DESIGN AND IMPLEMENTATION OF INTERNET OF THINGS IN NANOSATS

by

MTHOMASEBE ADONIS

Thesis submitted in fulfilment of the requirements for the degree

Master of Engineering (Electrical Engineering)

in the Faculty of Engineering & the Built Environment

at the Cape Peninsula University of Technology

Supervisor: Dr Marco Adonis

Co-supervisors: Dr Laban Mwansa and Prof François Rocaries

Bellville

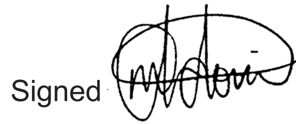
February 2024

CPUT copyright information

The thesis may not be published either in part (in scholarly, scientific or technical journals), or as a whole (as a monograph), unless permission has been obtained from the University.

DECLARATION

I, Mthomasebe Adonis, declare that the contents of this thesis represent my own unaided work, and that the thesis has not previously been submitted for academic examination towards any qualification. Furthermore, it represents my own opinions and not necessarily those of the Cape Peninsula University of Technology.

Signed  Date: 06 February 2024

ABSTRACT

A constellation of low earth orbit nanosatellites is often deployed to achieve what is difficult to do with a single much larger higher orbit satellite. This study looks at this common purpose constellation as a self-organising peer-to-peer network of equipotent, decentralised and distributed nodes. These nodes being much smaller in size and memory, are unable to individually store all the resources shared on the network. The Chord peer-to-peer algorithm is presented as a way for each node to keep only its allotted share of resources and thus reducing the memory load of each nanosat. Through consistent hashing of port and IP vs resource name, each resource can be looked up. Resource lookup simulations were done using internet connected MSP-430 wireless sensor nodes from the IoT-Lab platform. Other simulations were done using a developed (platform independent) Java application. The study found that the network can easily scale, without putting strain on the memory of each node, as each node only has to keep $log_2 n$ reference to other nodes. It was found that 66 such satellites in a mesh topology could be enough to cover the globe. UDP was considered as a transport layer protocol of choice. A lightweight encryption mechanism was found to provide security, by exchanging various key parameters instead of the encryption key itself. The study found that nanosats can be connected as Internet of Things devices without needing a different internet protocol (IP).

ACKNOWLEDGEMENTS

I wish to thank my supervisor Dr. Marco Adonis for his support and guidance in completing this work. A sincere thank you to Dr. Laban Mwansa and Prof. François Rocaries for their support at the beginning of this work. Thank you to AGI for allowing us to use the STK software for simulations. Thank you to the IoT Lab for letting us use their microcontroller nodes.

DEDICATION

I dedicate this work to my wife Lite and children Qhama and Niku. Thank you for your generous love and support.

TABLE OF CONTENTS

CLARIFICATION OF BASIC TERMS AND CONCEPTS x
LIST OF ABBREVIATIONS AND ACRONYMS: x
Chapter 1 : Introduction 1
1.1 Internet background 2
1.2 Nanosats 2
1.3 Importance of nanosatellites to South African government 3
1.4 Satellites background 3
1.5 Internet of Things 4
1.6 Research problem statement 6
1.7 Research goal 7
1.8 Research objectives 7
1.9 Research questions 7
1.10 Research significance 7
Chapter 2 : LITERATURE REVIEW 8
2.1 Space Communications Protocol Specifications (SCPS) 8
2.2 Connecting satellites to the Internet 9
2.2.1 Ground station 9
2.2.2 Global Educational Network for Satellite Operations (Genso) 9
2.2.3 Tracking and Data Relay System (TDRS) 11
2.2.4 Laser Communications Relay Demonstration (LCRD) 12
2.2.4 Space network ground network hybrid 13
2.3 Security 13
2.4 On board computer 14
2.5 IoT Technologies 14
2.5.1 Bluetooth 14
2.5.2 Wi-Fi 14
2.5.3 LoraWAN 14
2.5.4 Sigfox 15
2.5.5 Weightless 15
2.5.6 Enhanced Machine Type Communication (LTE-M) 15
2.5.7 Narrowband-IoT (NB-IoT) 15
2.5.8 Extended Coverage GSM-IoT 16
Chapter 3 : Research Design 17
3.1 Satellite coverage 17
3.2 Peer-to-peer system 17
3.3 Chord algorithm 18

3.4 IoT Lab ... 21
3.4.1 Setting up an experiment: ... 21
3.5 Java simulation ... 22
3.6 Node address ... 23
Chapter 4 : Results ... 25
4.1 Satellite coverage with STK ... 25
4.2 Network topology in Packet Tracer ... 27
4.3 IoT Lab ... 29
4.3.1 Chord in general ... 30
4.3.2 Static Chord Protocol ... 31
4.3.3 Node structure ... 32
4.3.4 Simple Look-up ... 32
4.3.5 Complex Look-up ... 33
4.3.6 Data Storage ... 35
4.3.7 Chord Tests ... 36
4.3.8 Dynamic Chord ... 40
4.3.9 Creating the network ... 40
4.3.10 Asking the network to join it ... 40
4.3.11 Joining the network ... 40
4.3.12 Leaving the network ... 40
4.3.13 Coping with crash ... 40
4.4 Java Simulation ... 41
4.4.1 UDP communication ... 43
4.4.2 TCP Connection ... 47
4.5 Node address ... 49
4.6 Security ... 51
4.6.1 Static key encryption ... 51
4.6.2 Randomly generated key encryption ... 52
4.6.3 Diffie Hellman key calculation ... 52
Chapter 5 : Discussion of results ... 54
5.1 Chord ... 54
5.2 Node addresses ... 54
5.3 Transport layer protocols ... 54
5.4 Security ... 55
5.5 Satellite coverage ... 55
Chapter 6 Conclusion ... 56
6.1 Future work ... 57

References 58

TABLE OF FIGURES

Figure 2-1: How GENSO will fit into the existing satellite communication framework 10
Figure 2-2: Dial tone model 10
Figure 2-3 : HumSAT Mission Concept 11
Figure 2-4 : TDRS constellation 12
Figure 2-5 : LCRD 12
Figure 3-1 : Chord logical ring 20
Figure 4-1 : Tracking Orbital line of ZACUBE 1 25
Figure 4-2 : Second track of orbital line of ZACUBE 1 26
Figure 4-3 : Third track of orbital line of ZACUBE 1 26
Figure 4-4 : Final track of orbital line of ZACUBE 1 27
Figure 4-5 : Packet Tracer Network topology of satellites – how each will be connected 27
Figure 4-6 : Topology for all 64 satellites covering the globe 28
Figure 4-7 : Each satellite providing internet for various devices 29
Figure 4-8 : Finger table and complex lookup method 30
Figure 4-9 : Complex lookup method 31
Figure 4-10 : Node structure in C 32
Figure 4-11 : Chord Request structure in C 32
Figure 4-12 : Node Array structure in C 33
Figure 4-13 : Finger Table structure in C 34
Figure 4-14 : Message structure in C 35
Figure 4-15 : Memory structure in C 35
Figure 4-16 : Temperature Data from Sensor n°3 36
Figure 4-17 : Communication succeed between node n°0 and node n°3 37
Figure 4-18 : Request temperature of sensor n°3 from sensor n°0 37
Figure 4-19 : Temperature response from sensor n°3 for sensor n°0 37
Figure 4-20 : Creation of message « test » and send it into the chord ring 38
Figure 4-21 : Save Message test to the responsible node 38
Figure 4-22 : Memory of message contained in the node 38
Figure 4-23 : Request the content of the message test 39
Figure 4-24 : Saving the content of the message test into the node 0 39
Figure 4-25 : The message test2 doesn't exist into the network 39
Figure 4-26 : Java application node 1 - Linux 41
Figure 4-27 : Node 2 of java application - Raspbian 41
Figure 4-28 : Node 3 of Java application - Raspbian 42
Figure 4-29 : Node 4 - Windows 42
Figure 4-30 Node 5 - Windows 43
Figure 4-31 Node 6 - Windows 43
Figure 4-32 : UDP datagram 44
Figure 4-33 : UDP request to join client message 44
Figure 4-34 Resending UDP messages at Application Layer until they are received 45
Figure 4-35 Server responds to 5th message 45
Figure 4-36 Broadcast message sent and received 46
Figure 4-37 All five broadcast messages on Wireshark 46
Figure 4-38 Port and IP address info of source and of broadcast 46
Figure 4-39 Packet data in plain text sniffed by unintended user 47
Figure 4-40 TCP socket establishing communication channel 47
Figure 4-41 Reading data from input stream of the channel 47
Figure 4-42 Writing data to output stream of the channel 48

Figure 4-43 TCP three way handshake to establish a channel 48
Figure 4-44 TCP data exchange only after handshake 49
Figure 4-45 Encrypting communication 51
Figure 4-46 Decrypting the message 51
Figure 4-47 Data sniffed on Wireshark by unintended user encrypted 51
Figure 4-48 Generating an encryption key randomly before data is sent 52
Figure 4-49 First key shared 52
Figure 4-50 Public key calculated and sent 52
Figure 4-51 Calculating the private key 53

CLARIFICATION OF BASIC TERMS AND CONCEPTS

CubeSat	A nanosatellite whose dimensions are 10 cm x10 cm x10 cm for one unit (1U) which can be extended to 1.5U, 2U, 3U, and 12U
Nanosat	A 1 to 10 kg satellite

LIST OF ABBREVIATIONS AND ACRONYMS:

CPUT	Cape Peninsula University of Technology
FSATI	French South African Institute of Technology
GSM IoT	Global System for Mobile communication Interne of Things
GSS	Ground Station Server
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IPSec	Internet Protocol Security
ISEB	International Space Education Board
ISP	Internet Service Providers
ISS	International Space Station
LEO	Low Earth Orbit
NAT	Network Address Translation
NASA	National Aeronautics and Space Administration
OWASP	Open Web Application Security Project
PDU	Protocol Data Unit
RAAN	Right Ascension of the Ascending Node
RIR	Regional Internet Registries
SACSA	South African Council for Space Affairs
SANSA	South African National Space Agency
SHA	Secure Hashing Algorithm
TCP	Transmission Control Protocol
TDRS	Tracking and Data Relay System
UDP	User Datagram Protocol

Chapter 1 : Introduction

This study focusses on the design and implementation of Internet of Things in nanosatellites. Available technologies and protocols are explored in supporting the implementation of IoT for nanosatellites. Nanosats have been in existence as early as the beginning of the space race between the USA and Russia (then USSR). In the early days, their existence was owed to launch-vehicle constraints on the USA side, where they did not have bigger vehicles like USSR did. These constraints were eventually overcome, and it became a norm to launch big satellites not only by both these countries, but by almost every country with space launching capabilities.

The recent resurgence of nanosatellite launches therefore are not because big ones cannot be launched, but because it is much cheaper to launch many nanosatellites from differing communities (e.g. academic researchers, business, governments etc.) in one launch and thereby sharing the cost of the launch. Even countries whose launching capabilities are not yet ready, like South Africa, can join with other countries launches at a fraction of the cost it would take to build their own, while not missing the present opportunity for research and business solutions.

Since the definition of the CubeSat standard in 1999, a lot of Nanosats that have been launched are in fact CubeSats. This standard encouraged companies to build commercially off-the-shelf CubeSat components knowing that any buyer can fit that in their nanosat. This means, for example, a university building a CubeSat, when they need a power supply unit or a radio communication system, they can just buy a space ready unit that they can plug in without worrying about whether the size will fit.

The possibilities that these nanosatellites have revealed has raised interest in universities, business and countries. The African Union's (AU) Pan African University has among its research post graduate programs an Institute of Space Sciences, which is hosted by the Cape Peninsula University of Technology (CPUT). This is in line with the AU's agenda 2063 defining "The Africa we want". Some the goals outlined in the agenda are ocean economy, with ports operations as a priority area; and climate resilient economies with natural disaster preparedness as its priority.

South Africa has a National Development Plan (NDP) aimed at eliminating poverty and inequality by 2030. Operation Phakisa (Phakisa means to hurry up) is the government's plan to speed up the realisation of the NDP. Through this operation, the government plans to unlock the economic potential of its oceans. The new marine domain awareness (MDA) CubeSats being built at CPUT are part of this operation.

CPUT in its own plan, identified space science and technology as one of its seven research focus areas. Not only has CPUT built and launched two CubeSats, it has also developed

commercially available VHF/UHF transceiver, S-band transmitter and S-band patch antenna to be used by other CubeSat developers.

Nanosatellites have a huge opportunity to provide solutions not only as single satellites but also as space wireless sensor networks. As wireless sensor networks, they join the Internet of Things gathering and processing data without human intervention.

1.1 Internet background

The story of the internet began when a global network of networks started in 1963 with a series of memorandums from J.C.R. Licklider to his colleagues about a vision of an Intergalactic Computer Network. Two years before this, Leonard Kleinrock published the first paper on packet switching. In 1967 Lawrence Roberts built ARPANET on these concepts, the first packet switched wide area computer network connecting scientists and engineers throughout the US.

In 1969, Kleinrock's lab was the first computer node to connect and send a message on this ARPANET. In 1974, Robert Kahn and Vince Cerf developed TCP/IP. The 4th version of TCP/IP IPv4 was invented in 1980 and became the standard used in ARPANET and later most other networks. IPv6 was invented in 1998 and is phasing out IPv4 as the protocol of the future. Unlike IPv4 which could allocate just over 4 billion (1 followed by 9 zeros) addresses, IPv6 can allocate up to 340 undecillions (1 followed by 36 zeros).

As internetworking of computers was being firmly set in place, machine to machine communication was also being developed. The natural evolution of this would be the internetworking of real-world physical things, and these things would have sensors and/or actuators. This would be turning real world things to cyber things. In a presentation he made in 1999 Kevin Ashton called this The Internet of Things, where the gathering of data did not need human intervention. He wanted to convince his company that the solution to their supply chain challenge would be to put RFID's on their consumer products. Internet of Things has been growing rapidly since. This growth has been compounded by the growing improvement in wireless communication systems. Some of these are Bluetooth, Wi-Fi, LoraWAN, Sigfox, Weightless, enhanced Machine Type Communication (LTE-M), Narrowband-IoT (NB-IoT) and Extended Coverage Global System for Mobile communication IoT (GSM-IoT).

1.2 Nanosats

There is a growing number of nanosats being launched worldwide by academic institutions, governments and private companies. Since design of CubeSats in 1999 by California Polytechnic State University and Stanford University, over 1300 CubeSats have been launched (Kulu, 2021). South Africa has had six successful satellite launches and more launches are scheduled including the cluster of CubeSats for the Phakisa project (Royi, 2022)

. In treating nanosats as IoT's this issue of security must be considered carefully. Spoofing and jamming would be some of what hackers could do to nanosats. How should these nanosats be like other IoT's ? Should IoT standards be modified to suit these nanosats? What would these modifications be?

FSATI at CPUT launched a nanosat TshepisoSat (Code name ZACUBE-1) in 2013 and as part of the Phakisa project they plan on launching more in the coming years (Zaidi and Van Zyl, 2017). This is in collaboration with the Department of Planning, Monitoring and Evaluation. The Department wants to use these to address poverty, unemployment by bringing resources to rural areas. An example of this would be cheaper satellite-based internet. Other uses would be ship tracking, fire tracking and ocean colour monitoring (Sulaiman *et al.*, 2022). In 2019, F'SATI launched ZACUBE-2 which is a 3U form factor CubeSat. South Africa has two CubeSats part of the QB50 programme, that launched in 2017, namely nSIGHT 1 (Wiid *et al.*, 2017) and ZA-AeroSat (Woldai, 2020).

1.3 Importance of nanosatellites to South African government

When Dr Val Munsami (now SANSA CEO), was still deputy director at the Department of Science and Technology, told parliament that it was important to use satellites to study the Southern Atlantic Anomaly (SAA), which is a dip in the Earth's magnetic field which could cause not only an upset to satellite electronics but also an increase in cancer cases due to radiation (Parliamentary Monitoring Group, 2011).

South Africa has two bodies under the Minister of Trade and Industry responsible for space activities. The first is the SACSA underpinned by the Space Affairs Act No. 84 of 1993 (South Africa, 1993). SACSA regulates space affairs, keeps a register of space entities and issues licences for space activities, e.g. ZACUBE-1's license. The second is the SANSA necessitated by South African Space Agency Act (Act No. 36 of 2008) to "provide for the promotion and use of space and co-operation in space-related activities, foster research in space science, advance scientific engineering through human capital, support the creation of an environment conducive to industrial development in space technologies within the framework of national government policy"(South Africa, 2008)

1.4 Satellites background

The story of man-made objects flying into space began in 1954 when the International Council of Science made a call for satellites to be launched from July of 1957 to map the Earth's surface. The Soviet Union was the first country to heed this call by launching SPUTNIK 1 (СПУТНИК is a Russian word for satellite) on October 4, 1957 from Baikonur in Kazakhstan. This satellite was a 2mm thick metal sphere of 58 cm diameter, weighing 84 kg with four antennas. It was launched into orbit by a Russian built R-7 rocket, the most powerful intercontinental ballistic missile in its day. The orbit was elliptical with the furthest point at 939

km and the nearest point 215 km from the surface of the earth. The resulting orbital period was 96 minutes.

As the world was stunned by this new breakthrough into space, a month later the Soviet Union sent Sputnik 2 into space, much larger and carrying a dog on a one-way mission. On January 31 of 1958, the United States successfully joined the space race by launching a javelin-shaped 13 kg Explorer 1. The race exploring space remained between these two countries until Canada and the United Kingdom joined successfully launching their satellites from the United States in 1962. By 1970 Italy, France, Australia, West Germany, Japan, China and India by 1975. South Africa only joined in 1999 when it launched SUNSAT (Stellenbosch University Satellite). It had however been tracking other countries satellites from as early as 1958 from its Hartebeesthoek station.

Though these satellites started small (like the 13kg Explorer 1), over time much bigger satellites were built like the International Space Station (ISS), which is bigger than a soccer field and over 400 000 kg. Building these became a long and expensive feat which needed careful thought. The high cost of mistakes in design can be seen in the explosion of the European Space Agency's Ariana 5 rocket where $7 billion and 10 years worth of development went up in flames within 40 seconds of launch. This is said to have been caused by a failure to convert a 64-bit float to a 16-bit float (as the 64-bit number to be converted was larger than the largest 16-bit number possible).

In an effort to make affordable space programme to universities, professors Bob Twiggs of Stanford University and Jordi Puig-Suari California Polytechnic State University developed specifications for CubeSats in 1999. These would be 10 cm x 10 cm x 10 cm per unit weighing not more than 1.33 kg. These could be totally built from commercially off the shelf components. They would be launched as secondary payloads of other mission launches. Some are launched from the ISS, and this reduces the cost even further, as these CubeSats don't have to pass the same vibration tests as those that would deploy en route to a mission. This is because they could be packed in bag with soft cushion, shielding them from launch induced vibrations.

