UNIVERSITY OF SOUTHAMPTON

Faculty of Engineering and Physical Sciences
School of Electronics and Computer Science

An Investigation into Passive Information Gathering for Uncooperative Ad-Hoc Mesh Network Assessment.

DOI: 10.5258/SOTON/D30111

by

Michael William O'Sullivan
PhD
ORCiD: 0000-0002-4216-4287

A thesis for the degree of
Doctor of Philosophy

July 2025

University of Southampton

Abstract

Faculty of Engineering and Physical Sciences
School of Electronics and Computer Science

Doctor of Philosophy

An Investigation into Passive Information Gathering for Uncooperative Ad-Hoc Mesh Network Assessment.

by Michael William O'Sullivan

An Ad-Hoc Mesh Network is a wireless network formed spontaneously and dynamically by a group of wireless devices without the need for infrastructure or centralised administration. Its life cycle is divided into four phases: formation, operation, maintenance, and disbandment. This thesis aims to illustrate the information available to an adversary of an Ad-Hoc Mesh Network during phases 1 and 2 of the life cycle.

The first phase focuses on the formation stages of the life cycle and investigates the nodes' initial spatial distribution. The goal is to establish a foundation for a methodology that predicts the locations of missing nodes. It is assumed that the adversary is aware of the Network Topology Generator (TG) being used, enabling them to estimate the placement of unobserved nodes based on observed node positions. The outcome for this phase of this thesis demonstrates 78%accuracy in correctly classifying the TG that produces select topology using a Gaussian Naive Bayes probabilistic clustering.

The second phase identifies the routing algorithm responsible for distributing network traffic throughout an Ad-Hoc Mesh network. From an adversarial perspective, this knowledge will enable them to follow data paths from a target node to network collection locations. The goal is to examine variations in network traffic that an outside party can identify using data available to an adversary or encrypted inter-node interactions. The research shows that of the various machine learning techniques, Support Vector Machine, Decision Tree and Random Forest gave the most accurate results of 99%. These accuracies were maintained as the sample size was reduced to 5 consecutive packets from 5 randomly chosen nodes.

In the final phase, a methodology is created to track influential nodes and investigate an extension of the operation stage of the life cycle. By predicting influential nodes, an adversary can target those nodes that will carry the most data, potentially causing data interception or network disruption. This research identifies that a Temporal Graph Networks link prediction methodology gave a 90% accuracy when determining which nodes will maintain or gain influence in the near future (roughly 9 mins of its operation). It also demonstrates that the mobility model used to generate the data does not statistically affect the outcome.

Contents

List of Figures

List of Tables

Acknowledgements

I want to sincerely thank Dr. Paul Brittan for all of his help and inspiration during the course of my studies. Sincere thanks goes out to my PhD supervisors, Prof. Vladimiro Sassone and Dr. Leonardo Aniello, for their excellent guidance and support. Lastly, I would like to thank my wife for her unwavering support and understanding over the past few years; without her, I could not have finished my studies.

Definitions and Abbreviations

AES	Advanced Encryption Standard
AH	Authentication Header
AMI	Advanced Metering Infrastructure
AODV	Ad Hoc On-Demand Distance Vector
ARP	Address Resolution Protocol
AS	Autonomous Systems
BRITE	Boston University Representative Internet Topology
BRP	Bordercast Resolution Protocol
CNA	Computer Network Attack
CND	Computer Network Defence
CNE	Computer Network Exploitation
CNN	Convolutional Neural Network
CNO	Computer Network Operation
CS	Continuous Simulation
CSV	comma-separated values
DANETs	Drone Ad Hoc Networks
DAPs	Data Aggregation Points
DARPA	Defence Advanced Research Projects Agency
DT	Decision Tree
DES	Discrete Event Simulation
DSDV	Destination Sequenced Distance Vector
DSR	Dynamic Source Routing
ESP	Encapsulating Security Payload
FANETs	Flying Ad Hoc Networks
FSS	Forward Sequential Selection
GCN	Graph Convolutional Network
GLMR	Gator Landmine Replacement
$GT-ITM$	Georgia Tech Internetwork Topology Model
HNA	Host and Network Association packet
IARP	IntrA-zone Routing Protocol
c *ICT*	Inter-Contact Times
IERP	IntEr-zone Routing Protocol

IoT	Internet of Things
IP	Internet Protocol
IPsec	Internet Protocol Security
LAN	Local Area Network
MAC	Media Access Control
MANETs	Mobile Ad Hoc Networks
MCs	Mesh Clients
MID	Multiple Interface Declaration packet
ML	Machine Learning
MRs	Mesh Routers
MWSN	Mobile Wireless Sensor Network
*NS*3	Network Simulator version 3
NPART	Node Placement Algorithm for Realistic Topologies
OLSR	Optimized Link State Routing
PCAP	Packet Capture
PDS	Privacy Database Structure
PR	Proactive Routing
QoS	Qualify of Service
RF	Random Forest
RERR	Route Error packet
RR	Reactive Routing
RREP	Route Reply packet
$RREP - ACK$	Route Reply Acknowledgement packet
RREQ	Route Request packet
SBFS	Sequential Backward Floating Selection
SBS	Sequential Backward Selection
SDN	Software Defined Networks
SFFS	Sequential Forward Floating Selection
SFS	Sequential Forward Selection
SVM	Support Vector Machine
TC	Topology Control packet
TG	Topology Generator
TGN	Temporal Graph Network
TTL	Time-to live Exceeded
TPL	Truncated Power-Law
UAVs	Unmanned Ariel Vehicles
UBI	Upper Bound Interchange
UDP	User Datagram Protocol
VANETs	Vehicular Ad Hoc Networks
*V*2*I*	Vehicle to Infrastructure
*V*2*RL*	Vehicle to Real

$V2U$	Vehicle to User
$V2V$	Vehicle to Vehicle
$WIN-T$	Warrior Information Network – Tactical
$WINS$	Wireless Integrated Network Sensors
$WLANs$	Wireless Local Area Networks
WSN	Wireless Sensor Network
ZRP	Zone Routing Protocol

Chapter 1

Introduction

This thesis explores how passive information gathering can uncover critical insights about uncooperative ad hoc mesh networks. In which we mean, **passive gathering** refers to that which is observing network traffic without directly interacting with the network, while **uncooperative** means that the target network does not under the control of the person analysing the network.

Ad hoc mesh networks operate using a decentralised topology, where devices (or nodes) work together to implement routing protocols. Each node not only transmits its own data but also forwards traffic for others nodes , this transmission of data following a specific algorithm to reach the intended destination. Unlike traditional networks, ad hoc mesh networks lack fixed infrastructure, and communication is limited to nodes within each other's transmission range. These networks have a wide range of applications, from wireless sensor networks and vehicular ad hoc networks (VANETs) to mobile ad hoc networks (MANETs). Their use extends from everyday scenarios to high-stakes environments like military operations.

Many aspects of ad hoc mesh networks remain active areas of research, including routing protocols Cheng and Hancke (2016); Cheng and Lin (2017) and security Xu et al. (2016, 2017)). To evaluate proposed solutions, researchers often rely on simulations, which allow for preliminary testing and validation of effectiveness. Simulations typically involve testing various approaches on different network topologies. This is to ensure that the results are not limited to specific configurations. From the paper Günes and Akgün (2011), designing and selecting appropriate test network topologies is a crucial step in any network protocol simulation.

1.1 Applications of Ad Hoc Mesh Networks

Ad hoc mesh networks play a key role in various fields:

- **Mobile Ad Hoc Networks (MANETS)**

  Military Operations: These networks, such as the Joint Tactical Radio System (JTRS) published in (Schiavone et al. (2006)) and Warrior Information Network–Tactical (WIN-T) which can be found on the website (Win), provide dynamic communication in environments without fixed infrastructure.

  Challenges: Frequent topology changes due to node mobility and interruptions in line-of-sight (LOS) connectivity.

- **Gator Landmine Replacement (GLMR) program**

  This has been developed as a networked munition system. Networked munitions are designed to replace traditional anti-personnel persistent and non-persistent munitions. The sensor and communications capabilities of the system will notify a soldier that someone is close to the munition when an enemy enters a networked munitions field, as described in the project website [1]

- **Vehicular ad hoc networks (VANETs)**

  This is where the concept of mobile ad hoc networks has been applied to the domain of vehicles. Information can be relayed via vehicle-to-vehicle or vehicle-to-infrastructure communications for road safety, navigation, and other road services.

- **Mobile wireless sensor network (MWSN)**

  This is where a network is made up of mobile sensor nodes in a wireless sensor network (WSN). This is again typified by the rapid topology-changing environment in the area of environmental monitoring or surveillance.

- **Drone ad hoc networks (DANETs)**

  Also called Flying Ad Hoc Networks (FANETs), these are employed for military reconnaissance and civil applications like photogrammetry and search-and-rescue operations.

Despite their varied applications, research into adversarial routing identification and undisclosed routing in these networks remains limited. This is increasingly critical as ad hoc mesh networks are deployed in military and defence contexts, as set out in the project web-page [2]

1.2 Evolution and Importance of Ad Hoc Mesh Networks

Originally developed for military applications by organizations like DARPA Wang et al. (2009), ad hoc mesh networks were valued for their self-healing capabilities,

[1] https://www.army.mil/article/165263/peo_ammo_stands_up_new_product_management_office
[2] https://mwi.usma.edu/era-drone-swarm-coming-need-ready/

allowing nodes to join or leave the network without disrupting overall performance. Early civilian adoption was limited due to challenges like low throughput and weak security. However, advances in multi-hop communication and dynamic topology have led to broader research and real-world applications in areas such as Vehicular Ad Hoc Networks (VANETs), Mobile Wireless Sensor Networks (MWSNs), and Disruption-Tolerant Networks (DANETs).

Examples of Emerging Use Cases:

- **VANETs:** In urban environments, vehicles communicate dynamically with minimal constraints on power, storage, or computational resources (Lèbre et al. (2014)).

  Subcategories of VANETs include:

  - Vehicle-to-Real (V2RL): Real-world autonomous vehicle applications, such as Google's driverless car project (Luettel et al. (2012)).
  - Vehicle-to-Vehicle (V2V): Direct communication between cars to enhance traffic safety and coordination (Biswas et al. (2006)).
  - Vehicle-to-Infrastructure (V2I): Interaction between vehicles and roadside units or smart infrastructure (Tee and Lee (2010)).
  - Vehicle-to-User (V2U): Systems that enable direct communication between vehicles and personal devices (Birk (2011)).

- **Wireless Integrated Network Sensors** (WINS):

  WINS technology is widely used in fields such as healthcare, industrial automation, and environmental monitoring. A subset of WINS, known as Personal Area Networks (PANs), supports wearable devices for tracking physiological signals. While PANs are commonly wireless, wired implementations are often preferred for short-range applications, such as EEG monitoring in telemedicine Martin et al. (2000) & Jovanov et al. (1999).

Types of Ad Hoc Mesh Networks

Ad hoc mesh networks can be classified into two main categories:

- **Homogeneous Networks**: In these networks, all nodes are identical and perform the same functions. One example is client-based wireless mobile networks, where peer-to-peer communication occurs between mobile devices. A notable example is drone swarms, which rely on uniform nodes for collective operation.
- **Heterogeneous Networks**: These networks consist of different types of nodes with specialised roles, such as mesh routers (MRs) and mesh clients (MCs). A

common example is infrastructure-based backbone networks used in community setups, where routers serve as access points for local users. This model is often seen in community and neighbourhood networks Akyildiz et al. (2005).

1.3 Research Gaps and Motivations

Research on ad hoc mesh networks has largely focused on improving **routing protocols** and **security**. However, there is limited investigation into scenarios where the network's routing mechanisms are undisclosed or adversarial entities aim to identify them. This gap is critical, especially for military and defence applications, where understanding adversarial strategies and network vulnerabilities could have significant implications, such as vulnerability to adversarial attacks, inability to predict network behaviour or compromised network resilience. This thesis addresses this gap by exploring methodologies to gather actionable insights from passive observations.

The study aims to:

- **Expose security weaknesses** in ad hoc mesh networks by examining how passive data gathering can reveal crucial information about network topology, routing algorithms, and influential nodes.
- **Investigate the impact of topology generators** (TGs) on network simulations, highlighting how biases introduced by these TGs can assist adversaries to infer node placements.
- **Identify and predict influential nodes** in dynamic, mobile networks, providing insights into how attackers might target key nodes for interception or disruption.
- **Develop machine learning methods** to classify routing algorithms used in encrypted networks, simulating real-world attack strategies.
- **Strengthen network security** by proposing ways to minimise TG bias and better predict node importance, helping network operators defend against these emerging threats.

1.4 Aim of Research

Ad Hoc Mesh Networks are wireless networks that form naturally and flexibly among wireless devices without requiring centralised administration or infrastructure. The lifecycle of these networks can be divided into the following phases as set out in Zhan et al. (2022):

1. **Formation** During this phase, wireless devices that are in close proximity to one another construct the network. To build a network topology and choose a routing protocol, the devices talk to one another.
2. **Operation**: After the network has been established, the devices begin exchanging data with one another. The optimal path for data transfer is chosen using the routing protocol.
3. **Maintenance**: During this phase, the network is examined to make sure it is operating properly. The network topology is monitored by the routing protocol, which also updates the routing tables to reflect any changes.
4. **Disbandment**: Either all of the devices depart the network or the network is no longer required. To update their routing tables and rearrange the network architecture, the remaining devices connect.

This research explores what information an adversary can access during the **Formation** and **Operation** phases. It demonstrates how passive data gathering can reveal critical details throughout the lifecycle, dividing the study into three phases.

The first phase focuses on the formation stages of the life cycle and investigates the nodes' initial spatial distribution. The goal of the research is to lay the foundation for a methodology to predict the location of missing nodes. The initial step is to determine the Network Topology Generator (TG) that generated the node placement of observed nodes.

1.5 Research Phases

1.5.1 Phase 1: Formation Phase Analysis

As set out in the paper by Alrumaih and Alenazi (2023), topology generators allow network node's placement to be designed before simulation. Simulating these network topology provides a practical method for designing, testing, and implementing complex real-world networks. This enables the pre-planning of ad hoc mesh network deployments.

Additionally, Camilo et al. (2007) discuss sensor networks in which nodes can be either randomly deployed or arranged using a topology generator. Random placement often results in uneven area coverage, whereas a topology generator allows for calculated node placement. This approach also carries risks that an adversary could use the generator's instructions to deduce the locations of any unobserved nodes by analysing the nodes that are visible, potentially targeting key links during or after initial placement, as noted by Bhunia et al. (2018).

Hric et al. (2016) introduces an approach to predict missing nodes using network metadata, differing from methods that primarily predict missing edges, such as those proposed by Cimini et al. (2015); Musmeci et al. (2013). While edge prediction is popular, it does not address scenarios where entire nodes must be forecasted but remain unobserved.

1.5.2 Phase 2: Identifying the Routing Algorithm

The second phase of the research focuses on identifying the routing algorithm responsible for distributing traffic in an ad hoc mesh network, addressing part of the second life cycle stage. From an adversarial perspective, understanding these routing mechanisms enables tracking of data paths from target nodes to network collection points.

With Ad Hoc Mesh Networks, unlike traditional networks that broadcast packets over segments, they use point-to-point links for data distribution. Once the network is discovered, a path is established for data flow.

Research on mesh network security and attacks, such as that by Siddiqui et al. (2007), underscores the importance of understanding the network's architecture and routing scheme for effective exploitation. This phase investigates variations in network traffic that adversaries can detect, relying on data accessible to them—such as encrypted inter-node communications—to ensure realistic analysis.

1.5.3 Phase 3: Tracking Influential Nodes

The final phase develops a method for identifying and tracking **influential nodes** during the Operation stage and predicts their roles beyond the observed data.

Influential nodes are those responsible for a significant portion of the network's data flow. Predicting these nodes allows an adversary to strategically target them for interception or disruption. Successfully identifying these critical nodes provides an advantage in compromising the network's overall functionality.

1.6 Ad Hoc Mesh Network Vulnerabilities

Research highlights key vulnerabilities that adversaries can exploit during the network lifecycle:

- During the **Formation** phase, adversaries can infer the location of unobserved nodes using topology generators and observed node placements.

- In the **Operation** phase, routing algorithms can be identified to trace data paths and predict traffic patterns.
- By tracking the **influential nodes** within a network, an adversary can target those nodes that will become important for data transfer either to disrupting or intercepting communication.

This study demonstrates how adversaries can exploit passive data gathering to uncover crucial network details, even in secure or encrypted environments.

1.6.1 Vulnerability in Ad Hoc Mesh Networks

A vulnerability is any weakness in a device, network, or system that an attacker can exploit. In the case of uncooperative networks, identifying these vulnerabilities is the first step toward gathering useful intelligence. Ad hoc mesh networks, in particular, have several security gaps that adversaries can take advantage of, depending on the communication layers involved.

The type of attack used depends on the attacker's goal—whether it's to disrupt network operations or silently intercept information. Some common tactics include passive eavesdropping, jamming, MAC address spoofing, and traffic replay attacks Sen (2013). Each of these methods is designed to manipulate network behaviour in a specific way, either by injecting malicious activity or by passively monitoring data flows.

To counter such threats, the military has developed a structured approach to cyber defence and offensive operations. One such framework is the Computer Network Operation (CNO) strategy, introduced by Scaparrotti, which divides cyber activities into three main categories:

- **Computer Network Attack** (CNA): Actions aimed at disrupting or disabling network functionality.
- **Computer Network Defence** (CND): Security measures designed to protect and secure network infrastructure.
- **Computer Network Exploitation** (CNE): Gaining unauthorised access to extract valuable intelligence.

As part of CNE, the Cyber Kill Chain, developed by Lockheed Martin, outlines the key steps attackers follow to infiltrate a network. The first stage —reconnaissance —focuses on gathering intelligence about a network's structure and routing processes. In traditional networks, attackers might use several techniques to map out their targets, including:

- **Host Detection**: Identifying active devices within the network.
- **Port Enumeration**: Scanning for open and accessible network ports.
- **Vulnerability Assessment**: Analysing security flaws that could be exploited.

Shaikh et al. (2008) highlight how attackers use these techniques to uncover critical details, such as which devices are running key services, what ports are left exposed, and even the type and version of operating systems in use. Similarly, White et al. (2019) discuss how passive network sniffing can reveal the overall topology of a system.

When it comes to ad hoc mesh networks, things get even more complex. Because these networks are decentralised and mobile, attackers have additional opportunities to exploit shifting node positions, unstable connections, and dynamic routing protocols. Unlike traditional networks, where devices typically remain in fixed locations, ad hoc mesh networks constantly evolve—making security both more challenging and more critical.

1.6.2 Information Gathering

The goal of information gathering is to collect as much relevant data as possible about a specific target. Depending on the method used, a significant amount of valuable information can be uncovered. In broadband wireless networks, the use of air as the medium for data transmission makes them particularly susceptible to eavesdropping. This vulnerability arises from the open nature of wireless communications. Attackers can exploit intercepted communications to identify and disable sensor nodes by uncovering their physical locations. Additionally, adversaries may access application-specific message content, such as message IDs, timestamps, and other metadata, alongside the nodes' positions Padmavathi et al. (2009).

In traditional networks, information gathering can be classified into two main types: **active** and **passive**.

1.6.3 Active Gathering

Active information gathering involves directly interacting with the target network to collect data. While this method can yield more extensive information than passive gathering, it is often unauthorised and carries the risk of triggering the target system's security alerts. The direct interaction between the attacker and the target increases the likelihood of detection. A common example of active gathering is **port scanning**, a technique used to identify open or closed ports at an IP address. By scanning ports,

attackers can determine which services are actively running on the target network. Each active service represents a potential vulnerability that could be exploited.

1.6.4 Passive Gathering

Passive information gathering seeks to learn about the target network's architecture without directly interacting with its components. This method is safer and less intrusive, as it does not engage the target system. Instead, the attacker queries the target system indirectly, often through intermediary systems, minimizing the risk of detection. The primary goal of passive information gathering is to observe the target network and collect data discreetly. This technique is often preferred in scenarios where avoiding detection is critical.

1.6.5 Information Gathering on Ad-Hoc Mesh Networks

When applied to Ad Hoc Mesh Networks, information gathering aims to determine what data can be passively collected from the network. This research focuses exclusively on passive methods. Examples include:

- **Promiscuous Interception**: Listening to data transmissions from endpoint or transition nodes without participating in the communication. Only the necessary operational data is intercepted and analysed.
- **Observation**: Examining the spatial distribution of nodes within the network to infer patterns or structures.

Ad Hoc Mesh Networks, like other wireless systems, are inherently vulnerable to eavesdropping due to their use of air as a transmission medium. As noted by Khan et al. (2008), wireless connections are particularly susceptible to interception. Passive listening devices, or "sniffers," can be strategically positioned to capture transmitted data discreetly. This vulnerability has been demonstrated in the context of the Internet of Things (IoT), as shown by Crnogorac et al. (2022). By leveraging these passive techniques, attackers can gather critical data about network operations without alerting the target system, emphasizing the need for enhanced security measures in wireless environments.

1.7 Topology Bias

Chapter 4 explores the impact of Network Topology Generators (TGs), which are commonly used to create network topologies based on predefined models, real-world

measurements, and tuneable parameters. While TGs are designed to generate representative topologies, their differing assumptions and methods often lead to diverse outputs. These variations can influence simulation outcomes, as highlighted by Magoni and Pansiot (2001); Heckmann et al. (2003).

This raises the hypothesis that the choice of topology generator (TG) may introduce bias in Ad-Hoc Mesh Network simulations, as TGs determine the initial placement of nodes. Given that network topology significantly influences how the network evolves, knowledge of the TG could enable adversaries to infer the original positions of nodes.

Given that TGs differ in methodology, researchers often select a TG based on the specific mathematical or physical model they need. To minimise bias, it is recommended to use topologies from multiple TGs during experiments, ensuring results are not skewed by a specific TG or subset of TGs. This research examines the initial node placement generated by various TGs, analysing differences to reduce the number of TGs required while maintaining representative topology diversity. The chapter addresses the following research questions:

- **Research Question 1**: How to measure the difference between topologies generated by distinct TGs?
- **Research Question 2**: How to choose what TG, or TGs, to reduce such a bias?

The approach involves creating a **numeric representation of topologies** using features like inter-node distance, clustering, and node density. Topologies are modelled as vectors, enabling comparison through distance metrics. Bias is quantified using Hedges' g effect size measure to compute a **bias index** that evaluates differences between topologies generated by specific TGs or subsets versus all TGs.

Machine learning techniques are employed to assess classification accuracy—determining how accurately the source TG of a topology can be identified. The methodology to reduce bias uses the bias index to select TGs, depending on the allowable number of TGs in use.

Key findings and contributions include:

- Using a single TG likely introduces bias, but among individual TGs, **NPART** is the least biased.
- For two TGs, **BRITE** and **NPART** minimise bias.
- Topologies can be classified with nearly **78% accuracy** using specific topology features.

- A framework to minimise bias based on TG features like inter-node distance, spatial distribution, and clustering.

This research, the first to systematically investigate TG-induced bias in Ad-Hoc Mesh Networks, demonstrates significant differences in TG outputs and proposes practical methods for minimizing bias.

1.8 Routing Algorithm Reconnaissance

With the node placement biases established, the next logical step was to investigate how these placements influence routing behaviors within ad hoc mesh networks. Chapter 5 explores the routing algorithms used in the target mesh network, aiming to classify which specific algorithm is in use. Most research on ad hoc mesh networks, such as Ian F. and Xudong (2005) and Grossglauser and Tse (2002), has primarily focused on enhancing network capacity or scalability, as detailed by Ou et al. (2004) and Eriksson et al. (2007), rather than addressing security. Originally, mesh networks were designed with minimal or no security measures, relying instead on enforcement at the application layer, as discussed by Lopez et al. (2021). Securing ad hoc mesh networks and their protocols poses challenges due to their open medium, dynamic topology, and distributed nature, as explored in Shi-Chang et al. (2010), Lou et al. (2009).

The inherent characteristics of ad hoc mesh networks, as highlighted by Olakanmi and Dada (2020), create numerous vulnerabilities, including intrusion, data modification, and identity tracing. Data is transmitted from a source node to a destination through multiple intermediary nodes, exposing these networks to potential attacks. Adversaries targeting such networks are likely to operate in hostile environments, prompting network operators to implement stronger security measures, such as traffic encryption, to protect data distribution.

If the development of ad hoc mesh networks follows a trajectory similar to traditional networks, packet encryption will likely become a standard security feature. This underscores two primary motivations for this research: (1) understanding how a network attack might be executed and (2) addressing scenarios where the entire network is not visible to the attacker.

In ad hoc mesh networks, mobility introduces a new dimension, allowing attackers to analyze how data flows through the network. These flows depend heavily on the routing algorithm in use. Identifying the algorithm from a non-intrusive standpoint could provide attackers with critical insights. To account for the anticipated encryption of future networks and ensure the relevance of this research, the study relies on metadata from encrypted traffic.

Research Questions This phase of the research seeks to address the following questions:

- **Research Question 3**: What is the most accurate machine learning based approach to detect the routing algorithm used in an ad-hoc mesh network?
- **Research Question 4**: How is the detection accuracy affected if we consider routing algorithms of the same class?
- **Research Question 5**: How is the detection accuracy affected if we reduce either the number of nodes from which data is gathered or how much data is collected from each node?

RQ3 and RQ4 are investigated by creating multiple networks, capturing traffic routed through these networks during various simulations, and applying machine learning techniques to predict routing algorithms. The analysis identifies methods yielding the highest accuracy.

RQ5 builds on this by selecting the most effective combinations of packet fields and gradually reducing the dataset size to evaluate the point where accuracy becomes unacceptable.

The study examines two datasets. The first contains data from two classes of routing algorithms: AODV (Reactive Routing) and OLSR (Proactive Routing). The second includes three algorithms: AODV, OLSR, and Dynamic Source Routing (DSR), with DSR providing an additional example of a reactive routing algorithm.

1.9 Influential Node Detection

Chapter 6 explores the use of link prediction to assess node importance, aiming to forecast which nodes will become influential in the near future. Identifying influential nodes is crucial across various domains, from technological to biological networks, as discussed in the works of Zhang et al. (2013) Buldyrev et al. (2010), Fath et al. (2007). This research focuses specifically on wireless sensor networks (WSNs), which rely on peer-to-peer communication to relay sensor data to sink nodes.

In such networks, data flow toward the sink nodes can create bottlenecks when one or more nodes become primary facilitators of this flow, potentially leading to resource issues. For an adversary, the ability to predict these facilitator nodes offers strategic advantages, such as capturing nodes to disrupt or manipulate data flow or increasing opportunities for intercepting content. On the other hand, network operators would benefit from identifying these nodes in advance to optimise their performance and

adapt to the data flow demands. Facilitator nodes can be identified by determining which nodes are most influential within the network.

In mobile networks, identifying future flow facilitators becomes increasingly complex. As nodes shift positions to gather data, existing links break, and new connections are formed, complicating the prediction of influential nodes. This highlights the temporal nature of the network: nodes deemed highly influential at the beginning of a deployment may not hold the same status midway or at its conclusion.

This phase of research raises the questions

- **Research Question 6**: When considering a dynamic ad-hoc mesh network with a temporal-dependent, what type of link prediction is the most effective?
- **Research Question 7**: Considering the same temporal-dependent ad-hoc mesh network and how the mapping of the participating node's influence changes over time. How accurate is predicting the node's influence compared with those calculated using the ground truth?
- **Research Question 8**: Is there any significant impact on predicting nodal influencer ranking by the original mobility model for the data generation?

1.10 Structure of the Thesis

- **Chapter 2:** Background.
- **Chapter 3:** Related Work.
- **Chapter 4:** Research on Topology Bias.
- **Chapter 5:** Routing Algorithm Reconnaissance.
- **Chapter 6:** Influential Node Detection.
- **Chapter 7:** Discussion.
- **Chapter 8:** Conclusions and Future Work.

Chapter 2

Background

This chapter aims to review the approaches taken for data generation, machine learning techniques and other concepts used within the research undertaken for this thesis.

2.1 Data Generation

The data for this research and analysis is generated via a simulated model. A simulation is an algorithm that simulates a real-world event and can be employed to test various strategies. The capacity to compare millions of possibilities using a simulation is extremely advantageous when dealing with a very complex system. It should be kept in mind that simulations can never be as precise as their real-world equivalents.

Beyond the costs associated with using real-world data, simulations are used for a variety of other reasons. It enables tests that may be repeated with established outcomes by utilising predetermined parameters. Furthermore, this is quite rare in practice; it is more likely that a smaller dataset is accessible. It is typical in a simulated environment to be able to collect data from all involved nodes. There will be situations where using real-time data is better, such as when determining how environmental and geographic factors affect a network that may span kilometres. The capacity to acquire all data is achievable when using live data, but it would be costly and impractical.

The type of simulation used is in the form of discrete event simulation (DES) as opposed to continuous simulation (CS). DES operates a sequence of discrete events that occur in different time intervals, whereas CS continuously tracks system responses through the duration of the simulation. This is done to imitate a system that consists of a network of separate nodes and the data that travels between them. DES

was chosen as the data generation method because the transmissions take the form of discrete occurrences known as packets.

2.1.1 NS3

NS3.29 is the simulation tool used for this study. NS3, a network simulator described by Riley and Henderson (2010), employs discrete event simulation to model network behaviour. To create an effective test environment, the nodes in the simulation must be distributed across the area in such a way that they can form and break connections, which helps generate a representative dataset for analysis. The nodes are placed within the predefined test region using various Topology Generators (TGs), and the simulation runs for a specified duration. Each node in the simulation functions as a Wi-Fi device, using the 802.11a protocol. As the simulation runs, each node collects Packet Capture (PCAP) files, recording any network traffic entering or leaving that node.

2.2 Topology Generation

Before starting any simulation, the participating nodes need to be positioned within the test area. To avoid introducing any bias, the nodes must be placed randomly. If done manually, there's a natural tendency to place nodes in areas where none are present, which could bias the experiment. To prevent this, Topology Generators (TGs) are used, which employ specific algorithms to generate pseudo-random node placements. The main concern is whether these TGs introduce bias themselves, and whether this can be measured. This issue is addressed in the first phase of the research, with further details in 4, which has been published in O'Sullivan et al. (2020).

