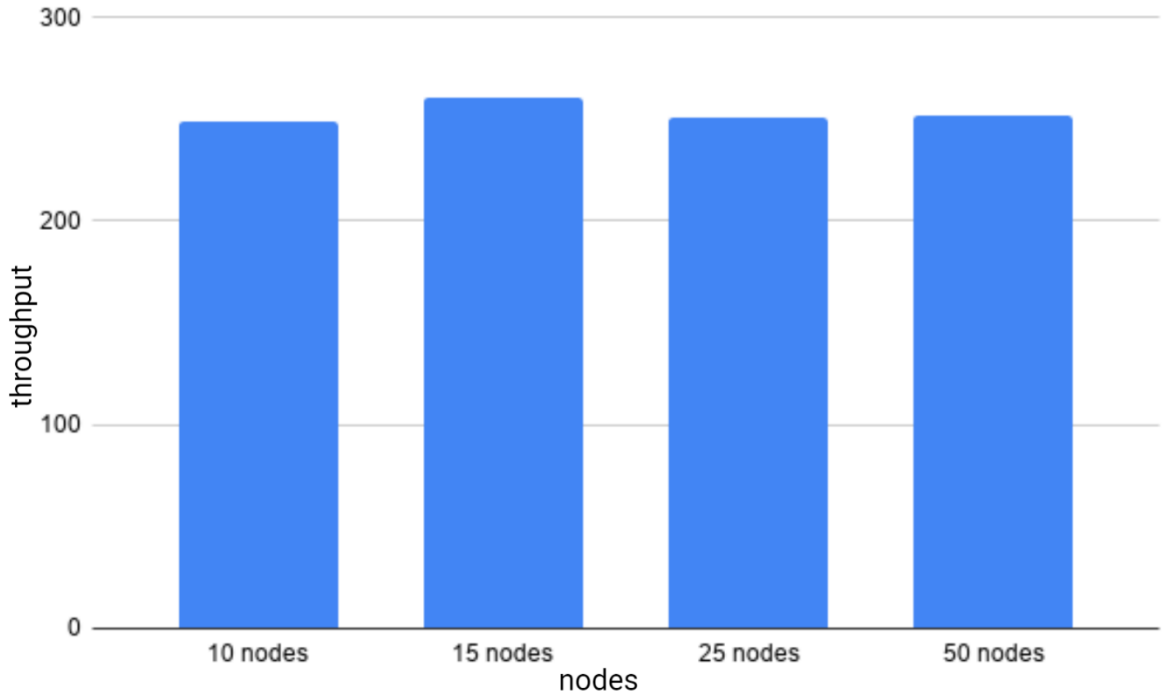


Figure 6.65: 54mbps – 128 bytes – throughput

The final metric we focused on was the throughput. The bump to 54mbps provided only slight changes to our results when the packet size was set to 128 bytes. In more detail, when the nodes were 10, we saw a 1.6% increase in the throughput, when they were 15, we saw a 2.6% decrease, when they were 25, we saw a 0.4% decrease and when they were 50, we saw a 2.9% increase.

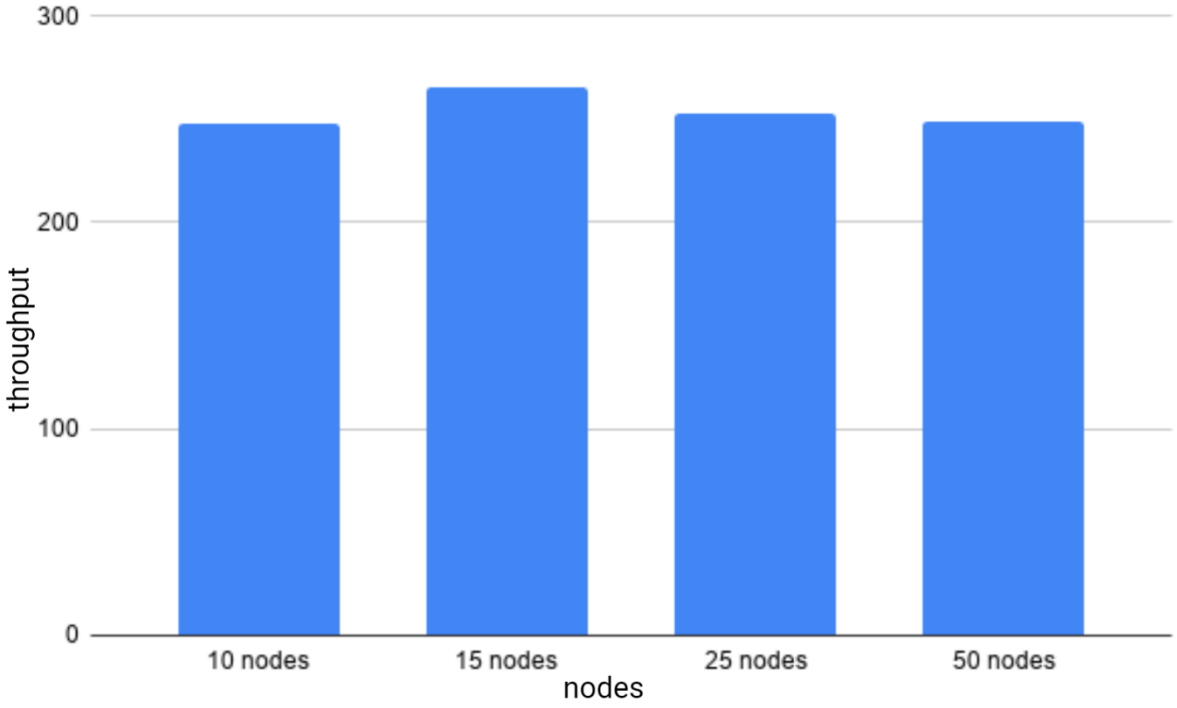


Figure 6.66: 54mbps – 768 bytes – throughput

Increasing the packet size to 768 bytes provided very small changes, a 0.4% decrease when the nodes were 10, a 1.9% increase when the nodes were 15, a 0.8% increase when the nodes were 25 and a 1.2% decrease when the nodes were 50. Increasing the bitrate to 54mbps resulted in a 1.2% decrease, a 3.3% decrease, a 4.5% increase and a 5.5% increase respectively.