1.5 Internet of Things

The field of IoT has grown significantly since the term was coined in 1999 (Madakam, Ramaswamy and Tripathi, 2015). The question in this day is no longer whether a device **can** be connected to the internet, but rather **should** it be connected. As an example, a farmer no longer needs to ask if a cow can be connected to the internet for its movements to be tracked, but rather should it be? (Is it ethical for a cow to have an IP address?).

Big technology companies have since been investing a lot of resources in IoT. Several of these have commercial IoT consumer products. At this stage, several of these products could

be considered nice to haves and not yet absolute necessities (like the way cell phones have now become a necessity). These range from Phillips Hue lights that can change colour when an airplane passes (these can now be programmed through IFTT web-based service)(Ronen and Shamir, 2016) to Japanese designed diapers that alert a caregiver to attend to the baby's bottom (Sen *et al.*, 2020). Most IoT devices however are industrial.

It had been estimated that 50 billion devices will be connected to the internet by year 2020 (Cisco, 2016). With such a growing number of internet-connected devices, hackers are also finding themselves with a lot more fruit to choose from. It is reported that in October of 2016 a hundred thousand IoT devices were hacked through a distributed denial of service attack (Park and Tyagi, 2017). Moving forward from here security must be an important focus of IoT research and development. The IoT Village, a connected devices student hacker competition hosted by DefCon in Las Vegas runs workshops on hacking connected devices like medical devices, home appliances, routers, and storage devices (Miloslavskaya and Tolstoy, 2019). Initiatives such as these contribute to better IoT security.

1.6 Research problem statement

Nanosats are like IoT devices, in size and power constraints - and in being application specific. In design and implementation of nanosats, these constraints must be kept in mind as suitable IoT protocols are investigated to connect these.

The physical size constraint brings about a memory size constraint. It is because of these constraints that this study must investigate how a constellation of nanosats can be connected to be a peer-to-peer network of nodes distributing resources.

These constraints should not be looked at in isolation, because a peer-to-peer constellation can fully replace a much more expensive large single satellite, with no such constraints. The self-organising nature of such a constellation means that nanosats can be launched and added to the constellation over a long period. The period can be long as the lifetime of the previously launched nanosats. In other words, if a constellation of nanosats is launched in 2013 and in 2020, they are still functional, more satellites can be added to that same constellation network.

This self-organising nature allows single nanosats to be added as budgets of organisations needing mission launches permits. This also means that additions to constellations can be done as problems arise. The time between the problem arising and the launch would obviously include the time it takes to design, build, test functionality and launch readiness, finding a launching vehicle.

One of the problems faced by South Africa is wildfires that can ravage a forest so quickly long before the nearest department of fire fighters is alerted. The challenges these forest fires bring include the destroyed environment, death and displacement of wild animals, danger to neighbouring community members and their properties, and the much more money and other resources that must be spent putting out the fire. Therefore, detecting these fires early, before doing much damage would be in the best interest of all parties. Adding nanosatellites to the constellation, equipped with the latest fire detecting sensors could detect the fire as quickly as it takes for a nanosatellite to pass over. A nanosatellite in the low earth orbit with a sensor that has a ninety-degree field of view can have a big enough swath (the area imaged earth's surface) big enough to cover the whole country at a time. At each pass, it can scan the whole country (and neighbouring countries) for fires and alert relevant authorities as soon it sees one.

The farming community faces challenges such as locust swarms eating produce. If their movements could be detected early, the results could be mitigated. Determining when plants are ripe by nanosatellite image processing could reduce waste of overripe produce. Similarly, geologists and mineral explorers could use satellite image sensors (such as near infrared sensors) to aid their work in discovering potentially mineral rich areas.

The area of healthcare also faces a challenge, though top healthcare facilities and experts exist in the country, yet they are inaccessible to a large population of the poor, even more so

for those in rural communities. Delivering health services using electronic communication means (e-health, telemedicine) would be welcome innovation for these challenges. The issue of the high cost of data sold by the five network providers in South Africa would also add a challenge to this. Therefore, the use of network of nanosatellites as backbone to provide internet could add impetus in solving these challenges.

With a coastline of nearly three thousand kilometres, South Africa is burdened with having to monitor the movements of ships in its waters. Ensuring that every ship is known and that there are no unaccounted-for ships (possibly pirates) is done by vessel tracking services using an automatic identification system (AIS). This vessel tracking along with collision detection could also be done by use of nanosatellites, there by possibly reducing the need for base stations which are unable to track out of range vessels.

In this study an infrastructure will be designed that will address the above

1.7 Research goal

To design and implement an infrastructure to implement Internet of Things for nanosats.

1.8 Research objectives

a. Review of technologies and protocols suitable for IoT in nanosats.
b. Design and propose a communication infrastructure for nanosats.
c. Assess IP security framework.

1.9 Research questions

a. Should nanosats be connected as Internet of Things?
b. How will these nanosatellites connect to the Internet?
c. What protocol should be used in connecting them?
d. What sensors will be used?
e. What will be the security considerations?

1.10 Research significance

The Internet of Things has been a growing area of research since 1999. This timeline coincided with the development of 10-cm cube shaped nanosats. The contribution of this study is to look at the nanosats with the lens of IoT. This is significant because, IoT is a fast-growing research area with a number of subcategories. Nanosat research would benefit from that.

Chapter 2 : LITERATURE REVIEW

In this section, the Internet of Things technologies and space-based technologies and protocols are reviewed. Though IoT is a much newer phenomenon, space-based communications dates decades earlier. These phenomena are reviewed in silos and not as a single tribal unit.

The various communication links between satellites and the Internet are reviewed. Common Internet of Things technologies are also reviewed to see their applicability in satellites.

2.1 Space Communications Protocol Specifications (SCPS)

The Consultative Committee for Space Data Systems (CCSDS) published the SCPS to improve Internet protocols for space communication (Sanchez *et al.*, 2013). The specifications that were added are:

File Protocol (SCPS-FP) is an extension of the File Transfer Protocol (FTP) that can be expanded for full internet compatibility and scaled to meet resource constraints. Some of its capabilities are: automatic restart of a failed transfer, pausing a file transfer to be restarted later from where it was interrupted, etc. (Council of the Consultative Committee for Space Data Systems, 1999b)

Security Protocol (SCPS-SP) deals with integrity, confidentiality and authentication. SCPS-SP is comparable to IPsec. It assumes IP style addressing. The protocol also assumes that the two PDU will be indexed in the security association whose attributes (each a minimum of eight bits in size) encipher/decipher key, confidentiality algorithm identifier, integrity key, etc. When a PDU is received the IP number field will be examined to check if security services were applied by the sender (Council of the Consultative Committee for Space Data Systems, 1999a).

Network Protocol (SCPS-NP) is comparable with IP. Like IPv4, it uses four octets for addressing. IPv6 types of address are used for programs without bit efficiency issues. This protocol uses the same broadcast address definitions as IP.(Council of the Consultative Committee for Space Data Systems, 1999c)

Transport Protocol (SCPS-TP) services provided are full reliability (provided by TCP), best effort reliability, and minimal reliability (provided by UDP). Among TP extensions are to reduce handshaking needed for TCP connections; to deal with data corruption, link outages,

congestion, header compression, long delays, large amounts of data in transit, etc. (Council of the Consultative Committee for Space Data Systems, 2006).

2.2 Connecting satellites to the Internet

There are several ways that can be used to communicate with a satellite.

Some of these could be via ground station, a network of ground stations or a network of satellites.

2.2.1 Ground station

This is the most common way of communicating with satellites. Because nanosats are in the Low Earth Orbit (LEO), they orbit the earth roughly every ninety minutes (Cates, Cirillo and Stromgren, 2006). The Line of sight (LoS) of the satellite from the ground station can be as short as 15 minutes (Ge *et al.*, 2018). For instance, the in-view time is of an Iridium satellite with its orbital period of a hundred minutes from a single fixed point on earth is 9 minutes (Fossa *et al.*, 1998)

2.2.2 Global Educational Network for Satellite Operations (Genso)

Genso is an International Space Education Board (ISEB) project that was established in 2006 to connect amateur ground stations. Genso seeks to have a global coverage to allow satellites to be accessible in a continuous twenty-four hours in a day instead of the current few minutes fifteen times a day. To the already known nanosat framework, Genso adds a Mission Control Client , a Ground Station Server (GSS) and an Authentication Server as seen in Figure 2.1 (European Space Agency, 2008). This works in a way similar to cellular network (Mishra, 2004) that allows communication to continue when transitioning from one cell to another. As seen in Figure 2.2, a user can connect to the Internet, is authorised through the request server and given access to receive data from their satellite from various GSS's. Each GSS will prioritise its own satellite over others (Kief *et al.*, 2011).

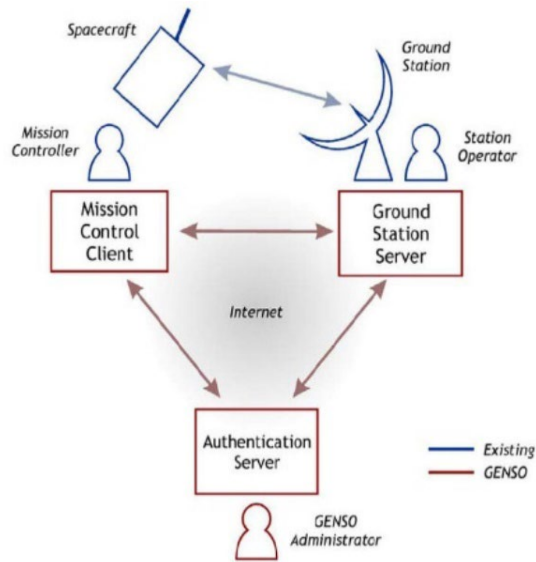


Figure 2-1: How GENSO will fit into the existing satellite communication framework

(European Space Agency, 2008)

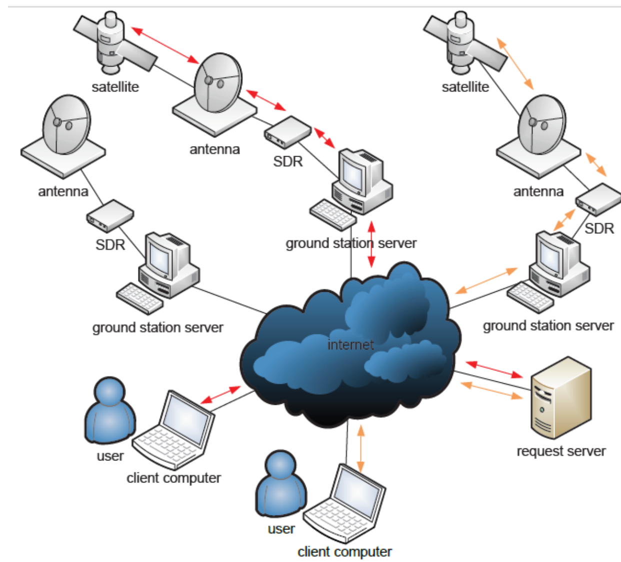


Figure 2-2: Dial tone model

(Kief *et al.*, 2011)

Figure 2.3 shows the Humsat mission concept developed by the University of Vigo in Spain that uses the Genso network and allows users to not only access the network but to also add their existing network of sensors to relay data through the satellites (Page *et al.*, 2010)

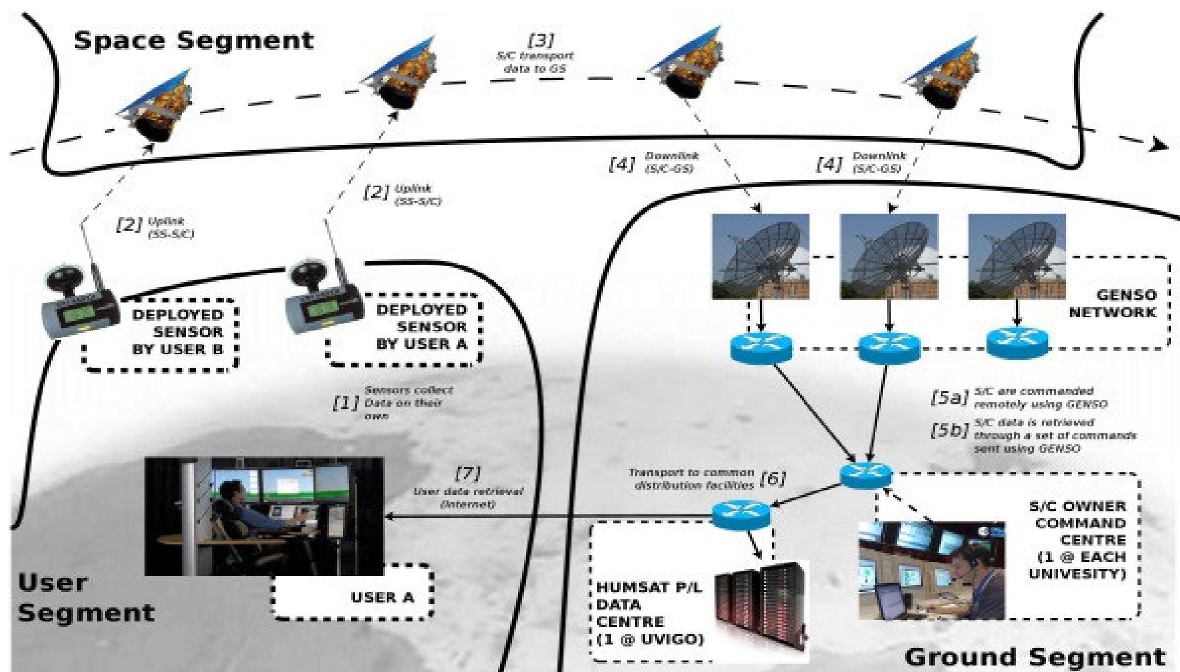


Figure 2-3 : HumSAT Mission Concept

(Tubío-Pardavila R *et al.*, 2014)

2.2.3 Tracking and Data Relay System (TDRS)

TDRS is a space-based network of distributed satellites that provide continues relay of information to and from other satellites. The system consists of nine operational satellites (plus one failed launch in 1986 and two satellites retired in 2009 and 2011) that are owned and managed by the NASA Goddard Space Flight Centre. NASA added to this network by launching TDRS-M in August of 2017 (Israel and Shaw, 2018), pushing the total number of these to ten (Campbell, 2015). Figure 2.4 shows how communication takes place from a LEO satellite (e.g. Hubble or ISS) to TDRS satellite in GEO to a ground station (*MIT System Architecture Group*, 2017).

The Hubble Space Telescope, whose data is responsible for more than fourteen thousand scientific papers uses TDRS for relaying its data (Garner, 2015).

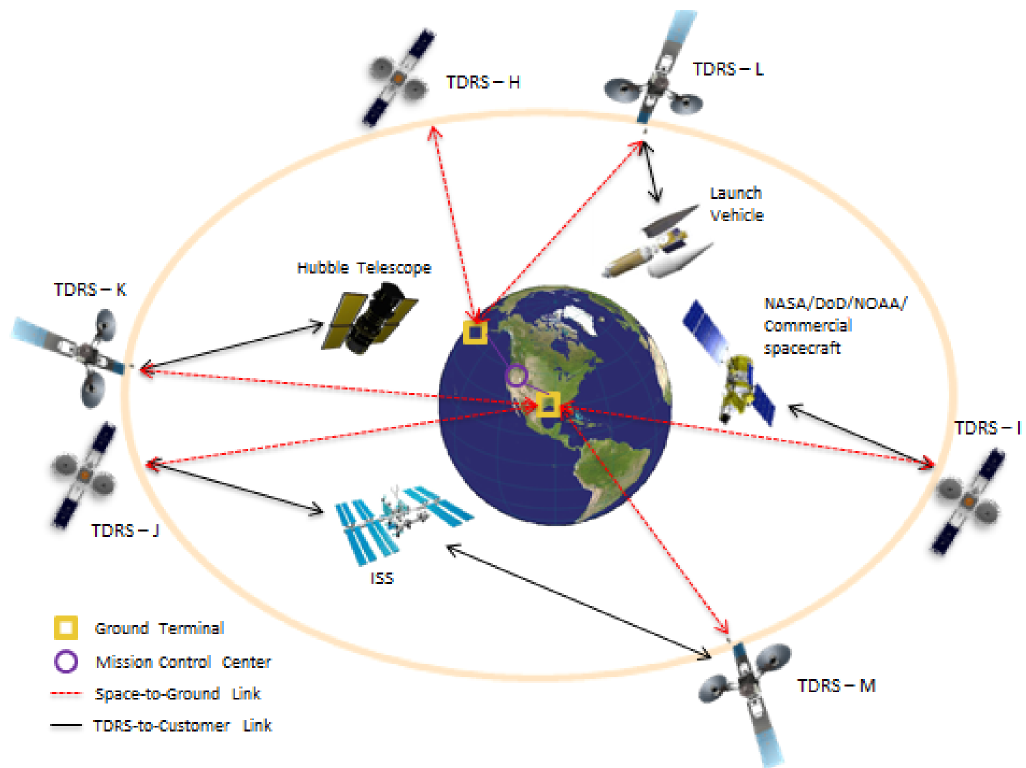


Figure 2-4 : TDRS constellation

(Garner, 2015)

SpaceX's Falcon 9 spacecraft used this system (Bhasin *et al.*, 2014) to relay data when it was delivering cargo to the ISS.

2.2.4 Laser Communications Relay Demonstration (LCRD)

NASA along with the United States Air Force and the Massachusetts Institute of Technology is working on LCRD. This optical communication promises to provide ten to hundred times faster data rates than current radio frequency satellite communication (Garner, 2017). Figure 2.5 shows the model concept of what was planned for launch in 2021.