The node placement can be based on predefined models or real-world measurements, as described in 3.

2.2.1 Boston University Representative Internet Topology

The first TG used in this study is the Boston University Representative Internet Topology (BRITE), which was introduced by Medina et al. (2001b). BRITE is a universal, model-based TG that allows for the addition of new models. Unlike model-driven TGs, which are based on a specific design model, BRITE is designed to be flexible and extendable. It uses various models for node placement, such as the Flat model, Hierarchical model, and other sub models.

- **Flat Router model**:

  This model focuses on router-level topologies, specifically for router networks. It offers two versions: RouterWaxman and RouterBarabasiAlbert, both of which place nodes randomly or using a heavy-tailed approach.

- **Flat AS-level model**:

  This is similar to the Flat Router model but is used for generating AS-level topologies. It also includes two versions: ASWaxman and ASBarabasiAlbert.

- **Hierarchical Topologies model**:

  This model creates Internet-like topologies using either a top-down or bottom-up approach, building router-level topologies first, and then adding AS nodes.

For our experiments, the Flat AS-Level model was chosen because it is more representative of a mesh network, generating router-level topologies with interconnected routers. In studies of node placement, the connection style was less important than the node placement itself.

The configuration file for the BRITE topology generator can be summarised:

- **Topology Type**:
  - **1**: AS-level
  - **2**: Router-level
  - **3**: Imported File
- **N**: Number of nodes
- **HS**: 1000 Size of the plane (horizontal scale)
- **LS**: 1000 Size of the plane (vertical scale)
- **NodePlacement**:
  - **1**: Random
  - **2**: Heavy-tailed
  - **3**: Imported
- **GrowthType**
  - **1**: Incremental
  - **2**: All at once)
- **PreferentialConnectivity**:

  - **0**: None
  - **1**: On

- **EdgeConnection**
  - **1**: Random
  - **2**: Nearest Neighbor
  - **3**: Mixed

- **Bandwidth**:
  - **1**: Constant
  - **2**: Uniform
  - **3**: Heavy-tailed
  - **4**: Exponential

- **BWMin**: Minimum bandwidth value
- **BWMax**: Maximum bandwidth value
- **OutputBRITE**:
  - **1**: Enabled
  - **0**: Disabled

- **OutputOTTER**
  - **1**: Enabled
  - **0**: Disabled)

- **OutputNS**:
  - **1**: Enabled,
  - **0**: Disabled

- **OutputDML**:
  - **1**: Enabled
  - **0**: Disabled

- **OutputJavasim**:
  - **1**: Enabled
  - **0**: Disabled

2.2.2 Node Placement Algorithm for Realistic Topologies

The Node Placement Algorithm for Realistic Topologies (NPART), as proposed by Milic and Malek (2009), generates network topologies based on real-world properties. Unlike purely random networks, NPART generates topologies that respect several features of real networks. Some sociological and technological factors that influence network structure were identified in earlier research Aha and Bankert (1996), which guided the development of this algorithm. Key aspects include:

- New participants are more likely to join areas with high connectivity.
- Every participant is expected to have at least one connection to the rest of the network, possibly creating a large number of pendant nodes.
- A pendant node can become a seed for a new subnetwork.
- The network specifies the area it occupies, rather than the other way around.

NPART offers various configuration options for generating topologies, including models based on real-world cities such as Berlin (275 nodes) and Leipzig (346 nodes), along with uniform, grid, quasi-grid, and random waypoint models.

The configuration file for NPART is of the following style:

- **Area size**:

  Defining the geographical boundaries of the simulated network.
- **Node density**:

  Specifying the average number of nodes per unit area.
- **Placement algorithm type**:

  Choosing between different algorithms like ”uniform random”, ”grid based”, or ”city-based”.
- **Geographical data file**:

  Specifying a file containing real-world geographical information to guide node placement

2.2.3 Georgia Tech Internetwork Topology Model

The Georgia Tech Internetwork Topology Model (GT-ITM), introduced by Calvert et al. (1997), is another model-based TG designed for wide-area networks. It offers two main node placement options: Flat Random Graphs and Hierarchical models.

- **Flat Random Graphs**:

  This model distributes nodes randomly across the test area. While not intended to mirror real-world topologies exactly, it is simple and widely used for testing. A variation of this model uses Waxman probability to generate more realistic topologies.

- **Hierarchical model**:

  This model connects smaller components according to a larger-scale structure, suggesting that it is more suitable for clustering.

For our study, the Flat Random Graph model was selected because it better represents ad hoc mesh networks.

The simple configuration parameters that govern this generator can be summarised as:

- **Network size**:

  Specifies the desired number of nodes in your network.

- **Hierarchy levels**: Set the number of levels in the network hierarchy (e.g., how many transit levels).

- **Connection probabilities**:

  Defines the likelihood of connections between nodes at different levels.

- **Seed value**:

  Sets a seed value for the random number generator to ensure reproducibility of your generated network.

2.3 Routing Protocols

The mechanism to route data within Ad-hoc mesh networks can be divided into 3 classes: **Proactive, Reactive and Hybrid**. This grouping describes how the network finds and maintains the routing tables for the participating nodes. There are differences in the way the data packets are presented to the network between these classes, and there are subtle differences between the different algorithms within the same class.

The routing protocols are responsible for establishing paths between the source and the destination. It is also responsible for maintaining the path between these two nodes until the communication has finished, as described in Jorg (2003)

2.3.1 Reactive Routing

The first routing algorithm to consider is the reactive routing protocol, these are also known as on-demand routing algorithms. The differentiating factor for the reactive protocols is that it does not need to maintain a route between all pairs of network nodes. It establishes the route from the source node to the destination node when a data packet needs to be transmitted.

2.3.1.1 Ad-hoc On-Demand Distance Vector (AODV)

Like other Reactive Routing protocols, AODV published in Perkins et al. (2003) determines the route to the destination when a node needs to send data to that destination. The entries in the node's routing table are kept for as long as they are needed. If a source node needs to send data to a destination, the routing table is first queried, if no entry exists for the destination then the routing request mechanism is activated. During this phase, a route request packet (RREQ) is sent out to all neighbouring nodes. These nodes, in turn, will check their routing tables for the destination in question. If there is no entry for the destination, the RREQ is rebroadcasted. When the destination node is reached, or an intermediate node has a current route to the destination, a route reply message (RREP) is then sent back to the source, creating a path from source to the destination in its routing table. Other packets used within this protocol include 'HELLO' messages, which are used locally to determine a node's neighbours but perform poorly in an 802.11 environment. A route error message (RERR) is generated when a link between nodes is lost.

AODV packets These packets are transmitted by the nodes participating in the AODV protocol and are characterised into several types.

1. **Address Resolution Protocol** , referred to as (ARP) is generated when a node does not know the MAC layer address of the next hop in the path. The ARP packet is broadcast and asks if any of the nodes are using the address. When a node recognises the address as its own, it replies to ARP with its own MAC address which can then update the originator node's address cache.

2. **Route Request** , referred to as (RREQ), is a multi-cast message transmitted by a node at the initiation of the route discovery process when it needs to transmit data to a destination. It is received by all of the node's neighbours, who, if they do not have a route to the required destination, will in turn rebroadcast to its neighbours.

3. **Route Reply** , referred to as (RREP), is the message sent in response to receiving the RREQ in one of two cases. First, if the node is the destination, or secondly, if

the node has a valid route to the destination. In either case, the node transmits the RREP back to the node that sent the RREQ message, which is then forwarded to the originator of the Routing Request along the same path the original RREP was sent.

4. **Route Error** , referred to as RERR, is the message sent when a link cannot transmit a packet between two nodes due to connection interruption. The RERR is sent by the node that has detected the interruption back along the path to the originating source.

5. **Route Reply Acknowledgement** , referred to as RREP-ACK, is the message that is sent in response to a RREP message. This is done when the original RREP has asked for an acknowledgement by setting a flag in the message.

6. **Time-to-Live Exceeded** This packet occurs when a routing request exceeds a time-to-live (TTL) value of a packet that reaches zero and notification is then returned to the originator. It is a mechanism that is designed to prevent unnecessary network-wide dissemination of routing request packets, and its value is set in the header of a routing request. The mechanism that describes this behaviour is complicated but to simplify it, the node that sends out the routing request with the TTL flag set to a predefined value (usually 2). If the routing request times out without a corresponding routing reply, the originator broadcasts the request again with the TTL incremented by a predefined value (usually 2). This mechanism continues until the TTL reaches a third value (usually 7).

7. **HELLO** HELLO messages are used for detecting and monitoring links to the neighbour's status. Depending on the configuration, the node may offer connectivity information by broadcasting local HELLO messages. In 802.11, broadcasting is often done at a lower bit rate than unicasting, thus HELLO messages can travel further than unicast data. When not implemented the network uses layer 2 feedback. A Link is considered to be broken if frame transmission results in a transmission failure for all retries. This mechanism is meant for active links and works faster than using the HELLO method.

8. **Data Packet** The remainder of the packets are classified as data packets. These packets contain user data that is being transmitted from the source to destination nodes and are of various sizes depending on the amount of data being transmitted.

2.3.1.2 Dynamic Source Routing (DSR)

DSR published in Johnson and Maltz (1996) & Johnson et al. (2001) is another Reactive Routing protocol, where it only determines a route when the source node needs to

send data to a destination. There are two mechanisms for routing: discovery and maintenance. The discovery mechanism builds a route from source to destination within the header of a ROUTE REQUEST packet which it sends to its neighbours. Each node that forwards this ROUTE REQUEST adds to the sequence of the header. When the packet reaches the destination it replies to the source with a ROUTE REPLY including the route taken in the reply. The maintenance mechanism used within this protocol utilises the fact that in each hop of the transmission between source and destination, a delivery receipt is generated on successful delivery. If a particular node does not receive this receipt within a defined limit, it will notify the originator with a ROUTE ERROR. The source will remove this link from its routing table and perform a discovery mechanism.

DSR Packets The packets that are transmitted by the nodes are characterised into several types.

1. **Address Resolution Protocol** As described in the AODV packet section.
2. **Route Request** is part of the route discovery phase where the initiating node transmits a local broadcast packet, which is received by all the nodes within the node's wireless transmission range. The message identifies the node that initiates the request, the target and a unique request ID. It also contains a record listing the address of each intermediate node through which this particular message has been forwarded.
3. **Route Reply** is the packet returned by the route discovery back to the initiator. It contains a copy of the accumulated route recorded in the Route Request. If the initiator has already received another ROUTE REQUEST message from the bearing with the same request ID, it is discarded.
4. **Route Error** is the packet that is returned to the original sender if the original route request is re-transmitted more than the maximum number of allowed hops with no receipt confirmation. The route error packet will identify the link over which the packet could not be forwarded.
5. **Acknowledgement** 3 mechanisms are used to confirm the reachability of the next-hop when transmitting from source to destination. A route request acknowledgement may be used. The network can use a low-level acknowledgement signal provided by the node's MAC layer protocol. Alternatively, the sender may be able to 'hear' that the host is transmitting the packet again; this is termed 'passive acknowledgement'.
6. **Data Packet** As described in the AODV packet section.

2.3.2 Proactive Routing

Unlike reactive routing, in proactive routing, each node in the network has a routing table to determine the path a data packet needs to travel when being transmitted to other nodes in the network. The individual nodes hold records for all the destinations in the routing table, with the number of hops required to reach each destination. The routing entry is tagged with a sequence number created by the destination node. To maintain stability, each station broadcasts and modifies its routing table from time to time.

2.3.2.1 Optimized Link State Routing Protocol (OLSR)

OLSR published in Clausen and Jacquet (2003) is a routing algorithm within the Proactive style, and it operates as a table-driven system for the routing mechanism. Nodes within the network use a 'HELLO' message to discover their direct neighbours - (1-hop neighbour) and their neighbours that need a second hop to reach - (2-hop neighbour). Using this information, multipoint relays (MPRs) are elected, and these become a set of nodes that together can re-transmit network changes via Topology Control (TC) messages. Via this mechanism, each node maintains a network topology and chooses the MPR that offers the best route to the destination that they need to transmit to.

1. **Address Resolution Protocol** As described in the AODV packet section.

2. **HELLO** message is transmitted at regular intervals to detect neighbours and the state of the links to them. The initiating node (A) sends an empty HELLO message to a neighbouring node (B) which registers the initiator as an asymmetric neighbour because B cannot find its address in the HELLO message. B then sends a HELLO declaring A as an asymmetric neighbour. When A receives this message it finds its address in it and therefore sets B as a symmetric neighbour. This time A includes B in the HELLO it sends, and B registers A as a symmetric neighbour upon reception of the HELLO message.

   Upon receiving a HELLO message from a symmetric neighbour, all reported symmetric neighbours, not including addresses belonging to the local node, are added or updated in the two-hop neighbour set. Entries in the two-hop neighbour set are all based on main addresses, so for all received entries in the HELLO message, the MID set is queried for the main address.

3. **Topology Control** (TC). TC messages are flooded using the multipoint relay (MPR) optimisation. This is done on a regular interval, but TC messages are also generated immediately when changes are detected in the MPR selector set.

4. **Multiple Interface Declaration** referred to as (MID) messages and is used to declare the presence of multiple interfaces on a node.
5. **Host and Network Association** referred to as (HNA). This message is used when a node announces itself as a gateway.
6. **Data Packet** As described in the AODV packet section.

2.3.2.2 Destination Sequenced Distance Vector Protocol (DSDV)

DSDV set out in (Perkins and Bhagwat (1994)) is another type of proactive routing protocol. Each node's routing table is updated periodically. These tables contain Node, Destination, Next Hop, Distance and Sequence Number. The Distance field specifies the hop count from the source to the destination, and the sequence number is generated from the destination and ensures route freshness. The tables can be updated in one of two methods, a full dump of all routing information or as an incremental dump of information changed since that last full dump.

1. **Address Resolution Protocol** As described in the AODV packet section.
2. **Periodic Updates** and is sometimes referred to as full dump, Every node periodically updates all of its selected routes on all of its interfaces, including any recently retracted routes. When the neighbours receive this packet, it will update their routing table and retransmit the update packet. The update data is also kept for a while to wait for the arrival of the best route for each particular destination node in each node before updating its routing table and retransmitting the update packet.
3. **Trigger Updates** are small updates in-between the periodic updates. These are sent out whenever a node receives a DSDV packet that causes a change in its routing table.
4. **Data Packet** As described in the AODV packet section.

2.3.3 Hybrid Routing

Hybrid routing, sometimes referred to as balanced-hybrid routing, is a third classification of routing algorithm. It is a mix between reactive and proactive routing and takes the strengths of each type of the two classes of routing.

2.3.3.1 Zone Routing Protocol (ZRP)

An example of hybrid routing is ZRP, as published in Haas and Pearlman (1998) which maintains routing information for a local neighbourhood. This is likened to the proactive routing algorithm. Additionally, the ZRP uses reactive routing to acquire routing information for destinations outside the local neighbourhood. ZRP defines a zone around nodes, and within these zones proactive routing is used, outside of it, nodes use reactive routing.

ZRP refers to the locally proactive routing component as the Intra-zone Routing Protocol. The globally reactive routing component is named Inter-zone Routing Protocol described in Haas et al. (2001).

1. **Address Resolution Protocol** As described in the AODV packet section.
2. **IntrA-zone Routing Protocol** referred to as IARP. IARP maintains routing information for nodes that are within the routing zone of the node.
3. **IntEr-zone Routing Protocol** referred to as IERP. IERP is a family of reactive routing protocols that offer enhanced route discovery and route maintenance services based on local connectivity monitored by IARP
4. **Bordercast Resolution Protocol**, referred to as BRP. BRP uses a map of an extended routing zone to construct bordercast trees for the query packets.
5. **Data Packet** As described in the AODV packet section.

2.4 Mobility Models

Mobility models are important for analysing algorithms and protocols in wireless local area networks (WLANs) and self-organizing wireless ad hoc networks, as discussed by Bettstetter (2001). These models can be categorised in different ways, such as whether they are random or depend on temporal or spatial aspects.

For the generation of data for this research, examples of these models were used to inform the simulation about the nodal movement during the simulation. Several mobility models can synthesise the movement during the simulation of an ad-hoc mesh network. The specific models explored during this research are:

2.4.1 Column Mobility Model

This model simulates nodes that move together in a spatially dependent manner. A group of nodes travels in unison from one location to another, maintaining a

controlled proximity to one another. For instance, a line of nodes could move across a field, such as in a search line as defined in Camp et al. (2002).

2.4.2 Probabilistic Random Walk Mobility Model

This model is an example where the movement is random, where at each step, a calculation utilises a set of probabilities to determine the next direction that the node moves. It uses a probability matrix that defines the probabilities of a node moving forwards, backwards, or remaining still in both the x, y and optionally z directions, as defined in Camp et al. (2002).

2.4.3 SMOOTH Mobility Model

The SMOOTH model, or Smooth Random Mobility Model, accounts for temporal dependency in node movement. It is based on real-world human movement patterns and imitates mobility traces collected from various scenarios, as detailed by Munjal et al. (2011). The model incorporates several factors, including:

- Flights: The straight-line distance between consecutive waypoints. The distribution of mobile nodes follows a truncated power law (TPL). ICTs (Inter-contact Times): The amount of time between two nodes' successive contacts.
- Pause-times: The time a node pauses at a waypoint.
- Waypoint Preferences: Mobile nodes tend to visit popular waypoints.
- Movement Patterns: Nodes usually visit the closest waypoint first, although this is not always the case.
- Non-uniform Distribution: Mobile nodes are not evenly distributed throughout the network.
- Communities of Interest: Nodes with similar interests often form communities and tend to move within these groups.

2.5 Topology Features

When considering the research for the phase covered in Chapter 4 the focus is on the features that are likely to be the most representative to show variance within topologies. Consideration is given to the features that characterise the placement of

the nodes within the test plane and the relationships between nodes. Features are extracted by looking at the following aspects of a generic topology: inter-node distance, node spatial distribution, node density, shared node neighbours and node clustering coefficient.

2.5.1 Inter-Node Distance.

For this feature, the following is considered (i) the minimum distance, (ii) the maximum distance, (iii) the distance range '(max distance) - (min distance)' (iv) the mode, i.e. the most frequent value [1], (v) how many times the mode occurs, (vi) the mean value and (vii) the standard deviation.

2.5.2 Spatial Distribution.

By taking inspiration from the Quadrat Method set out in Greig-Smith (1952) which is used to test the Complete Spatial Randomness hypothesis, the topology area is partitioned into smaller squares. The number of nodes within these squares is calculated, and various features are extracted from these measurements. Figure 2.1 shows an example of topology area partitioning.

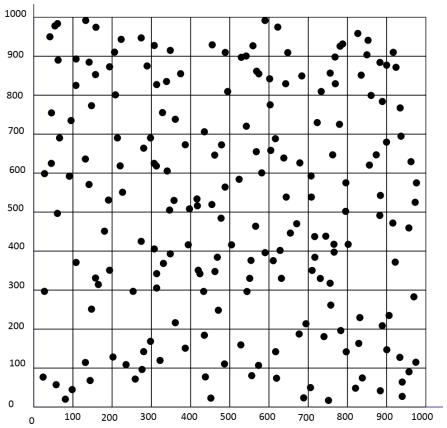

FIGURE 2.1: Partition of a Topology Area with 1000 Units Sides in 100 Smaller Squares, each with 100 Units Sides. This Partitioning is used to Compute Spatial Distribution Reatures.

The features extracted are (i) the minimum number of nodes in a square, (ii) the maximum number of nodes, (iii) the value range '(max number) - (min number)' (iv) the mode and (v) how many times the mode occurs.

[1] If there are more most-frequent values, by convention the smallest is picked.

2.5.3 Node Density.

For a given radius around a node, the number of other connected nodes is calculated. This is classed as the node's density. The features are calculated based on increasing the radius from the centre node as can be seen in figure 2.2.

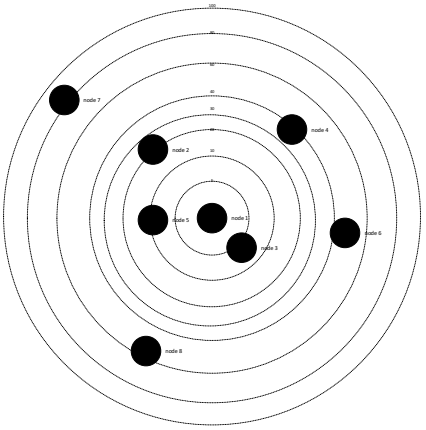

FIGURE 2.2: Node Density Feature.

2.5.4 Shared Neighbours Distribution.

For any given pair of nodes and and a set radius, the *shared neighbours* feature as defined in Nowak et al. (2014) are those nodes that are connected. Figure 2.3 shows an example where nodes 1 and 3 are shared neighbours of nodes 3 and 4.

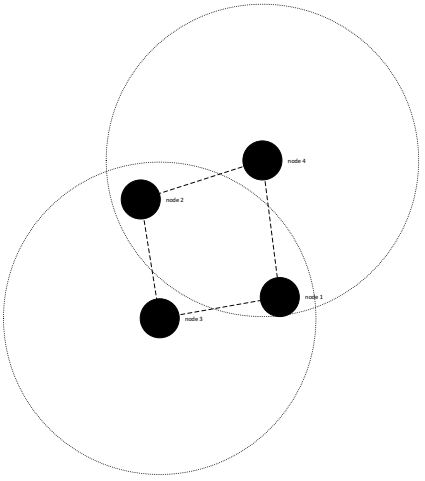

FIGURE 2.3: Nodes 1 and 2 are Shared Neighbours of Nodes 3 and 4. In this case, the Value of *Shared Neighbours Count* for Nodes 3 and 4 is 2.

These features are generated corresponding to the average shared neighbours count for a given radius, defined as follows

2.5.5 Clustering Coefficient.

The clustering coefficient as defined in Holland and Leinhardt (1971) of a node is a measure based on the number of node pairs that lie within a given radius and are neighbours of each other. An example is reported in Figure 2.4.

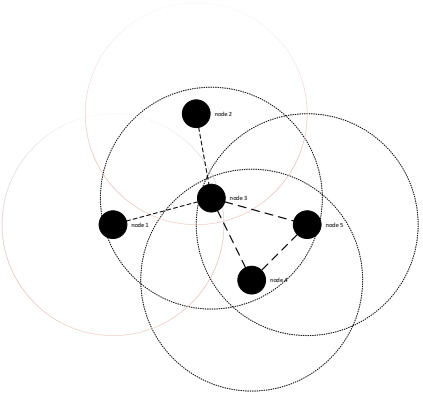

FIGURE 2.4: The Neighbours of Node 3 are Nodes 1, 2, 4 and 5. Among those Neighbours, there is a Pair of Nodes, i.e. nodes 4 and 5, which are Neighbours of Each Other, while Nodes 1 and 2 are not Neighbours of any other Node.

Features extracted corresponding to the average cluster coefficient for a given radius.

2.5.6 Sequential Feature Selection (SFS)

As described in Aha and Bankert (1996) and Rückstieß et al. (2011) is used to reduce the initial dimension feature space to the optimal number of features. The goal is to create a subset of features that explain the most variance in the dataset. This is done by either adding or removing one feature at a time and measuring the classifier performance until the subset of the desired features has been created. There are four different flavours of SFAs available:

- **Sequential Forward Selection** (SFS): SFS is a bottom-up search method that begins with an empty set of features and gradually adds features based on an evaluation function. At each iteration, the algorithm selects the best feature from the remaining ones that haven't yet been added to the feature set.
- **Sequential Backward Selection** (SBS): In contrast to SFS, SBS starts with a full set of features and removes them one by one, selecting which feature to eliminate at each step. The goal is to improve the criterion by iteratively removing less important features.
- **Sequential Forward Floating Selection** (SFFS): SFFS is an extension of SFS, incorporating an additional step that allows the removal of features once they've

been added to the set. This floating variant helps produce a more optimized set of features by adding or excluding features during the procedure.

- **Sequential Backward Floating Selection** (SBFS): Similar to SFFS, SBFS is the floating variant of SBS. It works by removing features but also allows for the reintroduction of previously excluded features to improve the feature set.

2.6 Machine Learning Algorithm

Various methodologies are employed in machine learning (ML). This section offers a basic overview of the ML techniques used in this research and does not attempt to cover the entire scope of the field. Generally, ML techniques are categorised into two main types: parametric and non-parametric algorithms, as outlined by Brownlee.

Parametric algorithms use a function to map input data to output data (predictions), incorporating weights and biases. Non-parametric algorithms, on the other hand, aim to find the best mapping function by using input data to organise the output into classes.

Throughout this research, several ML algorithms were used, evaluated, and compared based on their performance. Initially, the choice of algorithms wasn't clear, and relevant studies didn't provide a definitive recommendation. To avoid bias toward either parametric or non-parametric methods, random examples of both were chosen. These examples were assessed for their predictive accuracy to identify the best algorithm for the analysis.

Examples of parametric algorithms include Neural Networks and Probabilistic Clustering. For Neural Networks, a Convolutional Neural Network (CNN) was selected as a standard representative. For Probabilistic Clustering, Bernoulli Naive Bayes and Gaussian Naive Bayes were used. For non-parametric algorithms, the Support Vector Machine (SVM) was chosen initially. SVM was developed to enhance both training and prediction efficiency, using cluster centres obtained through k-means clustering (as described by Bang and Jhun (2014)). Random Forest was also selected because of its ability to handle datasets with a large number of predictor variables, as highlighted by Speiser et al. (2019)). Additionally, Deep Forest Neural Networks were chosen for their potential to combine the advantages of both Deep Neural Networks and Random Forests.

2.7 Unsupervised Learning

Unsupervised learning techniques are employed when no class information is provided. The algorithm is tasked with determining how many classes exist within the dataset. In this research, two algorithms were considered: the K-means distortion curve and affinity propagation. The K-means distortion curve was evaluated by calculating distortion for different numbers of clusters. Affinity propagation was iterated until convergence, and the number of clusters or cluster members was recorded.

2.7.1 K-Means Distortion Curve

The distortion curve, figure 2.5, shows the distortion against the perceived number of clusters. The change in the incline of the graph shows the optimal number of clusters, where for this plot the optimal number is 2.

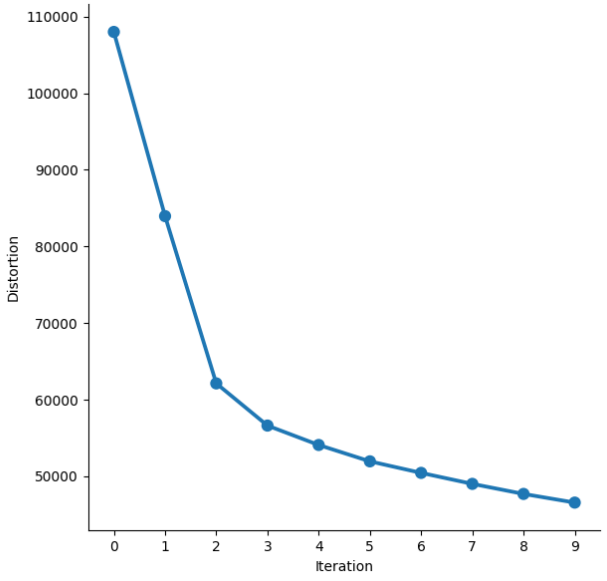

FIGURE 2.5: K-Mean Distortion Curve

2.7.2 Affinity Probability

When affinity probability clustering was used to give the optimal number of clusters, it gave a value of 148, which seemed excessive.

2.8 Semi-Supervised Clustering

The accuracy of the classifier is determined by using the topology generator ground truth. This method involves splitting the ground truth data into two parts: one used for training the classifier and the other for testing. The classification accuracy is then calculated based on these two sets. To ensure reliable results, cross-validation and a train/test split were employed during the analysis.

To minimise any potential bias that could arise from how the ground truth data is split, a well-established technique called k-fold cross-validation was applied, as outlined in Stone (1974). This method divides the data into k equally sized folds. The classifier is trained from scratch using all but one of these folds, which is held out for testing. This process is repeated k times, with each fold being used for validation once. In this analysis, we used 10-fold cross-validation, a common approach as shown in the work of Bengio BENGIOY and Grandvalet YVESGRANDVALET (2004). The final accuracy is calculated by averaging the accuracy values achieved in each of the k iterations, ensuring an unbiased evaluation of the classifier.

Additionally, the data was split into training and testing sets in a 25%/75% ratio. As shown in Tables 4.2 and 4.3 , the DBScan algorithm failed to classify the data properly, instead labelling all points as outliers.

2.9 Supervised Learning

Supervised learning techniques rely on labeled data, where each input is associated with a known output—for example, identifying the topology generator responsible for producing a given dataset. To evaluate model performance, the dataset is typically split into training and testing subsets. This separation ensures that the model is evaluated on previously unseen data, preventing overlap between training and testing phases and reducing the risk of overfitting.

2.9.1 Convolutional Neural Network Analysis

Convolutional Neural Networks (CNNs) are typically used for image classification, where they identify patterns within images to classify them. An input image is made up of three colour channels: red, green, and blue. This allows the CNN to extract colour-related features from the image for classification. The process involves applying filters, known as convolutional kernels, of a predefined pixel size over the image. These kernels are three-dimensional and examine the interaction between the

colour channels. The output of this process is a single value that represents a grid cell, or pixel, in the resulting feature map.

Although CNNs are most commonly used in image identification, this research repurposes the CNN technique to work with data generated from an Ad-Hoc Mesh Network.

2.9.2 Probabilistic Clustering

Probabilistic clustering, as described by Friedman et al. (1999), consists of algorithms designed to find the best fit for classifying data. One of these algorithms is Naïve Bayes, as defined by McCallum et al. (1998). Naïve Bayes is a probabilistic classifier that groups data into distinct classes. It calculates the mean and standard deviation of the selected features, and then determines the probability that a new data point belongs to any of the existing classes. The ”naïve” in its name comes from the assumption that the features are independent of each other, which simplifies the classification process but may not always hold true in practice.

2.9.2.1 Bernoulli

Bernoulli Naïve Bayes, defined in Larkey and Croft (1996) uses the Bernoulli distribution, named after Swiss mathematician Jacob Bernoulli as published in the paper Khoshnevisan (2002) it assumes that all the features are binary i.e. they have only two values.