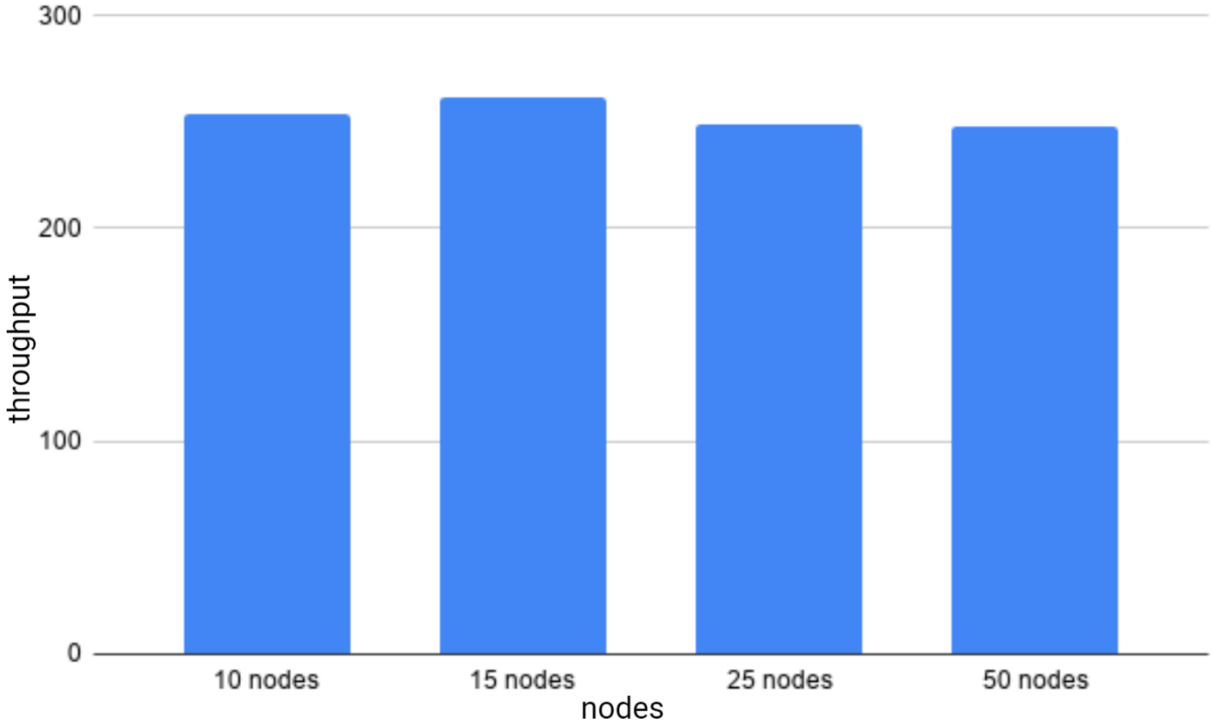


Figure 6.67: 54mbps – 1408 bytes – throughput

Increasing the packet size to 1408 bytes provided once again very small changes, a 2.4% increase when the nodes were 10, a 1.5% decrease when the nodes were 15, a 1.6% decrease when the nodes were 25 and a 0.4% decrease when the nodes were 50. Increasing the bitrate to 54mbps resulted in a 5.4% increase, a 0.4% decrease, a 3.3% increase and a 9.3% increase respectively.