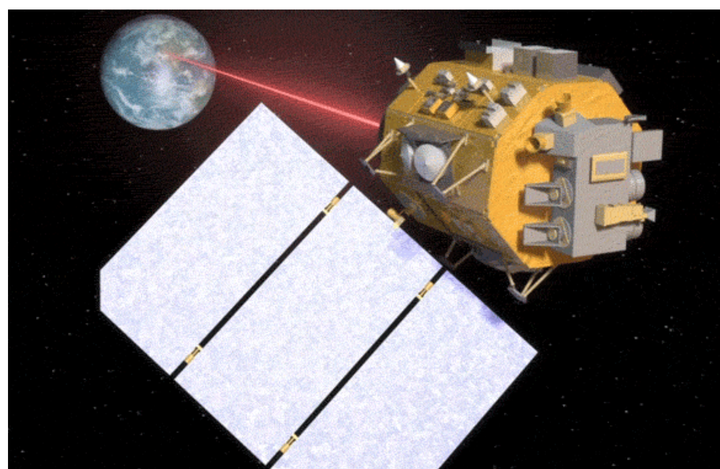


Figure 2-5 : LCRD

(Garner, 2017)

2.2.4 Space network ground network hybrid

Clyde Space, an American start-up, is implementing a hybrid model that uses a network of three ground stations that track a network of three satellites that orbit the earth every eight hours. This gives the three satellites constant communication with the ground network. This network of satellites can then relay data from any satellite in the Medium Earth Orbit (MEO) to the ground stations. Clyde Space service who would be owners of the satellites would access that data from the Internet (Ewig, 2017).

2.3 Security

Typically Internet of Things security can be divided into seven sections (Tuwanut and Kraijak, 2015) which are

- communication layer security,
- privacy protection,
- access control,
- user authentication,
- data integrity,
- data confidentiality and
- availability at any time.

The Open Web Application Security Project (OWASP) has a list of ten areas of IoT vulnerability (OWASP, 2014), some of which are:

- insecure web interface,
- insufficient authentication/authorization,
- insecure network services,
- lack of transport encryption,
- insecure software/firmware,
- poor physical security, etc.

Solutions propose the following: look at security across the entire ecosystem (as hackers will be looking for areas of weakness), assume a possibility of physical access by hackers (where a component can be moved to a hostile network) anticipate a node being isolated and security measures must still apply, ensure that data is protected throughout its entire life cycle and ensure that there are no moments, thoroughly review encryption, use white lists to ensure only those devices that need access are granted instead of the entire Internet, e.g. If the computers that are going to be accessing TshepisoSat are only those physically located at CPUT Bellville and have a range of IP addresses from 196.21.120.0 to 196.21.120.255, then every address outside of this range should not be granted access.

2.4 On board computer

In a nanosat, the on-board computer (OBC) allows all the various subsystems to communicate with each other. The OBC is also responsible for most of the data processing and decision making on the satellite. Some subsystems based on how much processing they need may have their own dedicated processor. An example of this is the F'SATI ZaCube-2 that has a dedicated processor for the attitude determination and control system (ADCS) and another for its primary payload which is the software defined radio ship monitoring automatic identification system communication. These processors also can act as backup should the OBC fail.

On board computers can either be designed from scratch or one can be bought commercially off the shelf (cots). A popular choice is the MSP430 by Texas Instruments (TI) used in the ZACube-1 (Earth Observation Portal, 2016). The MSP430 has been independently (not by TI) tested to handle an ionizing dose of 30 krad of radiation with no anomalies detected (Vladimirova *et al.*, 2007).

2.5 IoT Technologies

2.5.1 Bluetooth

Bluetooth, originally envisioned in 1994 and designed by Ericsson engineer Jaap Haartsen, was founded in 1998 by Ericsson, and four other founding members of The Special Interest Group which were IBM, Intel, Nokia and Toshiba. It was designed to replace cable connections like that of RS-232 and was to have longer a longer range than the direction sensitive infrared links.(Haartsen, 2000). The first version started with a range of about 10 meters (about five times more than infrared) at a data transfer rate of 732 kbps. The current version 5.0 can have a range of up to 200 meters and at a rate of 2 Mbps (Collotta *et al.*, 2018). A known draw-back of Bluetooth is its high power consumption. This can become unacceptable for IoT battery powered devices that can't be recharged. For the nanosat use case, the best range possible is still way far from acceptable ranges.

2.5.2 Wi-Fi

Wi-Fi started in 1997 under the IEEE 802.11 group of standards, with data rates about 2 Mbps (Crow *et al.*, 1997). The latest 802.11ax version known commercially as Wi-Fi 6 has improved significantly, designed to handle a data rate of up to10 Gbps (Rochim *et al.*, 2020). Though Wi-Fi gives better range and data throughput, it unfortunately is too power hungry to be the first choice for a battery powered IoT device(Friedman, Kogan and Krivolapov, 2013).

2.5.3 LoraWAN

LoRa is a low power, low data throughput, **Lo**ng **Ra**nge radio frequency modulation scheme developed by Semtech. LoraWAN is a wide area network protocol that is supported, standardised and promoted by the Lora Allliance. The network is made up of eight channel

gateways that act as base stations (LoRa Alliance Technical Commitee, 2017). Lora nodes connect to these gateways, which are listening for lora packets. These gateways are in turn connected to the internet. To build a gateway, all that is required is a lora concentrator board to receive lora packets, an antenna and computer (even a single board computer like a raspberry pi) connected to the internet whether connected via ethernet or Wi-Fi. Generally, these gateways would cover a range of 20 km. The Semtech SX1280 LoRa module was recently tested to have a range up to 133 kilometres at a data rate of 0.595 kbit/s (Janssen *et al.*, 2020) The Things Network have however registered 832 km as the record range at 25 mW in early 2020 (The Things Network, 2020)

2.5.4 Sigfox

Sigfox is a cellular network specifically for IoT things. Like other cellular network, the network coverage is provided by Sigfox directly (in US and parts of Europe) or by Sigfox network operators (partners companies in the rest of the world) through their base stations. It uses the ISM bands. These base stations generally cover a radius of 40 km in rural areas (Mekki *et al.*, 2019). Each node can send a maximum of 140 messages, 12 bytes in size, per day. In a day a node can receive no more than four 8-byte messages.

2.5.5 Weightless

Weightless is a wireless technology that combines both frequency and time division multiple access (FDMA and TDMA) (Ali *et al.*, 2017). It is standardised by the Weightless Special Interest Group. The first company to develop and deploy its hardware is Ubiik (an IoT firm). Unlike many IoT technologies, Weightless allows sending firmware to nodes wirelessly, which means software updates can be sent without going to the nodes. It transmits at 50mW and have a communication range of 5-10 km and a data throughput of 1 to 10 Mbps (Ali *et al.*, 2017).

2.5.6 Enhanced Machine Type Communication (LTE-M)

Long Term Evolution(LTE) based enhanced Machine Type Communication (eMTC), also known as LTE-M is a standard developed by the Third Generation Partnership Project (3GPP). LTE-M uses the licensed spectrum (and not ISM bands). With a channel bandwidth of 1.4 MHz and bit rates of 1Mbps, it can support high bandwidth applications like voice over LTE.(Soussi *et al.*, 2017)

2.5.7 Narrowband-IoT (NB-IoT)

Like LTE-M, NB-IoT is also a 3GPP standard also using licensed spectrum. It uses a much narrower channel bandwidth of 200kHz. Its service is offered by cell network providers and thus has the same coverage as is provided by LTE towers. NB-IoT data throughput rate is 250 kbps, and unlike LTE-M does not support transmission of voice.(Ha *et al.*, 2018)

2.5.8 Extended Coverage Global System for Mobile communication IoT (GSM-IoT)

EC GSM IoT is a standard based on the legacy second generation (2G) general packet radio service (GPRS) and enhanced GPRS (eGPRS or EDGE). EC-GSM co-exists with 2G,3G and 4G mobile networks. It is achieved by a software update on an existing network. It is particularly useful because of its support for legacy 2G systems.(Lippuner *et al.*, 2018)

Chapter 3 : Research Design

In this study, all experiments were done using software simulations. The cost of doing these experiments with real nanosatellite nodes and the time it would take to build them would not justify doing this study.

3.1 Satellite coverage

In order to see how many nanosats would be needed to have an around the globe coverage of the earth, the STK software was used. Analytical Graphics Inc. (AGI) has developed satellite System Tool Kit (STK) to be able to design and model a space mission. Students and researchers are allowed to download and use the desktop application.

From AGI's website (currently www.agi.com), a student signs up for an account. Then under products, STK is downloaded and installed. After installing, the license manager is used to request a license. The requested license must then be added to STK.

A scenario which meets the mission design specifications is created. When adding the satellites to the scenario, either a new satellite is created, or an already previously launched satellite can be chosen (e.g. the International Space Station can be chosen). When creating a new satellite, firstly the satellite is given a name and then the type of orbit is chosen, e.g. circular. The height of the satellite above earth's surface is then chosen as altitude in km's.

The orbital inclination is then chosen as the angle between the orbit path line and the equator (in other words, it is the tilt between the equatorial plane and the orbital plane). Along with it, the swivel of the orbital plane along the north-south pole axis is chosen. This is the angle between the vernal equinox (axis and where the satellite crosses the equatorial plane as it ascends from the south pole towards the north pole. This angle is referred to as the right ascension of the ascending node (RAAN). Right ascension lines are to the sky above the earth (celestial sphere) what the lines of longitude are to the earth's surface.

Once this information about the orbit has been added, a sensor (e.g. a camera) is added and a field of view of the sensor is added. A circle that shows the satellite's coverage will be observable as the satellite orbits the earth.

3.2 Peer-to-peer system

Lloyd Wood describes satellite constellation as a "number of similar satellites, of a similar type and function, designed to be in similar, complementary, orbits for a shared purpose, under shared control"(Wood, 2003). Andy Oram, on the other hand describes a peer-to-peer system as a "self-organizing system of equal, autonomous entities (peers) which aims for the shared usage of distributed resources in a networked environment avoiding central services" (Oram, 2001).

Though Wood is talking about satellites and Oram about computing devices, the two of them are describing the same concept, and that of similar, equipotent (no super node), decentralised (no central server listening to all requests) and distributed nodes with a common purpose. It is for this reason that this study looks at a satellite constellation as nothing more than a peer-to-peer network of nodes.

When building a peer-to-peer system, there are several protocols and algorithms to choose from. Some of these are Gnutela (Ripeanu, 2001), Napster (Giblin, 2012), Pastry (Rowstron and Druschel, 2001), Kademlia (Maymounkov and Mazières, 2002) and Chord (Stoica *et al.*, 2003). The algorithm that was chosen for this study is Chord.

3.3 Chord algorithm

Chord is a peer-to-peer distributed hash table algorithm. A hash table is a data structure that stores data in key value pairs. A hash function produces the key by hashing the value or data. The data in the hash table is then stored at the index of the hashed key. To search for the data, the same hash function is used to produce the index (key) of the data (value) that is being searched. This data structure makes it very quick to search unsorted data. It also makes it very easy to add more data in the structure.

In the design of the structure, collisions must be catered for. A collision happens when two or more different values (data) produce the same key (data index or address). Suppose there was a majorly simplified hashing function that took a letter of the alphabet (the value) and divided it by 10, and returned the remainder of that division as the hash key produced. If small 'a' was value 1, capital letter 'A' value 27, using such a hash function the letters c(3),m(13),w(23),G(33) and Q(43) would all hash to the same key 3. This is what is meant by a collision. In this simplified case, changing the divisor from 10 to a number greater or equal to 2 times 26 will remove all these collisions.

A distributed hash table (DHT) is thus like a normal hash table except that the data is distributed in various computing devices in a network. These computing devices in the network are referred to as nodes. The DHT is an overlay network (a virtual network built on top of a physical network). Each of these nodes uses the same hashing algorithm to store and to lookup data.

In the Chord algorithm, each node in the network keeps a total of m references where the size of the network is 2^m. The list of references that each node keeps is referred to as the finger table. It is also often referred to as the routing table, because it contains the routes to the different nodes.

The first node in the network is Node 0, and the last node is Node 2^m -1. In the example below, where m = 4 (and therefore the last node is 2^4 -1,i.e 15), each node has 4 references and the

network has 2^4 (=16) nodes. In Chord, each node N keeps references to nodes N + 2^0 ; N+ 2^1 ; N + 2^2 +;……;up to node N + $2^{(m-1)}$. In the example below, each node keeps up to node N + $2^{(4-1)}$ (which is N + 2^3 , i.e. N + 8).

These nodes form a logical ring, as seen in the figure below, where the next node after 2^m-1 in the ring goes back to node 0. In other words, in a network where m = 4 and node 2^4-1 is node 15, the next node after node 15 is node 0. To cater for this clockwise wrapping around the ring, finger[i] of each node must be modulo of N + 2^i divided by 2^m where i is the index of the references (fingers) of a node. Node N finger[i] = (N + 2^i) % 2^m .For node 8 finger[3] = (8 + 2^3)%16 = (8 + 8)%16 = 16%16 = 0. In other words, for node 8, the finger at index 3 (i.e. the fourth reference) is modulo of (8 + 2^3) divided by 16 which is - the remainder of (16) divided by 16, i.e. zero. The modulo sign represented by % means the remainder after dividing the two numbers, e.g. 16/16 is 1 remainder 0, therefore 16%16 is 0. Similarly, Node 10 finger [3] is (10 + 2^3)% 16 = (10 +8) %16 = 18%16 = 2 (18/16 is 1 remainder 2).

Table 3-1 : Chord Nodes 0 to 15

Node 0

i	finger[i]
0	1
1	2
2	4
3	8

Node 1

i	finger[i]
0	2
1	3
2	5
3	9

Node 2

i	finger[i]
0	3
1	4
2	6
3	10

Node 3

i	finger[i]
0	4
1	5
2	7
3	11

Node 4

i	finger[i]
0	5
1	6
2	8
3	12

Node 5

i	finger[i]
0	6
1	7
2	9
3	13

Node 6

i	finger[i]
0	7
1	8
2	10
3	14

Node 7

i	finger[i]
0	8
1	9
2	11
3	15

Node 8

i	finger[i]
0	9
1	10
2	12
3	0

Node 9

i	finger[i]
0	10
1	11
2	13
3	1

Node 10

i	finger[i]
0	11
1	12
2	14
3	2

Node 11

i	finger[i]
0	12
1	13
2	15
3	3

Node 12

i	finger[i]
0	13
1	14
2	0
3	4

Node 13

i	finger[i]
0	14
1	15
2	1
3	5

Node 14

i	finger[i]
0	15
1	0
2	2
3	6

Node 15

i	finger[i]
0	0
1	1
2	3
3	7

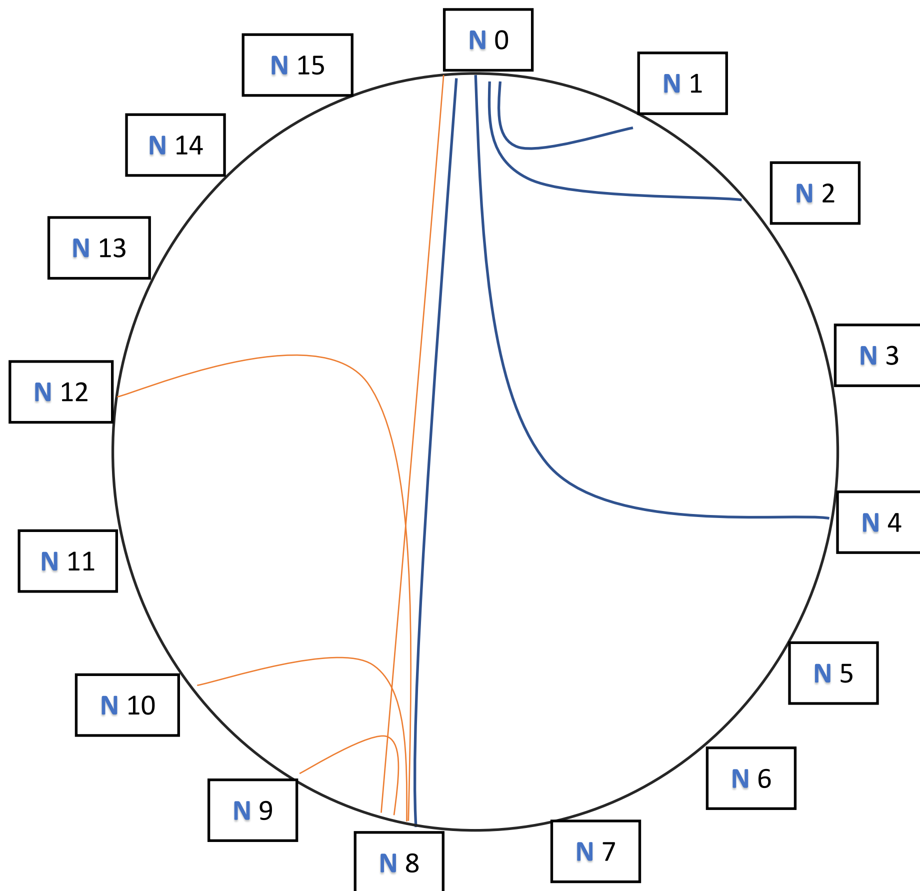


Figure 3-1 : Chord logical ring

The Chord protocol dictates that in the likely event that nodes do not yet exist in the node id space, then the references to the next existing nodes is what goes into the finger table (also known as routing table). In the example above, if the ring only had nodes 1,3,4 and 12 then the finger tables of those four nodes would only contain either 1,3,4 or 12 like Table 3-2 . (The shaded numbers are the missing nodes, that are then replaced by next existing nodes.).

In the Node 1 routing/finger table below, the finger[0] reference should contain a reference to Node 1 + 2^0, which would be Node 2. However, because there is no Node 2, finger [0] is Node 3, which is the next node that exists in the ring. Finger[1] is Node 1 + 2^1 which is Node 3, which exists. Finger[2] would be Node 1 + 2^2, which would be Node 5 if it existed. Node 12 is the next existing node, and it therefore takes the place of the non-existent Node 5. Finger[3]

would be Node 1 + 2^3, which would be Node 9. The non-existent Node 9 is replaced by the existing Node 12.

In the Node 3 finger table, finger[0] is Node (3 + 2^0) which is Node 4, and it exists. Nodes 5 (3 + 2^1), 7 (3 + 2^2), 11 (3 + 2^3) do not exist and are therefore replaced by Node 12. The same can be seen in Nodes 4 and 12 finger tables.

Table 3-2 : Routing tables for Nodes 1,3,4 and 12

Node 1

i	finger[i]	
0	3	*2*
1	3	*3*
2	12	*5*
3	12	*9*

Node 3

i	finger[i]	
0	4	*4*
1	12	*5*
2	12	*7*
3	12	*11*

Node 4

i	finger[i]	
0	12	*5*
1	12	*6*
2	12	*8*
3	12	12

Node 12

i	finger[i]	
0	1	*13*
1	1	*14*
2	1	*0*
3	4	*4*

3.4 IoT Lab

The choice of a microcontroller was guided by the on-board-computer (OBC) on the CPUT cubesat TshepisoSat. FIT Iot-lab was chosen because it is part of Onelab testbeds, which CPUT is a part of. This is an internet testbed that allows one to freely borrow fully kitted nodes to do experiments for about 20 minutes per session. On the IoT-Lab, there are MSP430 nodes with radio communication.