The following are the parameters set out in the documentation, all defaults were used where applicable. **Parameters:**

- **alpha**float or array-like of shape (n_features,), default **1.0**

  Additive (Laplace/Lidstone) smoothing parameter (set alpha=0 and force_alpha=True, for no smoothing).

- **force_alpha** bool, default **True**

  If False and alpha is less than 1e-10, it will set alpha to 1e-10. If True, alpha will remain unchanged. This may cause numerical errors if alpha is too close to 0.

- **binarize** float or None, default **0.0**

  Threshold for binarising (mapping to booleans) of sample features. If None, input is presumed to already consist of binary vectors.

- **fit_prior** bool, default **True**

  Whether to learn class prior probabilities or not. If false, a uniform prior will be used.

- **class_prior** array-like of shape (n_classes), default **None**

  Prior probabilities of the classes. If specified, the priors are not adjusted according to the data.

2.9.2.2 Gaussian

Gaussian Naive Bayes Bishop and Nasrabadi (2006) assume the data is distributed according to a Gaussian distribution within the features, it is used in cases when all features are continuous. he following are the parameters set out in the documentation, all defaults were used where applicable.

Parameters:

- **priors** array-like of shape (n_classes,), default **None**

  Prior probabilities of the classes. If specified, the priors are not adjusted according to the data.

- **var_smoothing** float, default **1e-9**

  Portion of the largest variance of all features that is added to variances for calculation stability.

2.9.2.3 Multinomial

Multinomial Naive Bayes Devroye et al. (2013) assume the data is distributed according to a multinomial distributed data distribution within the features. For this experimentation, we used sklearn's implementation as documented at sci (d), with no change to the default values, unless stated..

The following are the parameters set out in the documentation, all defaults were used where applicable.

Parameters:

- **alpha** float or array-like of shape (n_features,), default **1.0**

  Additive (Laplace/Lidstone) smoothing parameter (set alpha=0 and force_alpha=True, for no smoothing).

- **force_alpha** bool, default **True**

  If False and alpha is less than 1e-10, it will set alpha to 1e-10. If True, alpha will remain unchanged. This may cause numerical errors if alpha is too close to 0.

- **fit_prior** bool, default **True**

  Whether to learn class prior probabilities or not. If false, a uniform prior will be used.

- **class_prior** array-like of shape (n_classes,), default **None**

  Prior probabilities of the classes. If specified, the priors are not adjusted according to the data.

2.9.3 Support Vector Machine

Support vector machine (SVM) with k-means clustering is a class of supervised learning that can be used to classify data and was used as the representative method for this experiment.

The SVM aims to create a hyperplane that separates the classes from data points from each class with the greatest margin between the individual classes. The technique can then be used to classify new data points based on which side of the hyperplane it lies. In regards to their use within this experiment, the mean value for each field was calculated per sample, giving one value per field in each sample.

A single run of the analysis was thought not to be representative of the results so a k-fold cross-validation technique was used with a k value of 5 with a train/test dataset split of 80/20%. This meant that the whole sample set was split into 80% which was used to train the SVM, and 20% used to validate the result. This was repeated 5 times with the accuracy noted and the average accuracy used as the final result.

The following are the parameters set out in the documentation, all defaults were used where applicable.

Parameters:

- **C , float** default **1.0**

  Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty. For an intuitive visualization of the effects of scaling the regularization parameter C

- **kernel - 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'** default **'rbf'**

Specifies the kernel type to be used in the algorithm. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape (n_samples, n_samples). For an intuitive visualization of different kernel types.

- **degree, int** default **3**

  Degree of the polynomial kernel function ('poly'). Must be non-negative. Ignored by all other kernels.

- **gamma - 'scale', 'auto' or float** default **'scale'**

  Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. if gamma='scale' (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma, if 'auto', uses 1 / n_features if float, must be nonnegative.

- **coef0 , float** default **0.0**

  Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'. shrinkingbool, default **True**

  Whether to use the shrinking heuristic. probabilitybool, default **False**

  Whether to enable probability estimates. This must be enabled prior to calling fit, will slow down that method as it internally uses 5-fold cross-validation, and predict_proba may be inconsistent with predict.

- **tolfloat** default **1e-3**

  Tolerance for stopping criterion.

- **cache_sizefloat** default **200**

  Specify the size of the kernel cache (in MB).

- **class_weightdict or 'balanced'** default **None**

  Set the parameter C of class i to class_weight[i]*C for SVC. If not given, all classes are supposed to have weight one. The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y)).

- **verbosebool** default **False**

  Enable verbose output. Note that this setting takes advantage of a per-process runtime setting in libsvm that, if enabled, may not work properly in a multithreaded context. max_iterint, default **-1**

  Hard limit on iterations within solver, or -1 for no limit.
  decision_function_shape'ovo', 'ovr', default **'ovr'**

  Whether to return a one-vs-rest ('ovr') decision function of shape (n_samples, n_classes) as all other classifiers, or the original one-vs-one ('ovo') decision

function of libsvm which has shape (n_samples, n_classes * (n_classes - 1) / 2). However, note that internally, one-vs-one ('ovo') is always used as a multi-class strategy to train models; an ovr matrix is only constructed from the ovo matrix. The parameter is ignored for binary classification.

- **break_tiesbool** default **False**

  If true, decision_function_shape='ovr', and number of classes ¿ 2, predict will break ties according to the confidence values of decision_function; otherwise the first class among the tied classes is returned. Please note that breaking ties comes at a relatively high computational cost compared to a simple predict.

- **random_stateint, RandomState instance or None** default **None**

  Controls the pseudo random number generation for shuffling the data for probability estimates. Ignored when probability is False. Pass an int for reproducible output across multiple function calls.

2.9.4 MeanShift

Parameters:

- **bandwidthfloat** default **None**

  Bandwidth used in the flat kernel.

  If not given, the bandwidth is estimated using sklearn.cluster.estimate_bandwidth; see the documentation for that function for hints on scalability (see also the Notes, below). seedsarray-like of shape (n_samples, n_features), default **None**

  Seeds used to initialize kernels. If not set, the seeds are calculated by clustering.get_bin_seeds with bandwidth as the grid size and default values for other parameters.

- **bin_seeding** bool, default **False**

  If true, initial kernel locations are not locations of all points, but rather the location of the discretized version of points, where points are binned onto a grid whose coarseness corresponds to the bandwidth. Setting this option to True will speed up the algorithm because fewer seeds will be initialized. The default value is False. Ignored if seeds argument is not None.

- **min_bin_freqint**, default **1**

  To speed up the algorithm, accept only those bins with at least min_bin_freq points as seeds.

- **cluster_all** bool, default **True**

  If true, then all points are clustered, even those orphans that are not within any kernel. Orphans are assigned to the nearest kernel. If false, then orphans are given cluster label -1.

- **n_jobsint** default **None**

  The number of jobs to use for the computation. The following tasks benefit from the parallelization:

  The search of nearest neighbours for bandwidth estimation and label assignments. See the details in the docstring of the NearestNeighbors class.

  Hill-climbing optimization for all seeds.

  None means 1 unless in a joblib.parallel_backend context. -1 means using all processors. See Glossary for more details.

- **max_iterint** default **300**

  Maximum number of iterations, per seed point before the clustering operation terminates (for that seed point), if has not converged yet.

2.9.5 Agglomerative Clustering

Parameters:

- **n_clusters** int or None, default **2**

  The number of clusters to find. It must be None if distance_threshold is not None. metricstr or callable, default **euclidean**

  Metric used to compute the linkage. Can be “euclidean”, “l1”, “l2”, “manhattan”, “cosine”, or “precomputed”. If linkage is “ward”, only “euclidean” is accepted. If “precomputed”, a distance matrix is needed as input for the fit method. If connectivity is None, linkage is “single” and affinity is not “precomputed” any valid pairwise distance metric can be assigned.

- **memory** str or object with the joblib. Memory interface, default **None**

  Used to cache the output of the computation of the tree. By default, no caching is done. If a string is given, it is the path to the caching directory.

- **connectivity** array-like, sparse matrix, or callable, default **None**

  Connectivity matrix. Defines for each sample the neighbouring samples following a given structure of the data. This can be a connectivity matrix itself or a callable that transforms the data into a connectivity matrix, such as derived

from kneighbors_graph. Default is None, i.e, the hierarchical clustering algorithm is unstructured.

For an example of connectivity matrix using kneighbors_graph

- **compute_full_tree** 'auto' or bool, default **auto**

  Stop early the construction of the tree at n_clusters. This is useful to decrease computation time if the number of clusters is not small compared to the number of samples. This option is useful only when specifying a connectivity matrix. Note also that when varying the number of clusters and using caching, it may be advantageous to compute the full tree. It must be True if distance_threshold is not None. By default compute_full_tree is "auto", which is equivalent to True when distance_threshold is not None or that n_clusters is inferior to the maximum between 100 or 0.02 * n_samples. Otherwise, "auto" is equivalent to False.

- **linkage** 'ward', 'complete', 'average', 'single', default='ward'

  Which linkage criterion to use. The linkage criterion determines which distance to use between sets of observation. The algorithm will merge the pairs of cluster that minimize this criterion.

  'ward' minimizes the variance of the clusters being merged.

  'average' uses the average of the distances of each observation of the two sets.

  'complete' or 'maximum' linkage uses the maximum distances between all observations of the two sets.

  'single' uses the minimum of the distances between all observations of the two sets.

- **distance_threshold** float, default **None**

  The linkage distance threshold at or above which clusters will not be merged. If not None, n_clusters must be None and compute_full_tree must be True.

- **compute_distances** bool, default **False**

  Computes distances between clusters even if distance_threshold is not used. This can be used to make dendrogram visualization, but introduces a computational and memory overhead.

2.9.6 DBScan

Parameters:

- **epsfloat** default **0.5**

The maximum distance between two samples for one to be considered as in the neighborhood of the other. This is not a maximum bound on the distances of points within a cluster. This is the most important DBSCAN parameter to choose appropriately for your data set and distance function.

- **min_samples** int, default **5**

  The number of samples (or total weight) in a neighborhood for a point to be considered as a core point. This includes the point itself. If min_samples is set to a higher value, DBSCAN will find denser clusters, whereas if it is set to a lower value, the found clusters will be more sparse.

- **metric** str, or callable, default **euclidean**

  The metric to use when calculating distance between instances in a feature array. If metric is a string or callable, it must be one of the options allowed by sklearn.metrics.pairwise_distances for its metric parameter. If metric is "precomputed", X is assumed to be a distance matrix and must be square. X may be a sparse graph, in which case only "nonzero" elements may be considered neighbors for DBSCAN.

- **metric_params** dict, default **None**

  Additional keyword arguments for the metric function.

- **algorithm** 'auto', 'ball_tree', 'kd_tree', 'brute', default **auto**

  The algorithm to be used by the NearestNeighbors module to compute pointwise distances and find nearest neighbors. See NearestNeighbors module documentation for details.

- **leaf_size** int, default **30**

  Leaf size passed to BallTree or cKDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

- **pfloat** default **None**

  The power of the Minkowski metric to be used to calculate distance between points. If None, then p=2 (equivalent to the Euclidean distance).

- **n_jobs** int, default **None**

  The number of parallel jobs to run. None means 1 unless in a joblib.parallel_backend context. -1 means using all processors.

2.9.7 Spectral Clustering

Parameters:

- **affinity** array-like, sparse matrix of shape (n_samples, n_samples)

  The affinity matrix describing the relationship of the samples to embed. Must be symmetric.

- **n_clusters** int, default **None**

  Number of clusters to extract.

- **n_components** int, default **n_clusters**

  Number of eigenvectors to use for the spectral embedding.

- **eigen_solver** None, 'arpack', 'lobpcg', or 'amg'

  The eigenvalue decomposition method. If None then 'arpack' is used. See [4] for more details regarding 'lobpcg'. Eigensolver 'amg' runs 'lobpcg' with optional Algebraic MultiGrid preconditioning and requires pyamg to be installed.

- **random_state** int, RandomState instance, default **None**

  A pseudo random number generator used for the initialization of the lobpcg eigenvectors decomposition when eigen_solver == 'amg', and for the K-Means initialization. Use an int to make the results deterministic across calls.

- **n_init** int, default **10**

  Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia. Only used if assign_labels='kmeans'.

- **eigen_tol** float, default **auto**

  Stopping criterion for eigendecomposition of the Laplacian matrix. If eigen_tol="auto" then the passed tolerance will depend on the eigen_solver:

  If eigen_solver="arpack", then eigen_tol=0.0;

  If eigen_solver="lobpcg" or eigen_solver="amg", then eigen_tol=None which configures the underlying lobpcg solver to automatically resolve the value according to their heuristics. See, scipy.sparse.linalg.lobpcg for details.

- **assign_labels** 'kmeans', 'discretize', 'cluster_qr', default **kmeans**

  The strategy to use to assign labels in the embedding space. There are three ways to assign labels after the Laplacian embedding. k-means can be applied and is a popular choice. But it can also be sensitive to initialization. Discretization is another approach which is less sensitive to random initialization [3]. The cluster_qr method [5] directly extracts clusters from eigenvectors in spectral clustering. In contrast to k-means and discretization, cluster_qr has no tuning parameters and is not an iterative method, yet may outperform k-means and discretization in terms of both quality and speed. For a

detailed comparison of clustering strategies, refer to the following example: Segmenting the picture of greek coins in regions.

- **verbosebool** default **False**

  Verbosity mode.

2.9.8 Birch Clustering

Parameters:

- **threshold** float, default **0.5**

  The radius of the subcluster obtained by merging a new sample and the closest subcluster should be lesser than the threshold. Otherwise a new subcluster is started. Setting this value to be very low promotes splitting and vice-versa.

- **branching_factor** int, default **50**

  Maximum number of CF subclusters in each node. If a new samples enters such that the number of subclusters exceed the branching_factor then that node is split into two nodes with the subclusters redistributed in each. The parent subcluster of that node is removed and two new subclusters are added as parents of the 2 split nodes.

- **n_clusters** int, instance of sklearn.cluster model or None, default **3**

  Number of clusters after the final clustering step, which treats the subclusters from the leaves as new samples.

  None : the final clustering step is not performed and the subclusters are returned as they are.

  sklearn.cluster Estimator : If a model is provided, the model is fit treating the subclusters as new samples and the initial data is mapped to the label of the closest subcluster.

  int : the model fit is AgglomerativeClustering with n_clusters set to be equal to the int.

- **compute_labels** bool, default **True**

  Whether or not to compute labels for each fit. copybool, default **True**

  Whether or not to make a copy of the given data. If set to False, the initial data will be overwritten.

2.9.9 Decision Tree

Decision Tree published in Quinlan (1986) is a widely used machine learning algorithm that performs both classification and regression tasks. It models decisions and their possible consequences in a tree-like structure, where each internal node represents a decision or a test on a particular feature, each branch represents the outcome of that decision, and each leaf node represents the final output or class label.

The process of building a Decision Tree involves recursively splitting the data into subsets based on feature values. The aim is to create branches that result in the purest possible divisions, where each subset contains data points that belong to the same class or have similar output values. To determine the best feature to split the data at each node, algorithms like Gini impurity or Information Gain are often used, which measure how well the data is separated by a particular feature.

Once the tree is built, predictions are made by following the path from the root node to a leaf node, based on the values of the input features. The model outputs the class or predicted value associated with the leaf node that the input data reaches.

Decision Trees are particularly known for their simplicity and interpretability. Since they closely mimic human decision-making processes, they are easy to visualise and understand. However, they are prone to over-fitting, especially when the tree is too deep, meaning it can fit the training data very well but fail to generalise to new, unseen data. Techniques like pruning (removing parts of the tree that don't improve the model) and ensemble methods like Random Forests can be used to mitigate this issue.

In summary, Decision Trees are powerful tools for classification and regression tasks, offering both transparency and versatility in modelling complex decision-making processes.

The following are the parameters set out in the documentation, all defaults were used where applicable.

Parameters:

- **criterion: gini, entropy, log_loss** default **gini** The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "log_loss" and "entropy" both for the Shannon information gain, see Mathematical formulation.
- **splitter: best, random**, default **best**

  The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

- **max_depth, int** default **None**

  The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

- **min_samples_split: int or float** default **2**

  The minimum number of samples required to split an internal node

  - If int, then consider min_samples_split as the minimum number.
  - If float, then min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.

- **min_samples_leaf: int or float** default **1**

  The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

  - If int, then consider min_samples_leaf as the minimum number.
  - If float, then min_samples_leaf is a fraction and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.

- **min_weight_fraction_leaf** float, default **0.0**

  The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.

- **max_features** int, float or “sqrt”, “log2”, default **None**

  The number of features to consider when looking for the best split:

  - If int, then consider max_features features at each split.
  - If float, then max_features is a fraction and max(1, int(max_features * n_features_in_)) features are considered at each split.
  - If “sqrt”, then max_features=sqrt(n_features).
  - If “log2”, then max_features=log2(n_features).
  - If None, then max_features=n_features.

- **random_state: int, RandomState instance or None** default **None**

  Controls the randomness of the estimator. The features are always randomly permuted at each split, even if splitter is set to ”best”. When max_features ¡ n_features, the algorithm will select max_features at random at each split before finding the best split among them. But the best found split may vary across different runs, even if max_features=n_features. That is the case, if the

improvement of the criterion is identical for several splits and one split has to be selected at random. To obtain a deterministic behaviour during fitting, random_state has to be fixed to an integer.

- **max_leaf_nodes** int, default **None**

  Grow a tree with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.

- **min_impurity_decrease** float, default **0.0**

  A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

- **class_weight** dict, list of dict or "balanced", default **None**

  Weights associated with classes in the form class_label: weight. If None, all classes are supposed to have weight one. For multi-output problems, a list of dicts can be provided in the same order as the columns of y.

  The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y))

  For multi-output, the weights of each column of y will be multiplied.

- **ccp_alpha** non-negative float, default **0.0**

  Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than ccp_alpha will be chosen. By default, no pruning is performed.

- **monotonic_cst: array-like of int of shape (n_features)** default **None**

  Indicates the monotonicity constraint to enforce on each feature.

  - **1**: monotonic increase
  - **0**: no constraint
  - **-1**: monotonic decrease

  If monotonic_cst is None, no constraints are applied.

  Monotonicity constraints are not supported for:

  - multiclass classifications (i.e. when n_classes ¿ 2),
  - multioutput classifications (i.e. when n_outputs_ ¿ 1),
  - classifications trained on data with missing values.

2.9.10 Random Forest

A Random Forest published in Breiman (2001) is an ensemble learning method used for both classification and regression tasks. It builds multiple Decision Trees during the training process and merges their results to improve the model's accuracy and robustness. The key idea behind Random Forest is to create a forest of trees, where each tree is trained on a random subset of the training data, and the final prediction is made by averaging the results of all individual trees in the case of regression or by using majority voting in the case of classification.

The process of building a Random Forest involves two main components:

- **Bootstrapping** (Bagging): For each tree in the forest, a random subset of the training data is selected with replacement (i.e., some data points may be repeated in the subset). This subset is used to train the tree, allowing each tree to be trained on slightly different data.
- **Feature Randomness** : During the construction of each tree, only a random subset of features is considered for splitting at each node, rather than evaluating all features. This helps create diversity among the trees and prevents over-fitting by reducing the model's reliance on any single feature.

Once all the trees are trained, the Random Forest makes its prediction by aggregating the predictions of all the trees. For classification tasks, it uses majority voting, where the class that most trees predict becomes the final output. For regression tasks, it uses the average of all tree predictions.

The strength of Random Forest lies in its ability to handle large datasets with high dimensionality, as well as its robustness to over-fitting, especially when compared to a single Decision Tree. Because each tree is trained on a random subset of the data and features, the model can generalise better and perform well even on unseen data. Furthermore, Random Forests provide a measure of feature importance, allowing users to understand which features are most influential in making predictions.

Despite its many strengths, Random Forest can be computationally expensive and may not perform well in real-time applications due to the complexity of combining multiple trees. However, it remains one of the most widely used and powerful algorithms in machine learning.

In conclusion, Random Forest is a versatile and highly effective ensemble method, leveraging the power of multiple Decision Trees to enhance predictive performance and reduce the risk of over-fitting, making it an excellent choice for many machine learning tasks.

The following are the parameters set out in the documentation, all defaults were used where applicable.

Parameters:

- **n_estimatorsint** default **100** - The number of trees in the forest.
- **criterion "gini", "entropy", "log_loss"** default **gini**

  The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "log_loss" and "entropy" both for the Shannon information gain, see Mathematical formulation. Note: This parameter is tree-specific.
- **max_depthint** default **None**

  The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- **nmin_samples_splitint or float** default **2**

  The minimum number of samples required to split an internal node: If int, then consider min_samples_split as the minimum number. If float, then min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.
- **min_samples_leafint or float** default**1**

  The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression. If int, then consider min_samples_leaf as the minimum number. If float, then min_samples_leaf is a fraction and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.
- **min_weight_fraction_leaf float** default **0.0**

  The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.
- **max_features"sqrt", "log2", None, int or float** default **"sqrt"**

  The number of features to consider when looking for the best split: If int, then consider max_features features at each split. If float, then max_features is a fraction and max(1, int(max_features * n_features_in_)) features are considered at each split. If "sqrt", then max_features=sqrt(n_features). If "log2", then max_features=log2(n_features). If None, then max_features=n_features.

- **max_leaf_nodesint** default **None**

  Grow trees with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes. min_impurity_decreasefloat, default **0.0**

  A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

  The weighted impurity decrease equation is the following:

  N_t / N * (impurity - N_t_R / N_t * right_impurity - N_t_L / N_t * left_impurity)

  where N is the total number of samples, N_t is the number of samples at the current node, N_t_L is the number of samples in the left child, and N_t_R is the number of samples in the right child.

  N, N_t, N_t_R and N_t_L all refer to the weighted sum, if sample_weight is passed.

- **bootstrapbool** default **True**

  Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.

- **oob_scorebool or callable** default **False**

  Whether to use outofbag samples to estimate the generalization score. By default, accuracy_score is used. Provide a callable with signature metric(y_true, y_pred) to use a custom metric. Only available if bootstrap=True.

- **n_jobsint** default **None**

  The number of jobs to run in parallel. fit, predict, decision_path and apply are all parallelized over the trees. None means 1 unless in a joblib.parallel_backend context. -1 means using all processors. See Glossary for more details.

- **random_stateint, RandomState instance or None** default **None**

  Controls both the randomness of the bootstrapping of the samples used when building trees (if bootstrap=True) and the sampling of the features to consider when looking for the best split at each node (if max_features ¡ n_features).

- **verboseint** default **0**

  Controls the verbosity when fitting and predicting.

- **warm_startbool** default **False**

  When set to True, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new forest. See Glossary and Fitting additional trees for details.

- **class_weigh - "balanced", "balanced_subsample" , dict or list of dicts**default **None**

  Weights associated with classes in the form class_label: weight. If not given, all classes are supposed to have weight one. For multi-output problems, a list of dicts can be provided in the same order as the columns of y.

  The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y))

  The "balanced_subsample" mode is the same as "balanced" except that weights are computed based on the bootstrap sample for every tree grown.

  For multi-output, the weights of each column of y will be multiplied.

- **ccp_alphanon-negative float** default **0.0**

  Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than ccp_alpha will be chosen. By default, no pruning is performed. See Minimal Cost-Complexity Pruning for details.

- **max_samplesint or float** default **None**

  If bootstrap is True, the number of samples to draw from X to train each base estimator. If None (default), then draw X.shape[0] samples. If int, then draw max_samples samples. If float, then draw max(round(n_samples * max_samples), 1) samples. Thus, max_samples should be in the interval (0.0, 1.0].

- **monotonic_cstarray-like of int of shape (n_features)** default **None**

2.9.11 Deep Forest Neural Network

Deep Forest, as described in Zhou and Feng (2017) is an alternative to the traditional deep neural network. It uses a mixture of random forest as a probability distribution over classes and conventional classification. Deep neural networks use the last layer to classify the different classes while having 'hidden layers' trained via back-propagation. Deep forest uses a layer-by-layer processing approach; each layer is an ensemble of decision tree forests that receive feature information processed by its preceding layer. Each layer is a mix of 'completely random forests' and 'random forests', with each forest producing an estimate of class distribution and then averaging across all trees in the same forest. The ultimate layer's predictions are averaged, and the overall predictions are used in evaluation.

2.10 Centrality

The use of centrality metrics is used in Chapter 6 to identify influence nodes set out in Tulu et al. (2018b), the following measures are used during the analysis for this chapter.

2.10.1 Eigenvector Centrality

'Eigenvector centrality' is the first centrality to take into account. This is a measure of a node's influence within a network. It assigns relative scores to each node in the network based on the hypothesis that connections to high-scoring nodes add more to a node's score than do equal connections to low-scoring nodes. In a study that was reported in the paper Tulu et al. (2018a), the nodes that disseminated information through the network the slowest were identified by using eigenvector centrality, which consistently performed the worst among the centrality measures.

2.10.2 Degree Centrality

The next centrality is that of 'degree centrality' published in Freeman (1978). This is the simplest to understand and is defined as the number of links a node has. Assuming that there is a maximum of one edge per pair of nodes, its value ranges from 0 to the 'number of nodes in the network' (if the network allows a node to have an edge back to itself) or 'number of nodes in the network' -1 if not.

The input parameters required where as follows:

- **G**: graph - A NetworkX graph.

In the research, it is assumed that if this value is low (zero) for a particular node, then the node has minimal or no links and so is isolated from the network. Whereas if the value is high (one), the node has the maximum number of links.

2.10.3 Closeness Centrality

The next centrality is that of 'closeness centrality' from the same paper Freeman (1978). It calculates the distances from a single point to all other points in the network. A low value for this centrality indicates the more central position the node is in the network, and a high value indicates they are less central.

The specific parameters for this centrality are:

- **G**: graph - A NetworkX graph.
- **u**: node, optional - Return only the value for node u
- **distance** - edge attribute key, optional default **None**

  Use the specified edge attribute as the edge distance in shortest path calculations. If None (the default) all edges have a distance of 1. Absent edge attributes are assigned a distance of 1. Note that no check is performed to ensure that edges have the provided attribute.

- **wf_improved: bool** optional default **True**

  If True, scale by the fraction of nodes reachable. This gives the Wasserman and Faust improved formula. For single component graphs it is the same as the original formula.

Where applicable all defaults are used.

2.10.4 Betweenness Centrality

Betweenness centrality, described in the same paper Freeman (1978) is an indication of the number of times the node is involved in the shortest path between other nodes in the network. A high betweenness centrality value indicates that the node is involved in the highest number of shortest paths.

The input parameters required where as follows::

- **G**: graph - A NetworkX graph.
- **k: int** - optional default **None**

  If k is not None use k node samples to estimate betweenness. The value of k ¡= n where n is the number of nodes in the graph. Higher values give better approximation.

- **normalized: bool** - Set to **True**

  If True the betweenness values are normalized by 2/((n-1)(n-2)) for graphs, and 1/((n-1)(n-2)) for directed graphs where n is the number of nodes in G.

- **weight: None or string** - optional default **None**

  If None, all edge weights are considered equal. Otherwise holds the name of the edge attribute used as weight. Weights are used to calculate weighted shortest paths, so they are interpreted as distances.

- **endpoints: bool** - optional

  If True include the endpoints in the shortest path counts.

- **seed: integer** - random_state, or default **None**

  Indicator of random number generation state. See Randomness. Note that this is only used if k is not None.

2.10.5 Bridging Coefficient

The bridging coefficient described in Hwang et al. (2006) is a metric used to measure the strength of connections between different groups or clusters within a network. In the context of network analysis, it identifies how well a node acts as a bridge between two distinct clusters or communities. The concept is derived from the idea that certain nodes in a network have the ability to connect otherwise separate groups, facilitating the flow of information or resources between them.

In a graph or network, nodes are often grouped together based on their proximity or connection patterns, forming clusters. The Bridging Coefficient quantifies how much a node is involved in connecting different clusters, acting as an intermediary between them. This is particularly useful in the study of social networks, biological networks, and communication systems, where the ability of a node to connect distinct communities can play a key role in the dynamics of the network.

To calculate the Bridging Coefficient for a node, the following steps are typically followed:

- **Identification of Clusters**: First, the network is divided into clusters or communities. This can be done using various clustering algorithms or community detection techniques.
- Node's Role as a Bridge: Next, for each node, the number of distinct clusters it connects is determined. If a node is only connected to other nodes within the same cluster, it is not considered a bridge. However, if the node connects nodes from two or more different clusters, it plays the role of a bridge.
- **Coefficient Calculation**: The Bridging Coefficient is then computed based on how many clusters a node connects and the relative strength of these connections. A higher Bridging Coefficient indicates a more significant role in connecting diverse parts of the network.

In real-world networks, nodes with a high Bridging Coefficient are considered critical for communication, information exchange, and resilience. For example, in social networks, individuals with high bridging scores may act as central figures that

connect different social groups or communities. In biological networks, such nodes may represent key proteins that facilitate interactions between different biochemical pathways.

However, the concept of the Bridging Coefficient also has its limitations. It does not account for the flow of information or the intensity of interactions between the connected clusters, which may be critical in some applications. Additionally, the coefficient depends on the clustering algorithm used to divide the network, which may vary in effectiveness depending on the network's structure.

Overall, the Bridging Coefficient serves as a useful tool for identifying key nodes that facilitate the connectivity between different parts of a network, enabling better understanding of network structure and function.

2.10.6 Bridging Nodes

Bridging Nodes described in Liu et al. (2019) refer to nodes within a network that connect two or more distinct clusters or communities. These nodes are critical in maintaining the overall connectivity of the network by serving as intermediaries between otherwise isolated groups. In network analysis, bridging nodes are often viewed as vital for the flow of information, resources, or influence across different parts of a system, as they facilitate communication between otherwise disconnected clusters.

In graph theory and network science, a community or cluster refers to a group of nodes that are more densely connected to each other than to the rest of the network. Bridging nodes, therefore, play an essential role in linking separate communities, making them key to understanding the structure and behaviour of complex networks, such as social networks, communication networks, or biological systems.

The concept of a bridging node is tied closely to the idea of modularity and community detection, as it highlights the nodes responsible for bridging gaps between different network sections. These nodes may exhibit different characteristics depending on the network's context, but they often have higher degrees of connectivity than other nodes, allowing them to connect multiple groups.

