Communication in Drones using UDP

Piyush Prajapati

October 2025

Contents

1 Research Papers **3**
1.1 Terminologies . . . 3
1.1.1 Dynamic Space Partition Algorithm (DSP) . . . 3
1.1.2 Voronoi Tessellation . . . 3
1.1.3 Lawn Mower Patterns in Robotic Exploration . . . 4
1.1.4 Cellular Decomposition in Coverage Path Planning . . . 4
1.1.5 Swarm Coverage Path Planning (SCoPP) . . . 5
1.2 Paper : DSP with Archimedian Spirals . . . 5
1.2.1 Summary of Core Work . . . 5
1.2.2 Identified Limitations . . . 5
1.2.3 Scope for Improvement . . . 6
1.3 Paper : Tsunami: Scalable, Fault Tolerant Coverage Path Planning for UAV Swarms . . . 6
1.3.1 Overview . . . 6
1.3.2 System Architecture and Workflow . . . 6
1.3.3 Fault Tolerance and Adaptability . . . 7
1.3.4 Experimental Evaluation . . . 7
1.3.5 Discussion and Advantages . . . 8
1.3.6 Limitations and Future Work . . . 8

2 BATMAN: Better Approach To Mobile Ad-hoc Networking **9**
2.1 Important Concepts Required . . . 9
2.1.1 Mobile Adhoc Network (MANET) . . . 9
2.1.2 Optimized Link State Routing (OLSR) . . . 9
2.1.3 Multihop Communication . . . 9
2.2 Introduction . . . 9
2.3 Operational Principles . . . 10
2.3.1 Originator Messages (OGMs) and Neighbor Discovery . . . 10
2.3.2 Link Quality and Metrics . . . 10
2.3.3 batman-adv and Layer 2 Routing . . . 10
2.3.4 Flooding, Propagation, and Message Overhead . . . 10
2.4 Advantages and Use Cases . . . 10
2.5 Limitations and Challenges . . . 10
2.6 Potential Improvements and Extensions . . . 11

3 Understanding Communication **12**
3.1 MAC and IP Addresses . . . 12
3.2 OSI Layers . . . 12
3.3 MANET and WANET . . . 13
3.4 Routers and Access Points . . . 13
3.5 UDP vs TCP . . . 13

4 Simulation of Drone Communication using UDP Sockets and Wifi-Direct **14**
4.1 Simulation . . . 14
4.1.1 Overview . . . 14
4.1.2 System Architecture . . . 14
4.1.3 Operation . . . 14
4.1.4 Results . . . 15

4.1.5 Limitations 15
4.2 Wi-Fi Direct for Drone Communication 15
4.2.1 Introduction 15
4.2.2 Wi-Fi Direct for Drone-to-Drone Communication 15
4.2.3 Limitations for Multi-Drone Swarms 16
4.3 Conclusion 16

5 Establishing an Ad-Hoc Network Between Two Devices **17**
5.1 IBSS (Ad-Hoc) Mode 17
5.2 Overview of Steps 17
5.3 Detailed Procedure 17
5.3.1 Wireless Interface Preparation 17
5.3.2 Setting IBSS Mode and Ad-Hoc Parameters 18
5.3.3 IP Configuration 18
5.3.4 Testing Connectivity 18
5.4 Results 18
5.5 Discussion and Limitations 18

Chapter 1

Research Papers

1.1 Terminologies

1.1.1 Dynamic Space Partition Algorithm (DSP)

Dynamic Space Partitioning (DSP) refers to the process of recursively dividing a multidimensional space into smaller, more manageable subregions that can adapt dynamically to new data or environmental changes. It serves as a foundational concept in computational geometry, robotics, and spatial data analysis.

Formally, consider a space R^n containing a finite set of data points or objects $S = \{s_1, s_2, \ldots, s_m\}$. A dynamic partition function f maps S into disjoint regions $\{R_1, R_2, \ldots, R_k\}$ such that:

$$\bigcup_{i=1}^{k} R_i = R^n, \quad R_i \cap R_j = \emptyset \text{ for } i \neq j$$

Each region R_i may evolve with time t as:

$$R_i(t + \Delta t) = f(R_i(t), \Delta S)$$

where ΔS represents the set of new insertions, deletions, or updates.