The firmware for the nodes was written in C programming language on Codeblocks IDE. The firmware was compiled into a hex file using the MSP430 GCC compiler. This MSP430 compiler was downloaded and added on the toolchain settings of Codeblocks.

3.4.1 Setting up an experiment:

An account was created with www.iot-lab.info and verified by checking email following the link sent. IoT-Lab uses secure socket shell (SSH) to connect to it's nodes, and therefore SSH keys need to be generated before attempting to connect.

Generating SSH public/private key pair:

On the Linux machine, this was done on the terminal by typing: *ssh-keygen -t rsa* the location of the two keys was given at the end of key generation.

On the Windows machine, this was done by downloading and installing PuTTYgen (a key generator by PuTTY). In PuTTYgen, under parameters, RSA (Rivest–Shamir–Adleman cryptosystem) was selected and the number of bits in a generated key was left to 2048 bits. A prompt appeared to move the mouse around to generate randomness for this key generation. When done, 15 to 20-character paraphrase was needed to encrypt the private key. The private key was saved.

The public key was copied as a single line of text - not broken into multiple lines. When the key was broken into multiple lines this caused the error "*No supported authentication methods available server sent public key*" . It is thus advisable to rather copy the public key from PuTTYgen textbox, than from a saved file. Once logged in to iot-lab, under the profile the key was registered by pasting the public key and clicking on update keys.

To load the experiment, *Testbed* then *new experiment* were clicked. Then Nodes was clicked. Under architecture, the WSN430 (cc11010) was selected. Then any of the sites that have these nodes was selected. At the final time, of the experiment the only site that had these WSN430 nodes is the Strasbourg site in France. In the past (at the start of the experiments), there were other sites as well that had these nodes. A node is then selected "add to experiment" is clicked. The firmware (as a .hex file) was loaded to the node by clicking on the microcontroller sign and navigating to where firmware had been saved. This was repeated for all the chosen nodes, and the experiment was submitted.

Once the firmware was loaded on the nodes, then on the Linux machine terminal *ssh* followed by full username which includes the site of the node, followed by enter was typed i.e *ssh mthiadonis@strasbourg.iot-lab.info*.

When the windows machine was used, because there is no Unix-like terminal to connect via SSH, PuTTY was downloaded and installed. To generate SSH keys on the Windows machine PuTTygen had to be downloaded, installed and used. Once PuTTY was opened, under category, SSH was selected and Auth was clicked. The private key that was generated using PuTTygen and saved was then browsed and found. Then again under category, *Session* was selected, and under *hostname or ip* address *strasbourg.iot-lab.info* typed and open was clicked. At the next prompt to login, the username was typed i.e. *mthiadonis*.
On both the windows (PuTTY terminal) and the Linux machine terminal to interact with each node the command *nc* followed by nodename followed by port number eg. *nc wsn430-103 20000* was then typed.

3.5 Java simulation

The code for this simulation was written on one windows machine, in the Netbeans IDE. It was compiled and built with JDK 1.8_0261. The compiled binaries were built as a jar file (Node.jar). This jar file was run on five devices. These were two Windows 10 laptops, a Linux (ubuntu) desktop, a raspberry pi 2 (with ethernet) and a raspberry pi 3 (with wi-fi). All these devices were connected to the same home network router by wi-fi (except the raspberry pi 2, which had no wi-fi module, and was connected via ethernet cable). On all five devices the jar

was run on the command prompt for Windows and bash terminal for Linux by typing the same command *java -jar Node.jar*.

Java being a cross platform language meant there was no need to compile separate binaries for the separate platforms (namely Windows, Ubuntu and Raspbian). Each of the devices had to have the java runtime environment (JRE) installed on it. The same node.jar was run on Java Virtual Machine (part of the JRE) installed on the different platforms. If this program was written and compiled in a language that is not cross platform e.g. C/C++, then it would have needed to be compiled for each platform to produce platform specific binaries.

There are two TCP/IP transport protocols in which a communication channel could be established, before sending data packets. The first is the transmission control protocol (TCP) itself, and the other is user datagram protocol (UDP). TCP ensures that a connection is established before data is sent. It does this by a node sending a synchronise message to request a connection. The responding node sends an acknowledgement, and along with it its own synchronisation (request for connection) message. The first node then sends a message acknowledging this request.

After this initial handshake agreement to communicate is set, then actual data is sent. The node receiving the data needs to acknowledge receipt of that. If the sending node does not receive the acknowledgement, it can assume that the sent packet was lost, it can resend it is acknowledged. In this way TCP guarantees that every packet sent is received and confirmed.

UDP does not do this at all, as in does not establish connection before communicating. UDP simply sends the datagrams, with no concern of whether they are received or not. This lack of acknowledgements makes communicating with UDP faster in terms of number of actual data bits sent per time (not faster in terms of time it takes for message from node A to make it to node B). What UDP lacks at the transport layer of the TCP/IP model, the programmer will have to fill at the layer of the communication application.

3.6 Node address

The node address was produced by hashing the port number and the IP address using SHA-1 hashing algorithm. SHA-1 produces a 160-bit output, no matter how large the input is. The hashing function which takes port and IP was first tested to see that it produces a 160-bit address in binary, which equates to a 40-character string in hexadecimal.

Node addresses (i.e hash outputs of port and IP) of 253 nodes were tested to see if there were any duplicate addresses as the address size was reduced.

The format of the node address was chosen to be hexadecimal. The format could have been either hexadecimal, binary or decimal.

Hex: b57e3b31a5c8ca9afdbb09808bbfff0312c73705

Binary:10110101 01111110 00111011 00110001 10100101 11001000 11001010 10011010 11111101 10111011 00001001 10000000 10001011 10111111 11111111 00000011 00010010 11000111 00110111 00000101

Decimal: 1 036 142 379 933 200 953 385 190 273 773 802 349 822 357 223 173

Binary has 160 characters (bits) – this is fixed length, no matter what the hash input is

decimal has 49 characters (for this specific hash) – theoretically between zero and 49 digits in length

Hexadecimal has 40 characters (binary ÷ four). Like Binary, this is fixed length. It therefore made sense to write the node addresses in hexadecimal.

Chapter 4 : Results

4.1 Satellite coverage with STK

Below are four pictures of the CPUT ZACUBE-1 orbit line. The satellite orbits at the altitude of 600 km. The circles are the coverage of the sensor at that altitude and at 90 degrees field of view. The size of that circle has a direct proportional relationship with the altitude and the field of view of the sensor on the satellite. It is observed that with eight such satellites following each other in the same orbital plane, an around the globe coverage can be achieved with a swath width of around 5000km (that is about the longitudinal width of Sub-Saharan Africa).

Adding another eight adjacent orbital planes, with each having eight satellites, (thereby having 64 satellites) the whole globe could be covered with sensors. This is similar to Iridium's full global coverage at 777 km altitude with 66 satellites (Tan *et al.*, 2019).

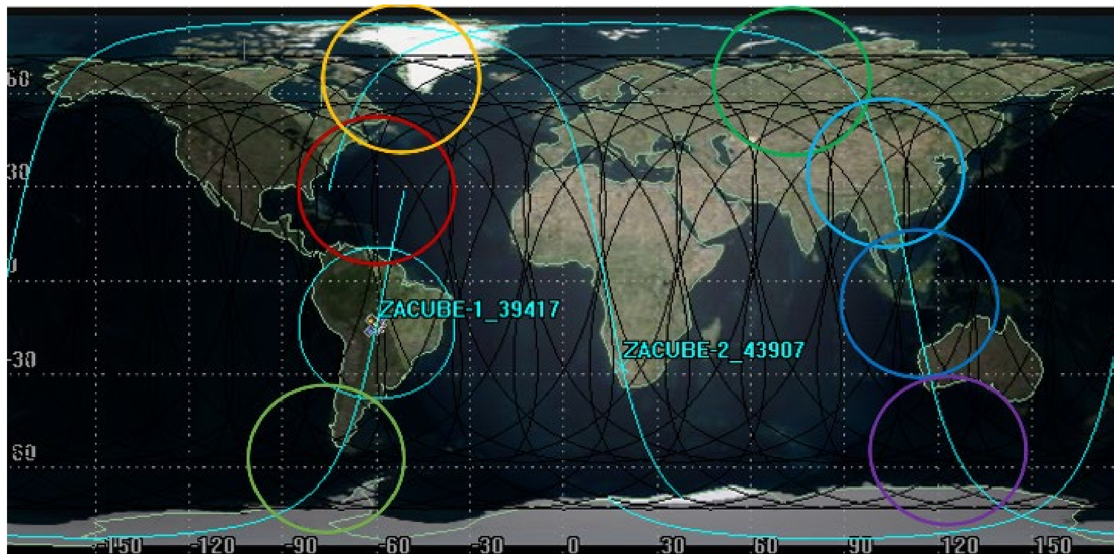


Figure 4-1 : Tracking Orbital line of ZACUBE 1 (image generated using STK simulation tool)

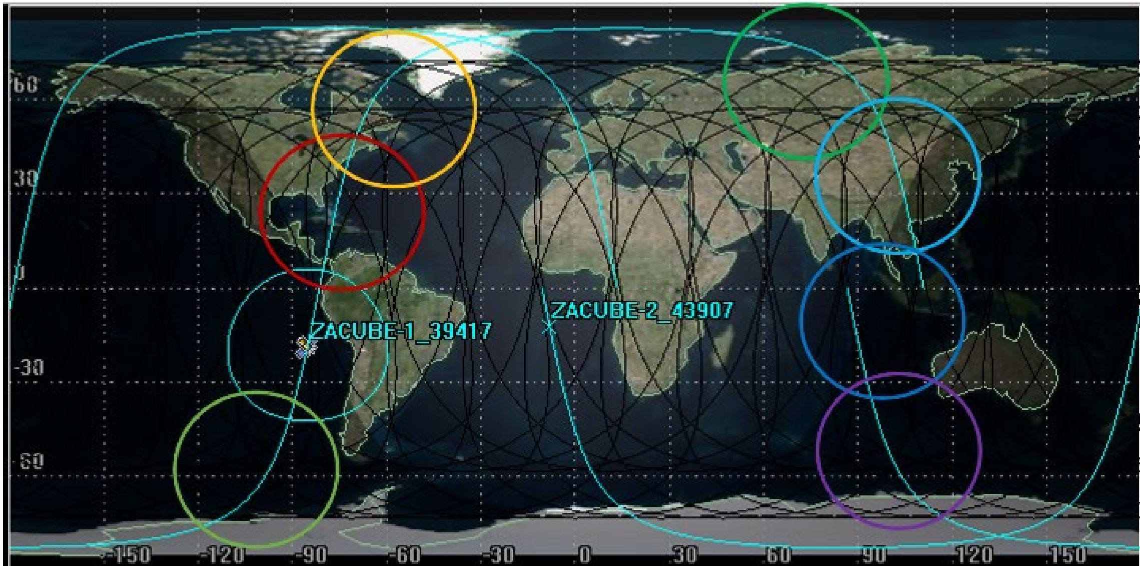


Figure 4-2 : Second track of orbital line of ZACUBE 1 (image generated using STK simulation tool)

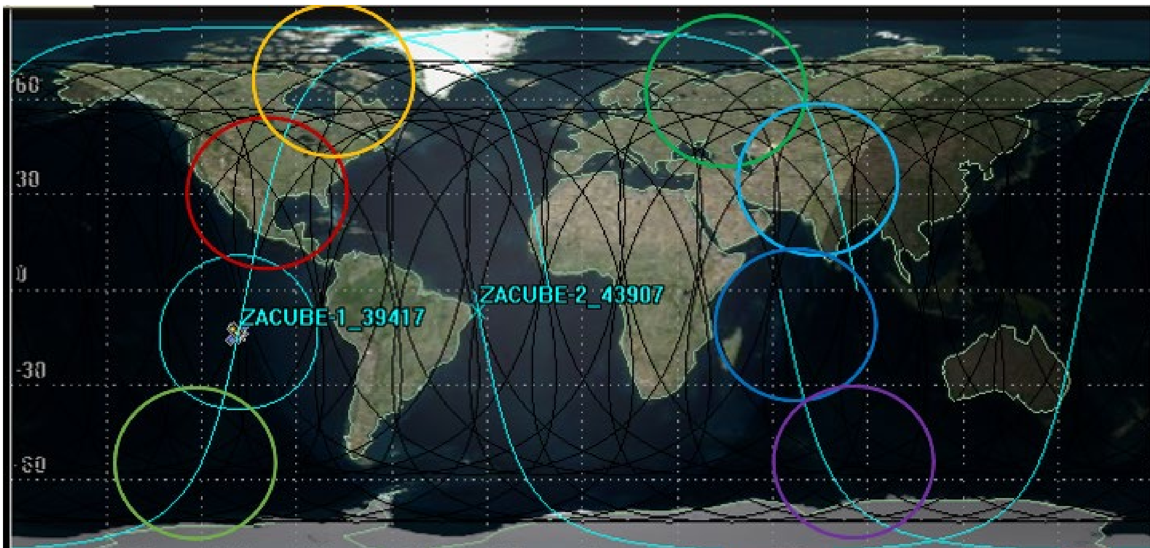


Figure 4-3 : Third track of orbital line of ZACUBE 1 (image generated using STK simulation tool)

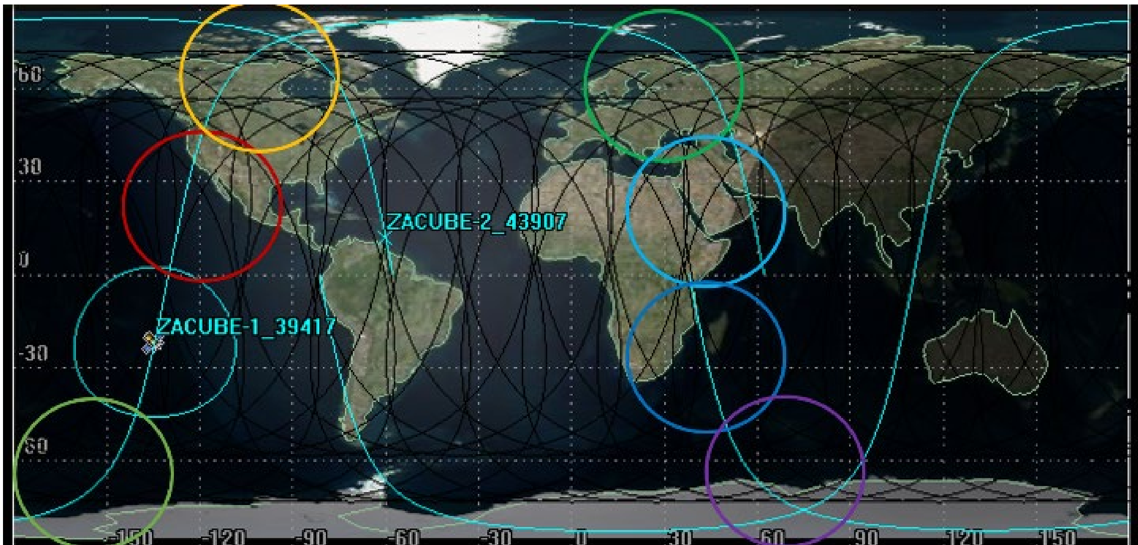


Figure 4-4 : Final track of orbital line of ZACUBE 1 (image generated using STK simulation tool)

The global coverage of earth with these satellites present an opportunity to not only use them as sensors, but as part of internet infrastructure. This is particularly useful for remote areas that are difficult and expensive to provide internet for. The time it would take for a data packet to travel from a user on earth to the satellite and back to another earth user would be below two hundred milliseconds. This latency is in the acceptable range for voice communications.

4.2 Network topology in Packet Tracer

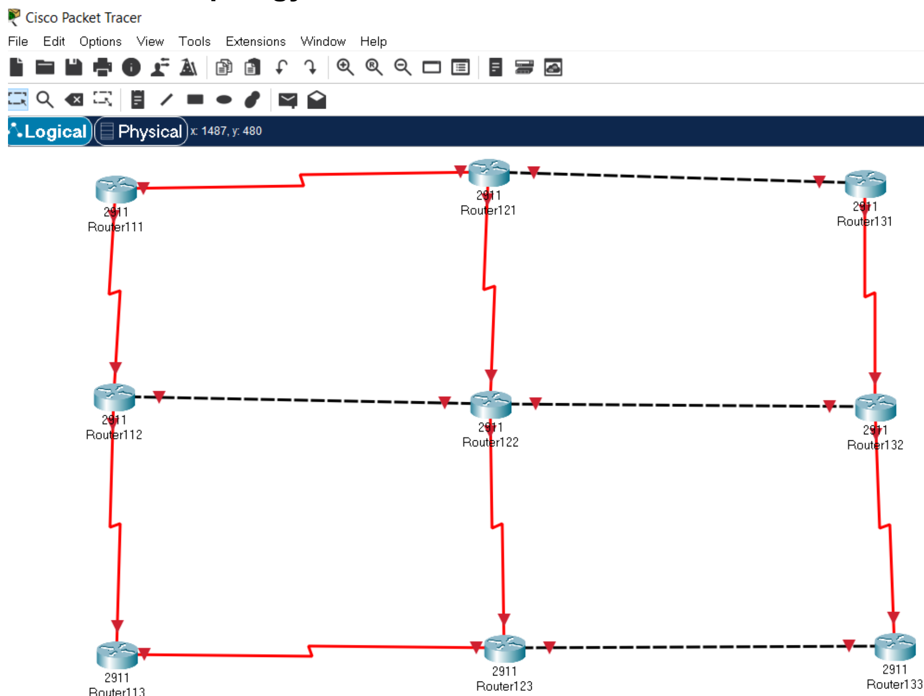


Figure 4-5 : Packet Tracer Network topology of satellites – how each will be connected

To uniformly cover the globe, it is recommended that each satellite has link to four others. Like router 122 in the centre, each should be connected to two satellites in the same orbital plane as itself, e.g. routers 123 and 121 above and below. It would also have a link to one satellite in each of the two adjacent orbital planes, like Router 112 and 132.