Here is an overview of how bridging nodes are identified and why they are important:

- **Identification of Communities**: The first step in identifying bridging nodes is to divide the network into distinct clusters or communities, typically using community detection algorithms such as modularity optimization or spectral clustering.

- **Connectivity Analysis**: Once the network is divided into communities, the nodes that appear in more than one community are considered potential bridging nodes. These nodes typically have edges (connections) that link different clusters. The more clusters a node connects, the more influential it becomes as a bridging node.
- **Node's Role in the Network**: Bridging nodes often serve as important conduits for information flow within the network. They can connect isolated groups and facilitate communication between them, allowing the network to function as a whole rather than as fragmented sections. In social networks, for example, bridging nodes may represent individuals who connect disparate social groups, while in communication networks, they could represent routers or switches that link different sub-networks.
- **Significance of Bridging Nodes**: The presence of bridging nodes can significantly impact the structure and resilience of the network. They are often crucial for ensuring the network's connectivity, robustness, and functionality. Their removal can result in fragmentation of the network, leading to disconnection between otherwise connected communities.
- **Applications and Implications**: In social networks, bridging nodes might correspond to individuals who connect different social groups or subcultures, potentially enabling the spread of information or ideas. In biological networks, bridging nodes could represent proteins or molecules that link different biological pathways or systems, making them critical for cellular function. In communication systems, bridging nodes might be routers or switches that link various network segments and enable the efficient transfer of data.

Although bridging nodes are essential for maintaining network connectivity, they can also be a vulnerability point. If a bridging node fails or is removed, the network may become fragmented into separate, isolated communities. This makes bridging nodes a critical focus in studies of network resilience and fault tolerance.

Overall, bridging nodes are fundamental components of networks that maintain inter-community connections and ensure the cohesion and functionality of the overall system. Their role is especially important in large, complex networks where various sub-networks or groups need to interact and collaborate. Identifying these nodes helps in understanding the flow and dynamics within networks, and can provide insights into improving the efficiency, resilience, and performance of the network as a whole.

2.10.7 Bridging Centrality

Bridging centrality is a combination of the bridging coefficient and betweenness centrality. It is a metric to quantitatively measure the extent of the bridging capability of all nodes or edges in the network described in Hwang et al. (2008)

2.10.8 Data Collection

The methodology for data generation remained consistent throughout each phase of the research. Network traffic data was gathered using the ns-3 discrete-event network simulator (Riley and Henderson (2010)), version 3 release 29. This simulator allows the creation of ad-hoc mesh networks where nodes exchange messages using a predefined routing algorithm. During each simulation, the generated traffic is recorded as PCAP files at each participating node.

For each run, a set number of nodes are randomly placed within a test area using one of three node placement algorithms: NPART, BRITE, or GT-ITM. Each algorithm is used in one-third of the simulation runs. Throughout each run, packets are sent continuously from randomly selected source nodes to randomly chosen destination nodes. The random selections are determined by the pseudo-random number generator built into the simulation software.

The data collected in these simulations consists of data packets traversing a mesh network, which is formed by spatially distributed nodes. These networks are referred to as multi-hop ad-hoc mesh networks because the route from the source to the destination involves several intermediary hops, as explained by Pešović et al. (2010). The term ”ad-hoc” highlights that the network’s infrastructure is not predefined.

2.10.9 Data Extraction

Network traffic is captured by every node within the simulation and then aggregated into a single comma-separated values (CSV) file. The captured traffic includes all packets exchanged between the intermediate nodes along the route from source to destination. Each captured packet contains both layer 3 (IP) routing information and layer 2 (MAC) routing information. The layer 3 information includes the source and destination IP addresses, while the layer 2 information contains the MAC address of the next hop along the route.

2.10.10 Link/Edge Prediction

Link or Edge Prediction is the process of forecasting the likelihood that an edge will occur between two named nodes at a specific time in the future. This probability is calculated using a model trained on previous occurrences of edges. Examples of link prediction applications include predicting friendships in social networks (as discussed in Berahmand et al. (2022)), co-authorship in citation networks (Makarov et al. (2019)), and drug discovery (Wu et al. (2018)).

The most common method for predicting links is topology-based. This approach assumes that the frequency of links during the training period determines their temporal effect. Because there is no time element considered, this method can be viewed as a static model. It works by analysing network structure and predicting new links by matching nodes with similar patterns.

During the research on Node Importance in Temporal Dynamic Networks, outlined in 6, the first method employed in the analysis phase uses this static model, assuming that all edges between two nodes are equally important. To do this, each edge is assigned a weight based on its frequency during the simulation. The technique used was a Graph Convolutional Network (GCN) with the 'GCNLayer' described by Kipf and Welling (2016), with no weight applied to the links.

The second method follows the same approach with the 'GCNLayer', but this time, link weights are determined by the occurrence of different packet types. To calculate the weight of a link, we count the number of times each packet type occurs during a specific time slice. This value is then multiplied by the frequency of the packet's occurrence, as outlined in Table 2.1. For consistency across experiments, the same packet values are used throughout. The time slice consists of 200 packets, which provided sufficient data for centrality measures, ensuring representative granularity.

Our use of 'GCNLayer' we followed the parameters set out in the documentation, all defaults were used where applicable.

- **Parameters**:
  - **in_feats (int)** – Input feature size; i.e, the number of dimensions of h(l)j.
  - **out_feats (int)** – Output feature size; i.e., the number of dimensions of h(l+1)i.
  - **norm (str, optional)** – How to apply the normalizer. Can be one of the following values:
    - right, to divide the aggregated messages by each node's in-degrees, which is equivalent to averaging the received messages.
    - none, where no normalization is applied.

    * both (default), where the messages are scaled with 1/cji above, equivalent to symmetric normalisation.
    * left, to divide the messages sent out from each node by its out-degrees, equivalent to random walk normalisation.
- **weight (bool, optional)** – If True, apply a linear layer. Otherwise, aggregating the messages without a weight matrix.
- **bias (bool, optional)** – If True, adds a learnable bias to the output. Default: **True**.
- **activation (callable activation function/layer or None, optional)** – If not None, applies an activation function to the updated node features. Default: None.
- **allow_zero_in_degree (bool, optional)** – If there are 0-in-degree nodes in the graph, output for those nodes will be invalid since no message will be passed to those nodes. This is harmful for some applications causing silent performance regression. This module will raise a DGLError if it detects 0-in-degree nodes in input graph. By setting True, it will suppress the check and let the users handle it by themselves. Default: **False**.

TABLE 2.1: Assigned Weights

Packet Type	**Weight**
Route Request	1
Route Reply	2
Route Error	3
Route Reply Acknowledgement	4
Time-to-live exceeded	5
Data	6

The next method applied to link prediction is the GraphSAGE algorithm, as introduced by Hamilton et al. (2017). GraphSAGE uses a stochastic generalization of graph convolutions, specifically designed for link prediction tasks. For this approach, the same packet weighting method is applied to generate link weights.

When using 'GraphSAGE', all the defaults where used as set out in the following documentation, where applicable.

- **Parameters**:
    - **in_feats (int)** – Input feature size; i.e, the number of dimensions of h(l)j.
    - **out_feats (int)** – Output feature size; i.e., the number of dimensions of h(l+1)i.

- **norm (str, optional)** – How to apply the normalizer. Can be one of the following values:
  - right, to divide the aggregated messages by each node's in-degrees, which is equivalent to averaging the received messages.
  - none, where no normalization is applied.
  - both (default), where the messages are scaled with 1/cji above, equivalent to symmetric normalisation.
  - left, to divide the messages sent out from each node by its out-degrees, equivalent to random walk normalisation.
- **weight (bool, optional)** – If True, apply a linear layer. Otherwise, aggregating the messages without a weight matrix.
- **bias (bool, optional)** – If True, adds a learnable bias to the output. Default: **True**.
- **activation (callable activation function/layer or None, optional)** – If not None, applies an activation function to the updated node features. Default: None.
- **allow_zero_in_degree (bool, optional)** – If there are 0-in-degree nodes in the graph, output for those nodes will be invalid since no message will be passed to those nodes. This is harmful for some applications causing silent performance regression. This module will raise a DGLError if it detects 0-in-degree nodes in input graph. By setting True, it will suppress the check and let the users handle it by themselves. Default: **False**.

The final method used in the analysis incorporates the temporal aspect of the simulation, as outlined in the paper by Rossi et al. (2020). This approach utilises continuous-time dynamic graphs represented as a sequence of events to predict links. In this method, dynamic graphs update in real time by adding or removing link information, so each link only influences the graph for the duration of its existence. As with previous methods, the same weighting approach is used to generate edge weights.

The simulation runs for 1800 seconds, providing a baseline for the creation and destruction of links as the nodes move within the simulation area. To evaluate the accuracy of link prediction, the simulation data is split into three sections: training, validation, and testing, in a 70/15/15 ratio. The first 70% of the data is used for training the prediction methods, the next 15% is for validating the trained model, and the final 15% is used for testing the accuracy of the model. This split ensures that the data used for training is not reused for validation or testing, helping to prevent over-fitting.

2.11 Similarity Measures

The similarity measure are divided into Euclidean, Manhattan and Cosine, all of which are programmatically calculated in the sections using following equations.

2.11.1 Euclidean Distance

Euclidean distance is a widely used metric for calculating the distance between two points with numeric attributes. It is computed by taking the square root of the sum of the squared differences between the corresponding elements of two vectors. This method is particularly effective when working with low-dimensional data, where the magnitude of the vectors plays an important role in measurement. A Euclidean distance of zero means the two points are identical, whereas a large distance indicates little similarity between them.

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

2.11.2 Manhattan Distance

Also referred to as city block distance or taxicab geometry, Manhattan distance measures the distance between two points by summing the absolute differences of their Cartesian coordinates. This measure is more suitable for high-dimensional data, as the lower the value of the Manhattan distance, the closer the two points are to each other.

$$d(p,q) = \sum |(q_i - p_i)|$$

2.11.3 Cosine Similarity

Cosine similarity calculates the normalised dot product between two vectors. By measuring the cosine of the angle between them, this metric determines how similar two objects are. A cosine similarity value close to 1 indicates that the objects are highly similar, while a value near 0 signifies that they are less similar.

The Cosine Similarity is calculated using the SciPy library - Sci - scipy.spatial.distance

Parameters:

- **u: array_like of floats** - Input array.
- **v: array_like of floats** - Input array.

- **w: array_like of floats** - optional

The weights for each value in u and v. Default **None**, which gives each value a weight of 1.0

Chapter 3

Related Work

As outlined in chapter 1, this research is divided into three phases: the first phase examines topology generators (TGs), the second focuses on routing algorithms, and the third investigates influential nodes within complex networks.

The first phase concentrates on TGs, which are essential for simulating network topologies. These generators primarily differ in the methods they use to place nodes within a network, as explained by Sanni et al. (2013), and in how the nodes are represented, as discussed by Heckmann et al. (2003). Node placement strategies generally fall into two categories: predefined models or real-world measurements. Predefined models may follow a specific probability distribution, such as the Waxman model proposed by Waxman (1988), or maintain certain node-to-node distances, as seen in chain or grid node placement models. On the other hand, real-world measurements rely on actual data from existing network topologies to inform node placement.

TGs can be further categorised into two main types:

- **Placement model**: In this model, nodes are randomly positioned according to probability distributions, like the BRITE model introduced by Medina et al. (2001a).
- **Measurement model**: This approach positions nodes based on real-world data, such as power transmission data, to reflect actual network conditions. One example of this is the NPART model, proposed by Milic and Malek (2009).

Network topologies can also be classified into AS-level and router-level topologies, as noted by Heckmann et al. (2003). In AS-level topologies, each node represents an autonomous system (AS), and edges denote peering agreements between them. In router-level topologies, nodes represent routers, and edges represent direct connections between them.

Topologies can be categorised into **AS-level** and **router-level** types, as discussed by Heckmann et al. (2003). In AS-level topologies, each node represents an autonomous system (AS), with edges indicating peering agreements between the systems. On the other hand, router-level topologies treat nodes as routers, and edges signify one-hop connectivity between them. Router-level topologies offer two main options for node placement: random placement and heavy-tailed distribution. In the random placement model, nodes are distributed randomly across the test plane, while in the heavy-tailed distribution model, the plane is divided into squares. Nodes are then placed in each square based on a heavily-tailed distribution.

In the first phase of this research, Topology Generators (TGs) are examined. These generators primarily differ in how node placement is determined, as outlined by Sanni et al. (2013), and what each node represents, as explained by Heckmann et al. (2003). Node placement can follow either a *predefined model* or be based on *real-world measurements*. For predefined models, various probability distributions, such as the Waxman model Waxman (1988), may be used. Alternatively, specific strategies, like *chain node placement* (where nodes are placed along a line) or *grid node placement* (where nodes are placed at the intersections of squares), can be employed. In contrast, real-world measurements use actual data from existing network topologies to determine node positions.

TGs can be grouped into two main categories:

- **Placement model**: Nodes are randomly positioned based on a probability distribution, such as the BRITE model Medina et al. (2001a).
- **Measurement model**: Nodes are positioned based on real-world measurements, such as power transmission data, as seen with the NPART model Milic and Malek (2009).

As noted by Heckmann et al. (2003), topologies are typically divided into AS-level and router-level types. In AS-level topologies, nodes represent autonomous systems (AS), and edges represent peering agreements. In router-level topologies, nodes represent routers, and edges denote one-hop connectivity. Router-level topologies provide two primary placement options: random placement and heavy-tailed distribution. In the random placement model, nodes are randomly distributed across the plane, while in the heavy-tailed distribution model, the plane is divided into squares, and nodes are placed within each square according to a heavily-tailed distribution. Once the values are assigned, nodes are randomly positioned within each square.

Previous works, such as Sanni et al. (2013); Nowak et al. (2014); Medina et al. (2001b,a); Milic and Malek (2009), emphasise the importance of correctly representing network topologies in studies of ad hoc mesh networks. Magoni and Pansiot (2006,

2002, 2001) argue that topologies are typically modelled as undirected graphs, where network devices are represented as nodes and communication links as edges. Topology Generators (TGs) are commonly used to test network protocols, and the choice of the initial topology can significantly influence the results, as demonstrated by Magoni and Pansiot (2001); Heckmann et al. (2003). TGs can generate many nodes within the test plane and replicate a physical representation of the topology. However, few models are verified with real measurements, and there is often reliance on mathematical models or power/signal transmission data.

Most research in this field is aimed at creating new or improved TGs, or at highlighting the limitations or inadequacies of existing ones. While TGs provide valuable metrics for studying topologies, there appears to be a gap in research focused on assessing potential biases introduced by different TGs. Some works address this issue by examining the realism of generated topologies.

Magoni and Pansiot (2006) compare three TGs for Internet topologies, evaluating their effectiveness by comparing generated topologies with real-world Internet maps. Similarly, Magoni and Pansiot (2002) analyse six TGs in terms of their accuracy by comparing topology properties with real Internet maps. Several studies have explored various aspects of TGs, including how realistic the generated topologies are. Rossi et al. (2013) propose a framework for analysing Internet topologies using a multi-level approach, based on graph measures and reference datasets, to determine if Internet TGs meet their stated objectives and how realistic they are. In their work, Heckmann et al. (2003) compare three TGs by evaluating the similarity between generated topologies and real-world topologies.

The focus of many previous publications has been to analyse and compare existing topology generators (TGs). However, this research distinguishes itself by shifting the focus from the typical comparison of how well generated topologies replicate real-world networks, to exploring whether the choice of TG—either selecting one over another or using multiple generators—can introduce bias. This novel approach adds a new perspective to the previous work, which has mainly concentrated on comparing the availability and quality of TGs.

This research is particularly relevant to the wireless advanced metering infrastructure (AMI) network, a system used for communication between smart utility meters and utility companies. Unlike typical internet topologies, AMI networks do not rely on the full TCP/IP protocol stack and have a distinct physical topology. A key characteristic of AMI networks is the close correlation between building locations and node connectivity, as highlighted by Nowak et al. (2014). A noteworthy feature of these networks is the role of data aggregation points (DAPs), which are designed to collect and transmit data. Gallardo et al. (2021) propose a machine learning-based algorithm for optimizing the placement of DAPs in residential grids, creating a bipartite

structure with DAPs and feeder nodes. However, this research assumes that all nodes are identical, which is a critical difference. In the context of adversaries targeting these networks, identifying the TG used to place nodes could be a crucial factor in determining the location of missing nodes.

The second phase of this research is centred on reconnaissance within Ad-Hoc Mesh networks. The need for reconnaissance suggests the presence of hidden elements that could be uncovered through various methods. Initially, Ad-Hoc Mesh networks were designed with minimal or no security, assuming that security measures would be applied at the application layer, as noted by Lopez et al. (2021). These networks, with their open medium, dynamic topologies, and wide distribution, pose significant security challenges, as discussed in Shi-Chang et al. (2010), Lou et al. (2009) .

In Ad-Hoc Mesh networks, data flows from a source node to a destination through various intermediary nodes. This decentralised structure increases the potential for infiltration, data alteration, or identity tracing, as emphasised by Olakanmi and Dada (2020). As a result, it becomes clear that Ad-Hoc Mesh networks require robust security measures and careful consideration of traffic encryption to function securely, particularly in hostile environments.

One potential solution for encrypting traffic in these networks is the use of IPsec, as suggested by Witzke et al. (2012). IPsec offers two modes: Transport mode and Tunnel mode. The research recommends using IPsec in Tunnel mode to encrypt traffic in Ad-Hoc Mesh networks, as outlined by López-Millán et al. (2023). IPsec includes two key components: the Authentication Header (AH) and the Encapsulating Security Payload (ESP), as described by Raza et al. (2010). To ensure that data reaches its destination, either a new IP header is added, or routing information is left unencrypted (in plain-text) to preserve routing functionality.

A recent study by Anajemba et al. (2020) introduces an updated encryption standard, I-AES, in combination with a privacy database structure (PDS) to address privacy concerns in the context of the Internet of Things (IoT). Given that IoT networks share similarities with Ad-Hoc Mesh networks—both connect numerous low-powered devices—this approach could be applicable to both types of networks, providing further insights into securing communication within these environments.

Many studies, such as those by Siddiqui et al. (2007) and Rajendran et al. (2021), have pointed out security flaws and impersonation attacks within networks, stressing the importance of understanding the network architecture and routing schemes in ad hoc mesh networks. This knowledge is becoming increasingly critical, particularly in military contexts, where such security concerns are more pressing. Attackers often try to gather detailed information about the network, such as its layout, security measures, and data flow rules. This aligns with findings from Alshamrani (2020), which also explores how similar tactics are used to attack software-defined networks

(SDNs). However, ad-hoc mesh networks differ from SDNs because of their highly mobile nature, which adds an extra layer of complexity to their security challenges.

The focus of this research is on examining the variations in network traffic that might be detected by external sources. This study particularly looks at data that would be available if the inter-node connections were encrypted, aiming to enhance real-world accuracy. The ultimate goal is to explore how choosing a specific topology generator (TG) for node placement might lead to biased results and how this could impact security.

The final phase of this study investigates the identification of influential nodes in complex networks. There is significant cross-disciplinary interest in understanding these influential nodes, with applications in areas such as disease spread Wang et al. (2016), intelligent networks Faris et al. (2019), and even time series prediction (Pravilovic et al. (2017). In the context of wireless sensor networks (WSNs) and advanced metering infrastructures (AMIs), which are both types of ad-hoc mesh networks, the research focuses on how specific data aggregation points (DAPs) receive and process data. In these networks, the nodes not only collect data but also pass it on to DAPs. The nodes play a crucial role in connecting the network and ensuring that data is delivered efficiently.

The concept of identifying influential nodes extends to social networks as well. Research by Song et al. (2016) discusses how, over time, the influence of individuals in these networks changes as connections are made or broken. This concept of influence maximization, where a set of ”seed” nodes is selected to impact the largest number of people, has parallels in ad hoc mesh networks. For example, the connectivity between nodes in a mesh network can influence how information spreads or is blocked.

Furthermore, this research investigates the use of centrality metrics to analyse node importance. Jain and Reddy (2013) show how these metrics are applied to optimise the placement of sink nodes in wireless networks, especially when dealing with real-time video data. In ad-hoc networks, nodes with high centrality are often responsible for forwarding large amounts of data. If these nodes are overburdened, they can deplete their resources quickly, leading to issues such as delays or dropped connections.

Mobile sensor networks are another example where node centrality plays a crucial role, especially when static sensors cannot be installed in hazardous environments, as described by Howard et al. (2002). In military communications, nodes with high centrality may serve as local hubs, making them potential targets due to the traffic they handle, as Kim and Anderson (2012) note. In the case of disease dissemination, identifying nodes with high centrality is equally important, as they can accelerate the spread of infections, as discussed by Lawyer (2015).

In the study of complex networks, predicting future links is a key area of interest. Berahmand et al. (2022) focus on the random walk algorithm, which they prefer for predicting future links in a network. As part of this research phase, the use of various mobility algorithms, including the random walk approach, will be examined.

While using centrality measures for identifying influential nodes is not new, papers like Kim and Anderson (2012) suggest that betweenness and closeness centrality are particularly useful, even though they can be computationally expensive. Other approaches, such as LeaderRank Lü et al. (2011) and PageRank BrinS (1998), have been used to identify influential nodes. More recently, the ANiceRank algorithm Yao and Ji (2019) has been developed to enhance LeaderRank by considering the personal attributes of nodes.

Finally, the concept of temporal networks is discussed by Kim and Anderson (2012), who propose that temporal slices may lose their effectiveness when network topologies change too rapidly. However, during this research, it was found that node centrality tends to stabilise towards the end of simulations, meaning predictions for identifying influential nodes remain accurate using the proposed methods.

Other recent work, like that in Rossi et al. (2020), suggests a new framework for Temporal Graph Networks, which is applied to continuous-time dynamic graphs, adding another layer of complexity and understanding in the study of network behaviours.

Chapter 4

Phase 1: Topology Bias

As previously suggested, if a human were to place nodes within a test area, there is a likelihood that a bias could be induced in the resultant experiment. This phase of research then investigates the presence of bias in the initial placement of nodes in artificial Ad Hoc Mesh Network topologies produced by different TGs. A methodology is proposed to assess such bias and introduce two metrics to quantify the diversity of the topologies generated by a TG with respect to all the available TGs, which can then be used to select what TGs to use. The research set out to address the following questions:

- **Research Question 1**: How to measure the difference between topologies generated by distinct TGs? i.e. how to characterise the bias introduced by the choice of a specific TG rather than using all the TGs?
- **Research Question 2**: How to choose what TG(s), to reduce such a bias?

The significance of research experiments depends heavily on the representativeness of artificial topologies. Indeed, if they were not drawn fairly, obtained results would only apply to a subset of possible configurations, hence they would lack the appropriate generality required to port them to the real world. Although using multiple TGs could mitigate this issue by generating topologies in several different ways, this would entail a significant additional effort. Hence, the problem then arises of what TGs to choose, among several available generators, to maximise the representativeness of generated topologies and reduce the number of TGs to use. Experiments were carried out on three well-known TGs, namely BRITE, NPART and GT-ITM as described in section 2.2. In particular, *given a fixed number of available TGs*. A total of 35 features were extracted from the data, and the extent of these features is tabulated in 4.1. Visual inspection of the topologies from the NPART, BRITE and GT-ITM generators seemed identical and distinguishing which TG produced which topology structure

was impossible. The results of the unsupervised experiment gave the result of an optimum value of 2 classes.

TABLE 4.1: Final List of Features.

	Spatial Randomness
0	Minimum cell count
1	Maximum cell count
2	Cell count range
3	Cell count mode
4	Cell count modal count
	Inter-node distance features
5	Maximum euclidean distance
6	Minimum euclidean distance
7	Range of euclidean distance
8	Mean of euclidean distance
9	Mode of euclidean distance
10	Modal count of euclidean distance
11	Standard deviation of euclidean distance
	Node density
12	Node density with radius of 5 units
13	Node density with radius of 10 units
14	Node density with radius of 20 units
15	Node density with radius of 30 units
16	Node density with radius of 40 units
17	Node density with radius of 60 units
18	Node density with radius of 80 units
19	Node density with radius of 100 units
	Shared neighbours distribution
20	Shared neighbours distribution with radius of 5 units
21	Shared neighbours distribution with radius of 10 units
22	Shared neighbours distribution with radius of 20 units
23	Shared neighbours distribution with radius of 30 units
24	Shared neighbours distribution with radius of 40 units
25	Shared neighbours distribution with radius of 60 units
26	Shared neighbours distribution with radius of 80 units
27	Shared neighbours distribution with radius of 100 units
	Clustering coefficient
28	Clustering coefficient with radius of 5 units
29	Clustering coefficient with radius of 10 units
30	Clustering coefficient with radius of 20 units
31	Clustering coefficient with radius of 30 units
32	Clustering coefficient with radius of 40 units
33	Clustering coefficient with radius of 60 units
34	Clustering coefficient with radius of 80 units
35	Clustering coefficient with radius of 100 units

The first location of the target Ad-Hoc Mesh Network's nodes will, as was stated in the introduction, enable an adversary to set up an interception point to disrupt a key

network link. A side benefit is that knowing which TG was used to produce the placement can help locate unknown nodes when all of their locations are unknown.

4.1 Bias Index

A bias index was proposed for a TG concerning all the TGs. This was measured as the distance between the topologies generated by the selected TG and those generated by all the topologies. This distance is computed from 7 features for inter-node distances, 5 features for spatial distribution, and as many features as the 8 radii defined above for (i) node density, (ii) shared neighbours distribution, and (iii) clustering coefficient.

To estimate the standardised mean, *Hedges' g* is used as set out in Hedges (1981), to calculate the difference between two populations, i.e. the average distance between the elements of two different populations, measured in standard deviations. Although in its original form it only applies to single-dimension elements, it is proposed to extend Hedges' g to multiple dimensions to quantify the difference between topologies.

4.2 Methodology

Taking the basic premise set out in section 2.6 for machine learning algorithms, the programmatically implementation uses the library scikit-learn which is based on the paper -(Pedregosa et al., 2011) and the additional information need to replicate the experiments is set out as follows:

4.2.1 Bernoulli Naive Bayes

This is the first of three Naive Bayes methods use for the analysis of the data as described in section 2.9.2.1. For this experimentation we used sklearn's implementation as documented on the scikit-learn page for Bernoulli Naive Bayes page [1] (sci, a), with no change to the default values, unless stated.

4.2.2 Gaussian Naive Bayes

The second of Naive Bayes methods as described in section 2.9.2.2. For this experimentation, we used sklearn's implementation as documented on the scikit-learn

[1] https://scikit-learn.org/1.6/modules/generated/sklearn.naive_bayes.BernoulliNB.htmlbernoullinb

page for Gaussian Naive Bayes [2] (sci, c), with no change to the default values, unless stated.

4.2.3 MultinomialNB Naive Bayes

The last of Naive Bayes methods as described in section 2.9.2.3. For this experimentation, we used sklearn's implementation as documented on the scikit-learn page for MultinomialNB Naive Bayes page [3] sci (d), with no change to the default values, unless stated.

4.3 Experimental Analysis

To investigate how to measure the difference between topologies, two experiments are undertaken to identify if any features could give a better accuracy of predicting a TG than a random chance. Data was taken from 3000 generated topologies consisting of 1000 topologies for each of the 3 TG. Each generation contained 1000 nodes in a reference area that was 1000 x 1000 units long. To give more realistic results, the nodes are free to establish connections with other nodes placed within a specific distance. This distance (referred to as radius) is randomly chosen from a set of predefined distances (5, 10, 20, 30, 40, 60, 80, 100 units) to mitigate any effect connection distance could have on the simulations.

The first experiment was a path discovery exercise that generates network traffic using the network utility "Ping" and collects that traffic as PCAP (packet capture) data. While evaluating delivery and packet loss statistics, the second experiment looks at a network's throughput. The trials are repeated several times to produce a sufficient amount of data for the analysis phase. Relevant features are extracted from the data through processing, and these features are then the foundation for the machine learning methods discussed earlier in this work. This is done to compare the accuracy of forecasting TGs to that of making random decisions.

As experiment 2 supports the hypothesis, what single or combination of TG's is considered to minimise the *Hedges' g* and therefore best reflects the total population of topologies.

[2] https://scikit-learn.org/1.6/modules/generated/sklearn.naive_bayes.BernoulliNB.htmlsklearn.naive$_b$*ayes.GaussianNB.html*
[3] https://scikit-learn.org/1.6/modules/generated/sklearn.naive_bayes.BernoulliNB.htmlsklearn.naive$_b$*ayes.MultinomialNB.htm*

Algorithm	Correct	Incorrect
K Means	66.63	33.37
Mean Shift	33.13	67.87
Agglomerative Clustering	33.90	66.10
DBScan	0.0	100.00[a]
Spectral	25.76	74.24
Birch	0.03	99.97

TABLE 4.2: Analysis for Untrained Analysis of Features

[a] Note: All DBScan points marked as outliers.

Algorithm	Correct	Incorrect
Agglomerative Clustering	34.00	66.00
DBScan	0	100[a]
Spectral	34.05	65.95

TABLE 4.3: Clustering Algorithm using Trained Analysis

[a] Note: All DBScan points marked as outliers.

4.3.1 Experiment 1

The first of the experiments examined was the use of 'ping' across random network structures; this tests the reachability of a node on the network. A predefined number of nodes was placed in a test area of 1000 x 1000 using one of the three TGs. The number of nodes chosen were from a set {200, 250, 300, 350, 400,450, 500 or 600}. The start and finish nodes of the ping message were next selected at random from among the nodes included in the experiment. Every node's PCAP files were gathered to display the network discovery process and the ping's network traversal. Some nodes may produce empty files because they are not participating in the "ping"; these files are ignored. For the next five minutes after the experiment began, each node performed a random walk to give the 'ping' path a chance to break and create other paths. A total of 5790 runs were recorded, with 1930 runs from each of the classification's classes, to balance the classes.

A simple unsupervised technique gave an accuracy of 67%, this increased to an accuracy of 72% when the probabilistic algorithms were used. These techniques were supplied will all features, however using the sequential feature selection (SFS) to reduce the number of features involved in the analysis gave a final accuracy of 78.6%. This showed that the "Clustering coefficient with a radius of 20 units" by itself explained all the variance when comparing BRITE vs NPART and BRITE vs GT-ITM. While the greatest accuracy for NPART vs GT-ITM was explained by the Inter-node distance feature (Mode of Euclidean distance).