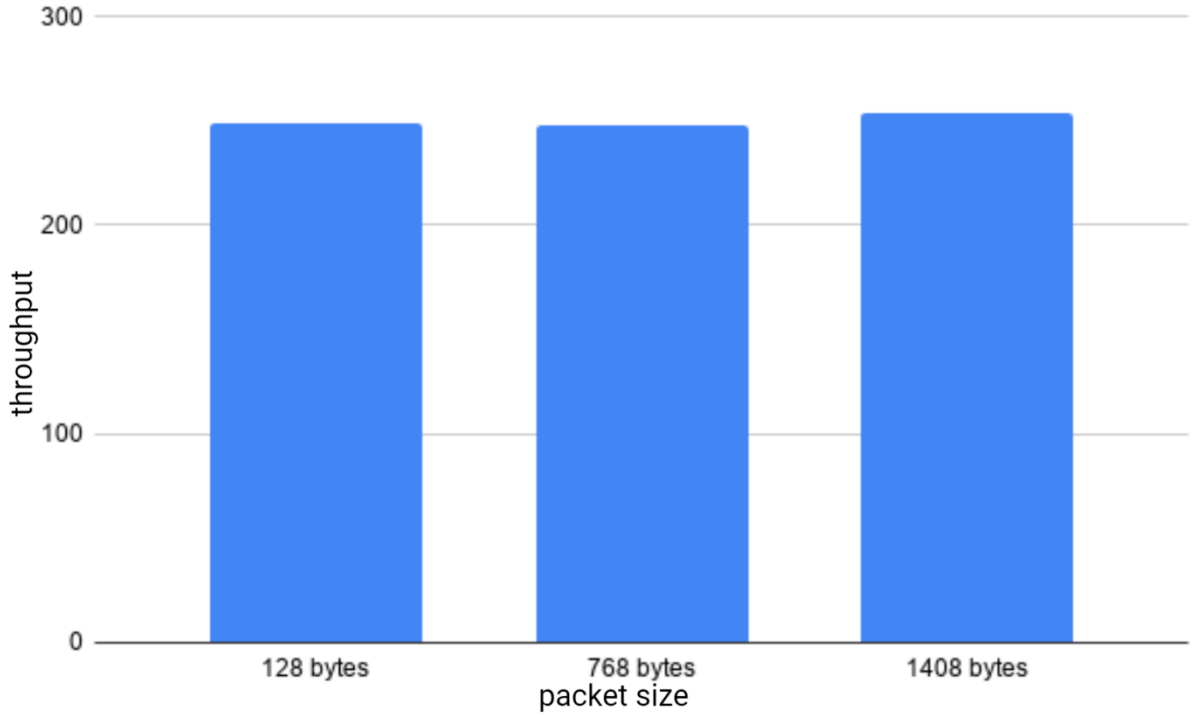


Figure 6.68: 54mbps – 10 nodes – throughput

Keeping the number of nodes static and changing the packet size while increasing the bitrate to 54mbps, we observed a 1.6% increase when the packet size was 128 bytes, a 1.2% decrease when the packet size was 768 bytes and a 5.4% increase when the packet size was 1408 bytes.

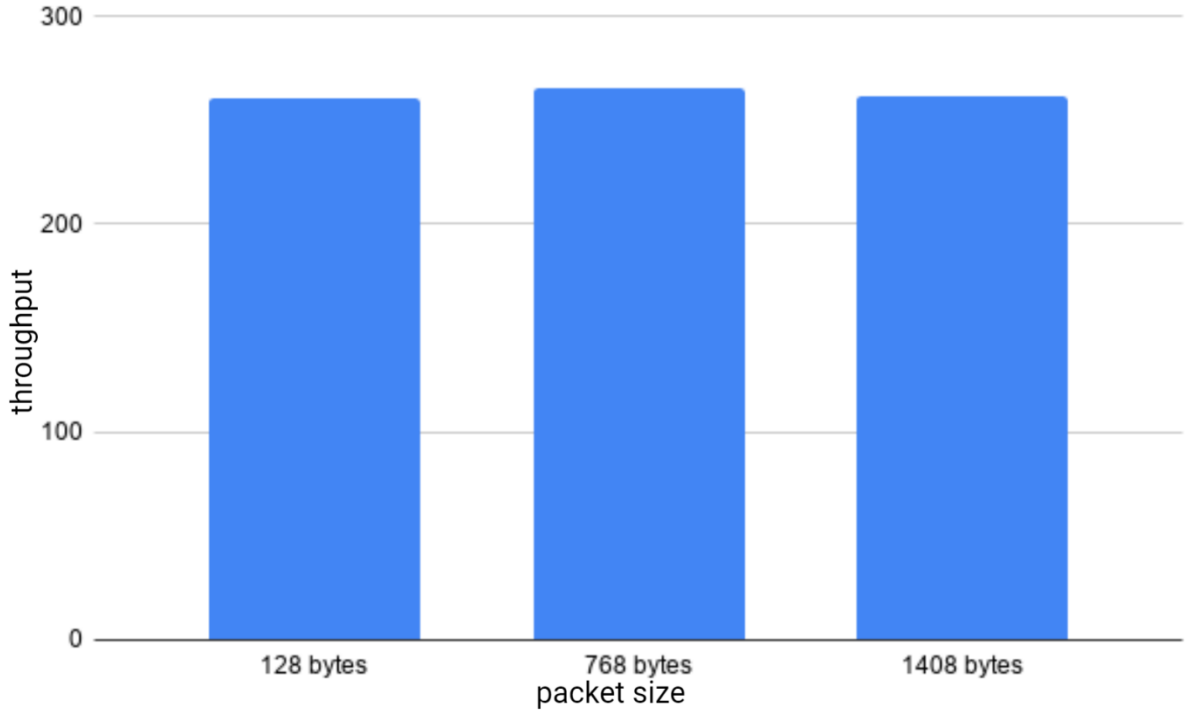


Figure 6.69: 54mbps – 15 nodes – throughput

Jumping to 15 nodes, we measured a 4.4% increase, a 6.9% increase and a 2.8% increase respectively. Jumping from 11mbps to 54mbps we measured a 2.6% decrease, a 3.3% decrease and a 0.4% decrease respectively.