Dynamic Space Partitioning enables efficient spatial querying, path planning, and data organization by maintaining hierarchical spatial relationships. It is frequently implemented through structures such as quadtrees, k-d trees, and bounding volume hierarchies (BVH), each offering different trade-offs between update complexity and query performance.

1.1.2 Voronoi Tessellation

Voronoi tessellation is a geometric partitioning technique used to divide a given space into regions based on proximity to a predefined set of seed points, also known as generators or sites. It is extensively applied in computational geometry, robotics, wireless communication, and spatial data analysis.

Given a finite set of seed points $P = \{p_1, p_2, \ldots, p_n\} \subset R^2$, the Voronoi cell corresponding to a point p_i is defined as:

$$V(p_i) = \{x \in R^2 \mid \|x - p_i\| \leq \|x - p_j\|, \ \forall j \neq i\}$$

Each region $V(p_i)$ thus contains all points closer to p_i than to any other seed p_j. The union of all Voronoi cells covers the entire domain:

$$\bigcup_{i=1}^{n} V(p_i) = R^2, \quad V(p_i) \cap V(p_j) = \emptyset \text{ for } i \neq j$$

Voronoi tessellations are particularly valuable in adaptive partitioning systems such as Dynamic Space Partitioning (DSP), where the partition boundaries can evolve dynamically as the set of generators changes. This dynamic behavior makes Voronoi-based approaches suitable for real-time robotic mapping, coverage optimization, and environmental modeling.

1.1.3 Lawn Mower Patterns in Robotic Exploration

The lawn mower pattern, also referred to as the *boustrophedon* pattern, is a deterministic path planning strategy used in robotic exploration and area coverage applications. The term originates from the alternating back-and-forth trajectory similar to that of a lawn mower covering a rectangular field.

Consider a rectangular workspace Ω with dimensions $L \times W$. The trajectory of a robot following a lawn mower pattern can be expressed as a parametric path:

$$P(t) = (x(t), y(t)) = \begin{cases} (vt, kd), & \text{if } k \text{ is even} \\ (L - vt, kd), & \text{if } k \text{ is odd} \end{cases}$$

where v is the constant linear velocity, d is the inter-track spacing, and k denotes the current traversal index. This formulation ensures systematic coverage of the entire region with minimal overlap.

Advantages:

- Guarantees complete coverage of a rectangular or structured region.
- Simple to implement and computationally efficient.
- Enables predictable and easily monitored robotic motion.

Limitations:

- Reduced efficiency in irregularly shaped or obstacle-dense environments.
- Predictable motion may be undesirable in surveillance or adversarial contexts.
- Lacks adaptability to dynamically changing conditions without external control feedback.

In coverage missions, the lawn mower pattern serves as a baseline path planning approach. More advanced strategies, such as those integrating *Archimedean spirals* or dynamic partitioning, improve upon it by enabling adaptive and continuous coverage suited for non-uniform terrains.

1.1.4 Cellular Decomposition in Coverage Path Planning

Cellular Decomposition is a fundamental strategy in Coverage Path Planning (CPP) that involves dividing a given environment into smaller, structured subregions called *cells*. Each cell represents a manageable area that can be systematically covered by a robotic agent or an Unmanned Aerial Vehicle (UAV). This decomposition allows for localized path optimization, simplified control, and efficient area management during exploration or surveillance missions.

Types of Cellular Decomposition:

- **Exact Cellular Decomposition:** The environment is partitioned into non-overlapping cells that exactly represent the area of interest. The resulting cells are typically simple polygons, such as rectangles or trapezoids, enabling efficient coverage with deterministic motion patterns like the lawn mower or spiral trajectories.
- **Approximate Cellular Decomposition:** The workspace is discretized into a grid composed of uniform, identically sized cells—commonly squares. This method is computationally efficient and well-suited for occupancy-based mapping, though it may introduce slight inaccuracies near irregular or curved boundaries.
- **Semi-Approximate Decomposition:** A hybrid model that combines the structural simplicity of regular cells with adaptive boundary fitting, improving representation accuracy while retaining computational tractability.