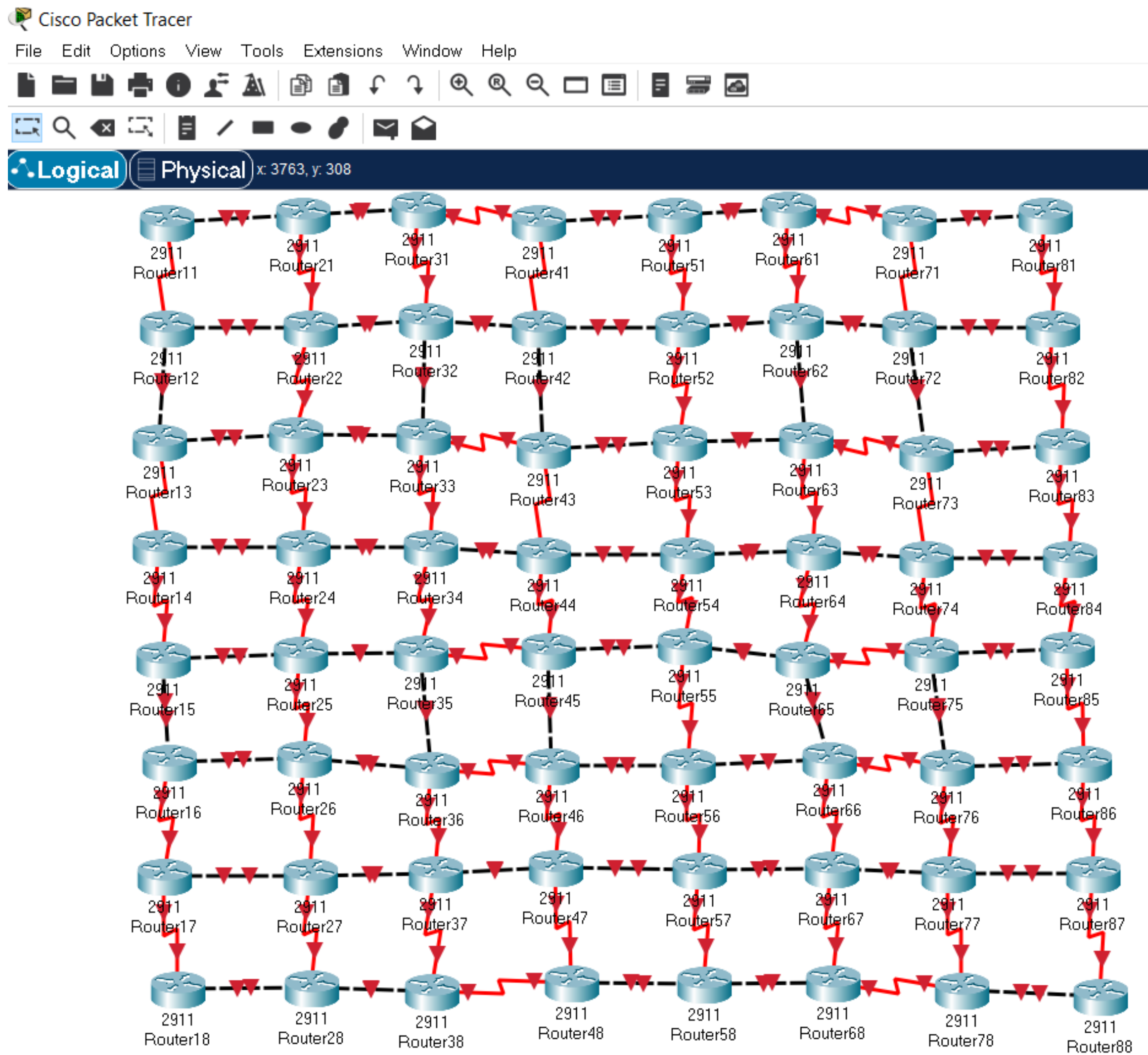


Figure 4-6 : Topology for all 64 satellites covering the globe

The sixty-four satellites would be like a net covering the globe. Each satellite would be like a knot of the net, and if it is disconnected, it would represent a hole (uncovered area) in the net.

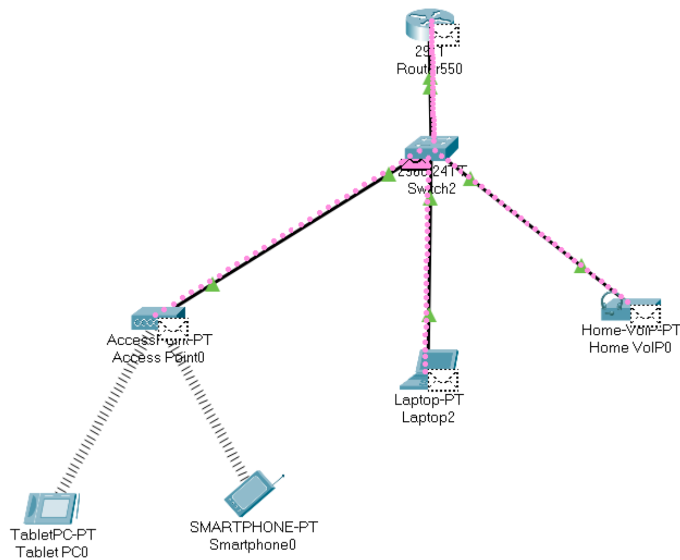


Figure 4-7 : Each satellite providing internet for various devices

Each of these satellites would be providing internet to various devices on the ground for a number of minutes before handing the connection over to another satellite.

4.3 IoT Lab

In recent years, a new way of thinking in terms of satellite design has emerged. Instead of large, complex and expensive systems, there is a great interest towards nanosatellites, which are lighter and significantly cheaper in terms of production. Several projects aim to launch dozen or even hundreds of nanosatellites in orbit. Although their lower cost allow them to be numerous, their miniaturization comes with limitations in memory, to which we've tried to find a solution.

Connecting the nanosatellite flying close by each other's (in clusters) with Chord is proposed, which is a P2P protocol that provides two functionalities. It evenly spread all the data shared on the network between the nodes and provide a look up method to retrieve that data.

To realize this project, many different tools were used. First, *Iot-Lab* was discoverd, which is a testbed platform where test programs can be tested on different boards remotely. It was

decided to work on the WSN430 v1.4 board, which is small but has enough capability for us to run our tests on. Just enough to run tests. It’s composed of 2 sensors (light and temperature) and a small memory (48kB Flash, 10kB RAM). Then to access the *IoT-Lab* servers it was necessary to setup SSH access with PuTTY software. Code Blocks was used as a programming environment and msp430_GCC with IoT-Labs 's library for the WSN430.

4.3.1 Chord in general

With chord, nodes and keys are assigned an m-bit identifier using consistent hashing. Each identifier is mapped and sorted around a circular space (called the *chord ring*) in ascending order. Each node has a *successor*, which correspond to the next node on the chord ring in a clockwise direction and each key is stored at its successor node.

The main purpose of chord is to serve as a data bank where the data is stored and indexed on the nodes and can be retrieved from any one of them thanks to a lookup method. When sending or searching for a data on the network, one must use its related key.

The most basic and the simplest lookup method is to pass on the request for a key to the next successor. The request thus passes through all nodes until the node responsible for this key (the one storing the data related to the key) is reached. It can then send an answer request which will contain the related data to its successor. This answer will also be passed on from successor to successor, until it reaches back the original node. This lookup method has a response time equals to the number of nodes in the network since the request has to pass through every node of the network.

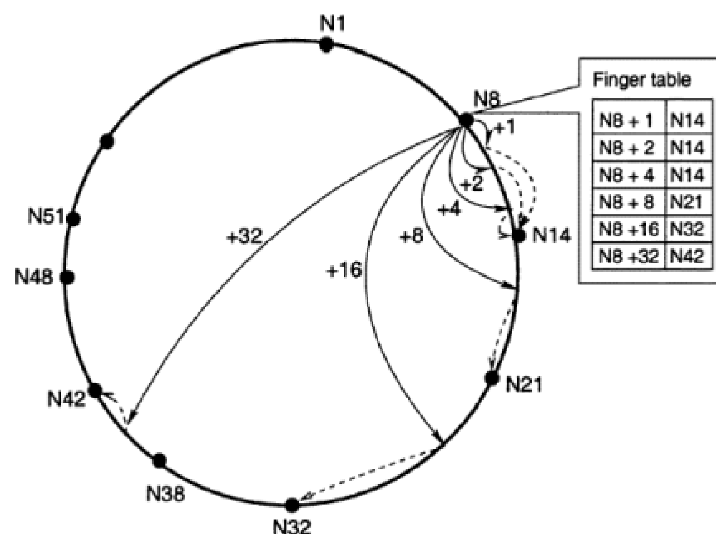


Figure 4-8 : Finger table and complex lookup method (Stoica *et al.*, 2003)

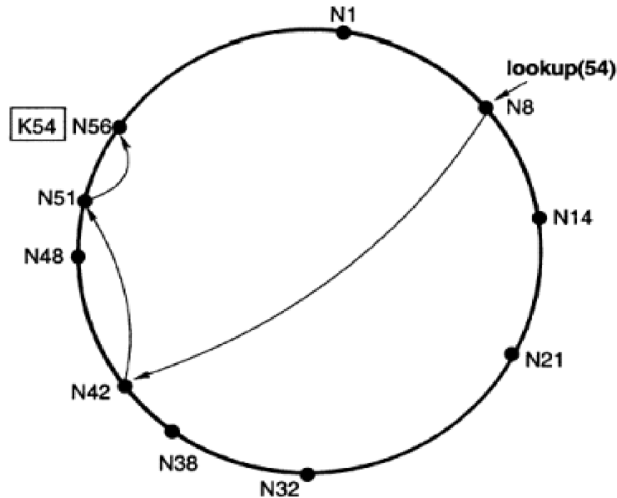


Figure 4-9 : Complex lookup method (Stoica *et al.*, 2003)

With the complex method, each node has a routing table also called finger table. The role of this finger table is to give information to his node about the nodes to which he can talk to within the network. The routing table is built using binary research O(log N). The first entry of the finger table of a node N is the immediate node's successor of N on the ring or the node with the closest ID to $N+2^0$. Then to fill the finger table, use $N+2^k$ incrementing k until reaches an ID which is out of the circular identifier space.

As can be seen on the Figure 4.8, the immediate successor of the node 8 on the ring is the node 14 (or the node with the closest ID to $8+2^0$) so is the first entry of the finger table. And then, as said previously, increment k to find the other entry. For example $8+2^4$ is 24 and the closest ID on the ring is 32, so the node 8 can talk with the node 32 too.

In order to resolve a lookup request for a key k, nodes route the request to the node in their finger tables whose ID immediately precedes k until the key's successor is reached. For example, in the Figure 4.9, if node 8 wants the key 54 he needs to make a request to the node 42. If the finger table of the node 42is built, it will be seen that the closest node to node 56 he can talk to is the node 51. And so, the node 51 can talk with the node 56 which possess the key.

So the number of nodes that must be contacted to find a successor in an N-node network is O(log N) which is significantly more efficient and faster than the basic consistent hashing which requires all nodes to be called.

4.3.2 Static Chord Protocol

To answer the issues related to the small memory of nanosatellites and to the chord protocol itself, one must be able to figure out which one is responsible for a key and how to access the data store. In this part, consider that the number of nodes in the network is fixed and that there

is no joining, leaving and crashing.

4.3.3 Node structure

```
typedef struct ChordNOde_t ChordeNode_t;
struct ChordNode_t {
   uint16_t address; //16-bit MAC address
  Key key;
  IPV6 ipv6Node; //128-bit address
  FingerTable_t fingerTable;
};
extern ChordNode_t node;
```

Figure 4-10 : Node structure in C

To use the chord algorithm and realise a peer-to-peer sensor environment, the first step is to create the structure of a node. For this, created a C structure was created representing a node (Fig. 4.10). This structure contains 4 attributes necessary for the proper functioning of the communication with the chord algorithm. The structure called “ChordNode_t” has its own 16-bit MAC address, its own 128-bit IPv6, a 160-bit hash key from its IPv6 thanks to SHA1 (data encryption algorithm), the key of its predecessor and a FingerTable (optional: depends on whether it is in simple look up or complex look up).

4.3.4 Simple Look-up

Experiments started by the simple look-up, it consists to find the identifier that a node wants to communicate with it by asking if its successor node owns the key, otherwise the successor node sends back the same request to its direct successor node and so on until the successor (id) has been found. The successor result (ID) will be returned along the circle until the node which has issued the query.

```
//Chord packet info structure
typedef struct ChordRequest_t ChordeRequest_t;
struct ChordRequest_t {
   RequestType type;
  Key seekerKey;
  Key targetKey;
  uint8_t data[ 54 ];
  uint8_t length;
};
```

Figure 4-11 : Chord Request structure in C

To do this, a C structure named was created "Chord_Request" (Fig. 4.11.), which is a 94-byte frame and currently has a request type on 1 byte to know the type of the request (if it’s a temperature request of a sensor or to recover a message coming from a sensor for example). "SeekerKey" represents the hashed key of the node at the origin of the request on 20 bytes. "Target Key" represents the hashed key of the node with to be communicated to. "Data"

represents data to be transmitted (for example, the data of a message or the data of a temperature). Finally, "length" on 1 byte represents the size in byte of "Data».

In static, it is considered that each node knows its direct successor and only its direct successor in the chord circle. A function was created that allows to compare a key on 20 bytes with another key on 20 bytes, this function will be very useful to know if the Target Key of the frame “Chord_Request” is equal to the key of a node and thus to determine if this node is the successor (id).

The steps to carry out the simple look-up chord in static are the following ones:

- Create the frame Chord_Request with the type of request which one wishes (For example: a request of temperature). The key of the node to origin of the request. The key of the node to which one wishes to communicate. Data to be transmitted if there is and the size in byte of these data.
- Send the created frame to the MAC layer that will send this frame to the direct successor of the node that issued the request.
- Once the frame has been received by its direct successor, the "compare key" function will compare the key of Target Key with the key of the successor node. If it's equal, then the successor node will write in "Data", the data to be transmitted according to the type of request as well as the size of the data in byte in length and then send the results to the chord ring. If the keys are not equal, then the successor reviews the query to his direct successor and so on until the successor (id) has been found.

Simple look-up is a very easy search to set up, code and implement. However, its complexity is linear O (n) which is not very optimized for a very large number of sensors. However, this simple look up was enough to establish the base of network, so it can send and receive query.

4.3.5 Complex Look-up

As explained above, the simple look up is not very optimized if the number of sensors is high. A a distributed hash table is used to make the search faster, because its complexity is in log (n).

```
//For static network only. Contains info from all the nodes
typedef struct All_node_t All_node_t;
struct All_node_t {
   uint16_t address[ NB_NODE ];
  Key keys[ NB_NODE ];
  IPV6 ipv6[ NB_NODE ];
};
extern All_node_t nodeArray;
```

Figure 4-12 : Node Array structure in C

In static mode, it is considered that each node knows the keys of the nodes in the network, so it’s easier to know the successor nodes of a node (Fig. 4.12.). For one node to communicate

quickly with another node, each node will have its own fingerTable containing its own successors (Fig. 4.13.). The size of the fingerTable will depend directly on the number of nodes in the network and will be determined by this formula: log (number of Nodes) / log (2).

```
//Chord node environment structure
struct FingerTable_t {
  uint8_t nbFinger;
  uint16_t finger Addr[ 160 ];
  IPV6 fingerIPV6 [ 160 ];
  Key fingerKey [ 160];
};
```

Figure 4-13 : Finger Table structure in C

To fill the fingerTable, It is considered that the first box will always be the direct successor of node n and that the last of the table will be the node before it according to the chord ring. The operation (n + 2i) mod 2m with n the current node, i the ith box of the fingerTable and m the size of the fingerTable will allow to fill in the finger table. A function named "get_best_finger" was created to determine the best node with whom to communicate.

The steps to carry out a complex look-up in static are the following:

1) The IPv6 address of all nodes is hashed in order to get their corresponding keys, then sorts them in increasing order to create the chord ring and store the sorted keys with their corresponding IPv6 and MAC address in nodeArray. 2) The size of the fingerTable is determined according to the number of nodes in the network with the formula log (number of nodes) / log (2) .3) The fingerTable is filled using the sorted keys and the operation (n + 2i) mod 2m. 4) A "Chord_Request" frame was created the same way as the simple look-up. 5) It uses the function "get_best_finger" to see if the key of the node that is going to communicate with is in the fingerTable otherwise the "get_best_finger" function always returns the node in the last box of the fingerTable (the node that is opposite to the current node in the chord ring). 6) The IPv6 obtained from the "get_best_finger" function was hashed and then assign it to TargetKey. 7) One sends the created frame to the MAC layer that will send this frame to the k node returned by the "get_best_finger" function. 8) Once the frame has been retrieved by the node k, the "compare key" function is used to compare the target key with the key of the k node. If it is equal, then node k will send a request with its data depending on the type of the request to the chord ring. Otherwise, the node k returns the request to the node returned by the "get_best_finger" function and the step (8) is repeated until the successor (id) has been found. The log(n) search is much more interesting for our problem as it allows us to minimize the impact of the number of node in the network on the lookup time.

4.3.6 Data Storage

This last part will be devoted to the storage of data through the chord ring. It will explain how it was possible to add, store and recover data on the network, to and from a node. For this part, the stored data will be represented by text messages to which a key is given.

```
//Creation structure
typedef creation_t creation_t;
struct creation_t {
   uint8_t ID [ OCTET_ID_MAX ];
   uint32_key_ID [ KEY_SIZE ];
   uint8_t data_ID [ OCTET_DATA_MAX ];
   uint8_t length_ID;
};
```

Figure 4-14 : Message structure in C

For the creation of a message, for each node a structure C "creation_t" (Fig. 4.14.) was created with a 20 bytes ID corresponding to the title of the message, Key_ID on 20 bytes corresponding to the hashed key of the title of the message. Data_ID on 50 bytes representing the contents of the message and finally length_ID on 1 byte corresponding to the size of the ID + Data_ID.

```
typedef struct memory_t memory_t;
struct memory_t {
   uint8_t storage_index;
   creation_t creation_mem[ 10 ];
};
```

Figure 4-15 : Memory structure in C

To save messages, one has also created a "memory_t" structure for each node (Fig. 4.15). This structure will aim to store the messages whose node will be responsible. It contains an allocated space to store up to 10 messages (the limit was decided arbitrarily) and a 1-byte integer "storage_index" to better distribute messages in memory.

Several essential functions were coded for a successful creation, sending, searching or saving a message. The function message_creation(uint8_t * ID, uint8_t * data) allows one to create a message according to the title and the content received as parameters. The find_best_storage_node() function finds the node that will be responsible for storing the message. The recording_creation() function is used to manage the storage of a message in the memory of a responsible node. Finally, the last essential function is research_message_node() to search for a message in the memory of a node n.