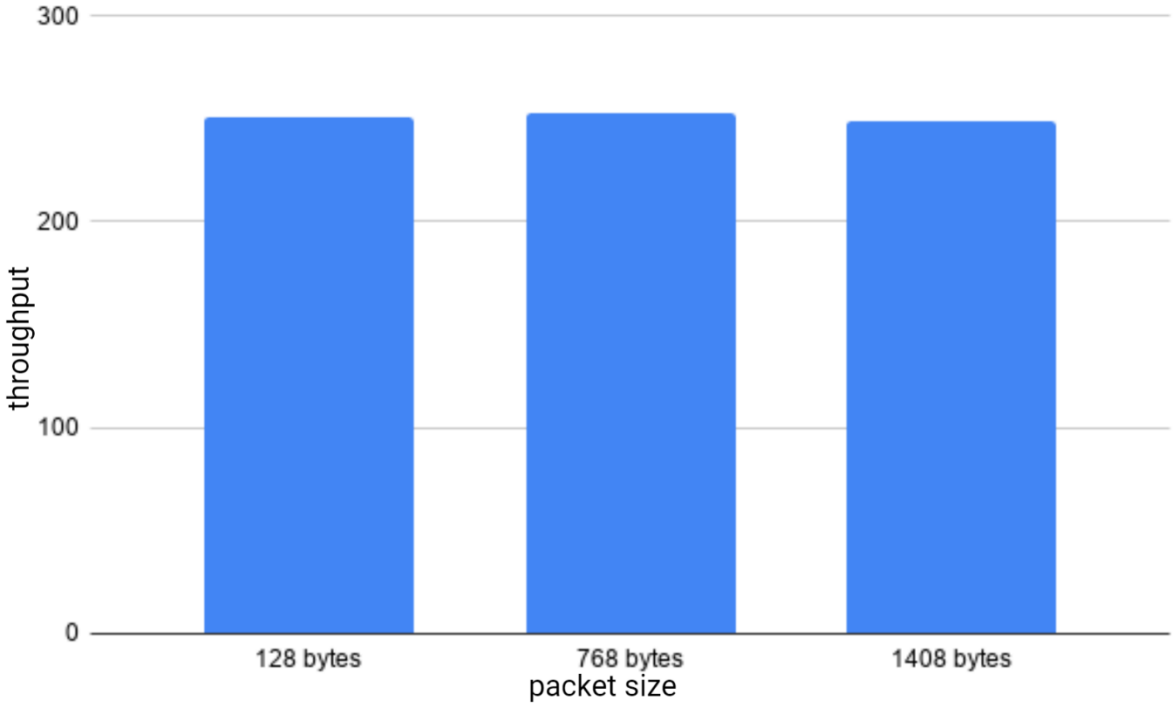


Figure 6.70: 54mbps – 25 nodes – throughput

Jumping to 25 nodes, we measured a 3.5% decrease, a 4.5% decrease and a 4.6% decrease respectively. Jumping from 11mbps to 54mbps we measured a 0.4% decrease, a 4.5% increase and a 3.3% increase respectively.

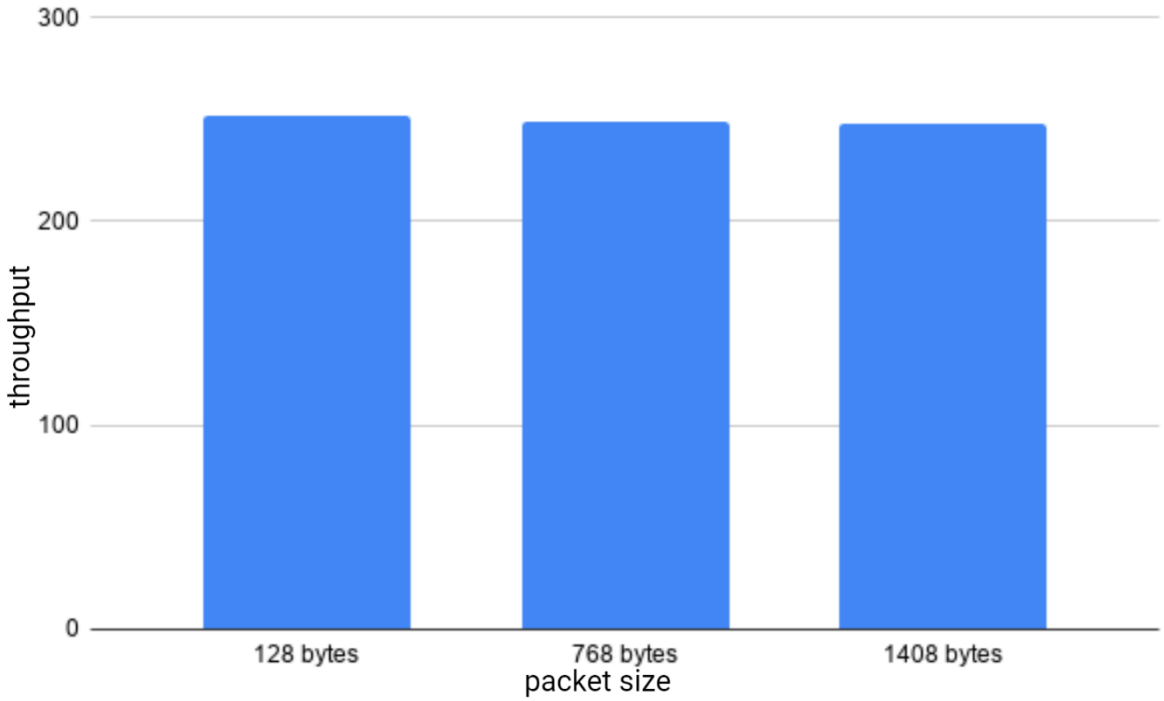


Figure 6.71: 54mbps – 50 nodes – throughput

Jumping to 50 nodes, we measured a 0.4% increase, a 1.6% decrease and a 0.4% decrease respectively. Jumping from 11mbps to 54mbps we measured a 2.9% increase, a 5.5% increase and a 9.3% increase respectively.