It was thought interesting to see how the wrongly predicted values were distributed. The BRITE class was always classified correctly, while NPART and GT-ITM are often misclassified as one another. This reinforced the finding of the unsupervised learning that 2 of the TGs were difficult to distinguish.

To get clearer results, the analysis was then broken down into paired comparisons. This showed that BRITE could be distinguished from either of the other two classes with 100% accuracy while NPART and GTITM gave a maximum of 64% accuracy, increasing to 77% when applying SFS.

Sequential Feature Selection and K nearest neighbour as set out in Benedetti (1977) classification is used for the analysis stage of this experiment. *k*-fold cross-validation set out in Stone (1974) used, and the accuracy score was averaged score used to test the accuracy. The use of Sequential Feature Selection is to reduce the feature set to a subset of the most relevant features to the problem by using accuracy as the performance measure. This not only resulted in providing the most relevant features but also gave the accuracy of the prediction. Assuming that the choice TG is fair then the probability of obtaining a correct prediction is ⅓ or 33.333%. Using the same experimental data, ignoring any features and randomly picking TG for each entry, using Python's (pseudo) random floating point number generator. Tabulating the results of both the random choice, and the choice guided by the Sequential Feature Selector

TABLE 4.4: Classification Accuracy against Random Selection.

Algorithm	**Classification Accuracy (%)**
GaussianNB	35.66
Random Selection	33.03

(Table 4.4), 33.03% is obtained compared with 35.66% when using SFS. This experiment was conducted using *k*-fold cross-validation to resample the data, by dividing the data into *k* group, where for this experiment case *k* is 5. Each group in turn was held as a test, whilst the other groups were used to train the model. Multiple runs consistently gave an average accuracy over the 5 folds, resulting in a 2% increase from random chance, which could not be said to be statistically relevant.

4.3.2 Experiment 2

The second experiment undertaken consisted of the same 1000 by 1000 test area was used, and the TG's were used to place several nodes there. The nodes were then randomly assigned to preconfigured topologies from all 3 TG, ranging in size from 50 to 100 nodes. Each node employed a random walk for the next five minutes after the experimental start. These nodes were then picked at random to send packets to a sink node, which created the traffic. This simulated a situation where there were

competing routes and bandwidth restrictions. As opposed to a PCAP file, timing and delivery statistics were taken for this experiment. These measurements were 1) an average of all flows and 2) an average of all packets.

Again, using Sequential Feature Selection and K nearest neighbour, the features used in this experiment were reduced to the subset of the most relevant to the problem by using accuracy as the performance measure. This not only gives an indicator of which were the most relevant features, but also an idea of the accuracy of the prediction, (using the best features prediction against using a randomly picked prediction for TG). Using an NS3 library "ns3-network-performance-tool-v2" [4] a description can be accessed using the link E2E, the features measured during the analysis were Network Throughput, Packet Loss Ratio, End to End Delay (Min, Max, Median and Average) and Jitter, which were measured both as an averaging of all network flows through the network and as an averaging of all packets transmitted during the experiment.

TABLE 4.5: Classification Accuracy and Corresponding Features set for the First 5 Iterations of FSS.

Features	Accuracy	Feature IDs
1	**0.6259981**	**11**
2	0.61746557	11,1
3	0.53926863	11,1,4
4	0.52700189	11,1,4,5
5	0.45196711	11,1,4,5,0

Using the same assumption as with Experiment 1, where the choice of TG was 'fair', then the probability of getting a correct choice was ⅓ or 33.333%. With the same Python's (pseudo) random generator, a random choice as to the TG against the actual was measured, with similar results of 0.3330 accuracy recorded. With the prediction guided by the Sequential Feature Selector (Table 4.5), a maximised result of 0.6259981 is obtained from the best feature.

A similar analysis was used in Experiment 2, and k-fold cross-validation was used, where k equals 5. The 'minimum end-to-end delay' measuring feature was the only combination of characteristics that provided the highest accuracy; hence, any additional features decreased accuracy. When the choice of TG was based on the characteristic of "minimum delay incurred for a packet to traverse the network" averaged over all packets during the respective experiment, the result of 62.599% can be regarded to be statistically meaningful.

[4] https://github.com/neje/ns3-network-performance-tool-v2

TABLE 4.6: Average Value for Combination of TGs for Experiment 2 —Single Topologies

Feature	BRITE	NPART	GT-ITM
trans-pack	54.8453	54.8136	54.8445
trans-flow	55.0582	55.0284	55.0594
throu-pack	27355.6239	27688.5487	27026.1971
throu-flow	589032.0771	621210.2999	601630.2906
tx-pack	1072.1736	1071.5547	1072.1566
tx-flow	34992.4028	35040.2717	35646.5816
rx-pack	730.6027	742.4705	721.8805
rx-flow	730.6027	742.4705	721.8805
loss-all	31.6733	30.7880	32.4961
loss-flow	31.6727	30.7874	32.4961
E2E-min-all	16.5445	14.3950	16.5833
E2E-min-flow	0.6874	0.6867	0.6874
E2E-max-all	3385.2774	3229.6150	3418.5973
E2E-max-flow	3385.2774	3229.6150	3418.5973
E2E-ave-pack	320.4106	307.5570	327.3486
E2E-ave-flow	277.3359	271.2663	283.9076
E2E-jit-pack	298.9616	292.2095	306.5563
E2E-jit-flow	318.7785	310.9312	326.4664

Note: Average rounded to four decimal places.

TABLE 4.7: Bias Index for Combination of TGs for Experiment 2

Feature	BRITE	NPART	GT-ITM	BRITE-NPART	BRITE-GT-ITM	NPART-GT-ITM
trans-pack	0.0015746	0.00295112	0.00145226	0.0007233	0.0014921	**0.0007701**
trans-flow	0.0013882	0.00285795	0.00156149	0.0007778	0.0014483	**0.0006716**
throu-pack	0.0005424	0.01966216	0.02048955	0.0102483	0.0102094	**0.0003426**
throu-flow	0.0355235	0.03898646	0.00612829	**0.0030575**	0.0202070	0.0168320
tx-pack	0.0015762	0.00295155	0.00145106	**0.0007227**	0.0014923	0.0007708
tx-flow	0.0090793	0.00752873	0.01709739	0.0085780	**0.0037669**	0.0049980
rx-pack	0.0028131	0.02350926	0.02233989	0.0110867	0.0123047	**0.0007952**
rx-flow	0.0028131	0.02350926	0.02233989	0.0110867	0.0123047	**0.0007952**
loss-all	0.0009769	0.02053885	0.02092578	0.0104674	0.0106442	**0.0001252**
loss-flow	0.0009768	0.02053895	0.02092593	0.0104680	0.0106443	**0.0001253**
E2E-min-all	0.0176351	0.03613450	0.01859785	0.0094410	0.0176339	**0.0086659**
E2E-min-flow	0.0325921	0.04506267	0.03106953	**0.0122679**	0.0355718	0.0129038
E2E-max-all	0.0121476	0.03290375	0.02178707	0.0108917	0.0167048	**0.0056328**
E2E-max-flow	0.0121476	0.03290375	0.02178707	0.0108917	0.0167048	**0.0056328**
E2E-ave-pack	0.0051636	0.02549998	0.02164926	0.0108394	0.0130691	**0.0019803**
E2E-ave-flow	**0.0000014**	0.01668297	0.01805647	0.0090210	0.0087372	0.0006081
E2E-jit-pack	**0.0002610**	0.01800765	0.01968096	0.0098381	0.0094060	0.0007424
E2E-jit-flow	0.0006104	0.01925311	0.02005763	0.0100234	0.0100316	**0.0003010**

Note: Highlighted values are the lowest value for each feature.

4.3.3 Features Analysis

Carrying out a more detailed analysis of which features are weighted most for classification by applying the Forward Sequential Selection (FSS) method as a sequential feature selector. FSS works sequentially, starting with an empty set of features and at each iteration adding the feature to the set that yields the highest accuracy.

TABLE 4.8: Bias Index of the Considered TGs.

Topology Generator(s)	Bias index
NPART	1.890
GT-ITM	2.145
BRITE	4.282
BRITE + NPART	0.908
BRITE + GT-ITM	0.976
NPART + GT-ITM	2.430

TABLE 4.9: Classification Accuracy for the Three Classification Algorithms.

Algorithm	Classification Accuracy (%)
GaussianNB	77.95
BernoulliNB	58.56
MultinomialNB	70.07

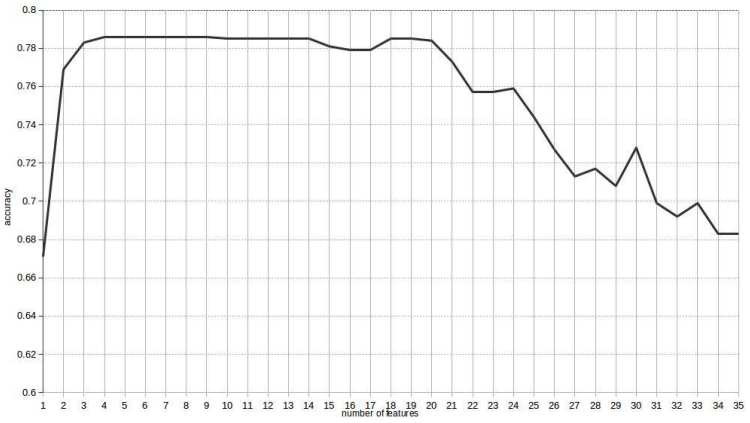

FIGURE 4.1: Classification Accuracy by Varying the Number of Used Features for a Specific Fold.

Figure 4.1 shows how the classification accuracy varies with the number of considered features, for a single fold, and that the highest accuracy is achieved with 4 features.

TABLE 4.10: Final List of Features

Feature rank	Feature description
1st	31. Shared neighbours distribution with 30 units radius
2nd	6. Minimum Euclidean distance
3rd	30. Shared neighbours distribution with 20 units radius
4th	14. Node density with 20 units radius

Table 4.10 lists the four features yielding the highest classification accuracy. About the three TGs employed in our tests, this feature analysis identifies the key differences in

the topologies produced by the various TGs. The exact application such topologies must be employed for, or how much those qualities matter for the intended scenario, will determine if this can introduce bias.

TABLE 4.11: Table of Accuracy against Number of Features for all Generators

Number of features	Accuracy	Combination of features
1	0.668	31
2	0.769	31,6
3	0.783	31,6,30
4	**0.786**	**31,6,30,14**
5	0.786	31,6,30,14,13
6	0.786	31,6,30,14,13,12
7	0.786	31,6,30,14,13,12,15
8	0.786	31,6,30,14,13,12,15,16
9	0.786	31,6,30,14,13,12,15,16,20

4.3.4 Mislabelled Points

Taking the fact that when predicting the class of new data points when using a probabilistic clustering, this gives only an average accuracy of 77.95%, the breakdown of those mislabelled was investigated.

When all classes were analysed it was found that out of those mislabelled points, 47.8% of GT-ITM points are misclassed as NPART while 52.2% of NPART are misclassed as GT-ITM. BRITE data points are never misclassed, which meant that when comparing between pairs of TG's BRITE vs NPART or BRITE vs GT-ITM the were no misclassifications. hen looking at the comparison of NPART vs GT-ITM, it was found that out of those data points that were misclassified, 75% of NPART was misclassed as GT-ITM and 25% of GT-ITM were misclassed as NPART.

When all three classes are analysed together, BRITE is always categorised correctly and GT-ITM and NPART are mislabelled equally. As a sample of training vs testing data was randomly selected, this could explain the slight difference in percentage.

4.4 Research Question 1

How to characterise the bias introduced by the choice of a specific TG rather than using all the TGs?

A TG's bias index about all other TGs was proposed. The difference between the topologies produced by the chosen TG and those produced by all the topologies was

used to measure this. This distance is calculated using seven features for the distances between nodes, five features for the spatial distribution, and as many features for the node density, shared neighbour distribution, and clustering coefficient as there are radii for each of the eight parameters that were previously determined.

TABLE 4.12: Classification Accuracy for the Three Classification Algorithms. As 10-Fold Cross-Validation is Used, the Classification Accuracy a_l for each Fold is reported as well.

Algorithm	**Fold-wise Classif. Accuracy (%)**	**Ave Acc (%)**
GaussianNB	78.6, 78.6, 77.6, 77.6, 77.6, 82.3, 78.3, 74.6, 79.0, 75.3	77.95
BernoulliNB	57.3, 56.6, 55.7, 58.7, 60.7, 60.7, 56.0, 60.3, 64.0, 55.6	58.56
MultinomialNB	71.7, 67.7, 78.3, 70.7, 69.0, 70.3, 66.0, 72.0, 64.7, 70.3	70.07

As previously mentioned, the bias index is calculated for each TG and between each pair of TGs. The results are shown in table 4.8. BRITE topologies seem to deviate significantly from NPART and GT-ITM topologies, and vice versa. This implies that using each TG separately would result in a collection of topologies that is very different from the set of topologies that includes all of the topologies. Conversely, NPART appears to generate topologies that are less different from all accessible TGs' topologies overall. According to Table 4.8, the findings also indicated that BRITE and NPART, which have the lowest combined bias, seem to be the best options for a pair of TGs.

The accuracy of the categorisation is evaluated to find out how well topologies can be divided according to their TG. The higher the classification accuracy, the more markedly diverse topologies generated by various TGs are from one another. While selecting and optimising classifiers to improve classification accuracy is standard routine, in this case, our primary focus is on establishing whether the accuracy can be appreciably greater than 1/3. Three different probability distributions the multinomial, Bernoulli, and Gaussian were employed to test if the outcomes are consistently independent of the particular distribution. These are all illustrations of the Naive Bayes classifier, which was selected due to its ease of use. The classification accuracy for each of the three techniques is displayed in table 4.9, where it is evident that every value is noticeably greater than 1/3. The accuracy results are an average of k-folds, the individual values are set out in the Table 4.12

4.5 Research Question 2

How to choose what TG(s), to use to reduce such a bias?

Results from both experiments showed that when using the artificial networks produced by a single TG, a detectable bias was introduced which could then

potentially jeopardise related experiments. This problem can be mitigated by considering topologies generated by all three of the TGs to generate topologies. If however only two TGs could be chosen, BRITE and NPART proved to be the TG pair with the lowest bias. Finally, when only one TG was chosen, it would be logical to choose the of TG that shows the least bias, which was the NPART TG.

It has been established that one or more of our features were biased, which allowed the K nearest neighbour to categorise the findings with an accuracy of 62.599%. The average values of the characteristics noted in experiment 2 are tabulated in Table 4.6. It demonstrated that there was little difference for each TG or TG combination. This might be due to the different population sizes, so to counteract this, the *Hedges' g* is computed, as explained in section 4.1. This uses the standardised mean difference between two populations, and in this instance, the population would be each TG and each pair of TGs, compared to all TGs.

Using *Hedges' g* to look at the mean and standard deviation of each feature, this gave a measure of distance between that of a single TG or pair of TG's from all populations within the experiment. If all TGs are used it would be like comparing two identical populations and hence would have given a value of zero. Therefore looking at the value of *Hedges' g* that is closest to zero for the majority of features. As can be seen from Table 4.7, the combination of the NPART TG and GT-ITM TG gives the lowest *Hedges' g* for the greatest number (⅔) of features.

Chapter 5

Phase 2: Routing Algorithm Reconnaissance in Ad-Hoc Mesh Networks

This phase of the research demonstrated that it is possible to predict the routing algorithm based on the data transmitted by nodes in an Ad-hoc Mesh Network. The investigation aimed to address the following questions:

- **Research Question 3**: What is the most accurate machine learning (ML)-based approach for detecting the routing algorithm used in an Ad-hoc Mesh Network?
- **Research Question 4**: How does the detection accuracy change when considering routing algorithms within the same class?
- **Research Question 5**: How is the detection accuracy affected by reducing either the number of nodes from which data is collected or the amount of data gathered from each node?

As previously mentioned, the study experiments rely on simulation data. In these simulations, network traffic is encrypted, as would be expected in real-world scenarios, meaning that only the metadata of the network is available for analysis. There are several encryption methods, but for this study, IPsec was employed, as outlined in Witzke et al. (2012).

IPsec operates in two modes: Transport mode and Tunnel mode. Research suggests that Tunnel mode, as discussed by Reddy and Thilagam (2013), is optimal for encrypting traffic in Ad-hoc Mesh Networks. IPsec defines two key components: the Authentication Header (AH) and the Encapsulating Security Payload (ESP), as explained by Raza et al. (2010). (2010). If each node had to decrypt the header data to

route the traffic, it would require additional computational power to manage the overhead. As a result, to maintain efficiency, either a new IP header would be added, or routing information would remain unencrypted (in plain text).

In a related study, Anajemba et al. (2020) explored privacy issues within the Internet of Things (IoT), recommending advancements in encryption standards (I-AES) combined with a privacy database structure (PDS). Similar to IoT networks, Ad-hoc Mesh Networks involve numerous lower-powered devices connected through fewer higher-powered devices.

The research in this phase focused on the routing of data within such networks. Routing can be divided based on how each node manages its knowledge of the network. These methods generally involve either storing this information in a table updated periodically (Proactive Routing) or requesting routing information when data transmission begins (Reactive Routing). This distinction forms the basis of the Proactive and Reactive routing methods.

'Proactive Routing' (PR) involves a predefined table that holds the 'next hop' on the path to a destination. This table is updated periodically via broadcasts, which spread routing information across the network. On the other hand, 'Reactive Routing' (RR) only discovers routes when a packet needs to be sent to a specific destination, populating the routing table when necessary. When both methods are combined in the same network, it is referred to as 'Hybrid Routing' (HR).

Hybrid Routing blends Proactive and Reactive approaches. The network is divided into zones, where nodes within a zone use Proactive Routing to maintain routing information for other nodes in the same zone. To communicate outside of the zone, Reactive Routing is used to discover the path. Thus, nodes apply different routing algorithms based on the destination node's location in the network.

At the time of writing, there was limited simulation software available to fully emulate Hybrid Routing for this research. Given that creating a custom simulator could inadvertently introduce biases that would affect the study, the decision was made not to use Hybrid Routing in the investigation.

5.1 Reconnaissance in Ad-Hoc Mesh Networks

Reconnaissance attacks on ad-hoc mesh networks are not a new concept. The work of Alshamrani (2020) , which discusses attacks on software-defined networks (SDNs), provides relevant insights. It suggests that if a part of the network can be compromised, it could generate specific traffic patterns that an attacker could observe. A non-intrusive method of gathering this information would be especially

advantageous. The task of identifying the routing algorithm becomes even more crucial when investigating encrypted traffic, as in ad-hoc mesh networks, particularly when probing port usage or embedded services.

From an adversarial perspective, a target network can span a large area. If an attacker is located at one end of the network, they may not have visibility of traffic occurring at the other end. In cases where data is being gathered within a limited range, it might not capture all nodes involved in a transaction. To address this, a holistic approach was taken in the research, focusing less on specific nodes and instead analysing packets transmitted or received by individual nodes. Early attempts to differentiate point-to-point traffic based on well-defined port numbers showed limitations, as some programs use arbitrary ports to hide their activities Karagiannis et al. (2004).

5.2 Machine Learning for Traffic Analysis in Ad-Hoc Mesh Networks

Machine learning (ML) has found various applications within ad-hoc mesh networks, and its use in military mobile ad-hoc networks (MANETs) is discussed in Kant et al. (2008). Since this study focuses on how data is routed through the network, ML is primarily applied to route optimization, as demonstrated in the work of Mao et al. (2018), Stampa et al. (2017). Upon reviewing the existing literature, it's clear that ML has been used extensively in many areas of ad-hoc mesh networks, including guiding mobile nodes, as noted in Kuskonmaz et al. (2019).

Machine learning has also been applied in signal intelligence. For example, Jagannath et al. (2019) highlight the use of deep learning for modulation classification. Additionally, automata have been employed for channel assignment learning Sarao (2019). While there is ample precedence for applying ML in ad-hoc mesh networks, there is no universally accepted method or technique for doing so. As a result, the first step in this phase of the research was to identify which ML technique yields the most accurate results.

5.3 Methodology

Taking the basic premise set out in section 2.6 for machine learning algorithms, the programmatically implementation uses the library scikit-learn which is based on the paper -Pedregosa et al. (2011) and the additional information need to replicate the experiments are set out as follows:

5.3.1 SVM with K-means

For Support Vector Machine as described in section 2.9.3. The implementation followed the documentation [1] - sci (f).

5.3.2 Decision Tree

For Decision Tree as described in section 2.9.9. The implementation followed the documentation [2] - sci (b).

5.3.3 Random Forest

For Random Forest as described in section 2.9.10. The implementation followed the documentation [3] - sci (e).

5.3.4 Convolutional Neural Network

For Convolutional Neural Network as described in section 2.9.1. The specific architecture can be summarised as follows:

Architecture Breakdown

- **Convolutional Layers:**
  - **conv1:** A convolutional layer that takes 1 input channel (grayscale image) and produces 8 output channels with a kernel size of (3, 5).
  - **conv2:** Takes the output of conv1 (8 channels) and produces 16 channels with a kernel size of (3, 5).
  - **conv3:** Takes the output of conv2 (16 channels) and produces 32 channels with a kernel size of (3, 5).
- **Fully Connected Layers (Linear Layers):**
  - **fc1:** A fully connected layer that takes the output from the final convolutional layer (flattened) and maps it to 128 features.
  - **fc2:** A fully connected layer that reduces the dimensionality from 128 to 64.
  - **fc3:** A final fully connected layer that outputs 3 values (perhaps corresponding to 3 classes for classification).

[1] https://scikit-learn.org/1.5/modules/generated/sklearn.svm.SVC.html
[2] hhttps://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
[3] https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.RandomForestClassifier.html

- **Activation, Pooling and Regularisation**

  ReLU activation functions are applied after each convolutional and fully connected layer. Max-pooling with a window size of (2, 2) is applied after the first two convolutional layers to reduce spatial dimensions. Additionally, dropout layers can be introduced after fully connected layers to reduce overfitting by randomly deactivating a fraction of neurons during training. This is a common regularisation technique in CNNs.

5.3.5 Bernoulli Naive Bayes

For an explanation of the parameters used for Bernoulli Naive Bayes, see previous section 4.2.1

5.3.6 Gaussian Naive Bayes

For an explanation of the parameters used for Gaussian Naive Bayes, see previous section 4.2.2

5.3.7 Deep Forest Neural Network

This is a neural network implementation of a Deep Forest Neural Network or Neural Decision Forest (NDF), which combines a deep neural network (to extract features) with decision trees (for classification) and expands section 2.9.11 and for the experiment during this phase the following architecture is employed:

Architecture Breakdown

The neural network consists of three primary classes:

- **FeatureLayer** (the feature extractor based on convolutional layers)
- **Tree** (a single decision tree, where each tree makes decisions based on features extracted from the input)
- **Forest** (a collection of decision trees that aggregate predictions)
- **NeuralDecisionForest** (the entire model that combines the feature layer with the forest of trees)

- **FeatureLayer (Feature Extraction) Convolutional Layers:**

  - **Conv1:** 2 input channels → 8 output channels, kernel size (3,3), followed by ReLU and max-pooling.
  - **Conv2:** 8 input channels → 16 output channels, kernel size (3,3), followed by ReLU and max-pooling.
  - **Conv3:** 16 input channels → 32 output channels, kernel size (3,3), followed by ReLU and max-pooling.
  - **Dropout:** Applied after each convolutional layer to prevent overfitting.

- **Tree (Decision Tree)**

  Each tree in the forest is a decision tree. It has a depth (depth), number of input features (n_in_feature), a feature mask (which decides which features are used for splitting at each node), and leaf probabilities (pi), which define the class distribution at each leaf.

  - **Features:** Each tree uses a subset of the features (used_feature_rate).
  - **Decision Layer** A linear layer followed by sigmoid activation that calculates the decision at each node in the tree.
  - **Route Probability@** This is computed based on the decision at each node and passed through the tree layers to determine the final leaf probabilities.

- **Forest (Random Forest of Decision Trees)**

  The forest consists of n_tree trees, where each tree independently computes class probabilities. The final output probability is the average of the probabilities predicted by each tree.

- **NeuralDecisionForest**

  The complete model consists of the feature layer (CNN) and the forest (decision trees). The feature layer processes the input data, and its output is fed into the forest for classification.

5.4 Research Question 3

To address the research question, a dataset was compiled with two classes—AODV and OLSR—based on simulations. Before any meaningful investigation could be undertaken, it was crucial to investigate how different combinations of network packet fields influenced machine learning accuracy. This was done by applying the same machine learning technique to various field combinations and comparing the results.

To answer Research Question 3 (RQ3), the research process was divided into three stages: (i) selecting suitable machine learning algorithms, (ii) extracting feature

vectors from captured data, and (iii) feeding these vectors into the ML algorithms to assess their accuracy in detecting the routing algorithm. For stage (i), all seven machine learning algorithms introduced in Section 2.6 were tested to identify the most accurate ones. The four algorithms with the highest accuracy were then chosen for further analysis.

In stage (ii), each feature vector represented a sequence of P consecutive packets sent by each of the N nodes. This approach allowed us to capture patterns in packet transmission over time and space, which could help distinguish between the routing algorithms. The average accuracy of these models over multiple experiments is shown in Table 5.2.

It was found that the Random Forest (RF) algorithm achieved the highest accuracy across several combinations. To simplify the analysis, the field combinations that yielded the highest accuracy for the Support Vector Machine (SVM) algorithm were used in subsequent experiments. The results showed that for the Convolutional Neural Network (CNN), the highest accuracy—91%—was achieved when using the IP header length as a single field. For SVM and RF, combining IP header length, Subtype frame, and UDP length resulted in accuracies of 99.98% and 99.87%, respectively. For Decision Trees (DT), a combination of frame length, IP header length, and UDP length resulted in 99.94% accuracy.

TABLE 5.1: Packet Header Fields used to generate Feature Vectors.

Field	**Description**
frame.len	Length of frame
ip.hdr len	Length of header
wlan.fc.subtype	Management frames subtype
udp.length	Length of the UDP header plus the UDP data

During the investigation, data was collected from network simulations, where for each packet, all header fields up to Layer 3 were extracted and included in the feature vector. The only exceptions were fields that were specific to the particular simulation run (such as 'frame.number' and 'frame.time epoch') or the network topology used (such as 'ip.src', 'ip.dst', 'wlan.ta', and 'wlan.ra'). The resulting feature vectors were composed of the four header fields listed in Table 5.1. To determine which field combinations provided the best accuracy, all possible 15 combinations of these four fields were tested.

The research identified the top three algorithms for routing algorithm detection: Support Vector Machine (SVM) with K-Means, Decision Tree (DT), and Random Forest (RF). Both the SVM and RF algorithms produced the best results when using the field combination of 'wlan.fc.subtype', 'ip.hdr len', and 'udp.length'. The DT algorithm performed best with a combination of 'frame.len', 'ip.hdr len', and 'udp.length'.

TABLE 5.2: Results from Field Combinations, 2 Classes

Combination	CNN Acc (%)	SVM Acc (%)	RF Acc (%)	DT Acc (%)
FL, HL, S, UL	0.7599	0.9998	0.9984	0.9992
FL, S, UL.	0.7361	0.9899	0.9971	0.9971
FL, HL, UL.	0.7353	0.9996	0.9984	**0.9994**
FL, HL, S	0.6025	0.9998	0.8642	0.9905
HL, S, UL	0.8054	**0.9998**	**0.9987**	0.9993
S, UL.	0.7193	0.9983	0.9786	0.9972
HL, S	0.8156	0.9998	0.6661	0.9900
FL, HL	0.5681	0.9947	0.9652	0.8357
FL, S	0.5281	0.9817	0.6546	0.9731
FL, UL	0.7383	0.9918	0.9917	0.9867
HL, UL	0.7332	0.9809	0.9917	0.9993
FL only	0.5299	0.8082	0.6451	0.5002
HL only	**0.9152**	0.9123	0.8807	0.8357
S only	0.8745	0.8896	0.6510	0.9366
UL Only	0.8374	0.9376	0.9600	0.9867

FL = Frame len, S = Subtype, HL = Header Len,
UL = UDP Len

Accuracy was found to decrease when considering routing algorithms within the same 'Reactive Routing' **(RR)** class. Reducing the number of nodes in the network or the amount of data collected from each node had minimal impact on detection accuracy.

The machine learning algorithms investigated were from two categories: parametric and non-parametric. Parametric algorithms use weights and bias to match input data to output data, while non-parametric algorithms construct a mapping function to find the best fit from input data to output classes. Out of the seven techniques initially considered, four were chosen for further analysis due to their high accuracy: CNN, SVM, RF, and DT.

For the four selected algorithms, SVM, RF, and DT used the mean value of each sample to classify the different routing algorithms, correlating to the number of control packets within the sample. The CNN technique, on the other hand, focused on groups of packets and their interactions, providing high accuracy when handling full-size samples but showing decreased accuracy as sample size was reduced. All four techniques demonstrated that the field 'frame.len' by itself gave the lowest predictive accuracy.

When using convolutional neural networks for classification, the results showed that CNN could match the performance of SVM when applied to two classes, but its accuracy dropped when used with three classes. This suggests that CNN struggles when combining information from different channels. This issue warrants further investigation, as CNN's accuracy was based on a single feature.

Two scenarios were considered during the research: one involved classifying data from Proactive Routing **(PR)** and Reactive Routing **(RR)**, and the other focused on classifying a single **(PR)** example and two **(RR)** examples. The findings were as follows: in the first scenario, SVM achieved 100% accuracy, which dropped to 97.96% in the second scenario. For RF, accuracy was 97.96% for two classes and 96.51% for three classes. For DT, the first scenario resulted in 99.90% accuracy for two classes and 99.80% for three classes.

In the second scenario, the minimum sample size required to maintain accuracy was investigated. It was found that SVM maintained its accuracy even when the sample size was reduced to the smallest size—five nodes with five consecutive packets. This indicates that it is highly likely that the routing protocol used within an Ad-Hoc Mesh Network can be predicted with high accuracy based on traffic from just a small sample (five) of randomly chosen nodes, using a minimum of five consecutive packets, whether they are transmission or reception packets.