The steps for saving a message through the chord ring are as follows: 1) Receiving the ID and content of the message typed by the user with the sms_package_creation() function to hash the message ID on 160 bits as well as determined the size of the message. 2) Determine the node that will be responsible for this message with the function find_best_storage_node(). 3) Create a Chord_Request with the following parameters: type ="SET_CREATION", SeekerKey

(the Key_ID of the message), TargetKey (the node that will be responsible of this message thanks to the function find_best_storage_node()), length (the size in byte of the message), data (the content of the message). 4) Once the node responsible for the key has received the frame, the latter must store this message with the function recording_creation(). 5) The receiving node must then notify the one who send the message that he received it

The steps to find a message from a node are as follows:

- Hashing the ID of the searched message to 160 bits.
- Find the node that should be responsible for this key with find_best_storage_node().
- Send a Chord_Request with the following parameters: type="ASK_CREATION", SeekerKey = Key_ID of the message, TargetKey = the node which must be responsible for this message thanks to the function find_best_storage_node, length = 20, data = the key of the Message ID.
- Once the responsible node has received the frame, it must check for whether it holds the key in its memory with the function research_message_node(). If the node holds the key of the message, then review the message content at node n otherwise it sends an error message to node n.
- Node n stores the content of the message sent by the node responsible for it.

4.3.7 Chord Tests

This part is intended to show the tests performed for complex look-up, the save and the request of a message. PuTTY as well as a man-machine interface was used to perform these tests.

Test 1: Complex lookup

The objective of this test is to recover the value of the temperature in °C coming from the sensor no. 3 (Fig. 4.16.) from the sensor no. 0.

```
cmd > temperature
Temperature measure: 34.107
```

Figure 4-16 : Temperature Data from Sensor n°3

```
cmd >
radio >
Chord request received
        Type : ASK_TEMPERATURE_REQUEST
        Seeker : 3589fad7 387c324a 08c7252a 8a1c4abe db0172a5
        Target : 9346c55b ee20c874 12932cc6 da360265 efd2543c
Comparing the 2 keys...
        This node is the target. Analysing request type...
        Requested for temperature. Mesuring...
        Sending temperature answer with temperature data...
get_best_finger():
        Type : TEMPERATURE_RESPONSE
        Seeker : 9346c55b ee20c874 12932cc6 da360265 efd2543c
        Target : 3589fad7 387c324a 08c7252a 8a1c4abe db0172a5
        Finger = 0
```

Figure 4-17 : Communication succeed between node n°0 and node n°3

```
cmd > ask temperature 0003

Address : 0003
IPV6 : 1234::2342::3455::4567::1234::2342::3455::0003
Test
KEY : 9346c55b ee20c874 12932cc6 da360265 efd2543c
get_best_finger():
        Type : ASK_TEMPERATURE_REQUEST
        Seeker : 3589fad7 387c324a 08c7252a 8a1c4abe db0172a5
        Target : 9346c55b ee20c874 12932cc6 da360265 efd2543c
        Finger = 3
Command line received : |ask| |temperature| |0003|
```

Figure 4-18 : Request temperature of sensor n°3 from sensor n°0

```
cmd >
radio >
Chord request received
        Type : TEMPERATURE_RESPONSE
        Seeker : 9346c55b ee20c874 12932cc6 da360265 efd2543c
        Target : 3589fad7 387c324a 08c7252a 8a1c4abe db0172a5
Comparing the 2 keys...
        This node is the target. Analysing request type...
        Received TEMPERATURE_RESPONSE :
                Temperature = 34,107
```

Figure 4-19 : Temperature response from sensor n°3 for sensor n°0

According to (Fig. 4.17) and (Fig. 4.18.), it is noted that the sensor No. 4 and the sensor No. 0 communicate well because the sensor No. 4 sends its temperature to the sensor No. 0 (Fig.

4.19.). It is consided that the complex look-up in static performs (not considering the joining, leaving and crashing of a node).

Test 2: Creating and sending a message

This test will focus on creating and sending a message on the chord ring. The goal is to send a message with ID: "test" and for content "chord protocol".

```
Address of node : 2
Test
Information about your creation :
        ID : test
        ID_KEY : f09a1b4f 18342648 c018edfe 0426eadc 09b1e0a1
        DATA : chord_protocol
        DATA Length : 34
End of information about your creation
Send your File to the node : f96cf61a e73712e7 24d4b1c5 4b19af79 2b110d2b
get_best_finger():
        Type : SET_CREATION
        Seeker : f09a1b4f 18342648 c018edfe 0426eadc 09b1e0a1
        Target : f96cf61a e73712e7 24d4b1c5 4b19af79 2b110d2b
```

Figure 4-20 : Creation of message « test » and send it into the chord ring

```
cmd >
radio >
Chord request received
        Type : SET_CREATION
        Seeker : f09a1b4f 18342648 c018edfe 0426eadc 09b1e0a1
        Target : f96cf61a e73712e7 24d4b1c5 4b19af79 2b110d2b
Comparing the 2 keys...
        This node is the target. Analysing request type...
Received SET_CREATION
From :f09a1b4f 18342648 c018edfe 0426eadc 09b1e0a1
Saving File...
```

Figure 4-21 : Save Message test to the responsible node

```
Information about your creation :
        ID : test
        ID_KEY : f09a1b4f 18342648 c018edfe 0426eadc 09b1e0a1
        DATA : chord_protocol
        DATA Length : 34
End of information about your creation
```

Figure 4-22 : Memory of message contained in the node

According to (Fig. 4.20.) and (Fig. 4.21.), it is observed that the message has been created and that it has been stored at the correct node responsible. When looking into the memory of the node responsible for the message, the information concerning the "test" message has been

stored (Fig. 4.22). It can therefore be concluded that the creation and sending of a message on the chord ring works in static.

Test 3: Request for a message

This test will focus on the request for a message. The goal is to request the content of a message by only knowing its ID, if no node is responsible for this ID, then the user must be informed. This continues the message from the previous test.

```
Chord request received
        Type : ASK_CREATION
        Seeker : 3589fad7 387c324a 08c7252a 8a1c4abe db0172a5
        Target : f96cf61a e73712e7 24d4b1c5 4b19af79 2b110d2b
Comparing the 2 keys...
        This node is the target. Analysing request type...
Received ASK_CREATION
From :3589fad7 387c324a 08c7252a 8a1c4abe db0172a5
Requested for message...Message Transfer...
get_best_finger():
        Type : GOT_CREATION
        Seeker : f09a1b4f 18342648 c018edfe 0426eadc 09b1e0a1
        Target : 3589fad7 387c324a 08c7252a 8a1c4abe db0172a5
        Finger = 0
```

Figure 4-23 : Request the content of the message test

```
Chord request received
        Type : GOT_CREATION
        Seeker : f09a1b4f 18342648 c018edfe 0426eadc 09b1e0a1
        Target : 3589fad7 387c324a 08c7252a 8a1c4abe db0172a5
Comparing the 2 keys...
        This node is the target. Analysing request type...
Received GOT_CREATION
RECEIVED MESSAGE
From :f09a1b4f 18342648 c018edfe 0426eadc 09b1e0a1
Saving DATA...
```

Figure 4-24 : Saving the content of the message test into the node 0

```
Chord request received
        Type : GOT_CREATION
        Seeker : 4ffed238 17ccada4 07578613 705e6a33 df4f0724
        Target : 3589fad7 387c324a 08c7252a 8a1c4abe db0172a5
Comparing the 2 keys...
        This node is the target. Analysing request type...
THIS FILE DOESNT EXIST INTO THE NETWORK !
```

Figure 4-25 : The message test2 doesn't exist into the network

According to (Fig. 4.23.), it can be seen that the node responsible for the "test" ID sends the node at origin of the request its content and information as can be seen in the figure (Fig. 4.24).

Moreover, when a query with an ID: "test2", it is seen that no node stores this message and the node at the origin of the request is informed (Fig. 4.25).

It can be concluded that the search for a message through the chord ring works.

4.3.8 Dynamic Chord

In order to not upload a new firmware, in every node of the network, each time a node is added or removed, a dynamic network is needed capable of letting new node join and leave at their whim. It also means that the network has a start and an end, with a first node creating the network, and a last one ending it.

4.3.9 Creating the network

The node, which creates the network, becomes its first member. As it is alone, it is its own successor, which implicitly means that it is responsible for all the possible keys of the network. It also does not have any successor or finger, since he does not have anyone to talk to.

4.3.10 Asking the network to join it

Before joining a network, the node and the network must exchange information relative to the new node's theoretical key / place in the network and the ability of the network to receive the new node.

4.3.11 Joining the network

If the answer is positive, all nodes should be notified to allow them to update their finger table. Also, the new node and its successor must exchange data, as the new node take the responsibility of a part of its successor's key. The node is completely part of the network once it has downloaded all the data related to the key he is in charge of, and all nodes in the network has updated their finger table.

4.3.12 Leaving the network

A node leaving the network imply that its successor will take responsibility for all of its keys, and that the other nodes won't be able to talk to it anymore. This is achievable in two steps. First the node must notify the network that it's leaving to prevent others to engage a transmission with it. As they are notified, they will update their finger table and will address the successor of the leaving node from there on. Finally, the leaving node must transmit its chord related data to it successor to ensure it continue to exist in the network. At this moment, it can discharge itself from that data and stop communicating with the network permanently.

4.3.13 Coping with crash

Although it is not possible for a network to prevent the failure of its nodes (as it can happen

unexpectedly), it is possible to counter its effects.

The first noteworthy effect when a node leaves the network unexpectedly is the loss of the information it contained. Chord can be seen as a databank that will lose part of its data in this case. One way to prevent it is to implement a redundancy of the data, by letting every successor save the data of its predecessor as a backup. This can be done every time a data is uploaded or at regular time intervals.

4.4 Java Simulation

The simulation was run on Linux (Ubuntu) Proline desktop with IPv4 address 10.0.0.107

```
mthiadonis@mthiadonis-PROLINE-DQ67SW:~$ ifconfig
eno1: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 00:22:4d:55:e4:0d  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 20  memory 0xfe600000-fe620000

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 244  bytes 18528 (18.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 244  bytes 18528 (18.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlxd46e0e04521f: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.107  netmask 255.255.255.0  broadcast 10.0.0.255
        inet6 fe80::5d58:c70f:197b:bb82  prefixlen 64  scopeid 0x20<link>
        ether d4:6e:0e:04:52:1f  txqueuelen 1000  (Ethernet)
        RX packets 400  bytes 139043 (139.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 241  bytes 49533 (49.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 4-26 : Java application node 1 - Linux

Another node was run on a Raspberry Pi 2 with no Wi-Fi module

```
pi@raspberrypi: ~
File  Edit  Tabs  Help
pi@raspberrypi:~ $ ifconfig
eth0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether b8:27:eb:0f:c8:ab  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

pi@raspberrypi:~ $
```

Figure 4-27 : Node 2 of java application - Raspbian

Another node was run on Raspberry Pi 3 with a Wi-Fi module, with IP address 10.0.0.144

```
File   Edit   Tabs   Help
pi@raspberrypi:~ $ ifconfig
eth0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether b8:27:eb:66:42:29  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.144  netmask 255.255.255.0  broadcast 10.0.0.255
        inet6 fe80::d34d:c2ff:7ee8:d016  prefixlen 64  scopeid 0x20<link>
        ether b8:27:eb:33:17:7c  txqueuelen 1000  (Ethernet)
        RX packets 38  bytes 6378 (6.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 80  bytes 13108 (12.8 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

pi@raspberrypi:~ $
```

Figure 4-28 : Node 3 of Java application - Raspbian

The next node was run on a Windows Lenovo laptop with IP address 10.0.0.100. This was the computer used writing and compiling the Java archive (jar) executables.

```
 Connection-specific DNS Suffix  . :

reless LAN adapter Local Area Connection* 2:

 Media State . . . . . . . . . . . : Media disconnected
 Connection-specific DNS Suffix  . :

reless LAN adapter Wi-Fi:

 Connection-specific DNS Suffix  . :
 Link-local IPv6 Address . . . . . : fe80::9161:a867:6413:5a22%4
 IPv4 Address. . . . . . . . . . . : 10.0.0.100
 Subnet Mask . . . . . . . . . . . : 255.255.255.0
 Default Gateway . . . . . . . . . : 10.0.0.254

\Users\adonismt>
```

Figure 4-29 : Node 4 - Windows

The next node was run on another Windows Lenovo laptop with IP address 10.0.0.125

```
   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::a000:5bcd:736:ab9%4
   IPv4 Address. . . . . . . . . . . : 10.0.0.125
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.0.0.254

Ethernet adapter Bluetooth Network Connection:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

C:\WINDOWS\system32>
```

Figure 4-30 Node 5 - Windows

The last Node.jar was run on a Windows Acer machine, which was also used to connect to the IoT lab via SSH using Windows PuTTY. Its IP address was 10.0.0.123

```
Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::19c6:9c0c:a90f:2c92%3
   IPv4 Address. . . . . . . . . . . : 10.0.0.123
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.0.0.254
```

Figure 4-31 Node 6 - Windows

These devices being on the same network, all had the same default gateway out of the network 10.0.0.254. The network used, could only accommodate a maximum of 253 nodes (with IP addresses from 10.0.0.0 to 10.0.0.255 minus the network address 10.0.0.0, the broadcast address 10.0.0.255 and the default gateway address 10.0.0.254)

4.4.1 UDP communication

On the server side of the node, UDP communication was achieved by creating a datagram socket bound to a port number 1234. Any port number greater than 1023 and less than 65 536 would have worked, provided that there is not another application that has taken that port number already. The command *netstat -an* can be used to view already used port numbers, to avoid a conflict.

```
DatagramSocket socket = new DatagramSocket(1234);
byte[] buffer = new byte[100];
DatagramPacket receivedpacket = new DatagramPacket(buffer, buffer.length);
System.out.println("Listening for UDP messages\n");
socket.receive(receivedpacket);
```

Figure 4-32 : UDP datagram

A buffer to hold byte data from incoming packets was created and bound to receive the datagram packet.

```
$ java -jar UDPServer.jar
Listening for UDP messages
```

The server then pauses and waits to receive a packet from any node wishing to communicate with it.

```
InetAddress ip = InetAddress.getByName ("10.0.0.100");
DatagramSocket datagramsocket = new DatagramSocket ( );
String message = "Request to join";
byte[] buffer = message.getBytes();
DatagramPacket packet = new DatagramPacket(buffer,buffer.length,ip,1234);
System.out.println("Sending UDP message:\n\"Request to join\" \nto 10.0.0.100 port 1234  ");
datagramsocket.send(packet);
```

The client side also needs a buffer to hold message sent in byte format. A datagram object is also needed, but unlike the server side it must be bound to the IP address of the server and the port that the server on which that server is listening to.

```
java -jar UDPClient.jar
Sending UDP message:
"Request to join"
to 10.0.0.100 port 1234
```

Figure 4-33 : UDP request to join client message

Once this packet is sent, only then can the server continue from waiting for a message. In other words, only after this can the server-side pass `socket.receive(receivedpacket);` line. Whether there is a server listening or not in UDP the client will consider the message sent. If there is no node listening, then the messages sent will be lost. The transport layer of the IP protocol will not resend this message when the listening node is online. This is because the transport layer did not require confirmation of receipt of messages.

To avoid the loss of the sent messages, the join message was sent several times with a delay in between of two seconds (two seconds chosen to not waste simulation time). This was done at application level as seen below.

```
for(int i=1;i<5;i++){
          try {
             Thread.sleep(2000);
             message =  "Request to join " +i;
             buffer = message.getBytes();
             packet = new DatagramPacket(buffer,buffer.length,ip,1234);
             datagramsocket.send(packet);

                   } catch (InterruptedException ex) {
             Logger.getLogger(UDPClient.class.getName()).log(Level.SEVERE, null, ex);
          }
     }
```

Figure 4-34 Resending UDP messages at Application Layer until they are received

The client was turned on with no listening server. The server was then only turned on five seconds later. The first four messages:

Request to join

Request to join 1

Request to join 2

Request to join 3

were lost, and only the fifth message *Request to join 3* was received

```
java -jar UDPClient.jar
Sending UDP message:
"Request to join"
to 10.0.0.100 port 1234

RESPONSE:
Your Request to join 4 has been received
```

Figure 4-35 Server responds to 5th message

In order for the server application to reply with the *RESPONSE:*, it needs to get the IP address of the client from the received packet clientaddress = receivedpacket.getAddress(). Similarly, to get the port number that the client opened to send the message (this should not to be confused with the port number that the client expected the server to be listening on, which was 1234), port = receivedpacket.getPort().

If the client does not know what the address of an active node is, it can send a broadcast message in the network. The broadcast message was sent to IP address 10.0.0.255, which is the last address in a network with subnet mask 255.255.255.0.

```
java -jar udpclient.jar
Sending UDP message:
"Request to join"
to 10.0.0.255 port 1234

RESPONSE:
Your Request to join has been
received
```

Figure 4-36 Broadcast message sent and received

As seen above, the broadcast request was received. The server was on before the client, and it therefore was able to respond to the first of the five messages.

Capturing from Wi-Fi

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

Packet list | Narrow & Wide | Case sensitive | Display filter | 10.0.0.100

No.	Time	Source	Destination	Protocol	Length	Info
5245	161.961042	10.0.0.125	10.0.0.255	BROWSER	243	Host Announcement MB-
11650	398.950584	10.0.0.100	10.0.0.255	UDP	57	56627 → 1234 Len=15
11657	400.942907	10.0.0.100	10.0.0.255	UDP	59	56627 → 1234 Len=17
11675	402.954603	10.0.0.100	10.0.0.255	UDP	59	56627 → 1234 Len=17
11678	404.955436	10.0.0.100	10.0.0.255	UDP	59	56627 → 1234 Len=17
11685	406.957124	10.0.0.100	10.0.0.255	UDP	59	56627 → 1234 Len=17

Figure 4-37 All five broadcast messages on Wireshark

The five UDP broadcast messages can all be seen on Wireshark (a network protocol analyser). The length of the first message is two characters shorter than the following four because of the space and message number at the end that is missing, for example *1* in *Request to join 1*.

```
13568 624.609045    10.0.0.103    10.0.0.255    DTLS    467 Continuation Data
> Frame 11650: 57 bytes on wire (456 bits), 57 bytes captured (456 bits) on interface \Device\NPF_{
> Ethernet II, Src: IntelCor_92:43:07 (d4:3b:04:92:43:07), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
> Internet Protocol Version 4, Src: 10.0.0.100, Dst: 10.0.0.255
> User Datagram Protocol, Src Port: 56627, Dst Port: 1234
> Data (15 bytes)
```

Figure 4-38 Port and IP address info of source and of broadcast

The destination port (marked Dst Port:) to the server can also be seen, and along with it the source port of the client which is 56627.