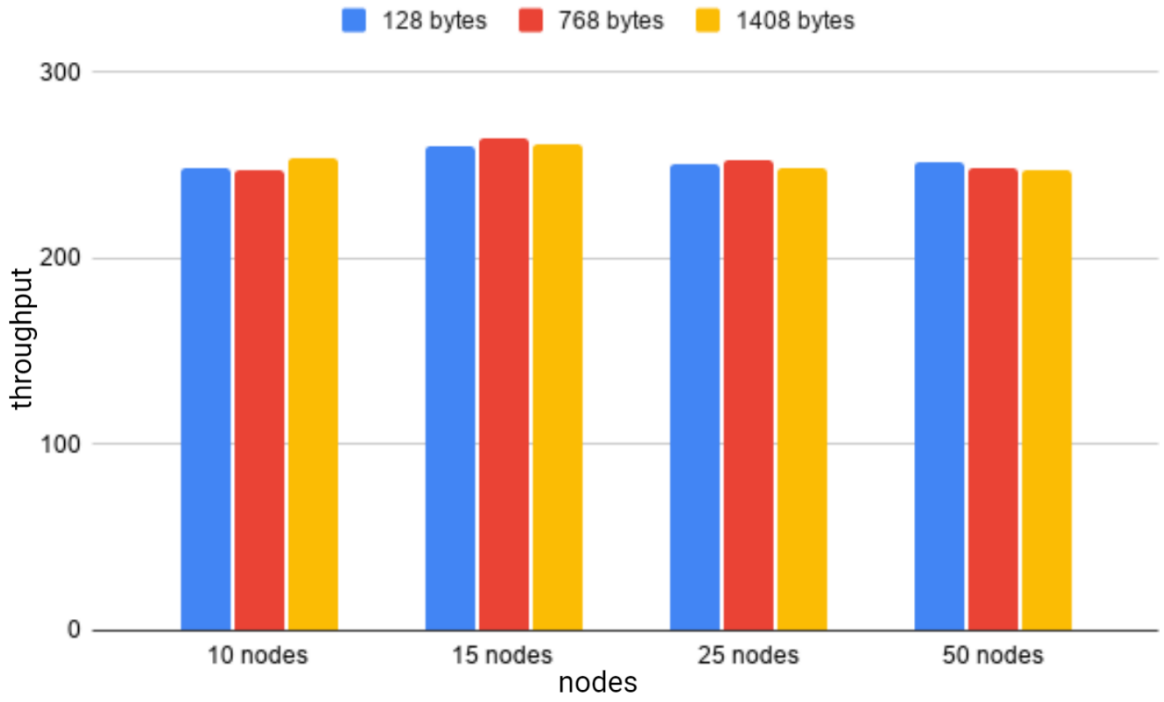


Figure 6.72: Throughput - 54mbps

Finally, studying the throughput results one last time, we can see that the best possible configuration for the best throughput result is the one with 15 nodes and a packet size of 768 bytes. The same was true when the bitrate was 11mbps. The results once again are not all that different, with the majority of changes being found in the scenarios where the nodes were 25 and 50. Here we can see that the throughput has increased slightly. We can also see that the results are weighted to an even larger degree, to the point where the packet size plays little role in the variation of the results. Again, this tells us that 54mbps is sufficient enough for the increase in the packet size and/or the number of nodes to not play a significant role in the results we got.

Chapter 7: Conclusions and/or suggestions for improvement

7.1 What we accomplished

Firstly, we made a very brief introduction to some basic networking concepts, like the different types of networks, how multiplexing and demultiplexing works and then we presented a brief study of the current status of the telecommunications. More specifically, we studied 5G and the Internet of Things, we presented some statistics and where things are heading for the future.

After that we went back in time and presented most of the IEEE 802.11 amendments, the changes they brought on, their most common use cases and their features. We focused more towards the latest protocols, as they are the ones being used now. At the end of that chapter, we also presented some of the alternative protocols to IEEE 802.11, namely Bluetooth, IEEE 802.15.7 and RFID. We presented their strengths, their weaknesses and where they are most typically used.

In the third chapter of this thesis, we introduced some of the most popular network simulators, OPNET - Riverbed Modeler, ns-3, OMNeT++ and J-Sim. We delved into detail about each one's system requirements, the installation process, how to create a new project and run a simulation, how they implement and handle wireless networks, their advantages and their disadvantages. In the end, we run benchmarks to measure their system resources utilization. We used the same scenario in all cases: AODV was used as the routing protocol, the simulation area was 650 m x 650 m, the packet size was 512 KB, the simulation time was 500 seconds and the number of nodes varied between 400, 800, 1200, 1600 and 2000. The host mobility was set to static, to eliminate any randomness to the end results due to the random mobility of the hosts. After running all 20 benchmarks, we came to the conclusion that ns-3 was the least demanding as far as RAM usage is concerned, with J-Sim, OPNET – Riverbed Modeler and OMNeT++ following behind. As for the CPU usage, OPNET – Riverbed Modeler achieved 100% utilization a bit earlier than ns-3 did, while J-Sim and OMNeT++ only reached a maximum of 24% CPU utilization.