Given that each simulation could generate up to 100,000 packets, processing the entire dataset for analysis would require excessive computational resources. To manage this, the output of the simulations was divided into slices of 500 consecutive packets per node to reduce computational cost. Subsequently, the number of techniques used for experimentation was reduced.

Initially, seven techniques were considered, but this was deemed too many for the full set of experiments. The techniques were then reduced to four. To determine which techniques to eliminate, all were tested against two classes (AODV and OLSR) and three classes (two from the **(RR)** class and one from the **(PR)** class), with the results summarized in Table 5.3. Based on these results, the final selection of algorithms was made, discarding Deep Forest Neural Network, Bernoulli Naive Bayes, and Gaussian Naive Bayes, as they demonstrated the lowest average accuracy.

TABLE 5.3: Analytic Test with 2 and 3 Algorithm and Average Accuracy obtained.

Result / Algorithm	**2 Class Acc**	**3 Class Acc**	**Ave Acc**
SVM with K-means	1.00	0.980	0.990
Random Forest	0.999	0.996	0.998
Convolutional NN	0.9895	0.932	0.960
Bernoulli Naive Bayes	0.9871	0.962	0.95655
Gaussian Naive Bayes	0.9579	0.926	0.88495
Deep Forest NN	0.919	0.670	0.7945
Decision Tree	0.999	0.997	0.998

To address **RQ3**, we examined the field combinations that yielded the highest accuracy for each of the selected machine learning algorithms. The results, summarized in Table 5.2, showed that for CNN, the single field 'ip.hdr len' achieved

an accuracy of 91.52%. For SVM and RF, the field combination of 'wlan.fc.subtype', 'ip.hdr len', and 'udp.length' resulted in an accuracy around 99%. Finally, for Decision Tree (DT), the combination of 'frame.len', 'ip.hdr len', and 'udp.length' reached an accuracy of 99.94%.

It's worth noting that for the 2-class analysis using the full sample size, SVM achieved the highest accuracy of 100%, while both RF and DT reached 99.99%. However, when the sample was reduced to just 5 nodes and 5 consecutive packets, SVM's accuracy dropped to 99%, while RF and DT performed slightly better with 99.1%. In the case of the 3-class analysis, DT maintained an accuracy of 99.1%, while RF decreased from 99.1% to 98.9% and SVM dropped from 99% to 96.5%.

TABLE 5.4: Reduced Aperture Sampling for CNN.

Nodes / Sample	**Classes**	**50**	**40**	**30**	**20**	**10**	**5**
500	2	0.880	0.864	0.892	0.869	0.887	n/a
	3	0.749	0.752	0.770	0.763	0.795	n/a
400	2	0.872	0.873	0.879	0.847	0.862	n/a
	3	0.735	0.744	0.791	0.755	0.804	n/a
300	2	0.885	0.892	0.884	0.863	0.866	n/a
	3	0.750	0.734	0.739	0.763	0.767	n/a
200	2	0.902	0.902	0.912	0.874	0.846	n/a
	3	0.736	0.763	0.776	0.789	0.745	n/a
100	2	0.859	0.835	0.822	0.793	0.745	n/a
	3	0.700	0.707	0.701	0.718	0.660	n/a
50	2	0.817	0.803	0.784	0.735	0.709	n/a
	3	0.715	0.702	0.709	0.693	0.642	n/a
40	2	0.807	0.803	0.751	0.735	0.691	n/a
	3	0.674	0.721	0.673	0.648	0.610	n/a
30	2	0.788	0.761	0.746	0.713	0.665	n/a
	3	0.709	0.705	0.681	0.717	0.668	n/a
20	2	0.775	0.752	0.722	0.697	0.662	n/a
	3	0.658	0.682	0.679	0.668	0.677	n/a
10	2	0.728	0.718	0.678	0.667	0.640	n/a
	3	0.590	0.613	0.587	0.560	0.533	n/a
5	2	n/a	n/a	n/a	n/a	n/a	n/a
	3	n/a	n/a	n/a	n/a	n/a	n/a

The analysis shows that SVM delivers the highest accuracy when distinguishing between two classes, while DT proves more effective when handling three classes. Notably, DT maintains its accuracy even when the sample size is reduced, indicating its robustness against limited data. These findings suggest that DT provides highly accurate predictions, particularly when the algorithms belong to different classes, as the differences in network traffic patterns are more pronounced.

However, when a third class is introduced, SVM experiences a slight drop in accuracy. This suggests that the additional class shares similarities with an existing one, making it harder for SVM to differentiate between them.

TABLE 5.5: Reduced Aperture Sampling for SVM with K Means.

Nodes / Sample	Classes	50	40	30	20	10	5
500	2	1.000	1.000	1.000	1.000	1.000	1.00
	3	0.980	0.980	0.980	0.980	0.977	0.976
400	2	1.000	1.000	1.000	1.000	1.000	1.00
	3	0.980	0.980	0.980	0.979	0.979	0.976
300	2	0.999	1.000	0.999	0.999	1.000	0.998
	3	0.980	0.979	0.979	0.979	0.9778	0.974
200	2	1.000	0.999	0.999	0.999	0.999	0.999
	3	0.979	0.979	0.979	0.979	0.975	0.972
100	2	0.997	0.998	0.999	0.998	0.996	0.990
	3	0.975	0.975	0.973	0.972	0.969	0.961
50	2	0.999	0.998	0.999	0.999	0.995	0.995
	3	0.975	0.975	0.974	0.972	0.967	0.957
40	2	0.998	0.999	0.999	0.997	0.998	0.994
	3	0.975	0.975	0.972	0.969	0.963	0.961
30	2	0.999	0.999	0.998	0.999	0.998	0.996
	3	0.975	0.973	0.972	0.969	0.962	0.966
20	2	0.9996	0.9986	0.9993	0.9989	0.9986	0.9971
	3	0.9740	0.9720	0.9706	0.9666	0.9676	0.9699
10	2	1.00	0.998	0.997	0.999	0.997	0.994
	3	0.972	0.964	0.969	0.975	0.978	0.974
5	2	0.998	0.999	0.999	0.998	0.995	0.990
	3	0.973	0.977	0.982	0.984	0.983	0.965

The combination of fields selected for the analysis reflects the underlying patterns of routing behaviour. While these fields do not directly identify the type of protocol used, they highlight the ratio of control packets to regular data packets—an indicator of the routing style. This explains the results of the two-class analysis, where AODV and OLSR were examined. These protocols use fundamentally different routing mechanisms: AODV, part of the reactive routing class, discovers routes only when needed, whereas OLSR, belonging to the proactive routing class, maintains routing tables through periodic broadcasts.

Ultimately, the results suggest that the metadata analysed—derived from a combination of packet fields—could potentially be used to identify individual packet types involved in the routing algorithms, offering insights into how data flows through the network.

5.5 Research Question 4:

The confusion matrices in Table 5.8, Table 5.9 ,Table 5.10 and Table 5.11 illustrate how each technique—SVM, Random Forest, Decision Tree, and CNN—misclassifies predictions among AODV, DSR, and OLSR. A slight imbalance exists in the three classes within the matrices for SVM, Random Forest, and Decision Tree.

TABLE 5.6: Reduced Aperture Sampling for Random Forest.

Nodes / Sample	Classes	50	40	30	20	10	5
500	2	1.000	1.000	1.000	1.000	1.000	0.999
	3	0.997	0.997	0.997	0.996	0.995	0.995
400	2	1.000	1.000	1.000	1.000	1.000	0.994
	3	0.997	0.997	0.997	0.995	0.995	0.993
300	2	1.000	1.000	1.000	1.0007	1.000	0.999
	3	0.996	0.996	0.996	0.995	0.993	0.993
200	2	1.0005	0.999	0.999	1.000	0.999	0.999
	3	0.994	0.994	0.994	0.993	0.992	0.993
100	2	0.998	0.999	0.999	0.998	0.997	0.995
	3	0.992	0.992	0.992	0.992	0.991	0.993
50	2	0.999	0.999	0.999	0.998	0.998	0.996
	3	0.993	0.992	0.992	0.993	0.993	0.993
40	2	1.000	0.999	0.999	0.999	0.998	0.992
	3	0.992	0.993	0.992	0.993	0.993	0.992
30	2	0.999	0.999	0.999	0.999	0.998	0.996
	3	0.993	0.993	0.993	0.993	0.994	0.994
20	2	0.999	0.999	0.999	0.999	0.998	0.996
	3	0.993	0.993	0.994	0.993	0.994	0.994
10	2	0.999	0.999	0.997	0.998	0.997	0.993
	3	0.994	0.995	0.996	0.995	0.996	0.991
5	2	0.999	0.999	0.999	0.998	0.996	0.991
	3	0.995	0.994	0.997	0.997	0.994	0.989

For example, across 6,000 runs, the class selection was not strictly divided into 2,000 instances per class due to the randomness of the training/testing split. As a result, some classes had slightly more instances while others had fewer. However, this imbalance is corrected when averaging results over multiple repetitions and incorporating them into other experiments.

To answer **RQ4**, the choice between SVM, RF, DT and CNN for the analysis is based on the results in Table 5.2. The highest accuracy for any combination for the CNN is 91.52%, while the accuracy for SVM, DT and RF is in the region of 99%.

The most suitable analytical technique for this study appears to be the Decision Tree (DT) method, as its accuracy remains consistent regardless of whether the sample size is reduced or the classification moves from two to three classes.

Several key points emerge from these findings. Firstly, increasing the number of fields analysed during the Convolutional Neural Network (CNN) evaluation seemed to dilute its accuracy. This is evident in the results, where using three fields yielded a maximum accuracy of 80%, compared to 91.52% when only a single field was used. In contrast, the Support Vector Machine (SVM) analysis showed the opposite trend—achieving a maximum accuracy of 99.97% with a combination of three fields, but only 88% when relying on a single field.

TABLE 5.7: Reduced Aperture Sampling for Decision Tree.

Nodes / Sample	Classes	50	40	30	20	10	5
500	2	0.999	0.999	0.999	0.999	0.999	0.999
	3	0.977	0.977	0.978	0.975	0.974	0.976
400	2	0.999	0.999	0.999	0.999	0.999	0.999
	3	0.977	0.975	0.999	0.974	0.974	0.975
300	2	0.999	0.999	0.999	0.999	0.998	0.998
	3	0.975	0.974	0.976	0.974	0.974	0.977
200	2	0.998	0.999	0.999	0.999	0.998	0.998
	3	0.975	0.974	0.974	0.976	0.976	0.982
100	2	0.998	0.998	0.997	0.996	0.997	0.995
	3	0.971	0.971	0.972	0.976	0.980	0.985
50	2	0.998	0.998	0.998	0.998	0.996	0.995
	3	0.975	0.976	0.979	0.981	0.988	0.989
40	2	0.998	0.998	0.998	0.998	0.997	0.994
	3	0.975	0.976	0.979	0.981	0.988	0.989
30	2	0.998	0.998	0.998	0.998	0.997	0.994
	3	0.978	0.980	0.983	0.986	0.991	0.992
20	2	0.998	0.999	0.999	0.998	0.998	0.995
	3	0.983	0.984	0.985	0.989	0.993	0.992
10	2	0.998	0.998	0.996	0.997	0.996	0.994
	3	0.990	0.991	0.994	0.994	0.994	0.992
5	2	0.998	0.998	0.998	0.996	0.994	0.991
	3	0.995	0.994	0.996	0.996	0.994	0.991

TABLE 5.8: Confusion Matrix for SVM with K-Means.

Predicted / Real	AODV	DSR	OLSR
AODV	1971	0	16
DSR	0	2018	0
OLSR	2	0	1993

TABLE 5.9: Confusion Matrix for Random Forest.

Predicted / Real	AODV	DSR	OLSR
AODV	1984	0	3
DSR	0	2018	0
OLSR	9	0	1986

This difference can be explained by the way each algorithm processes data. When CNN analyses multiple fields, its convolutional kernel searches for patterns across both connected packets and the relationships between fields, focusing on how they interact. With a single field, however, the kernel still looks across multiple packets but focuses solely on identifying patterns within that one field.

On the other hand, SVM treats each sample as a single data point for its analysis. In

TABLE 5.10: Confusion Matrix for Decision Tree.

Predicted \ Real	**AODV**	**DSR**	**OLSR**
AODV	1991	7	0
DSR	29	1991	5
OLSR	6	0	1971

TABLE 5.11: Confusion Matrix for CNN.

Predicted \ Real	**AODV**	**DSR**	**OLSR**
AODV	2631	83	39
DSR	98	2224	526
OLSR	271	693	2435

this study, SVM averaged the values from each packet within a sample for a particular field, collecting these data points from all samples to form its dataset. This means SVM is less concerned with how fields interact across packets and more focused on the statistical characteristics of individual fields.

As shown in Table 5.2, CNN produced higher accuracy when using a single field (such as Header Length), while SVM and Random Forest (RF) performed better with a combination of multiple fields (Header Length, Subtype, and UDP Length). Interestingly, differences between routing algorithm classes were more distinguishable when CNN analysed a single field across multiple packets, compared to analysing multiple fields simultaneously. Conversely, SVM and RF more effectively identified class differences when using multiple fields, especially when evaluating samples as a whole rather than examining multiple packets independently.

For each machine learning algorithm, 10,000 samples were used per class, with a standard k value of 5 — representing the number of groups into which data points were split. The CNN method included both training and testing phases, where data was randomly divided during each analysis. To minimize random error and align with the other ML techniques, the CNN analysis was repeated five times, resetting the model after each run, and the accuracy scores were averaged across all repetitions. This approach ensured consistency with the k value used in cross-validation.

During CNN model training, the training dataset was loaded in randomly selected batches until all data had been processed. This randomization introduced slight variation into each repetition, reinforcing the robustness of the results.

The decision not to use normalized data, as opposed to non-normalized data, is supported by the results shown in Table 5.12, which indicate a slight decrease in accuracy with normalization. While it is standard practice to normalize data as part of

TABLE 5.12: Testing Normalisation.

Result / Sample	**Normalised**	**UnNormalised**
500 x 50	0.968497	0.989498
400 x 50	0.988784	0.989284
300 x 50	0.988499	0.989213
200 x 50	0.989571	0.987570
100 x 50	0.985498	0.986998

the preparation for convolutional neural networks (CNN), a comparative run of the experiment using both normalized and non-normalized data revealed that normalization resulted in marginally lower accuracy, as reflected in Table 5.12. This may be attributed to the nature of the selected fields for testing—three of which were binary, taking values of either zero or a specific number, while the fourth had a limited range of values.

The confusion matrices presented in Tables 5.8, 5.9, 5.10 and 5.11 illustrate the number of correct and incorrect predictions, as well as how the misclassification were distributed across the classes. The results show minimal misclassification for SVM, Decision Tree (DT), and Random Forest (RF), aligning with the accuracy scores observed in Table 5.3. Notably, the CNN analysis reveals a pattern: when AODV is misclassified, it tends to be labelled as OLSR, while OLSR, when misclassified, is more likely to be classified as DSR. The proportions suggest that CNN perceives OLSR as being more similar to DSR, with AODV appearing further removed from both.

The reducing aperture sampling method explored in the research aimed to decrease sample sizes while repeating the SVM, RF, DT, and CNN analyses. This approach was applied to both the two-class (AODV and OLSR) and three-class (AODV, DSR, and OLSR) experiments. Field selections were guided by the combinations that produced the highest accuracy, as outlined in Table 5.3.

The findings highlight that for the two-class analysis, SVM, DT, and RF consistently achieved the highest accuracy, effectively distinguishing between AODV and OLSR. However, when a third class was introduced, RF maintained its accuracy, whereas SVM's performance showed a slight decline. In contrast, the CNN performed reasonably well with two classes but experienced a significant drop in accuracy when a third class was added. An edge case emerged when using either a minimal sample of five nodes or five packets — under these conditions, the CNN model failed, as the convolutional kernel was unable to effectively traverse the limited sample size.

5.6 Research Question 5:

This experiment aims to reflect real-world conditions, acknowledging that it's often impractical to capture all traffic transmitted to and from every node in a network. This mirrors a scenario where an Ad-Hoc Mesh network is spread across a large area, making it impossible for an observer to gather data from every point in the network. The analysis shows that the same level of accuracy can be achieved using data from just five nodes — whether connected or unconnected — and any five consecutive packets, with both SVM and RF performing reliably under these conditions.

To address RQ5, the experiment focuses on using smaller feature vectors to account for situations where fewer packets per node are collected, fewer nodes are observed, or both. The packets and nodes are reduced using the same dataset applied to RQ3 and RQ4.

Various combinations of node count and packets per node are tested by iteratively reducing both. The first step involves determining the number of packets, "X," to remove at each iteration. The last X packets generated by a node are discarded — not randomly, but consecutively — to better simulate real-world wireless networks, where packet losses typically happen in bursts, as described by Da Silva and Pedroso (2019).

Similarly, the number of nodes considered is reduced iteratively, with "Y" nodes randomly removed at each step. The same methodology used in RQ4 is applied here to assess how detection accuracy changes when fewer packets are collected or when data comes from a smaller subset of nodes.

Chapter 6

Phase 3: Influential Node Detection in Wireless Sensor Networks: A Temporal and Adversarial Perspective

The concept of influential nodes is a key research area, with applications spanning technological and biological networks, as demonstrated in works like Zhang et al. (2013) Buldyrev et al. (2010), Fath et al. (2007). This research phase focuses on Wireless Sensor Networks (WSNs), specifically scenarios where sensors collect data, and nodes relay this data to sink nodes for processing, as discussed in Jain and Reddy (2013).

Adversaries may attempt to exploit network communications by eavesdropping and analysing content. Identifying influential nodes in advance could enable them to target these nodes for maximum disruption. Conversely, network operators could use such insights to optimise network performance by mitigating resource bottlenecks at these critical nodes.

6.1 Influential Node Detection Problem Definitions

The problem of identifying influential nodes within complex networks is crucial for addressing challenges such as:

- **Adversarial Attack on Ad Hoc Mesh Network** Unmanned Aerial Vehicles (UAVs) often form cooperative networks to accomplish complex missions, as detailed in Lopez et al. (2021). Adversaries may attempt to intercept

communications, as suggested by Thandava Meganathan and Palanichamy (2015). A more effective strategy might be targeting aggregation points—nodes central to data flow—to disrupt operations or boost data interception.

- **Sensor Network Data Aggregation Points** WSNs comprise independent nodes that gather environmental data and relay it to aggregation points for processing. In hazardous environments, static sensors may not suffice, requiring mobile sensors to ensure adequate coverage, as discussed in Howard et al. (2002). Predicting nodes with higher data throughput can help optimise network performance.
- **Epidemiology** Influential spreaders of disease, as studied by Malliaros et al. (2016) and Sun et al. (2018), are analogous to facilitator nodes in networks. Targeting these nodes in UAV mesh networks could isolate significant portions of the network, disrupting data flow.

6.2 Facilitators within Ad-Hoc Mesh Networks

This research focuses on influential nodes, referred to as facilitators in this context. Facilitators are critical nodes through which the majority of data flows to reach the sink node. When a single node or a small group of nodes becomes the primary conduit for data flow, it can lead to resource strain on these nodes, potentially disrupting the network.

Similarly, military operations rely on mobile ad-hoc mesh networks to maintain communication as personnel move through the field. These networks share the dynamic and decentralised nature of the wireless sensor networks (WSNs) studied here.

This research develops a method to identify facilitator nodes by determining which nodes hold the greatest influence within the network. Identifying these influential nodes is crucial because their high level of traffic makes them more prone to resource depletion, posing operational challenges for the WSN. For attackers, pinpointing these nodes offers strategic advantages, such as capturing facilitator nodes to alter the network's data flow or enhancing content interception by predicting their location.

The challenge of identifying facilitator nodes becomes even more complex in networks with mobile structures. In such scenarios, as nodes shift to new positions, existing data links break and new ones form, altering the pathways to the sink node. Consequently, the nodes deemed most significant at the beginning of a deployment may lose their influence as the network evolves, underscoring the dynamic nature of facilitator nodes over time.

The questions that this phase of research aims to answer are:

- **Research Question 6**: When considering a dynamic ad-hoc mesh network with a temporal dependent, what type of link prediction is the most effective?
- **Research Question 7**: Considering how a node's influence changes over time. How accurate is the nodal influence generated from the predicted data compared with those calculated using the ground truth?
- **Research Question 8**: To mimic the real world, several mobility models will be used to guide individual node's movements during the simulations. Does the mobility model that generates the data impact predicting nodal influencer ranking?

Wireless Sensor Networks (WSNs) can exist in either static or dynamic forms. A static WSN, commonly seen in the utility industry, involves networks like smart meters that connect household devices and forward readings to central collection points (Parvin (2019)). These networks typically have a central controller, and data is distributed across the network to all nodes. However, as noted by Kariuki (2019), such environments face significant security challenges, particularly regarding attacks on the network.

Dynamic WSNs, on the other hand, often deploy mobile sensors to address scenarios where static sensors cannot be optimally placed, such as in hazardous or toxic conditions (Howard et al. (2002)). Mobile sensors improve coverage by adjusting their positions based on the environmental requirements.

A study by Song et al. (2016) explores tracking influential nodes in social networks, where users constantly add and remove connections, altering the impact of individual nodes. This process of ”friending” and ”unfriending” mirrors the formation and dissolution of data links in ad-hoc mesh networks. The study introduces the Upper Bound Interchange Greedy (UBI) method, an extension of influence maximisation. This approach identifies a set of ”seed” nodes in a social network to maximise their influence on others. The research utilised data from three real-world social networks: a mobile phone network where calls form edges between users, and two citation networks, HepPh and HepTh, where edges represent paper citations.

While the paper effectively examines influential node tracking, its analysis relies heavily on the UBI methodology and focuses on previously identified seed sets to evaluate nodal influence. Furthermore, it uses a limited number of networks as the basis for its conclusions.

In contrast, this phase of research employs centrality metrics to predict influential nodes using traffic data from prior activity. The methodology involves data gathered

from Monte Carlo experiments across 2,100 simulations. Unlike the Song et al. (2016). study, this research leverages scenarios generated by three distinct mobility models to provide a more robust and diverse analysis.

6.3 Methodology

Taking the basic premise set out in section 2.10.10 for link prediction, section 2.10 for the centrality and the additional information need to replicate the experiments are setout as follows:

6.3.1 Link prediction

The first set of parameters describe are those involve in the link prediction methodologies. These are based on the papers Wang et al. (2020) for Digital Graph Library and Rossi et al. (2020) for Temporal Graph Networks.

6.3.1.1 Weighted GCNLayer

For the methodology implementing the Weighted GCNLayer, a Graph Convolutional Network model was created consisting of two Deep Graph Library layers, the Deep Graph Library library **GCNLayers**, each GCNLayer performs message passing on all the nodes then applies a fully-connected layer this can be found in the documentation that can be found DGL. For this experiment the links were weighted following the weighting set out in the Table 2.1

From the links that represent the connections between nodes within the our network, 25% were removed and used as the test while the remainder was used to train the model. New links were calculated using the DGL built in 'DotPredictor' function.

6.3.1.2 Non Weighted and Weighted GraphSAGE

Similar to the Weighted GCNLayer description, the methodology implementing the Non Weighted and Weighted GraghSage, a Graph Convolutional Network model was created consisting of two GraphSAGE layers, the Deep Graph Library library **SAGEConv** was used for these layers, each computes new node representations by averaging neighbour information as set out in the documentation that can be found DGL. Two experiments were run in this section, the first where all links were treated as having the same weight and the second following the weighting set out in the Table 2.1. And as for the GCNLayer, 25% of the links were used for the test set and the

remainder was used to train the model. New links were calculated using the DGL builtin 'DotPredictor' function.

6.3.1.3 Weighted TGN

The weighted TGN was generated using the methodology set out in the paper Rossi et al. (2020). During the investigation is was found that the use of the memory of a node updated after an event was not necessary so was therefore not included in the experiment.

The model hyper-parameters as set out in the above paper, the specific parameters that are used during this experiment were as follows:

- **BATCH_SIZE** = 200 - The is the size of the batch for the CNN.
- **NUM_NEIGHBORS** = 10 - The number of neighbours to sample during each iteration.
- **NUM_EPOCH** = 50 - The number of epochs for each run.
- **NUM_HEADS** = 2 - The number of heads used in attention layer.
- **DROP_OUT** = 0.1 - The dropout probability for the CNN.
- **UM_LAYER** = 1 - The number of network layers for the CNN.
- **NODE_DIM** = 100 - The dimensions of the node embedding.
- **TIME_DIM** = 100 - The dimensions of the time embedding.
- **MESSAGE_DIM** = 100 - The dimensions of the messages.

6.3.2 Centrality

The centrality measure used within this phases utilises the Networkx suite of software, based on the paper Hagberg et al. (2008). Where applicable all defaults were used.

6.3.3 Similarity Measures

As referenced in section 2.11, the similarity measures are all programmatically calculated for Euclidean Distance, Manhattan Distance and Cosine Similarity using the equations set out in the referenced section.

6.4 Centrality Measures for Determining Influential Nodes in Ad-Hoc Mesh Networks

Identifying influential nodes using centrality measures is a well-established concept. Chen et al. (2012) suggests that betweenness and closeness centrality provide more accurate results, though they come with higher computational complexity. In contrast, Eigenvector centrality is considered less effective (Tulu et al. (2018a)). Algorithms such as LeaderRank (Lü et al. (2011)) and PageRank (BrinS (1998)) have been employed to identify influential nodes, with newer methods like ANiceRank (Yao and Ji (2019)) enhancing LeaderRank by factoring in the personal attributes of the nodes.

The concept of temporal networks is discussed by Kim and Anderson (2012), who propose that using temporal slices becomes problematic when the network topology changes too rapidly. This implies that node centrality may also shift at the same pace. Our research found that node centrality tends to stabilise toward the end of the simulation, and predictions of highly influential nodes remain consistent with the proposed methodology. Additionally, a recent framework by Rossi et al. (2020) introduced Temporal Graph Networks, which apply to continuous-time dynamic graphs.

In our study, we investigate the application of network centrality metrics to analyze data. Similar techniques are seen in Jain and Reddy (2013), where optimising the placement of sink nodes improved the quality of service (QoS) metrics for video data streams in the network. Their findings show that higher centrality correlates with increased network traffic through a node, highlighting its importance. As the data transfer on these nodes grows, their resources are consumed more rapidly, supporting the problem defined in Section 6.1.

This research emphasises the effectiveness of using QoS as a measure of influence. It also highlights the importance of considering a node's spatial placement and proximity to other nodes. Our findings reaffirm that betweenness and closeness centrality are superior indicators of influence, while bridging centrality provides valuable insights into potential weaknesses in the overall network structure.

6.5 Research Question 6:

To answer this question, four types of predictive methodologies described in 2.10.10 were used to analyse the full data set of 2100 network simulations, consisting of 700 simulations for each of the mobility models described in 2.4. Each simulation lasted for 30 minutes/1800 seconds. During the data generation stage, link weighting was

applied to three of the four techniques. Determining the types of packets that the connection traffic was made of was crucial for establishing the accuracy of the results.

The predictive efficiency is determined by comparing the links predicted by a specific link prediction method to the actual links in the ground truth data. The process involves training the model on the first 70% of the data (ordered by timestamp), validating it on the next 15%, and testing the link prediction on the final 15%. This last segment corresponds to 9 minutes (540 seconds) of link data.

Efficiency is calculated by matching the links predicted at each timestamp with the corresponding links in the ground truth, providing a measure of the method's accuracy.

TABLE 6.1: Link Prediction

Link Prediction	**Efficiency**
Non weighted GraphSage	0.619
Weighted GraphSage	0.7319
Weighted GCNLayer	0.8007
Weighted TGN	0.9036

The efficiency of the different predictive methodologies is tabulated in Table 6.1. It can be seen that three of the techniques that do not take into account the temporal the aspect of the data gave a lower score of 10%.

6.6 Research Question 7:

Taking the technique that gave the highest score in Section 6.5, a new dataset was generated for every simulation run. This second dataset consisted of the same data as the original simulated data, but with the last 15% of the data being replaced by predicted link data.

For each centrality measure, the four centralities: degree 2.10.2, closeness 2.10.3, betweeness 2.10.4, and bridging 2.10.7, were calculated on the original simulated data and this new hybrid data. This gave two time series of centrality.

As the simulation generates temporal-dependant data, the centralities are calculated in temporal slices. Two arbitrary values were used for the analysis: 200 packets per time-slices, and 3 seconds for packet 'expiration'. The time slice consisted of 200 packets, as these values gave enough data for the centrality measure to produce representative granularity.

The packet 'expiration' was the length of time after which the packet was removed from the simulation data. This built a mapping of the centralities over time, allowing for a visual comparison between the original data and the predictive data.

TABLE 6.2: Similarity Results

	Similarity Measures
Deg. Euclidean	3.2127
Deg. Manhattan	27.8883
Deg. Cosine	0.9696
Bet. Euclidean	0.1864
Bet. Manhattan	1.4979
Bet. Cosine	0.9910
Close. Euclidean	4.5931
Close. Manhattan	39.9619
Close. Cosine	0.9721
Bridg. Euclidean	0.0151
Bridg. Manhattan	0.1134
Bridg. Cosine	0.9959

Graphs were generated showing each centrality measure against the time series for the original data with the predicted data and a visual comparison was undertaken. This however proved to be inconclusive as the pairs of graphs were too similar. Therefore the next steps used to employ the three similarity measures set out in 2.11 were calculated and the results tabulated in Table 6.2.

6.7 Research Question 8:

TABLE 6.3: Mobility Results

	Column	**Probabilistic**	**SMOOTH**
Deg. Cosine	0.9677	0.9717	0.9718
Bet. Cosine	0.9911	0.9908	0.9909
Close. Cosine	0.9699	0.9746	0.9745
Bridg. Cosine	0.9958	0.9960	0.9962

To answer this question, the same methodology is employed as described in Section 6.7, but with more granularity. Each mobility model set out in Section 2.4 is treated as an individual class. As in Section 6.6, the Cosine Similarity measure and centrality measure divisions were repeated, and the results were tabulated in Table 6.3.