```
0000  ff ff ff ff ff ff d4 3b  04 92 43 07 08 00 45 00   ·······; ··C···E·
0010  00 2d 1d 91 00 00 80 11  00 00 0a 00 00 64 0a 00   ·-······ ·····d··
0020  00 ff dd 33 04 d2 00 19  c7 81 52 65 71 75 65 73   ···3···· ··Reques
0030  74 20 74 6f 20 6a 6f 69  6e 20 31                  t to joi n 1
```

Figure 4-39 Packet data in plain text sniffed by unintended user

The protocol analyser found the message not encrypted on the wire. It captured and presented it both in hexadecimal and in its character representation. The first letter of the message 'R' in UTF-8 and ASCII is decimal 82, 0101 0010 in binary, can be seen in its hexadecimal format 52 , which is the first letter highlighted in blue. The last character of the message '1' can be seen as hexadecimal 31.

4.4.2 TCP Connection

The TCP communication started with first establishing a communication channel between the client and the server. On the side of the server, like in the case of UDP an unused port had to first be chosen and bound to a server socket.

```
ServerSocket server = new ServerSocket(1342);
System.out.println("WAITING FOR CLIENT CONNECTION ");
Socket clientserverchannel;
int clientno = 0;

while(true){
            clientserverchannel = server.accept();
            clientno++;
            //more code not included
}
```

Figure 4-40 TCP socket establishing communication channel

The server socket then pauses, listening for requests. Once a client sends a message to the server at that port number, the server accepts the request to connect, and a communication channel is then established.

```
Scanner inputStream = null;
        try {
            inputStream = new Scanner(clientserverchannel.getInputStream());
        } catch (IOException ex) {
            Logger.getLogger(NikuThread.class.getName()).log(Level.SEVERE, null, ex);
        }
         String input = inputStream.nextLine();
```

Figure 4-41 Reading data from input stream of the channel

Once the channel was established, data stream was created to then scan and receive input from it.

```
PrintStream p = null;
        try {
            p = new PrintStream(clientserverchannel.getOutputStream());
        } catch (IOException ex) {
            Logger.getLogger(NikuThread.class.getName()).log(Level.SEVERE, null, ex);
        }
         p.println(response);
```

Figure 4-42 Writing data to output stream of the channel

To send a response back to the client, another data stream is needed. Like the input stream, this data output stream is also bound to the communication channel that was created when the server accepted the request.

From Wireshark the first data frame the server [10.0.0.100] received from the client [10.0.0.123] was a synchronization “[SYN] Seq = 0” message (No. 30). It must be remembered that in a client server model, though the server must be listening before the client communicates. However, a communication channel is always initiated by the client. The second frame from 10.0.0.100 to 10.0.0.123 is an acknowledgement of that synchronisation message [SYN, ACK] (frame 31). The TCP handshake is completed by message 32, which is an acknowledgement of frame 31.

*Wi-Fi

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
30	3.591582	10.0.0.123	10.0.0.100	TCP	66	61525 → 1342 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256
31	3.591865	10.0.0.100	10.0.0.123	TCP	66	1342 → 61525 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS
32	3.595302	10.0.0.123	10.0.0.100	TCP	54	61525 → 1342 [ACK] Seq=1 Ack=1 Win=65536 Len=0
100	7.251528	10.0.0.123	10.0.0.100	TCP	59	61525 → 1342 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=5
101	7.292137	10.0.0.100	10.0.0.123	TCP	54	1342 → 61525 [ACK] Seq=1 Ack=6 Win=131328 Len=0
102	7.308329	10.0.0.123	10.0.0.100	TCP	56	61525 → 1342 [PSH, ACK] Seq=6 Ack=1 Win=65536 Len=2
103	7.310364	10.0.0.100	10.0.0.123	TCP	70	1342 → 61525 [PSH, ACK] Seq=1 Ack=8 Win=131328 Len=16
104	7.368429	10.0.0.123	10.0.0.100	TCP	54	61525 → 1342 [ACK] Seq=8 Ack=17 Win=65536 Len=0
105	7.368485	10.0.0.100	10.0.0.123	TCP	56	1342 → 61525 [PSH, ACK] Seq=17 Ack=8 Win=131328 Len=2
106	7.374262	10.0.0.123	10.0.0.100	TCP	54	61525 → 1342 [RST, ACK] Seq=8 Ack=19 Win=0 Len=0

Figure 4-43 TCP three-way handshake to establish a channel

It is only after the three-way handshake is complete that the message containing actual data is sent and can be responded to. That is frame number 100 containing “hello” below. This message is acknowledged by frame 101.

*Wi-Fi

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
111	10.035419	10.0.0.129	239.255.255.250	SSDP	167	M-SEARCH * HTTP/1.1
30	3.591582	10.0.0.123	10.0.0.100	TCP	66	61525 → 1342 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2
31	3.591865	10.0.0.100	10.0.0.123	TCP	66	1342 → 61525 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 M
32	3.595302	10.0.0.123	10.0.0.100	TCP	54	61525 → 1342 [ACK] Seq=1 Ack=1 Win=65536 Len=0
100	7.251528	10.0.0.123	10.0.0.100	TCP	59	61525 → 1342 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=5
101	7.292137	10.0.0.100	10.0.0.123	TCP	54	1342 → 61525 [ACK] Seq=1 Ack=6 Win=131328 Len=0
102	7.308329	10.0.0.123	10.0.0.100	TCP	56	61525 → 1342 [PSH, ACK] Seq=6 Ack=1 Win=65536 Len=2

```
0000  d4 3b 04 92 43 07 40 f0  2f ba 4a 12 08 00 45 00   ·;··C·@· /·J···E·
0010  00 2d 4e 18 40 00 80 06  97 d4 0a 00 00 7b 0a 00   ·-N·@··· ·····{··
0020  00 64 f0 55 05 3e ac 54  16 19 a9 b3 04 a5 50 18   ·d·U·>·T ······P·
0030  01 00 ef bc 00 00 68 65  6c 6c 6f                  ······he llo
```

Figure 4-44 TCP data exchange only after handshake

From the above it, it is evident just how much slower and verbose the TCP communication is when compared to UDP.

As a connection oriented protocol, TCP was designed by (Vint and Kahn, 1974) to reliably deliver packets in their right order. If the packets take different routes due to network congestion, they may arrive at the destination not in the order they were sent. It is for this reason that they are numbered and reordered, something not done in UDP.

Some like (Gomez, Arcia-Moret and Crowcroft, 2018) argue that TCP can be tweaked to work for IoT devices. However for the slowness observed above, it becomes a challenge to use it for real time IoT applications (Masirap *et al.*, 2016).

4.5 Node address

The hashing function was tested by comparing its output with two online sha-1 hashes (Tools For Noobs, 2020) and (GIGA Calculator, 2022) to check if it produced the correct 40 character string output. It was initially found that the hash function in the java code sometimes produced 37-character strings like below. Below is a SHA-1 hash of IP address 192.168.0.1 and port 5000.

b57e3b31a5c8ca9afdbb9808bbfff312c7375 [37 chars] - written Java hash function

b57e3b31a5c8ca9afdbb09808bbfff0312c73705 [40 chars] - online hash

It was noted the initial hash function was deleting some of the zeros that have been shaded in blue above.

Node addresses with ip addresses from 192.168.8.2 to 192.168.8.253 all using port 5000 to communicate were tested for collisions. A collision is when two inputs into the hash function produce the same output. In other words where output1 = hashFunction(ip1, port1) is the same as output2 = hashFunction(ip2, port2). It was found that all 40-character string outputs (i.e node addresses) were unique. However, when the 40-character hexadecimal string is truncated and only the first 2 characters are taken (i.e when the address is 8 binary bits long) there are a number of duplicate addresses. In other words, instead of taking the whole *b57e 3b31 a5c8 ca9a fdbb 0980 8bbf ff03 12c7 3705* forty-character string, it was truncated to just *05* instead (in an attempt to save address space).

An example of these duplicates was observed as in the table below, when nodes with ip addresses 192.168.8.13, 192.168.8.159, 192.168.8.181 and 192.168.8.232 when the port is 5000 all had **b3** as their 8-bit address. Each hexadecimal character is 4 binary bits. This means that hashFunction(192.168.8.13,5000) truncated to 8 bits (2 characters) leads to an address b3. That is the same for hashFunction(192.168.8.159, 5000) , hashFunction(192.168.8.181, 5000) and hashFunction(192.168.8.232, 5000). When the address space was increased to 16 bits (4 hex characters) there were no collisions (i.e no 4-character string hash outputs were the same).

The collisions were counted by putting the node addresses in a Java HashMap and comparing the size of the hash map to the total number of ip addresses. The hash map by design only keeps unique values and not duplicates. There are 252 addresses from 192.168.8.2 to 192.168.8.253. If the outputs of the hash function are stored in a hash map, there should be 252 outputs in the hash map if there are no duplicates. If there are n duplicates, then there should be 252 – n outputs in the HashMap.

The table below showing the 8-bit collisions was produced by writing the truncated hash function outputs to a comma separated values (csv) file and highlighting duplicates.

Table 4-1 : Eight-bit node address collisions

be	24	de	f3	a3	33	49	a6	8e	6f	**b3**	3a	a2	be	65	a8
68	2a	5e	d7	85	6f	ee	13	17	0	5b	f0	ad	1d	cc	a6
ce	3e	0c	bf	27	8d	b8	c9	ea	ef	0d	5e	e7	97	80	96
e1	ce	f0	b4	b0	bc	8f	a1	c9	4d	75	ec	3	3b	af	82
cd	63	8a	f6	f0	e0	b6	0b	84	ad	3b	40	b2	8c	ba	1a
fb	9c	66	e6	fe	38	70	9f	e1	23	16	58	63	cb	87	11
26	40	c0	10	31	3b	d7	c2	95	26	2e	bc	8f	8c	d2	9d
89	97	4e	45	f4	a5	69	19	62	51	3	b1	5d	17	cd	15
42	a2	ce	47	3f	8b	5d	79	54	bd	8a	d5	2	ab	a5	a7
c8	0d	a1	9	98	8a	6c	8a	48	bc	cf	22	**b3**	9c	29	50
61	2a	79	49	57	af	37	25	78	48	88	2	0d	aa	83	e6
14	a4	**b3**	7	1c	1c	4a	46	cd	b5	16	59	29	e6	c4	b1
1f	d6	b9	78	e2	ec	e1	23	8a	9d	a9	5b	94	15	88	7d
f6	e6	69	fc	89	c4	b4	c1	80	2f	6b	52	9d	a9	6c	23
65	f7	3c	2a	11	**b3**	47	14	4a	74	7f	4b	66	90	58	6
93	18	b5	82	9d	b8	0c	b8	61	cd	22	f8	83			

The HashMap was found to have 249 node addresses, which include the first b3 address and excludes the next three. In the 16-bit address space, the first b3 address (192.168.8.13) hashed to b369. The 192.168.8.159 to b384, the 192.168.8.181 to b35d and the 192.168.8.232 hashed to b391.

4.6 Security

Earlier it was shown that a network analyser was able to sniff the packets sent between the nodes and read contents, because the communication was in plain text. This was the message *"Request to join 1"* If the messages passed are confidential, this would need to be avoided. To encrypt the packets, the two nodes communicating would have to agree on an encryption/decryption algorithm and keys.

4.6.1 Static key encryption

Encryption was achieved by adding key k to each letter of the message sent. If the message was "*Request*" and the key was '1', then the encrypted message would be "*Sfrvftu*", where 'R' changes to 'S' and 'e' changes to 'f' etc.

```
public String encrypt(String text,int key){
        char txt[] = text.toCharArray();
        for(int i=0;i<text.length();i++){
            txt[i]+= key;
        }
        return new String(txt);
    }
```

Figure 4-45 Encrypting communication

The decryption was just the reverse of this, subtracting the key from each received character i.e. txt[i] -= key. The encryption key was set to 1 and the "Request to join" message was transmitted.

```
Encrypt e = new Encrypt();
String message = e.encrypt(messagetxt,1);//encryption key set to 1
//...
datagramsocket.send(packet);
```

Figure 4-46 Decrypting the message

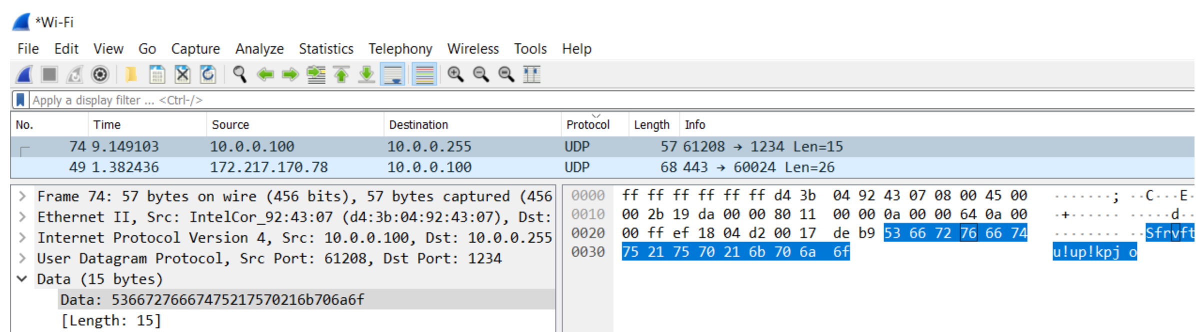


Figure 4-47 Data sniffed on Wireshark by unintended user encrypted

When the data in the packet which was sent over the network was analysed with Wireshark, it was found to be a ciphertext "Sfrvftu!up!kpjo" and not the unencrypted "Request to join" which was previously sniffed. The encryption key could be set (hard coded in the firmware) as a static value for all nodes, e.g. 1 in the case above. This would not be recommended, because finding the key (using a network analyser) for one message means one can encrypt all messages

between all nodes. In the given illustration above, an attacker could analyse frequently occurring characters like the "!" and guess that this might be the space character, which occurs frequently in a sentence. They could then see if there is a relationship between a space whose UTF-8 encoding value is 32 and "!" whose encoding value is 31. They could then try subtracting one from each character and see if the transmitted data makes any sense.

4.6.2 Randomly generated key encryption

One way to avoid this would be to get the server to randomly generate a key in a range. Only the range would need to be set. At the beginning of the communication with the client, this key is then exchanged. After this every subsequent message is encrypted with that key.

```
int min = 0;
int max = 1000;//the min and max values of the key would need to be set
int keyrange = max-min;
int key = (int) (Math.random( ) * keyrange + min);
```

Figure 4-48 Generating an encryption key randomly before data is sent

Every new communication channel established will thus have its own key. The key could also be changed in the middle of the communication, if the programmer wishes to do that.

4.6.3 Diffie Hellman key calculation

The second in which a key can be exchanged is using the Diffie Hellman key exchange algorithm (Diffie and Hellman, 1979) which is used widely in cryptography today to prevent the man in the middle attack(Mitra, Das and Kule, 2021).

In this algorithm to avoid a packet sniffer, the key used for encryption is never exchanged or shared between the two nodes. Parameters that make up the encryption key are what is exchanged. These parameters are used to randomly generate keys to calculate the secret key.

```
int mod = 3;
int gen = 37;
```

Figure 4-49 First key shared

The first key parameters shared are mod and gen, e.g. 3 and 37, and these are shared in plain text. Then each node generates its own random number. Mod is then raised to the power of this random number, and divided by gen. The remainder of this division is then sent in plain text to the other node as the public key.

```
int r = (int)( Math.random( )*20);
double publickey = Math.pow( mod, r)%gen;
buffer = String.valueOf( publickey ).getBytes ();
datagramsocket.send(new DatagramPacket(buffer, buffer.length, ip, port));
```

Figure 4-50 Public key calculated and sent

Each node receives the public key of the other and uses that to calculate the private key, by raising it to the power of r and finding the remainder when divided by gen.