In the fourth chapter, we set out to compare AODV, DSDV and DSR in OMNeT++ and find out which one is better in regards to the packet error rate, the queueing time and the minimum SINR. A total of 15 benchmarks were run, 5 for each routing protocol (one with 10 nodes, one with 20 nodes, one with 30 nodes, one with 40 nodes and, finally, one with 50 nodes). The results showed that AODV performed better that the rest of the routing protocols up until (and including) the 40 nodes mark as far as the packet error rate is concerned. With 50 nodes, DSDV performed slightly better than AODV, while DSR displayed the worst performance out of the three. In regards to the queuing time, DSR showed better performance for the first two rounds of benchmarks, but AODV showed better performance at the last three. It is also worth pointing out that DSR and DSDV were fairly consistent, while AODV showed quite a lot of inconsistency in its results. Finally, we studied the minimum SINR. DSDV displayed, for the most part, better results across the benchmarks, apart from the scenario with the 50 nodes, which came second to DSR. However, when DSDV was ahead, it was by a significant margin, like in the run with the 30 nodes, so it is safe to state that DSDV performed better.

In the fifth chapter, we presented CMM and SOLAR, which are synthetic mobility models that simulate the mobility of the nodes in an ad-hoc network. We delved into detail into how they work and then we set up a way to benchmark them to find out which one performs better. The winner would be selected

as the mobility model we would use in our custom network scenario in the following chapter. First we measured the minimum SINR, and we found out that the results were very close to one another, there was about a 3.7% difference between them. Then we measured the packet error rate, and the results were very helpful in helping us choose which model we were going to use. The results showed that SOLAR was 27.3% more likely to deliver a package with errors in comparison to CMM, so CMM was crowned as the winner.

In the sixth chapter, we created a custom network scenario trying to answer the following question: Would it be possible for a network that covers roughly the area of IHU's campus and has normal people who more around, going about their daily lives as the only network elements work, and, if so, what kind of results were to be expected? To get an answer to these questions, we chose an 850m x 850m area, and the number of nodes as well as the packet size and bitrate varied, to see where the "sweet spot" in regards to performance was. In total, we run 36 benchmarks for all the possible combinations and generated graphs for each and every one. We studied the effect of bitrate and the number of nodes on the packet error rate, on the RTT and on the throughput.

7.2 Why we used IEEE 802.11g for our custom scenario instead of a newer protocol

The CMM and SOLAR models were developed in 2008, as can be seen from the list of commits in their GitHub project (https://github.com/ComNets-Bremen/Mobility-Models/commits/master). As a result, in order to be fully compatible with OMNeT++, it requires an older version, version 5.5.1, which was released in 2019. The same thing applies to the INET Framework, it requires version 4.1.1 which was released in 2019 as well, while the latest version at the time of writing is version 4.3.2, which is compatible with OMNeT++ version 6.0 pre 10 or later.

Being forced to use older versions of OMNeT++ as well as INET Framework meant that there weren't yet any official implementations for the latest and greatest IEEE 802.11 versions. We tried to use the latest versions of OMNeT++ and INET Framework, but the code for the CMM and SOLAR models was incompatible, and we would have to significantly rewrite most of the files, to even have a chance of them not throwing up errors, let alone work correctly.

In addition, using IEEE 802.11n was also not an option, since it implements the use of multiple antennas, which was not supported by the CMM and SOLAR models. Manually trying to implement multiple antennas resulted only in failure, so IEEE 802.11g was the most recent, compatible version of IEEE 802.11. This is why the maximum data rate used in our simulations was 54 Mbit/s.

7.3 Conclusions

In this section we are going to present the summary from all the results we got from all the various simulations we run for this thesis. Firstly, we compared Riverbed Modeler, ns-3, OMNeT++ and J-Sim in regards to their RAM and CPU usage given the same network scenario and their scalability. We concluded that ns-3 is the least demanding in regards to RAM usage, followed by J-Sim and Riverbed Modeler. OMNeT++ was found to be the most demanding for RAM usage. As for the CPU usage, Riverbed Modeler was found to be the most demanding, followed by ns-3 and OMNeT++. J-Sim reached the smallest percent of CPU utilization. In regards to how scalable each network simulator is, OMNeT++ came out on top, followed by Riverbed Modeler, ns-3 and finally J-Sim.

Next, we compared AODV, DSDV and DSR using the same network scenario in OMNeT++. First, we turned our attention to the packet error rate. We found that AODV resulted in the smallest values, followed by DSDV. DSR resulted in the biggest values and these values were very inconsistent, showing

big fluctuations. After that we studied the queueing time. Once again, AODV came out on top, followed this time by DSR. DSDV came last, although it is worth pointing out that during the 5 benchmark runs there were some instances in which they briefly switched places. Finally, we studied the minimum SINR. DSDV resulted in the biggest values, thus being the better performer. DSR came second by default, since AODV produced only null measurements every time we tried to run the simulation.

In the next section, we set out to compare CMM and SOLAR, the two routing protocols. SOLAR came out on top in regards to the minimum SINR, although with an admittedly small difference of 3,7%. The differentiating factor that made us pick CMM over SOLAR was the packet error rate. The results showed that CMM was 27,3% less likely to present a packet error, which is a considerable percentage amount.

In the final section of this thesis, which is where the bulk of the various network simulations took place, we studied the effect of bitrate and the number of nodes on the packet error rate, the RTT and the throughput. The first result we extracted was that after a certain point, decreasing the bitrate and/or increasing the number of nodes resulted in a spike in the packet error rate, which, of course, is no surprise since it is known that when a network becomes congested and is not able to handle the increased traffic and starts throwing up errors. This is evident from the results we extracted, when the bitrate was 2mbps and the number of nodes 50 (which basically means that it's the most demanding scenario for our network), the maximum packet error rate we measured was 0.84%. In comparison, when the bitrate was increased to 11mbps, the maximum packet error rate dropped to 0.351% and when the bitrate was increased one final time to 54mbps, the maximum packet error rate dropped further to 0.145%.