Chapter 7

Discussion

This research demonstrates how information unintentionally “leaked” from an Ad Hoc Mesh Network can be exploited by an adversary. It spans three phases, covering initial network setup, ongoing operations, and predictions of future interactions. Each phase highlights methods to extract actionable intelligence from such networks.

7.1 Overview of Machine Learning Models

Before presenting results, it is important to understand the relative strengths and weaknesses of the machine learning models used in this study. This helps contextualise the observed performance differences across tasks.

TABLE 7.1: Strengths and Weaknesses of ML Models

Model	**Strengths**	**Weaknesses**
Support Vector Machine (SVM)	Handles high-dimensional data well; effective with small to medium datasets; kernel trick allows modelling of non-linear patterns.	Requires careful parameter tuning; less scalable to large datasets; limited interpretability.
Random Forest (RF)	Handles non-linearity and feature interactions well; reduces overfitting via ensembling; outputs feature importance.	Less interpretable; can be biased toward features with more levels.
Decision Tree (DT)	Interpretable; fast training and inference; can handle mixed feature types.	Prone to overfitting; instability due to high variance.
k-Nearest Neighbors (k-NN)	Simple to implement; non-parametric; no training phase.	Poor scalability; sensitive to irrelevant features and data scaling.
Naive Bayes	Fast and efficient; performs well on high-dimensional sparse data; easy to implement.	Assumes feature independence, which is rarely true; less accurate on complex relationships.
Convolutional Neural Network (CNN)	Excellent for grid-structured data (e.g., images); captures spatial hierarchies; automatically learns relevant features.	Requires large labelled datasets; computationally intensive; less interpretable.
Deep Forest (gcForest)	Capable of deep representation learning without backpropagation; less data-hungry than deep neural networks; interpretable structure.	Deep Forest implementations such as deep-forest remain relatively niche, with fewer contributors and limited integration into major ML pipelines.

7.2 Phase 1: Identifying Node Placement Bias

In the first phase of this thesis, a methodology was presented to characterise the bias introduced by the mechanism used to generate node placements within a test area. This software, called TG, generates node placements based on predefined network models or real-world measurements and the adjustable parameters that influence the

process. The experimentation in this phase focused on three common generators: NPART, BRITE, and GT-ITM. The first task was to use visual inspection of the generated topologies. This could not identify differences between the topologies, necessitating a systematic method to distinguish between them.

To analyse the generated topologies, spatial placement data was extracted for all three generators. Initial analysis used silhouette clustering on unlabeled data to estimate the number of natural "classes" or generators present. This class-agnostic algorithm suggested an optimal division into two classes, despite the actual number being three, indicating that the data lacked clear separation into distinct groups.

K-means clustering was applied to the data; this identified three clusters that aligned with the actual number of generators. This achieved an accuracy of 67%. Next, a probabilistic algorithm, Naive Bayes, was applied to the data and due to its simplicity, increasing the accuracy to 72%. This was further improved by applying sequential feature selection (SFS)—a method that optimises combinations of spatial placement measurements—the accuracy rose to 78.6%.

To generate the accuracy, the data was split into training and testing sets. The model was then trained on one portion and was then evaluated on the remaining test data to ensure an unbiased assessment of its performance. A examination of improperly classified samples revealed that errors were not evenly distributed among the three classifications. NPART and GT-ITM shared the faults evenly, however the BRITE class was continuously correctly classified. To enhance the outcomes, pairwise comparisons between the classes were conducted.

This demonstrated that BRITE could be 100% accurately differentiated from NPART and GT-ITM. However, distinguishing between NPART and GT-ITM achieved only 64% accuracy, which improved to 77% when using SFS.

SFS was again applied to identify the most informative features for classification. Among the many extracted features, the "Clustering coefficient with a radius of 20 units" was most effective in differentiating BRITE from NPART and GT-ITM. For distinguishing between NPART and GT-ITM, the "Inter-node distance feature – Mode of Euclidean distance" proved most informative.

7.2.1 Research Question 1

Research Question 1 (RQ1) addresses the question, "How can we characterise the bias introduced by using a specific TG instead of considering all TGs collectively?" To explore this, a "Bias Index" was proposed to quantify the differences between features observed in a single topology and the same features across the entire population of topologies.

The magnitude of these differences provided an objective measure of bias. Meanwhile, the variance in these differences revealed how distinct the TGs are, enabling the differentiation of topologies generated by different TGs based on specific characteristics, regardless of the overall extent of the differences.

7.2.2 Research Question 2

To answer Research Question 2 (RQ2), "How can we select the TG or TGs to minimise bias?". The ideal solution would be to use a combination of all three TGs, as this would provide the greatest diversity. However, this may not always be practical in all experiments. Therefore, if the choice is limited to two TGs, the combination that gave the lowest *Hedges' g* value is NPART and GT-ITM, indicating minimal bias. If only a single TG can be used, NPART is the preferred option, as it also produces the lowest *Hedges' g* in such cases.

7.2.3 Results and Observations

In this phase, the models were tasked with distinguishing between different node placement strategies — specifically, uniform grid layouts, random deployments, and clustered formations. The dataset included statistical descriptors of node degree, edge length distribution, clustering coefficients, and topological centrality measures. These features were extracted from synthetic networks generated under each strategy.

Table 7.2 presents the classification accuracy of each machine learning model.

TABLE 7.2: Node Placement Strategy Classification Accuracy

Model	**Accuracy (%)**
k-Nearest Neighbors (k-NN)	66.63
Mean Shift (MS)	33.13
Agglomerative Clustering (AC)	33.9
Spectral	34.05
Gaussian Naive Bayes	77.95

7.2.4 Justification of Model Selection

In Phase 1, the core objective was to classify the topology generator (TG) responsible for a given network topology based on spatial and structural features. This is essentially a supervised multiclass classification task on moderately dimensional, structured data. To this end, several machine learning models were selected to balance interpretability, computational efficiency, and classification accuracy.

Gaussian Naive Bayes (GNB) was chosen as the primary model due to its suitability for continuous-valued features and its assumption of feature independence, which holds reasonably well in the context of extracted network features such as inter-node distance, clustering coefficient, and spatial distribution. GNB provided a baseline with strong performance and low training overhead, making it a reliable and interpretable first-pass classifier.

In parallel, **k-Nearest Neighbors (k-NN)** was explored due to its simplicity and effectiveness in spatially driven classification problems. However, its sensitivity to feature scaling and the curse of dimensionality made it less robust than GNB in the presence of noisy or overlapping features across TGs.

For unsupervised analysis, clustering methods including **MeanShift**, **Agglomerative Clustering**, and **Spectral Clustering** were used to assess whether natural groupings existed among the topology features without relying on label supervision. This helped validate the underlying assumption that TGs produce distinct, learnable patterns in spatial distribution. These methods were particularly helpful in determining feature discriminability before committing to a supervised classification pipeline.

Ultimately, Gaussian Naive Bayes was retained for its combination of performance (as set out in Table 4.12, 78% classification accuracy for Gaussian Naive Bayes) and low complexity, while the clustering models informed the feature engineering and bias analysis pipeline. These decisions ensured that the methodology remained lightweight and reproducible while offering actionable insights for topology generator bias assessment.

7.2.5 Analysis

Gaussian Naive Bayes achieved the highest classification accuracy in this task, followed by k-Nearest Neighbors. Unsupervised clustering approaches (Mean Shift, Agglomerative, and Spectral Clustering) performed near chance level, highlighting the difficulty of separating node placement biases without explicit labels.

Gaussian Naive Bayes (GNB): The strong performance of GNB suggests that the features selected for this task (e.g., node degree distribution, clustering coefficient, and centrality) provided enough signal for statistical separation across placement types. Despite its simplifying assumption of feature independence, GNB benefited from the fact that different placement strategies lead to distinct distributions of local graph statistics, particularly in cases of tightly clustered vs. uniformly spread nodes.

k-Nearest Neighbors (k-NN): k-NN performed moderately well. Its effectiveness stemmed from capturing local similarity patterns in feature space, especially when certain topological metrics clustered tightly for specific node layouts. However, its performance likely suffered due to high-dimensional feature space and the curse of dimensionality. Feature scaling and the selection of distance metrics (e.g., Euclidean) would have further influenced the results.

Unsupervised Clustering (MS, AC, Spectral): These models performed poorly, with classification accuracy near random guessing. This reflects two primary challenges: (1) the features may not cluster cleanly in the high-dimensional space without supervision, and (2) unsupervised algorithms lack contextual understanding of class labels, which reduces their ability to discover structure tied to specific deployment strategies. Spectral Clustering, though more flexible than others, likely failed to identify meaningful graph partitions without clearer cluster boundaries in feature space.

7.2.6 Summary

This phase confirms that node placement strategies do manifest distinguishable structural characteristics in network topologies — but only when leveraged by models with strong probabilistic foundations (e.g., GNB) or local similarity metrics (e.g., k-NN). The poor performance of unsupervised methods emphasises the importance of labelled data and domain-informed feature engineering. These findings suggest that, although implicit, node placement leaves a detectable fingerprint, and that supervised learning is better suited to uncover these patterns than generic clustering.

7.3 Phase 2: Identifying Routing Algorithms

Building on the node placement analysis, the second phase examined routing algorithms. It focused on determining the routing algorithm used in an ad-hoc mesh network by analysing network traffic characteristics. Several assumptions were made about the network under investigation: it was uncooperative (the network did not voluntarily provide information about its routing algorithm), its traffic was encrypted (packet data could not be intercepted directly), and all data collection had to be passive—no active querying or participation in the network was allowed.

7.3.1 Research Question 3

RQ3 investigates: "What is the most accurate machine learning (ML) approach for detecting the routing algorithm in an ad-hoc mesh network?"

The process started by selecting data packet fields relevant to the routing mechanism. Irrelevant fields were eliminated, and combinations of relevant fields were tested with different ML algorithms to determine the most effective setup for accuracy. Table 5.2 summarises the results.

By holding the field combination constant and testing various algorithms, it was found that Support Vector Machine (SVM), Decision Tree (DT) and Random Forest (RF) achieved the highest accuracy at 99.99%–100%, while Convolutional Neural Networks (CNN) scored 98.95%. Other algorithms, such as Naive Bayes (95.79%) and Deep Forest (91.9%), performed slightly worse. Sequential testing revealed that SVM and RF were the most accurate, while other models still demonstrated strong performance.

7.3.2 Research Question 4

RQ4 explores: "How does detection accuracy change when analysing routing algorithms of the same class?"

Algorithms in the same class, such as AODV and DSR (Reactive Routing), share similar mechanisms for determining routes. This raised the question of whether they could still be distinguished, especially when mixed with data from other classes (e.g., Proactive Routing).

The analysis utilised the top-performing algorithms (SVM, RF, DT and CNN) from the previous question. For a 2-class comparison, SVM achieved perfect accuracy (100%), RF and DT scored 99.99%, and CNN reached 98.95%. However, in a 3-class scenario, DT retained the highest accuracy at 99.7%, with RF slightly lower at 99.6%, while SVM and CNN dropped to 98% and 81.2%, respectively. DT experienced the smallest performance drop (0.2%), demonstrating its robustness across both 2-class and 3-class analyses.

Further analysis (Table 5.2) highlighted the impact of field combinations. For CNN, accuracy decreased as more fields were added, with a 3-field combination achieving 81.2% accuracy compared to 91.52% with a single field. Conversely, SVM, DT and RF performed better with multiple fields, with SVM achieving 99.97% accuracy using three fields compared to 88% with one.

This discrepancy arises because CNN analyses interactions across multiple packets and fields, while SVM averages values from packets into a single data point. As a

result, CNN excels with single-field analysis over multiple packets, whereas SVM and RF benefit from multiple fields analysed per packet. Overall, RF emerged as the most reliable for distinguishing routing algorithms, particularly when transitioning from 2 to 3 classes.

7.3.3 Research Question 5

RQ5 examines: "How does detection accuracy change when reducing either the number of nodes sampled or the amount of data collected per node?"

For a 2-class analysis, SVM achieved the highest accuracy at 100%, while RF and DT followed closely at 99.99%. However, with reduced samples (e.g., 5 nodes and 5 consecutive packets), DT and RF slightly outperformed SVM, scoring 99.1% versus 99%. In a 3-class analysis, DT maintained its accuracy of 99.1% , RF dropped to 98.9% and SVM to 96.5%.

DT demonstrated no accuracy decline under reduced sampling conditions, indicating greater stability. SVM, DT and RF excelled when analysing algorithms from different classes, as their network traffic characteristics were distinct. However, when a third class was introduced, SVM's effectiveness diminished slightly, likely due to the similarity between classes diluting its classification ability.

This analysis suggests that SVM, DT and RF can achieve high accuracy even with minimal data—sampling any 5 nodes and 5 consecutive packets yields reliable results, regardless of whether the nodes are connected or independent.

The use of CNN for routing algorithm classification, rather than its typical application in image recognition, revealed intriguing behaviour. While CNN matched SVM's accuracy in 2-class scenarios, it struggled with 3-class problems, likely due to difficulties combining information from different channels. Further research is required to understand these limitations.

From Table 5.4, CNN's performance also varied significantly depending on the field combinations, excelling with single-field analysis but lagging with multiple fields. In contrast, SVM, DT and RF consistently benefited from analysing multiple fields together, further underscoring their suitability for scenarios involving complex, multi-class data.

Among the machine learning approaches tested, RF consistently offered the most robust performance across different conditions, making it the best choice for detecting routing algorithms in diverse scenarios.

7.3.4 Results

The classification accuracies of the four models on the routing protocol identification task are presented in Table 7.3.

TABLE 7.3: Routing Protocol Identification Accuracy

Model	Accuracy
SVM	99.0%
RF	99.8%
DT	99.8%
CNN	96.0%

7.3.5 Justification of Model Selection

In Phase 2, the research aimed to classify the routing algorithm used by an ad hoc mesh network based on passively observed packet metadata. This presented a multi-class classification problem over structured but noisy data, with a strong emphasis on maintaining high accuracy under conditions of limited data availability and encryption. Consequently, multiple machine learning models were selected based on their robustness, ability to generalise from small samples, and performance under dimensional constraints.

Support Vector Machines (SVM) were employed as a principal classifier due to their effectiveness in high-dimensional feature spaces and their strong theoretical foundations for binary and multi-class classification. SVMs are particularly well-suited for sparse datasets where only limited features can be derived from encrypted packet headers. Their use in conjunction with K-means clustering also facilitated semi-supervised exploration of latent structure in the dataset.

This classifier achieved very high accuracy (99.0%) in full-data settings, benefiting from its ability to create optimal decision boundaries in high-dimensional spaces. However, SVM performance was slightly less stable when the dataset was constrained to fewer packets or nodes, likely due to its sensitivity to the availability of support vectors and class boundary clarity in smaller sample sizes.

Decision Trees were included for their simplicity, speed, and high interpretability. Given that packet features are often hierarchical or categorical in nature (e.g., protocol type, TTL ranges), Decision Trees provided a natural fit for modelling rule-based distinctions between routing protocols.

This classifier performed strongly, achieving accuracy scores nearly equivalent to RF, especially under reduced-data conditions. Its interpretable rule-based structure was

well suited to the categorical and numeric packet fields, such as 'wlan.fc.subtype', 'ip.hdr len', and 'udp.length'.

Random Forests, as an ensemble of Decision Trees, were leveraged to reduce overfitting and improve generalisation. They consistently achieved high accuracy across a range of test conditions and proved resilient when the dataset was constrained to a small number of packets per node—a critical factor for adversarial scenarios.

This classifier achieved the highest accuracy across all experimental conditions, including multi-class classification and reduced-node/packet sampling. Its ensemble approach, which aggregates the outputs of multiple decision trees, allowed it to capture complex non-linear relationships between packet-level features while maintaining resilience to noise and overfitting. This made RF particularly effective when the feature space was shallow but structured, as is typical in encrypted metadata scenarios.

Convolutional Neural Networks (CNNs) were tested to explore whether spatial or sequential patterns within packet metadata could be exploited. While CNNs performed well on larger input sizes, their performance degraded with reduced data, and the computational complexity was not justified in all cases. However, they provided useful contrast during comparative evaluation.

This classifier achieved the lowest accuracy of the evaluated models. While CNNs are powerful for extracting spatial or sequential patterns, they typically require large, rich datasets to learn effective representations. In this context, the limited number of fields per packet and the abstract nature of the data made CNNs less effective compared to tree-based and kernel methods.

The selected models were benchmarked across various combinations of routing protocols (e.g., AODV, OLSR, DSR) and under progressive data reduction scenarios. This allowed for a systematic evaluation of each classifier's tolerance to limited visibility—an essential consideration for realistic passive reconnaissance.

In summary, the superior performance of Random Forest and Decision Tree models can be attributed to their capacity to handle structured yet limited features robustly, making them particularly well-suited for passive traffic analysis under constrained observation conditions.

7.3.6 Analysis

SVM, Random Forest and Decision Tree outperform CNN in this task. This can be attributed to their ability to model complex and non-linear patterns in the input

features. Routing protocol behaviours, especially in encrypted or aggregate contexts, often involve nuanced and non-linear characteristics.

- **SVM** excels due to its kernel trick, allowing it to find optimal hyperplanes in transformed feature spaces. It is particularly effective for non-linearly separable data and benefits from its margin-maximising objective.
- **Random Forest** leverages ensemble learning to average across multiple decision trees, reducing variance and enhancing generalisation. It captures feature interactions well and is robust against overfitting.
- **Decision Tree** overfits the training data more easily, especially in the absence of pruning or regularisation, limiting its performance on unseen data.
- **CNN** are typically effective when data exhibits spatial or temporal structure, as in images or sequences. However, in this experiment, the features are likely not spatially correlated in a way that CNNs can exploit. The curse of dimensionality is less of an issue for CNNs due to parameter sharing, but without meaningful local structure, their inductive bias becomes a limitation rather than a strength.

The SVM model achieved higher accuracy in routing algorithm classification due to the specific nature of the feature space derived from packet metadata. The packet headers used (e.g., TTL, source/destination, hop count) form a sparse but well-structured vector space. SVMs are particularly effective in such conditions because they construct optimal hyperplanes in high-dimensional space, maximising margin between class boundaries. This proved advantageous when distinguishing between protocols with subtle behavioural differences (e.g., OLSR vs AODV).

In contrast, Decision Trees and Random Forests are sensitive to feature splits and can suffer from overfitting in small or noisy datasets, especially when features have overlapping value ranges across classes. The CNN performed well on larger datasets but was less stable when using reduced-aperture data, where SVM retained consistent performance due to its robustness with small sample sizes.

7.3.7 Normalisation

It is standard practice to normalise data as part of preparing it for a convolutional neural network (CNN). However, a test comparing normalised and non-normalised data revealed that normalisation resulted in slightly lower accuracy, as shown in Table 5.12. This outcome may be due to the nature of the chosen fields, three of which were binary (taking values of either zero or a specific value), while the remaining field had a limited range of values.

The confusion matrices in Tables 5.8, 5.9, 5.10 and 5.11 provide a detailed breakdown of the predictions, showing both correct and incorrect classifications and how misclassification occurred. In the cases of SVM, Decision Tree and Random Forest, misclassification were minimal, consistent with the results in Table 5.3. For CNN, the analysis showed a distinct pattern: when AODV was misclassified, it was more often categorised as OLSR rather than DSR. Conversely, when OLSR was misclassified, it was usually labelled as DSR.

From these proportions, it can be inferred that CNN perceives OLSR as being closer to DSR, with AODV appearing more distinct and further away in its classification framework.

7.4 Phase 3: Identifying and Forecasting Influential Nodes in a Wireless Sensor Network

The final phase of the research focused on determining whether it was feasible to identify influential nodes in a wireless sensor network and predict their future behaviour. The study compared various link prediction methods and evaluated their accuracy based on how well their predictions matched the actual data. The findings revealed that the most accurate prediction method relied on timing.

Influential nodes in the network were defined as those with the highest centralities, serving as a proxy for their level of influence. The centralities considered in the analysis were degree, betweenness, closeness, and bridging. Degree centrality measures the number of direct connections a node has. While this provides some insight, it is not a precise indicator of influence because it focuses on quantity rather than the quality of connections.

Closeness centrality indicates how near a node is to all other nodes in the network. Nodes with high closeness centrality are important because they can quickly share information with other nodes in the network.

Betweenness centrality measures how often a node appears on the shortest paths between pairs of nodes. A node with high betweenness centrality is more influential as it plays a crucial role in data flow. Bridging centrality evaluates the importance of nodes in connecting distinct groups within the network. It is also useful for identifying critical nodes that may disrupt information flow in the network.

The analysis involved comparing centrality values derived from the original data with those from predicted link data. The first step was to track how centrality measures changed over time and compare the results. Two key observations emerged:

- Visually distinguishing between the data sets was challenging.
- The ranking of nodes by centrality stabilised over time, with higher-ranking nodes maintaining their status as the simulation progressed.

7.4.1 Research Question 6

RQ6 asks, "Which is the best technique for link prediction in a wireless network?" The research concluded that Temporal Graph Networks (TGNs) were the most effective approach, yielding prediction 10% higher than non-temporal techniques. This finding makes sense because the data from the simulation of a time-dependent ad-hoc mesh network naturally changes over time, and the temporal nature of the data provides relevant, dynamic information. Methods that do not account for this temporal aspect are missing vital details, reducing their effectiveness.

7.4.2 Research Question 7

RQ7 addresses the question, "How accurately can the predicted nodal influence be compared with ground truth values over time?" Visual comparisons of predicted and actual data were inconclusive. Therefore, we applied similarity measures like Euclidean Distance, Manhattan Distance, and Cosine Similarity. Euclidean and Manhattan distances were less effective at distinguishing between different mobility models because their measurements depend on the units they use.

On the other hand, Cosine Similarity proved more informative, providing a scale from 0 (completely dissimilar) to 1 (completely similar). The research demonstrated that Cosine Similarity yielded average values of 0.98 across the four centralities, suggesting that the predicted link data closely matched the original data.

7.4.3 Research Question 8

RQ8 asks, "Does the mobility model used to generate data affect the prediction of nodal influencer rankings?" The study used three mobility models that accounted for random dependencies on temporal or spatial aspects, raising the question of whether these models impact the prediction of nodal influence.

In general, the mobility model had minimal impact on the results. The degree of influence was only marginally affected—by about 0.01%—for the Degree and Closeness centrality measures when using Cosine Similarity. This suggests that the link prediction results were largely unaffected by the mobility model. The slight reduction in accuracy could be due to the column mobility model, which involves

spatial dependency and causes nodes to travel as a group. This model likely reduced the amount of new connection data available for individual nodes, thereby slightly influencing the prediction. This suggested that future work on this phase could include empirical curves showing the average number of new connections per node over time, but the phenomenon is consistent with both the mobility model design and the observed improvements in classifier stability.

While connection rate data was not collected during this experiments, this behaviour is reflected in the classification results: models such as Random Forest and Decision Tree performed better under mobility scenarios with lower neighbour entropy. This implies that reduced topological churn (i.e., fewer new neighbours) provides more stable patterns for classification.

7.4.4 Results and Observations

The third and final phase of the study investigated the feasibility of identifying influential nodes in a wireless sensor network (WSN) and forecasting their behaviour over time. This was achieved through link prediction techniques applied to time-dependent simulation data. The models were evaluated based on their ability to accurately reflect the true influence of nodes, measured using network centrality metrics.

Influence Metrics and Methodology

Influential nodes were defined using four key centrality measures:

- **Degree Centrality:** Number of direct links a node has.
- **Closeness Centrality:** How near a node is to all other nodes, indicative of efficient information dissemination.
- **Betweenness Centrality:** The frequency with which a node appears on shortest paths between other nodes.
- **Bridging Centrality:** A node's role in connecting distinct communities within the network.

Centrality values were computed on both original network data and the predicted links from various models. Temporal evolution of these metrics was analysed to observe whether node rankings remained consistent across time.

Key Observations

- Centrality rankings stabilised over time — nodes that became influential remained so as the simulation progressed.
- Visual differentiation between actual and predicted centrality plots was difficult, prompting the use of numerical similarity measures.
- Cosine Similarity scores averaged around 0.98 across all centrality metrics, indicating high similarity between predicted and actual node rankings.

Model Performance

Temporal Graph Networks (TGNs) outperformed other link prediction models, achieving up to 10% higher accuracy. Their advantage is attributed to their design, which captures the temporal dynamics inherent in ad-hoc mesh network data. Non-temporal models lacked this capacity, resulting in a loss of important temporal context.

Mobility Model Impact

Three different mobility models were used to simulate node movement, including spatial and temporal dependencies. Results showed negligible impact on influence prediction:

- Average differences in predicted centralities were less than 0.01% for Degree and Closeness centrality.
- The Column Mobility Model had a marginal effect due to group movement, reducing the diversity of link formation.

Both Column Mobility Model and the SMOOTH Mobility Model simulate correlated or community-based node movement. In these cases, nodes tend to maintain stable neighbour sets over time, resulting in fewer new link formations.

7.4.5 Justification of Model Selection

In Phase 3, the goal was to predict future influential nodes within a dynamic wireless sensor network using passive data. This task posed significant challenges due to the temporal and structural complexity of the network, requiring models capable of learning from both topological features and temporal dependencies. The nature of this

task aligned closely with the field of temporal graph learning, guiding the choice of machine learning techniques.

The **Temporal Graph Network (TGN)** was selected as the primary model due to its design for temporal link prediction in evolving graphs. TGN integrates historical interactions over time with structural embeddings, enabling it to learn patterns in how node influence changes dynamically. This was particularly relevant for predicting future facilitator nodes whose roles evolve as connections form and break over time. Its high accuracy (approx. 90%) validated its suitability for this context.

For comparison, simpler graph-based models such as **GraphSAGE** and **Graph Convolutional Networks (GCNs)** were also evaluated. These models provided a baseline for spatial inference but lacked temporal granularity. While they performed adequately when time windows were collapsed, they underperformed in real-time forecasting scenarios.

Additionally, a range of **centrality-based heuristics** (e.g., betweenness, eigenvector, degree centrality) were used as traditional benchmarks. These methods offered interpretability and helped validate model predictions but were inherently static and thus limited in temporal generalisation.

The inclusion of **similarity measures** such as Cosine and Euclidean distance further supported the analysis by enabling a measure of alignment between predicted and actual node importance rankings. These were useful in quantifying the predictive quality of learned embeddings relative to ground-truth rankings.

Model selection in this phase was thus driven by three core considerations:

- the **temporal nature** of the data,
- the **mobility and dynamism** of node roles,
- the need for **high-resolution link prediction** over time.

The adoption of TGN over more traditional or static models reflects the increasing need for adaptive learning in mobile network security and adversarial prediction. The results confirm that temporal representation learning is a critical capability for forecasting influential nodes in dynamic environments.

7.4.6 Analysis

These results confirm that influential node detection in a WSN is both feasible and accurate when temporal information is incorporated. The effectiveness of TGNs highlights the importance of temporal continuity in modelling evolving networks.

The consistently high Cosine Similarity scores validate that predicted links can reliably replicate the structure and influence hierarchy of the original network.

While some metrics like Euclidean and Manhattan distances were less informative due to their sensitivity to scale, Cosine Similarity emerged as a robust measure for comparing influence rankings.

The minimal influence of mobility models suggests that the underlying dynamics of node influence are governed more by link formation patterns than by specific movement behaviours. This robustness implies broader applicability of the approach across varied WSN scenarios.

7.4.7 Summary

Phase 3 demonstrates that the forecasting of influential nodes in wireless sensor networks is achievable with high accuracy using temporal graph-based models. This has practical implications for pre-emptive routing optimisation, load balancing, and failure recovery in dynamic wireless environments. Future work may explore integrating Graph Neural Networks (GNNs) or attention mechanisms to further enhance the granularity and interpretability of influence prediction.

7.5 Comparative Summary of Results Across All Phases

This study was structured into three distinct experimental phases, each focusing on a critical challenge in the analysis and optimisation of Wireless Sensor Networks (WSNs). The phases explored different Machine Learning (ML) and Graph-based techniques for node classification, behaviour prediction, and influence modelling. The table below and subsequent discussion summarise and contrast the main findings of each phase.

Overview Table

TABLE 7.4: Comparative Summary of Results Across Phases

Phase	Primary Task	Best Performing Techniques	Key Observations
Phase 1: Node Placement Bias	Identifying spatial clustering and node deployment bias	**Gaussian Naive Bayes** (77.95%)	Naive Bayes excelled due to its probabilistic handling of feature distributions. Mean Shift, AC, and Spectral performed poorly due to difficulty modelling sparsely clustered data.
Phase 2: Routing Protocol Identification	Classifying routing protocols from node behaviour	**Support Vector Machines** and **Random Forests**	Both models performed well due to their capacity to separate non-linear patterns and handle noisy or overlapping features. CNN underperformed due to sensitivity to poorly scaled input data and curse of dimensionality.
Phase 3: Influential Node Detectionn	Predicting and ranking key nodes in a dynamic WSN	**Temporal Graph Networks (TGN)** with Cosine Similarity (avg. 98%)	TGN effectively captured the temporal evolution of node centrality. Cosine similarity was robust in validating prediction quality. Mobility models had minimal impact.

Cross-Phase Analysis

Each phase addressed a distinct dimension of WSN behaviour, requiring tailored analytical approaches:

- **Phase 1** leveraged density-based and probabilistic models to detect node placement bias. It demonstrated that simple statistical classifiers such as Naive Bayes can outperform complex clustering methods when spatial distribution is highly structured.
- **Phase 2** showed the strength of supervised learning models in distinguishing between routing protocols. SVMs and Random Forests performed reliably due to their robustness in handling high-dimensional feature spaces and non-linear separability, while CNNs struggled due to insufficient feature scaling and dimensional complexity.
- **Phase 3** extended the analysis into temporal forecasting using dynamic graph learning. The success of Temporal Graph Networks highlighted the critical role of temporal data in modelling real-time systems. Centrality metrics and similarity comparisons validated the consistency and accuracy of predictions.

7.5.1 Rationale Behind Model Selection

The selection of machine learning models across all three experimental phases was guided by a combination of theoretical suitability, empirical effectiveness, and the nature of the data available. In Phase 1, simpler probabilistic models such as Gaussian Naive Bayes (GNB) were prioritised due to the statistical regularity and independence assumptions of spatial features. While more complex ensemble methods like Random Forests were later evaluated, initial emphasis was placed on interpretable models that could highlight the contribution of specific topology characteristics. Unsupervised methods, though initially considered for their class-agnostic approach, underperformed due to the subtlety of clustering differences and high-dimensional feature space.