```
datagramsocket.receive(receivedpacket);
receivedmessage =new String(receivedbuffer,0,receivedpacket.getLength());
int receivedpublickey = Integer.parseInt(receivedmessage);
int privatekey =(int)Math.pow( receivedpublickey, r)%gen;
```

Figure 4-51 Calculating the private key

For both nodes, this private key computes to the same number, as seen in Table 4.2 when the generated random numbers r1 for node 1 and r2 for node 2 were not the same.

Table 4-2 Private keys of Node 1 and 2 computed to the same number

MOD(Pk2^r1,gen)	MOD(Pk1^r1,gen)	MOD(mod^r,gen)		Agreed upon		Randomly generated	
Node 1 Private key	Node 2 Private key	Pub k1	Pub k2	mod	gen	r1	r2
36	36	36	3	3	37	9	1
33	33	12	9	3	37	8	2
27	27	4	27	3	37	7	3
26	26	26	7	3	37	6	4
4	4	21	21	3	37	5	5
26	26	7	26	3	37	4	6
27	27	27	4	3	37	3	7
33	33	9	12	3	37	2	8
36	36	3	36	3	37	1	9

It should be noted that the random numbers generated by nodes 1 and 2 were never shared, nor a power of or quotient which could be worked backwards. Instead a modulo % gen is what is shared, which is difficult to work backwards to the exact original number. The agreed upon gen (which is 37 in the table) must be a prime number and mod (which is agreed upon as 3 in the table) must be a primitive root of gen.

Chapter 5 : Discussion of results

This work looked at nanosats merely as things connected to the Internet. This is called The Internet of Things. This in contrast to the traditional design of the Internet which is meant for humans. It was shown that nanosats have enough similarities with other IoT devices that they could be treated as such. One of the draw backs of nanosats is physical size which is responsible for limited device on-board memory.

Table 5-1 Summary of results

What was evaluated	What was achieved	Comment
Chord	Load balancing	Each node only keeps 2^N references
Node addresses	Network size same as address size causes address duplicates	Increasing address size by order of two is enough to remove duplicates
Transport layer protocols	TCP and UDP can both be used for IoT	UDP has lower overheads and thus more preferred
Security	Static key, randomly generated key, mathematically computed key	Best is mathematically computed key
Satellite coverage	64 nanosats at 600 km	Mesh topology

5.1 Chord

It was shown that the nanosats can be connected as a peer-to-peer system. The Chord algorithm was presented as a way of load balancing the data on the network. The peer-to-peer system used the same algorithm to find data on the network as was used to store it. A routing table at each node kept a list of up to 2^N references when N was the network size.

5.2 Node addresses

It was however observed that in the design of peer network addresses, when the size (number of nodes) of the network (e.g. 256) was the same as size of the address space (e.g. eight-bits 00000000 to 11111111) there were duplicate addresses (called collisions). Such duplicates were considered unacceptable. It was then shown that increasing the address space to be any order of 2 higher than network size, the address hashing function used by Chord did not produce any collisions

5.3 Transport layer protocols

When looking at transport layer protocols to be used, it was seen that UDP was enough for communication. There however would be a necessity to guarantee that network join messages

are not lost. This was accomplished at application. TCP was found to be an overkill to use in these IoT devices. This use of UDP is good for two IoT constraints, namely low power and low memory size(Vasseur and Dunkels, 2010). IoT devices should by design consume as little power as possible (Heble *et al.*, 2018).

5.4 Security

Security threats are a major issue in IoT communications (Noor and Hassan, 2019). It was shown how a man in the middle could read data not intended for them on an unsecured communication channel. Three lightweight methods were evaluated, namely static key known by all nodes, randomly generated key shared at the start of the conversation, and an algorithm where the key is mathematically computed from key parameters that are shared. This third method was found to be the better of the three, as it already used extensively to secure channels on the Internet.

5.5 Satellite coverage

A minimum of sixty-four satellites were shown to be enough to cover the globe, if the nanosats are at an altitude of 600 km with a sensor field of view of ninety degrees. To provide stored data redundancy and network fault tolerance, these nanosats would be connected in a mesh topology. The number is not too far from Iridium's planned constellation of 77 and much closer to the actual number of 66 (Tan *et al.*, 2019). This number is much less than Starlink's planned constellation of 42 000 satellites, to provide high speed internet globally (Kassas *et al.*, 2021). If these nanosats are used to provide internet on the ground, this number is just enough to ensure that almost every square kilometre has coverage. This would be of great value particularly to rural areas with no cell towers, deep in the ocean and coast lines, north and south poles.

Chapter 6 Conclusion

The goal of this research was to design and implement an infrastructure for Internet of Things in nanosats. This was achieved through various simulations. On STK and Packet Tracer simulation tools, it was found that a peer-to-peer mesh network of nanosats with sensors could cover the globe. A distributed hash table was used to retrieve data stored each nanosat.

The first objective was to review technologies and protocols suitable to use for the Internet of Things in nanosats. Two Transport Layer protocols, namely UDP and TCP, were compared. They were found to be both suitable with their different pros and cons. UDP was however found to be much lighter, which makes it preferable for light weight devices like IoT's. Nanosats with IoT size processors can thus this transport layer in their application.

The second objective of this research was to design and propose a communication infrastructure for nanosats. This was accomplished by simulating a peer-to-peer network using the chord algorithm. This was done both using Java application run on multiple devices and running MSP430 nodes at FIT IoT-LAB. The code to run a distributed hash table can thus be successfully implemented in a nanosat microcontroller, with its space limitation.

The third objective was to assess Internet Protocol security framework. Three data encryption scenarios were compared. These were, having a fixed encryption key statically known by all nodes, randomly generating a key, and mathematically computing a key. It was found that it is more secure to mathematically compute the key. Each nanosat pair would have to do this calculation to get an encryption key before establishing a secure communication channel. Even if a key was leaked or cracked for one pair, that would not impact the keys of the rest of the network.

Nanosats can therefore be connected as Internet of Things devices without needing a separate Internet Protocol for them, because they are similar enough to other IoT devices. Not only can they be connected to the Internet, but they can be used as part of the backbone of the network, providing Internet to other devices. Common IoT Wireless Local Area Network protocols that only support less than a thousand-meter distances, like Wi-Fi and Bluetooth, should not be considered for use in nanosats.

This net of satellites would be connected to each other in mesh topology wirelessly as nodes and routers and to other router wireless routers that are already on The Internet. Security concerns can be dealt with using encryption methods. IPv4 can be used when connecting the nanosats as network nodes and Network Address Translation can be used to preserve number of Internet routable addresses used. Though the IPv6 header is a lot more optimised

compared to IPv4, it is still twice in size. IPv6 should thus be avoided if it is not necessary. When connecting them as routers however IPv6 should be considered for its abundance of available addresses, while the 4.3 billion IPv4 addresses are not enough for growing networks.

When set as IoT devices, nanosats should have the same sensors that are needed for their mission. There is no need to add more IoT specific sensors, only Internet connectivity. They should therefore collect the data necessary for their mission. The data collected can be distributed uniformly across the network using Chord. This should take away strain from single nodes doing all the work.

6.1 Future work

The low latency opens a possibility of internet applications that this network can be used for nanosats. It is recommended that such a study be done to better understand what kind of applications these would be.

References

Ali, A. *et al.* (2017) ‘Technologies and challenges in developing Machine-to-Machine applications: A survey’, *Journal of Network and Computer Applications*, 83(January), pp. 124–139. doi: 10.1016/j.jnca.2017.02.002.

Bhasin, K. *et al.* (2014) ‘Design concepts for a small space-based GEO relay satellite for missions between low earth and near earth orbits’, *13th International Conference on Space Operations, SpaceOps 2014*, (May), pp. 1–19. doi: 10.2514/6.2014-1688.

Campbell, A. (2015) ‘Tracking and Data Relay Satellite (TDRS)’. Available at: https://www.nasa.gov/directorates/heo/scan/services/networks/txt_tdrs.html (Accessed: 14 July 2017).

Cates, G. R., Cirillo, W. M. and Stromgren, C. (2006) ‘Low earth orbit rendezvous strategy for lunar missions’, *Proceedings - Winter Simulation Conference*, pp. 1248–1252. doi: 10.1109/WSC.2006.323220.

Cisco (2016) ‘Cisco Visual Networking Index: Forecast and Methodology, 2015–2020’.

Collotta, M. *et al.* (2018) ‘Bluetooth 5: A Concrete Step Forward toward the IoT’, *IEEE Communications Magazine*, 56(7), pp. 125–131. doi: 10.1109/MCOM.2018.1700053.

Council of the Consultative Committee for Space Data Systems (1999a) ‘SCPS Security protocol’.

Council of the Consultative Committee for Space Data Systems (1999b) ‘Space Communications Protocol Specification - File protocol’.

Council of the Consultative Committee for Space Data Systems (1999c) ‘Space Communications Protocol Specification - Network Protocol’, (+-+ + + + + + +).

Council of the Consultative Committee for Space Data Systems (2006) ‘SCPS Transport Protocol’.

Crow, B. P. *et al.* (1997) ‘IEEE 802.11 Wireless Local Area Networks’, *IEEE Communications Magazine*, 35(9), pp. 116–126. doi: 10.1109/35.620533.

Diffie, W. and Hellman, M. E. (1979) ‘Privacy and authentication: An introduction to cryptography’, *Proceedings of the IEEE*, 67(3), pp. 397–427.

Earth Observation Portal (2016) *Satellite Missions Catalogue:ZACUBE-2*. Available at: https://www.eoportal.org/satellite-missions/zacube-2#zacube-2-south-african-cubesat-2 (Accessed: 10 November 2022).

European Space Agency (2008) ‘How Genso Works’. Available at:

http://www.esa.int/Education/ESA_Academy.

Ewig, R. (2017) *Architecture — Audacy*. Available at: https://audacy.space/architecture (Accessed: 14 July 2017).

Fossa, C. E. *et al.* (1998) 'Satellite system', (2), pp. 152–159.

Friedman, R., Kogan, A. and Krivolapov, Y. (2013) 'On Power and Throughput Tradeoffs of WiFi and Bluetooth in Smartphones', *IEEE Transactions on Mobile Computing*, 12(7), pp. 1363–1376. doi: 10.1109/TMC.2012.117.

Garner, R. (2015) 'About the Hubble Space Telescope'. Available at: https://www.nasa.gov/mission_pages/hubble/story/index.html (Accessed: 14 July 2017).

Garner, R. (2017) 'First Steps Toward High-speed Space "Internet"'. Available at: https://www.nasa.gov/feature/goddard/2017/nasa-taking-first-steps-toward-high-speed-space-internet (Accessed: 14 July 2017).

Ge, H. *et al.* (2018) 'Initial assessment of precise point positioning with LEO enhanced Global Navigation Satellite Systems (LeGNSS)', *Remote Sensing*, 10(7). doi: 10.3390/rs10070984.

Giblin, R. (2012) 'The P2P wars: How code beat law', *IEEE Internet Computing*, 16(3), pp. 92–94. doi: 10.1109/MIC.2012.57.

GIGA Calculator (2022) *SHA1 Generator*. Available at: https://www.gigacalculator.com/calculators/sha1-online-generator.php (Accessed: 1 November 2022).

Gomez, C., Arcia-Moret, A. and Crowcroft, J. (2018) 'TCP in the Internet of Things: From Ostracism to Prominence', *IEEE Internet Computing*, 22(1), pp. 29–41. doi: 10.1109/MIC.2018.112102200.

Ha, S. *et al.* (2018) 'A novel solution for NB-IoT cell coverage expansion', *2018 Global Internet of Things Summit, GIoTS 2018*, pp. 10–14. doi: 10.1109/GIOTS.2018.8534519.

Haartsen, J. C. (2000) 'The Bluetooth radio system', *IEEE Personal Communications*, 7(1), pp. 28–36. doi: 10.1109/98.824570.

Heble, S. *et al.* (2018) 'A low power IoT network for smart agriculture', in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, pp. 609–614.

Israel, D. J. and Shaw, H. (2018) 'Next-generation NASA Earth-orbiting relay satellites: Fusing optical and microwave communications', in *2018 IEEE Aerospace Conference*, pp. 1–7. doi: 10.1109/AERO.2018.8396544.

Janssen, T. *et al.* (2020) ‘Lora 2.4 ghz communication link and range’, *Sensors (Switzerland)*, 20(16), pp. 1–12. doi: 10.3390/s20164366.

Kassas, Z. *et al.* (2021) ‘Enter LEO on the GNSS stage: Navigation with Starlink satellites’, *UC Irvine*, Report #:6. Available at: https://escholarship.org/uc/item/9td3n8wd.

Kief, C. *et al.* (2011) ‘GENSO, SPA, SDR and GNU Radio: The Pathway Ahead for Space Dial Tone’, *Infotech@Aerospace 2011*, p. AIAA 2011-1596. doi: 10.2514/6.2011-1596.

Kulu, E. (2021) *Nanosatellite Database by Erik*. Available at: http://www.nanosats.eu/ (Accessed: 9 February 2021).

Lippuner, S. *et al.* (2018) ‘EC-GSM-IoT network synchronization with support for large frequency offsets’, in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–6. doi: 10.1109/WCNC.2018.8377168.

LoRa Alliance Technical Commitee (2017) ‘LoRaWAN 1.1 Specification’, *LoRaWAN 1.1 Specification*, (1.1), p. 101. Available at: https://lora-alliance.org/resource-hub/lorawantm-specification-v11.

Madakam, S., Ramaswamy, R. and Tripathi, S. (2015) ‘Internet of Things (IoT): A Literature Review’, *Journal of Computer and Communications*, 03(05), pp. 164–173. doi: 10.4236/jcc.2015.35021.

Masirap, M. *et al.* (2016) ‘Evaluation of reliable UDP-based transport protocols for Internet of Things (IoT)’, in *2016 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, pp. 200–205. doi: 10.1109/ISCAIE.2016.7575063.

Maymounkov, P. and Mazières, D. (2002) ‘Kademlia: A Peer-to-Peer Information System Based on the XOR Metric’, in Druschel, P., Kaashoek, F., and Rowstron, A. (eds) *Peer-to-Peer Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 53–65.

Mekki, K. *et al.* (2019) ‘A comparative study of LPWAN technologies for large-scale IoT deployment’, *ICT Express*, 5(1), pp. 1–7. doi: 10.1016/j.icte.2017.12.005.

Miloslavskaya, N. and Tolstoy, A. (2019) ‘Internet of Things: information security challenges and solutions’, *Cluster Computing*, 22(1), pp. 103–119. doi: 10.1007/s10586-018-2823-6.

Mishra, A. R. (2004) *Fundamentals of Cellular Network Planning and Optimisation: 2G/2.5G/3G... Evolution to 4G*. Hoboken, NJ, USA: John Wiley & Sons, Inc.

MIT System Architecture Group (2017). Available at: http://systemarchitect.mit.edu/spaceCommunications.php (Accessed: 14 July 2017).

Mitra, S., Das, S. and Kule, M. (2021) ‘Prevention of the man-in-the-middle attack on Diffie--

Hellman key exchange algorithm: A review', in *Proceedings of International Conference on Frontiers in Computing and Systems*, pp. 625–635.

Noor, M. and Hassan, W. H. (2019) 'Current research on Internet of Things (IoT) security: A survey', *Computer Networks*, 148, pp. 283–294. doi: https://doi.org/10.1016/j.comnet.2018.11.025.

Oram, A. (2001) *Peer-to-Peer*. 1st Editio. Sebastopol, CA: O'Reilly & Associates, Inc.

OWASP (2014) *Principles of IoT Security - OWASP*. Available at: https://www.owasp.org/index.php/Principles_of_IoT_Security (Accessed: 15 July 2017).

Page, H. *et al.* (2010) 'Genso: a report on the early operational phase', in *International astronautical congress*.

Park, B. J. and Tyagi, A. (2017) 'Using Power Clues to Hack IoT Devices', 6(3).

Parliamentary Monitoring Group (2011) *SA National Space Agency & SA Council for Space Affairs roles and functions: briefing by Departments of Science and Technology & Trade and Industry*. Available at: https://pmg.org.za/committee-meeting/12783/ (Accessed: 16 February 2018).

Ripeanu, M. (2001) 'Peer-to-peer architecture case study: Gnutella network', *Proceedings - 1st International Conference on Peer-to-Peer Computing, P2P 2001*, pp. 99–100. doi: 10.1109/P2P.2001.990433.

Rochim, A. F. *et al.* (2020) 'Performance comparison of wireless protocol IEEE 802.11ax vs 802.11ac', in *2020 International Conference on Smart Technology and Applications (ICoSTA)*, pp. 1–5. doi: 10.1109/ICoSTA48221.2020.1570609404.

Ronen, E. and Shamir, A. (2016) 'Extended Functionality Attacks on IoT Devices: The Case of Smart Lights', in *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, pp. 3–12. doi: 10.1109/EuroSP.2016.13.

Rowstron, A. and Druschel, P. (2001) 'Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems', in Guerraoui, R. (ed.) *Middleware 2001*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 329–350.

Royi, N. (2022) 'The South African Maritime Domain Awareness Satellite Constellation and Beyond', *2022 International Conference on Electromagnetics in Advanced Applications, ICEAA 2022*, p. 380. doi: 10.1109/ICEAA49419.2022.9899898.

Sanchez, M. *et al.* (2013) 'Exploring the Architectural Trade Space of NASAs Space Communication and Navigation Program'.

Sen, P. *et al.* (2020) ‘Low-Cost Diaper Wetness Detection Using Hydrogel-Based RFID Tags’, *IEEE Sensors Journal*, 20(6), pp. 3293–3302. doi: 10.1109/JSEN.2019.2954746.

Soussi, M. El *et al.* (2017) ‘Evaluating the performance of eMTC and NB-IoT for smart city applications’, *arXiv*.

South Africa (1993) ‘Space Affairs Act No. 84 of 1993’, *Government Gazette*, (14917).

South Africa (2008) ‘South African National Space Agency Act 36 of 2008’, *Government Gazette*, 522(31729). Available at: http://www.gov.za/sites/www.gov.za/files/31729_1385_0.pdf (Accessed: 15 July 2017).

Stoica, I. *et al.* (2003) ‘Chord: A scalable peer-to-peer lookup protocol for Internet applications’, *IEEE/ACM Transactions on Networking*, 11(1), pp. 17–32. doi: 10.1109/TNET.2002.808407.

Sulaiman, A. *et al.* (2022) ‘Design, Implementation and Testing of Operational Modes in ADCS of a CubeSat’, *2022 13th International Conference on Mechanical and Aerospace Engineering, ICMAE 2022*, pp. 130–137. doi: 10.1109/ICMAE56000.2022.9852877.

Tan, Z. *et al.* (2019) ‘New Method for Positioning Using IRIDIUM Satellite Signals of Opportunity’, *IEEE Access*, 7, pp. 83412–83423. doi: 10.1109/ACCESS.2019.2924470.

The Things Network (2020) *LoRa World Record Broken: 832km/517mi using 25mW*. Available at: https://www.thethingsnetwork.org/article/lorawan-world-record-broken-twice-in-single-experiment-1 (Accessed: 10 December 2020).

Tools For Noobs (2020) *Online Hash Calculator*. Available at: https://www.tools4noobs.com/online_tools/hash/ (Accessed: 1 November 2022).

Tubío-Pardavila R, R. *et al.* (2014) ‘The humsat system: a CubeSat-based constellation for in-situ and inexpensive environmental measurements’, in *AGU Fall Meeting Abstracts*, pp. A23I--3365.

Tuwanut, P. and Kraijak, S. (2015) ‘A survey on IoT architectures, protocols, applications, security, privacy, real-world implementation and future trends’, *11th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM 2015)*, pp. 6 .-6 . doi: 10.1049/cp.2015.0714.

Vasseur, J.-P. and Dunkels, A. (2010) ‘Chapter 6 - Transport Protocols’, in Vasseur, J.-P. and Dunkels, A. (eds) *Interconnecting Smart Objects with IP*. Boston: Morgan Kaufmann, pp. 63–74. doi: https://doi.org/10.1016/B978-0-12-375165-2.00006-5.

Vint, C. and Kahn, R. (1974) ‘A protocol for packet network interconnection’, *IEEE Transactions of Communications*, 22(5), pp. 637–648.

Vladimirova, T. *et al.* (2007) ‘Characterising Wireless Sensor Motes for Space Applications 2 . Application of Wireless COTS Protocols to Small Satellites’, *2nd NASA/ESA Conf. (AHS)*, (Ahs).

Wiid, K. *et al.* (2017) ‘The Development of “nSight-1” - Earth Observation and Science in 2U’, *Small Satellite Conference*. Available at: https://digitalcommons.usu.edu/smallsat/2017/all2017/144 (Accessed: 23 January 2023).

Woldai, T. (2020) ‘The status of Earth Observation (EO) \& Geo-Information Sciences in Africa--trends and challenges’, *Geo-spatial Information Science*, 23(1), pp. 107–123.

Wood, L. (2003) ‘Satellite Constellation Networks’, in Zhang, Y. (ed.) *Internetworking and Computing Over Satellite Networks*. Boston, MA: Springer, pp. 13–34.

Zaidi, Y. and Van Zyl, R. (2017) ‘A low cost testbed and test-design methodology for nanosatellite sub-/systems’, *2017 IEEE AFRICON: Science, Technology and Innovation for Africa, AFRICON 2017*, pp. 416–423. doi: 10.1109/AFRCON.2017.8095518.