As for the RTT, when the bitrate was 2mbps and the number of nodes 50 (again, the most demanding scenario for our network), the RTT saw a huge spike, which we never observed since. When the bitrate was increased to 11mbps the RTT stayed well under 0.003s and when we increased the bitrate once again to 54mbps the RTT dropped again, to well under 0.025. In all the scenarios, apart from the one with the huge spike, our network proved capable of effectively handling all the traffic generated within our network.

Turning our attention to the throughput, when the bitrate was 2mbps, there were some scenarios when the throughput failed to reach 150, whereas most of the other scenarios were about 250. This shows us that at certain scenarios, our network becomes too congested to handle the traffic being generated effectively. As a result, its performance suffers. Increasing the bitrate to 11mbps and 54mbps solves this problem, where the throughput values are very close to one another across all the scenarios.

In summary, studying the packet error rate, the RTT and the throughput, the only scenarios where we observed a dip in our network's performance was when the bitrate was set to 2mbps. Jumping to 11mbps solved all our performance issues, and further jumping to 54mbps provided only a slight performance boost, and only at certain scenarios.

7.4 Future work

Due to time limitations (some of the network simulations took hours to be completed) we picked CMM over SOLAR for the routing protocol we would use in our custom network scenario. Of course, we based our decision on simulation results, which pointed to CMM as being the better performer, but it would be interesting to run all these simulations with SOLAR the same way we did with CMM to compare them more thoroughly and find out if there are specific network scenarios in which SOLAR would perform better than CMM.

Another step we could take to further refine our simulation results would be to make the increments in the packet size, the network speed and the number of nodes smaller, instead of having just three values for the packet size and the network speed and four values for the number of nodes. For example, we would like to be in a position to state that when the number of nodes is 21, the packet error rate is smaller than it is when the number of nodes is 22. This would allow us to really pinpoint the precise “sweet spot” for the packet error rate, the RTT and the throughput.

Finally, one more thing we could do, although extremely time consuming and difficult, would be to implement CMM and SOLAR to ns-3, Riverbed Modeler and J-Sim. That way, we could draw direct comparisons between them, and figure out if their results closely resemble one another or are way off. In the same fashion, we could try to make CMM and SOLAR compatible with the newest version of IEEE 802.11, instead of being forced to use an old amendment.

BIBLIOGRAPHY

[1] Wi-Fi NOW and Netradar, “Wi-Fi percentage of US smartphone traffic at 74%”, 2018. [Online]. Available: https://wifinowglobal.com/news-and-blog/wi-fi-percentage-of-us-smartphone-traffic-at-74-says-netradar/.

[2] Murat Miran Köksal, “A Survey of Network Simulators Supporting Wireless Networks”, Middle East Technical University, Ankara, 2008

[3] Hsin-Hung Hsieh, Bih-Hwang Lee, Huai-Kuei Wu, Wen-Pin Hsu and Hung-Chi Chien, “A Wireless Network Simulator Based on Design Patterns for WiMAX and LTE”, 2017 International Conference on Applied System Innovation (ICASI), 2017

[4] Mordor Intelligence LLP, “Fiber Optic Cable Market - Growth, Trends, and Forecasts (2020 - 2025)”, 2020. [Online]. Available: https://www.reportlinker.com/p05865746/Fiber-Optic-Cable-Market-Growth-Trends-and-Forecast.html.

[5] Adam Satariano (New York Times), “People think that data is in the cloud, but it’s not. It’s in the ocean.”, 2019. [Online]. Available: https://www.nytimes.com/interactive/2019/03/10/technology/internet-cables-oceans.html.

[6] Philipp Nattermann and Karolina Sauer-Sidor (McKinsey & Company), “The telecom sector in 2020 and beyond”, 2020. [Online]. Available: https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/the-telecom-sector-in-2020-and-beyond.

[7] Statista Research Department, “Mobile internet usage worldwide - statistics & facts”, 2021. [Online]. Available: https://www.statista.com/topics/779/mobile-internet/.

[8] Christian de Looper (Digital Trends), “What is 5G? Everything you need to know”, 2021. [Online]. Available: https://www.digitaltrends.com/mobile/what-is-5g/.

[9] International Data Corporation (IDC), “IDC Forecasts Worldwide 5G Connections to Reach 1.01 Billion in 2023”, 2019. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS45740019.

[10] S. O'Dea (Statista), “5G - Statistics & Facts”, 2021. [Online]. Available: https://www.statista.com/topics/3447/5g/.

[11] Laurence Goasduff (Gartner), “Gartner Predicts Outdoor Surveillance Cameras Will Be Largest Market for 5G Internet of Things Solutions Over Next Three Years”, 2019. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2019-10-17-gartner-predicts-outdoor-surveillance-cameras-will-be.

[12] 5G PPP Architecture Working Group, “View on 5G Architecture”, 2020

[13] Jeremy Horwitz (VentureBeat), “The definitive guide to 5G low, mid, and high band speeds”, 2019. [Online]. Available: https://venturebeat.com/2019/12/10/the-definitive-guide-to-5g-low-mid-and-high-band-speeds/.