In Phase 2, the decision to test Support Vector Machines (SVM), Random Forests (RF), and Decision Trees (DT) reflected their proven robustness in classifying non-linear and overlapping feature spaces typical of encrypted routing behaviour. Although CNNs are conventionally used in spatial domains, they were included to explore whether sequential or packet-based features might exhibit spatially structured patterns amenable to convolutional processing. Similarly, Deep Forest was tested for its promise of deep representation learning without requiring extensive data, though its relative novelty presented tuning challenges. The use of multiple field combinations

and variations in sampling size further justified ensemble and margin-based methods, which maintained high performance under constrained conditions.

For Phase 3, model selection was driven by the temporal nature of the data. Temporal Graph Networks (TGNs) were favoured for their ability to model dynamic link evolution, outperforming static graph learning methods. While Graph Neural Networks such as GraphSAGE were tested, they lacked the capacity to incorporate fine-grained temporal transitions between snapshots. Additionally, interpretability considerations played a role in evaluating centrality-based influence metrics, balancing model complexity with actionable insight. This tiered model selection strategy ensured alignment between the learning approach and the specific structural, behavioural, and temporal nuances of each phase.

Concluding Remarks

Collectively, the three phases provide a comprehensive framework for analysing WSN dynamics—from static spatial structure to temporal influence propagation. The study underscores the importance of selecting ML models that align with the underlying data characteristics: probabilistic models for spatial bias, ensemble classifiers for behavioural classification, and temporal graph learning for dynamic prediction.

This tiered approach could serve as a modular blueprint for future WSN optimisation systems, enabling both static analysis and real-time adaptive strategies.

The findings highlighted that even a small amount of information can be passively collected from an ad hoc mesh network, offering valuable insights without direct intervention.

Chapter 8

Conclusions

8.1 Overall Findings

This research demonstrates that passive information gathering can reveal critical insights about uncooperative ad hoc mesh networks, spanning three key phases: identifying node placement bias, detecting routing algorithms, and Influential Node Detection. Each phase builds upon the last, forming a cohesive exploration of how seemingly minimal, passively collected data can be used to extract meaningful intelligence.

Phase One established that the choice of topology generators (TGs) used to simulate ad hoc mesh networks introduces measurable bias into node placements. A "Bias Index" was proposed to quantify these differences, and experiments with NPART, BRITE, and GT-ITM revealed that BRITE-generated topologies were distinctly clustered. These findings underscore the importance of considering TG-induced bias when designing network simulations, as failing to do so may skew experimental results and limit real-world applicability.

Phase Two explored whether machine learning (ML) techniques could accurately identify routing algorithms based on passively collected network traffic. The study confirmed that even when traffic was encrypted, metadata analysis — focusing on fields like wlan.fc.subtype, ip.hdr_len, and udp.length — enabled accurate classification. Decision Tree (DT), Support Vector Machine (SVM), and Random Forest (RF) models achieved near-perfect accuracy (99.99–100%) for two-class routing algorithm detection, while DT remained the most stable when expanded to three classes. Notably, reducing the sample size — to as few as five nodes and five consecutive packets — had minimal impact on accuracy, suggesting that only small data samples are required for reliable routing algorithm identification.

Phase Three shifted focus to influential node detection within a dynamic wireless sensor network. The study compared link prediction techniques and found that Temporal Graph Networks (TGNs) outperformed non-temporal methods by 10%, leveraging time-dependent changes in node connectivity. Centrality metrics — including Degree, Betweenness, Closeness, and Bridging — provided a means of ranking node influence, with Cosine Similarity achieving an alignment score of 0.98 when comparing predicted and actual node rankings. Interestingly, mobility models had minimal effect on prediction accuracy, though spatially dependent models like column mobility showed a slight reduction, likely due to nodes moving in coordinated groups. Collectively, these findings highlight the broader implications of passive data gathering. Across all three phases, the research revealed that even limited, encrypted network data can be exploited to uncover node placement strategies, identify routing algorithms, and forecast influential nodes — all without direct network interaction. This not only exposes vulnerabilities in ad hoc mesh networks but also offers a framework for further studies into network dynamics, especially in adversarial contexts.

8.2 Further Work

The following are examples of further work that align with the 3 phases.

8.2.1 Network Topology Generator (Chapter 4)

Future research could refine methodologies for predicting missing node placements based on observed node locations. This could involve leveraging the random seed values used to initialise topology generators (TGs), which influence the deterministic placement of nodes. By understanding or controlling these seeds, it may be possible to better reconstruct or predict missing node positions based on observed patterns.

8.2.2 Routing Algorithm Prediction (Chapter 5)

Building on routing algorithm identification, future work could focus on using the identified algorithms to predict data packet paths, enhancing an adversary's ability to anticipate routing behaviour.

8.2.3 Influential Node Prediction (Chapter 6)

Further research could explore the accuracy and time horizons for future influential node detection nodes. Extending the forecasting capabilities of these algorithms would offer deeper insights into network dynamics.

Collectively, these areas of future work would expand our understanding of passive information gathering and its applications in uncooperative ad hoc mesh networks.

Bibliography

Deep graph library (dgl). URL https://docs.dgl.ai/en/0.8.x/index.html. ns3-network-performance-tool-v2.

https://github.com/neje/ns3-network-performance-tool-v2.

Scipy. URL https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.cosine.html.

Warfighter information network-tactical (win-t). https://gdmissionsystems.com/communications/warfighter-information-network-tactical. Accessed: 2023-06-26.

scikit-learn for naive_bayes bernoullinb, a. URL https://scikit-learn.org/1.6/modules/generated/sklearn.naive_bayes.BernoulliNB.html#sklearn.naive_bayes.BernoulliNB.html.

scikit-learn for decision tree classifier, b. URL hhttps://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

scikit-learn for naive_bayes gaussiannb, c. URL https://scikit-learn.org/1.6/modules/generated/sklearn.naive_bayes.BernoulliNB.html#sklearn.naive_bayes.GaussianNB.html.

scikit-learn for naive_bayes multinomialnb, d. URL https://scikit-learn.org/1.6/modules/generated/sklearn.naive_bayes.BernoulliNB.html#sklearn.naive_bayes.MultinomialNB.html.

scikit-learn for random forest, e. URL https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

scikit-learn for svm, f. URL https://scikit-learn.org/1.5/modules/generated/sklearn.svm.SVC.html.

David W Aha and Richard L Bankert. A comparative evaluation of sequential feature selection algorithms. In *Learning from data*, pages 199–206. Springer, 1996.

Ian F Akyildiz, Xudong Wang, and Weilin Wang. Wireless mesh networks: a survey. *Computer networks*, 47(4):445–487, 2005.

Thuraya NI Alrumaih and Mohammed JF Alenazi. Genind: An industrial network topology generator. *Alexandria Engineering Journal*, 79:56–71, 2023.

Adel Alshamrani. Reconnaissance attack in sdn based environments. In *2020 27th International Conference on Telecommunications (ICT)*, pages 1–5. IEEE, 2020.

Joseph Henry Anajemba, Celestine Iwendi, Mohit Mittal, and Tang Yue. Improved advance encryption standard with a privacy database structure for iot nodes. In *2020 IEEE 9th International Conference on Communication Systems and Network Technologies (CSNT)*, pages 201–206. IEEE, 2020.

Sungwan Bang and Myoungshic Jhun. Weighted support vector machine using k-means clustering. *Communications in Statistics-Simulation and Computation*, 43(10): 2307–2324, 2014.

Jacqueline K Benedetti. On the nonparametric estimation of regression functions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(2):248–253, 1977.

Yoshua Bengio BENGIOY and Yves Grandvalet YVESGRANDVALET. No Unbiased Estimator of the Variance of K-Fold Cross-Validation. *Journal of Machine Learning Research*, 5:1089–1105, 2004.

Kamal Berahmand, Elahe Nasiri, Saman Forouzandeh, and Yuefeng Li. A preference random walk algorithm for link prediction through mutual influence nodes in complex networks. *Journal of king saud university-computer and information sciences*, 34 (8):5375–5387, 2022.

Christian Bettstetter. Mobility modeling in wireless networks: categorization, smooth movement, and border effects. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(3):55–66, 2001.

Suman Bhunia, Paulo Alexandre Regis, and Shamik Sengupta. Distributed adaptive beam nulling to survive against jamming in 3d uav mesh networks. *Computer Networks*, 137:83–97, 2018.

Christoph Birk. Automotive it-services and applications. *Institute of Media Informatics Ulm University*, page 53, 2011.

Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

Subir Biswas, Raymond Tatchikou, and Francois Dion. Vehicle-to-vehicle wireless communication protocols for enhancing highway traffic safety. *IEEE communications magazine*, 44(1):74–82, 2006.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

PageL BrinS. Theanatomyofalarge scalehypertextualwebsearch engine. *ComputerNetworksandISDNSystems*, 30(1):107r117, 1998.

Jason Brownlee. Parametric and Nonparametric Machine Learning Algorithms. URL https://machinelearningmastery.com/parametric-and-nonparametric-machine-learning-algorithms/.

Sergey V Buldyrev, Roni Parshani, Gerald Paul, H Eugene Stanley, and Shlomo Havlin. Catastrophic cascade of failures in interdependent networks. *Nature*, 464 (7291):1025–1028, 2010.

Kenneth L Calvert, Matthew B Doar, and Ellen W Zegura. Modeling internet topology. *IEEE Communications magazine*, 35(6):160–163, 1997.

Tiago Camilo, Jorge Sá Silva, André Rodrigues, and Fernando Boavida. Gensen: A topology generator for real wireless sensor networks deployment. In *Software Technologies for Embedded and Ubiquitous Systems: 5th IFIP WG 10.2 International Workshop, SEUS 2007, Santorini Island, Greece, May 2007. Revised Papers 5*, pages 436–445. Springer, 2007.

Tracy Camp, Jeff Boleng, and Vanessa Davies. A survey of mobility models for ad hoc network research. *Wireless communications and mobile computing*, 2(5):483–502, 2002.

Duanbing Chen, Linyuan Lü, Ming-Sheng Shang, Yi-Cheng Zhang, and Tao Zhou. Identifying influential nodes in complex networks. *Physica a: Statistical mechanics and its applications*, 391(4):1777–1787, 2012.

Bo Cheng and G. Hancke. Energy efficient scalable video manycast in wireless ad-hoc networks. In *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, pages 6216–6221, Oct 2016. .

C. Cheng and S. Lin. A hole-bypassing routing algorithm for wanets. In *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, pages 547–550, Oct 2017. .

Giulio Cimini, Tiziano Squartini, Andrea Gabrielli, and Diego Garlaschelli. Estimating topological properties of weighted networks from limited information. *Physical Review E*, 92(4):040802, 2015.

Thomas Clausen and Philippe Jacquet. Optimized link state routing protocol (olsr). Technical report, 2003.

Jelena Crnogorac, Jovan Crnogorac, Mališa Vučinić, Enis Kočan, and Thomas Watteyne. Dense multi-channel sniffing in large iot networks. *IEEE Access*, 10: 105101–105110, 2022.

Carlos Alexandre Gouvea Da Silva and Carlos Marcelo Pedroso. Mac-layer packet loss models for wi-fi networks: A survey. *IEEE Access*, 7:180512–180531, 2019.

Luc Devroye, László Györfi, and Gábor Lugosi. *A probabilistic theory of pattern recognition*, volume 31. Springer Science & Business Media, 2013.

Jakob Eriksson, Michalis Faloutsos, and Srikanth V Krishnamurthy. Dart: Dynamic address routing for scalable ad hoc and mesh networks. *IEEE/ACM transactions on Networking*, 15(1):119–132, 2007.

Hossam Faris, Al-Zoubi Ala'M, Ali Asghar Heidari, Ibrahim Aljarah, Majdi Mafarja, Mohammad A Hassonah, and Hamido Fujita. An intelligent system for spam detection and identification of the most relevant features based on evolutionary random weight networks. *Information Fusion*, 48:67–83, 2019.

Brian D Fath, Ursula M Scharler, Robert E Ulanowicz, and Bruce Hannon. Ecological network analysis: network construction. *Ecological modelling*, 208(1):49–55, 2007.

Linton C Freeman. Centrality in social networks conceptual clarification. *Social networks*, 1(3):215–239, 1978.

Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *IJCAI*, volume 99, pages 1300–1309, 1999.

José Luis Gallardo, Mohamed A Ahmed, and Nicolás Jara. Clustering algorithm-based network planning for advanced metering infrastructure in smart grid. *IEEE Access*, 9:48992–49006, 2021.

Peter Greig-Smith. The use of random and contiguous quadrats in the study of the structure of plant communities. *Annals of Botany*, pages 293–316, 1952.

Matthias Grossglauser and David NC Tse. Mobility increases the capacity of ad hoc wireless networks. *IEEE/ACM transactions on networking*, 10(4):477–486, 2002.

M. H. Günes and M. B. Akgün. Link-level network topology generation. In *Proceedings of 31st International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 140–145, 2011.

Zygmunt J Haas and Marc R Pearlman. The performance of query control schemes for the zone routing protocol. *ACM SIGCOMM Computer Communication Review*, 28(4): 167–177, 1998.

Zygmunt J Haas, Marc R Pearlman, and P Samar. Interzone routing protocol (ierp), june 2001. *IETFInternet Draft, draft-ietf-manet-ierp-01. txt*, 2001.

Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

Oliver Heckmann, Michael Piringer, Jens Schmitt, and Ralf Steinmetz. On realistic network topologies for simulation. In *Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, pages 28–32. ACM, 2003.

L. V. Hedges. Distribution Theory for Glass's Estimator of Effect size and Related Estimators. *Journal of Educational and Behavioral Statistics*, 1981. .

Paul W. Holland and Samuel Leinhardt. Transitivity in Structural Models of Small Groups. *Comparative Group Studies*, 1971. ISSN 0010-4108. .

Andrew Howard, Maja J Matarić, and Gaurav S Sukhatme. An incremental self-deployment algorithm for mobile sensor networks. *Autonomous Robots*, 13(2): 113–126, 2002.

Darko Hric, Tiago P Peixoto, and Santo Fortunato. Network structure, metadata, and the prediction of missing nodes and annotations. *Physical Review X*, 6(3):031038, 2016.

Woochang Hwang, Young-rae Cho, Aidong Zhang, and Murali Ramanathan. Bridging centrality: identifying bridging nodes in scale-free networks. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 20–23, 2006.

Woochang Hwang, Taehyong Kim, Murali Ramanathan, and Aidong Zhang. Bridging centrality: Graph mining from element level to group level. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, 2008. Association for Computing Machinery. . URL https://doi.org/10.1145/1401890.1401934.

Akylidiz Ian F. and Wang Xudong. A Survey on Wireless Mesh Networks. *IEEE Communications Magazine*, 43(September):23–30, 2005. ISSN 01636804. .

Jithin Jagannath, Nicholas Polosky, Anu Jagannath, Francesco Restuccia, and Tommaso Melodia. Machine learning for wireless communications in the internet of things: A comprehensive survey. *Ad Hoc Networks*, 93:101913, 2019.

Aarti Jain and BVR Reddy. Node centrality in wireless sensor networks: Importance, applications and advances. In *2013 3rd IEEE International Advance Computing Conference (IACC)*, pages 127–131. IEEE, 2013.

David B Johnson and David A Maltz. Dynamic source routing in ad hoc wireless networks. *Mobile computing*, pages 153–181, 1996.

David B Johnson, David A Maltz, Josh Broch, et al. Dsr: The dynamic source routing protocol for multi-hop wireless ad hoc networks. *Ad hoc networking*, 5(1):139–172, 2001.

David Oliver Jorg. Performance comparison of manet routing protocols in different network sizes. *Computer Networks & Distributed Systems*, 2003.

Emil Jovanov, Dusan Starcević, Aleksandar Samardzić, Andy Marsh, and Zeljko Obrenović. Eeg analysis in a telemedical virtual world. *Future Generation Computer Systems*, 15(2):255–263, 1999.

Latha Kant, Kenneth Young, Ossama Younis, David Shallcross, Kaustubh Sinkar, A Mcauley, Kyriakos Manousakis, Kirk Chang, and Charles Graff. Network science based approaches to design and analyze manets for military applications. *IEEE Communications Magazine*, 46(11):55–61, 2008.

Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and KC Claffy. Transport layer identification of p2p traffic. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 121–134, 2004.

Gabriel G Kariuki. A survey on possible attacks in vehicular ad hoc network. *J. of Telecomm. and Information Technology*, 2019.

Shafiullah Khan, Noor Mast, Kok-Keong Loo, and A Silahuddin. Passive security threats and consequences in ieee 802.11 wireless mesh networks. *2; 3*, 2008.

Davar Khoshnevisan. Mathematical Probability. 2002.

Hyoungshick Kim and Ross Anderson. Temporal node centrality in complex networks. *Physical Review E*, 85(2):026107, 2012.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Bulut Kuskonmaz, Huseyin Ozkan, and Ozgur Gurbuz. Machine learning based smart steering for wireless mesh networks. *Ad Hoc Networks*, 88:98–111, 2019.

Leah S Larkey and W Bruce Croft. Combining Classifers in Text Categorization. *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 289–297, 1996. .

Marie-Ange Lèbre, Frédéric Le Mouël, Eric Ménard, Julien Dillschneider, and Richard Denis. Vanet applications: Hot use cases. *arXiv preprint arXiv:1407.4088*, 2014.

Wei Liu, Matteo Pellegrini, and Aiping Wu. Identification of bridging centrality in complex networks. *IEEE Access*, 7:93123–93130, 2019.

Martin Andreoni Lopez, Michael Baddeley, Willian T Lunardi, Anshul Pandey, and Jean-Pierre Giacalone. Towards secure wireless mesh networks for uav swarm connectivity: Current threats, research, and opportunities. *arXiv preprint arXiv:2108.13154*, 2021.

Gabriel López-Millán, Rafael Marín-López, Fernando Pereñíguez-García, Oscar Canovas, and José Antonio Parra Espín. Analysis and practical validation of a standard sdn-based framework for ipsec management. *Computer Standards & Interfaces*, 83:103665, 2023.

Wenjing Lou, Wei Liu, Yanchao Zhang, and Yuguang Fang. Spread: Improving network security by multipath routing in mobile ad hoc networks. *Wireless Networks*, 15(3):279–294, 2009.

Linyuan Lü, Yi-Cheng Zhang, Chi Ho Yeung, and Tao Zhou. Leaders in social networks, the delicious case. *PloS one*, 6(6):e21202, 2011.

Thorsten Luettel, Michael Himmelsbach, and Hans-Joachim Wuensche. Autonomous ground vehicles—concepts and a path to the future. *Proceedings of the IEEE*, 100 (Special Centennial Issue):1831–1839, 2012.

D. Magoni and J. Pansiot. Evaluation of internet topology generators by power law and distance indicators. In *Proceedings 10th IEEE International Conference on Networks (ICON 2002). Towards Network Superiority (Cat. No.02EX588)*, pages 401–406, 2002.

Damien Magoni and J J Pansiot. Analysis and Comparison of Internet Topology Generators. *NETWORKING 2002: Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications*, 2345:364–375, 2006.

Damien Magoni and Jean-Jacques Pansiot. Influence of network topology on protocol simulation. In *International Conference on Networking*, pages 762–770. Springer, 2001.

Ilya Makarov, Olga Gerasimova, Pavel Sulimov, and Leonid E Zhukov. Dual network embedding for representing research interests in the link prediction problem on co-authorship networks. *PeerJ Computer Science*, 5:e172, 2019.

Fragkiskos D Malliaros, Maria-Evgenia G Rossi, and Michalis Vazirgiannis. Locating influential nodes in complex networks. *Scientific reports*, 6(1):1–10, 2016.

Bomin Mao, Fengxiao Tang, Zubair Md Fadlullah, Nei Kato, Osamu Akashi, Takeru Inoue, and Kimihiro Mizutani. A novel non-supervised deep-learning-based network traffic control method for software defined wireless networks. *IEEE Wireless Communications*, 25(4):74–81, 2018.

Thomas Martin, Emil Jovanov, and Dejan Raskovic. Issues in wearable computing for medical monitoring applications: a case study of a wearable ecg monitoring device.

In *Digest of Papers. Fourth International Symposium on Wearable Computers*, pages 43–49. IEEE, 2000.

Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer, 1998.

A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: an approach to universal topology generation. In *MASCOTS 2001, Proceedings Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 346–353, Aug 2001a. .

Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. BRITE: Universal Topology Generation from a User's Perspective. Technical report, Boston, MA, USA, 2001b.

Bratislav Milic and Miroslaw Malek. NPART-node placement algorithm for realistic topologies in wireless multihop network simulation. In *Proceedings of the 2nd international conference on simulation tools and techniques*, 2009.

Aarti Munjal, Tracy Camp, and William C Navidi. Smooth: a simple way to model human mobility. In *Proceedings of the 14th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems*, pages 351–360, 2011.

Nicolò Musmeci, Stefano Battiston, Guido Caldarelli, Michelangelo Puliga, and Andrea Gabrielli. Bootstrapping topological properties and systemic risk of complex networks using the fitness model. *Journal of Statistical Physics*, 151:720–734, 2013.

S Nowak, M Nowak, and K Grochla. Properties of advanced metering infrastructure networks' topologies. *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–6, 2014. .

Oladayo Olufemi Olakanmi and Adedamola Dada. Wireless sensor networks (wsns): Security and privacy issues and solutions. In *Wireless Mesh Networks-Security, Architectures and Protocols*. IntechOpen, 2020.

Michael O'Sullivan, Leonardo Aniello, and Vladimiro Sassone. A methodology to select topology generators for ad hoc mesh network simulations. *J. Commun.*, 15(10): 741–746, 2020.

Canhui Ou, Hui Zang, Narendra K Singhal, Keyao Zhu, Laxman H Sahasrabuddhe, Robert A MacDonald, and Biswanath Mukherjee. Subpath protection for scalability and fast recovery in optical wdm mesh networks. *IEEE Journal on Selected Areas in Communications*, 22(9):1859–1875, 2004.

Dr G Padmavathi, Mrs Shanmugapriya, et al. A survey of attacks, security mechanisms and challenges in wireless sensor networks. *arXiv preprint arXiv:0909.0576*, 2009.

J Rejina Parvin. An overview of wireless mesh networks. *Wireless Mesh Networks-Security, Architectures and Protocols*, 2019.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Charles Perkins, Elizabeth M Royer, and S Das. Ad-hoc on demand distance vector routing (aodv). Technical report, Internet-Draft, November 1997. draft-ietf-manet-aodv-00. txt, 2003.

Charles E Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *ACM SIGCOMM computer communication review*, 24(4):234–244, 1994.

Uroš M Pešović, Jože J Mohorko, Karl Benkič, and Žarko F Čučej. Single-hop vs. multi-hop–energy efficiency analysis in wireless sensor networks. In *18th telecommunications forum, TELFOR*, 2010.

Sonja Pravilovic, Massimo Bilancia, Annalisa Appice, and Donato Malerba. Using multiple time series analysis for geosensor data forecasting. *Information Sciences*, 380:31–52, 2017.

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

Regan Rajendran et al. An optimal strategy to countermeasure the impersonation attack in wireless mesh network. *International Journal of Information Technology*, 13(3): 1033–1038, 2021.

Shahid Raza, Tony Chung, Simon Duquennoy, Thiemo Voigt, Utz Roedig, et al. Securing internet of things with lightweight ipsec, 2010.

K Ganesh Reddy and P Santhi Thilagam. Hierarchical wireless mesh networks scalable secure framework. *International Journal of Information and Network Security (IJINS) Volume*, 2(2):167–176, 2013.

George F Riley and Thomas R Henderson. The ns-3 network simulator. In *Modeling and tools for network simulation*, pages 15–34. Springer, 2010.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. In *ICML 2020 Workshop on Graph Representation Learning*, 2020.

R Rossi, S Fahmy, and N Talukder. A Multi-Level Approach for Evaluating Internet Topology Generators. *2013 IFIP Networking Conference*, pages 9 pp.–9 pp., 2013.

Thomas Rückstieß, Christian Osendorfer, and Patrick Van Der Smagt. Sequential feature selection for classification. In *AI 2011: Advances in Artificial Intelligence: 24th Australasian Joint Conference, Perth, Australia, December 5-8, 2011. Proceedings 24*, pages 132–141. Springer, 2011.

M L Sanni, A A Hashim, F Anwar, G S M Ahmed, and S Ali. How to model wireless mesh networks topology. *IOP Conference Series: Materials Science and Engineering*, 53 (1):012037, 2013.

Pushpender Sarao. Machine learning and deep learning techniques on wireless networks. *International Journal of Engineering Research and Technology*, 12(3):311–320, 2019.

Curtis M. Scaparrotti. Joint publication 3-13 information operations.

L. Schiavone, N. Browne, R. North, L. Schiavone, N. Browne, and R. North. Joint tactical radio system - connecting the gig to the tactical edge. In *MILCOM 2006 - 2006 IEEE Military Communications conference*, pages 1–6, Oct 2006. .

Jaydip Sen. Security and privacy issues in wireless mesh networks: A survey. In *Wireless networks and security*, pages 189–272. Springer, 2013.

Siraj A. Shaikh, Howard Chivers, Philip Nobles, John A. Clark, and Hao Chen. Network reconnaissance. *Network Security*, 2008(11):12–16, 2008. ISSN 1353-4858. . URL https://www.sciencedirect.com/science/article/pii/S1353485808701296.

Li Shi-Chang, Yang Hao-Lan, and Zhu Qing-Sheng. Research on manet security architecture design. In *2010 International Conference on Signal Acquisition and Processing*, pages 90–93. IEEE, 2010.

Muhammad Shoaib Siddiqui et al. Security issues in wireless mesh networks. In *2007 International Conference on Multimedia and Ubiquitous Engineering (MUE'07)*, pages 717–722. IEEE, 2007.

Guojie Song, Yuanhao Li, Xiaodong Chen, Xinran He, and Jie Tang. Influential node tracking on dynamic social network: An interchange greedy approach. *IEEE Transactions on Knowledge and Data Engineering*, 29(2):359–372, 2016.

Jaime Lynn Speiser, Michael E Miller, Janet Tooze, and Edward Ip. A comparison of random forest variable selection methods for classification prediction modeling. *Expert systems with applications*, 134:93–101, 2019.

Giorgio Stampa, Marta Arias, David Sánchez-Charles, Victor Muntés-Mulero, and Albert Cabellos. A deep-reinforcement learning approach for software-defined networking routing optimization. *arXiv preprint arXiv:1709.07080*, 2017.

Mervyn Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the royal statistical society. Series B (Methodological)*, pages 111–147, 1974.

Peng Gang Sun, Yi Ning Quan, Qi Guang Miao, and Juan Chi. Identifying influential genes in protein–protein interaction networks. *Information Sciences*, 454:229–241, 2018.

CATH Tee and A Lee. A novel routing protocol—junction based adaptive reactive routing (jarr) for vanet in city environments. In *2010 European Wireless Conference (EW)*, pages 1–6. IEEE, 2010.

Navamani Thandava Meganathan and Yogesh Palanichamy. Privacy preserved and secured reliable routing protocol for wireless mesh networks. *The Scientific World Journal*, 2015, 2015.

Muluneh Mekonnen Tulu, Ronghui Hou, and Talha Younas. Identifying influential nodes based on community structure to speed up the dissemination of information in complex network. *IEEE Access*, 6:7390–7401, 2018a. .

Muluneh Mekonnen Tulu, Ronghui Hou, and Talha Younas. Identifying influential nodes based on community structure to speed up the dissemination of information in complex network. *IEEE Access*, 6:7390–7401, 2018b.

Jungfang Wang, Bin Xie, and Dharma P Agrawal. Journey from mobile ad hoc networks to wireless mesh networks. In *Guide to wireless mesh networks*, pages 1–30. Springer, 2009.

Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks, 2020. URL `https://arxiv.org/abs/1909.01315`.

Zhen Wang, Chris T Bauch, Samit Bhattacharyya, Alberto d'Onofrio, Piero Manfredi, Matjaž Perc, Nicola Perra, Marcel Salathé, and Dawei Zhao. Statistical physics of vaccination. *Physics Reports*, 664:1–113, 2016.

B. M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, Dec 1988. ISSN 0733-8716. .

Ruffin White, Gianluca Caiazza, Chenxu Jiang, Xinyue Ou, Zhiyue Yang, Agostino Cortesi, and Henrik Christensen. Network reconnaissance and vulnerability excavation of secure dds systems. In *2019 IEEE European symposium on security and privacy workshops (EUROS&PW)*, pages 57–66. IEEE, 2019.

Edward L Witzke, Joseph P Brenkosh, Karl L Green, Loren E Riblett, and James M Wiseman. Encryption in mobile wireless mesh networks. In *2012 IEEE International Carnahan Conference on Security Technology (ICCST)*, pages 251–256. IEEE, 2012.

Zengrui Wu, Weihua Li, Guixia Liu, and Yun Tang. Network-based methods for prediction of drug-target interactions. *Frontiers in pharmacology*, 9:1134, 2018.

Y. Xu, J. Liu, Y. Shen, X. Jiang, and T. Taleb. Security/qos-aware route selection in multi-hop wireless ad hoc networks. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6, May 2016. .

Y. Xu, J. Liu, O. Takahashi, N. Shiratori, and X. Jiang. Soqr: Secure optimal qos routing in wireless ad hoc networks. In *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2017.

Yong Yao and Cong Ji. Identifying influential users by improving leaderrank. In *The International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery*, pages 459–467. Springer, 2019.

Guang Zhan, Zheng Gong, Quanhui Lv, Zan Zhou, Zian Wang, Zhen Yang, and Deyun Zhou. Flight test of autonomous formation management for multiple fixed-wing uavs based on missile parallel method. *Drones*, 6(5):99, 2022.

Xiaohang Zhang, Ji Zhu, Qi Wang, and Han Zhao. Identifying influential nodes in complex networks with community structure. *Knowledge-Based Systems*, 42:74–84, 2013.

Zhi-Hua Zhou and Ji Feng. Deep forest. *arXiv preprint arXiv:1702.08835*, 2017.