[14] Samsung, “Samsung Breaks 5G Speed Record, Reaching 5.23Gbps”, 2021

[15] E.J. Violette, R.H. Espeland, R.O. DeBolt and F.K. Schwering, “Millimeter-wave propagation at street level in an urban environment”, in IEEE Transactions on Geoscience and Remote Sensing, vol. 26, no. 3, 1988

[16] Dave Evans (Cisco Internet Business Solutions Group), “The Internet of Things: How the Next Evolution of the Internet Is Changing Everything”, 2011

[17] CISCO, “VNI Complete Forecast Highlights”, 2016

[18] US National Intelligence Council, “Global Trends 2040”, 2021

[19] P. Suresh, J. Vijay Daniel, Dr.V.Parthasarathy and R.H. Aswathy, “A state of the art review on the Internet of Things (IoT)”, 2014 International Conference on Science Engineering and Management Research (ICSEMR), 2014

[20] Ian Tiseo (Statista), “Global E-Waste - Statistics & Facts”, 2020. [Online]. Available: https://www.statista.com/topics/3409/electronic-waste-worldwide/.

[21] Don Torrieri, “Principles of Spread-Spectrum Communication Systems”. Springer, 2018

[22] IEEE, “IEEE 802.11-2012 - 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”. [Online]. Available: https://standards.ieee.org/standard/802_11-2012.html.

[23] George Vlantis, “TGn SB2: Presentation for 40MHz Coexistence CIDs”, 2009

[24] Keith Shaw, “The draft-n controversy”, 2006

[25] Jon Rosdahl, Bruce Kreamer and Adrian Stephens, “Plenary Presentation from WG11 to 802 EC”, 2009

[26] Project Zero team (Google), “Over The Air: Exploiting Broadcom’s Wi-Fi Stack (Part 1)”, 2017. [Online]. Available: https://googleprojectzero.blogspot.com/2017/04/over-air-exploiting-broadcoms-wi-fi_4.html.

[27] Vijay B T and Malarkodi B, “A study of IEEE 802.11aa”, 2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT), 2017

[28] Pablo Salvador, Luca Cominardi, Francesco Gringoli and Pablo Serrano ,“A First Implementation and Evaluation of the IEEE 802.11aa Group Addressed Transmission Service”, ACM SIGCOMM Computer Communication Review, 2014

[29] Md. Alimul Haque, Pritam Kumar and Amrendra Kumar Singh, “IEEE 802.11ac: 5th Generation Wi-Fi Networking”, 2012

[30] CISCO, “IEEE 802.11ax: The Sixth Generation of Wi-Fi White Paper”, 2020. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/wireless/white-paper-c11-740788.html.

[31] National Instruments, “Introduction to 802.11ax High-Efficiency Wireless”, 2020. [Online]. Available: https://www.ni.com/en-us/innovations/white-papers/16/introduction-to-802-11ax-high-efficiency-wireless.html

[32] Samsung Electronics, Electronics and Telecommunications Research Institute, Visible Light Communications Consortium and University of Oxford, “Visible Light Communication: Tutorial”, 2008

[33] Ngoc Quan Pham, Vega Pradana Rachim and Wan-Young Chung, “High-accuracy VLC-based indoor positioning system using multi-level modulation”, Optics Express Vol. 27, Issue 5, pp. 7568-7584, 2019

[34] Sebastian Rampfl, “Network Simulation and its Limitations”, 2013

[35] Anand Nayyar, “Flying Adhoc Network (FANETs): Simulation Based Performance Comparison of Routing Protocols: AODV, DSDV, DSR, OLSR, AOMDV and HWMP”, 2018 International Conference on Advances in Big Data, Computing and Data Communication Systems (icABCD), 2018

[36] Alexey V. Leonov, George A. Litvinov, “Simulation-Based Performance Evaluation of AODV and OLSR Routing Protocols for Monitoring and SAR Operation Scenarios in FANET with Mini-Uavs”, 2018 Dynamics of Systems, Mechanisms and Machines (Dynamics), 2018

[37] Anh Nguyen, Egemen K. Çetinkaya and James P.G. Sterbenz, “Mobile Wireless Networking: Wireless Network Simulation with ns-3”, 2016

[38] Atta ur Rehman Khan, Sardar M. Bilal and Mazliza Othman, “A Performance Comparison of Network Simulators for Wireless Networks”, 2012 IEEE International Conference on Control System, Computing and Engineering, 2012

[39] Jiming Chen, Jianhui Zhang, Weiqiang Xu, Lei Shu and Youxian Sun, “The Development of A Realistic Simulation Framework with OMNeT++”, 2008 Second International Conference on Future Generation Communication and Networking, 2008

[40] Arslan Musaddiq, Fazirulhisyam Hashim, “Multi-hop Wireless Network Modelling Using OMNET++ Simulator”, 2015 International Conference on Computer, Communications, and Control Technology (I4CT), 2015

[41] Hassen Redwan Hussen, Sung-Chan Choi, Jong-Hong Park, Jaeho Kim, “Performance Analysis of MANET Routing Protocols for UAV Communications”, 2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN), pp. 70-72, 2018

[42] Fei Yu, “A Survey of Wireless Sensor Network Simulation Tools”, 2011

[43] Harsh Sundani, Haoyue Li, Vijay Devabhaktuni, Mansoon Alam and Prabir Bhattacharya, “Wireless Sensor Network Simulators: A Survey and Comparisons”, International Journal Of Computer Networks, 2011

[44] Dj. Pesic, Z. Radivojevic and M. Cvetanovic, “A Survey and Evaluation of Free and Open Source Simulators Suitable for Teaching Courses in Wireless Sensor Networks”, 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 895-900, 2017

[45] C. Perkins, E. Belding-Royer and S. Das, “Ad hoc On-Demand Distance Vector (AODV) Routing”, 2003