Cellular Decomposition provides the spatial framework necessary for high-level planning algorithms such as Dynamic Space Partitioning (DSP) and facilitates task allocation in multi-robot systems.

1.1.5 Swarm Coverage Path Planning (SCoPP)

Swarm Coverage Path Planning (SCoPP) generalizes the principles of CPP to scenarios involving multiple UAVs or robotic agents operating collaboratively within the same environment. Its objective is to ensure full coverage of the target region while optimizing for efficiency, collision avoidance, and fault tolerance through distributed coordination.

The SCoPP framework is characterized by the following essential components:

- **Parallelization:** The overall workspace is divided into subregions that can be explored concurrently by different UAVs, leading to significant reductions in total mission time.
- **Coordination:** UAVs continuously exchange positional and task data to prevent overlaps, ensure collision-free paths, and maintain global coverage coherence.
- **Load Balancing:** Tasks are distributed proportionally among UAVs based on their energy levels, sensor ranges, and flight capabilities to achieve uniform resource utilization.
- **Scalability:** The underlying algorithms are designed to remain computationally feasible and efficient even as the number of UAVs or size of the operational domain increases.
- **Fault Tolerance:** In case of individual UAV malfunction or failure, neighboring units dynamically reassign coverage responsibilities to maintain mission completeness.

Modern SCoPP implementations often integrate techniques from Dynamic Space Partitioning and Cellular Decomposition, enabling real-time adaptability and optimized path generation in complex or evolving terrains. The synergy between spatial partitioning and multi-agent coordination forms the basis for advanced coverage missions, such as environmental monitoring, search-and-rescue operations, and agricultural surveying.

1.2 Paper : DSP with Archimedian Spirals

1.2.1 Summary of Core Work

The research explores an improved approach to Dynamic Space Partitioning (DSP) for multi-UAV wildfire monitoring systems by integrating an *Archimedean spiral*-based exploration pattern. Traditional DSP methods rely on random-walk exploration around virtual points, resulting in non-uniform coverage, low reproducibility, and unpredictable trajectories. To overcome these limitations, the proposed DSP-A (Dynamic Space Partitioning with Archimedean spiral) controller introduces a systematic, deterministic search pattern to enhance coverage efficiency and minimize overlap.

Each UAV is associated with a virtual DSP point, dynamically distributed through attractive and repulsive force interactions. The UAV navigates around its DSP point using a PID controller that produces smooth heading transitions, following a spiral trajectory defined in polar coordinates as:

$$r(\theta) = a + b\theta$$

The spiral's inter-loop distance is optimized based on the UAV's sensing radius f such that:

$$2\pi b = 2f$$

ensuring complete and efficient area coverage without redundancy.

Experimental results across multiple swarm configurations demonstrate that the DSP-A controller significantly outperforms the baseline DSP in coverage rate and consistency. For instance, with 10 UAVs, DSP-A achieved an average detection rate of 69.33%, compared to 43.92% for the baseline DSP. Even with larger swarm sizes, DSP-A maintained superior efficiency, evidencing scalability and robustness in distributed coverage operations.

1.2.2 Identified Limitations

Despite its notable improvements, the proposed DSP-A framework exhibits several limitations:

- **Limited Real-World Validation:** Experiments are primarily simulation-based, lacking field deployment under dynamic environmental conditions.

- **Neglected Environmental Factors:** The UAV dynamics are idealized, assuming stable flight conditions without wind disturbances or terrain constraints.
- **Communication Dependence:** System performance assumes continuous connectivity between UAVs, making it vulnerable to network disruptions.
- **Fixed Altitude and Energy Constraints:** The UAVs are modeled at constant altitude with no consideration for energy consumption, refueling, or battery limitations.

1.2.3 Scope for Improvement

To enhance realism, scalability, and operational resilience, the following research directions are proposed:

- **Resilience and Energy Management:** Incorporate refueling and energy-aware path planning models, allowing UAVs to autonomously return to base and rejoin the mission.
- **Communication Robustness:** Implement fallback or decentralized control mechanisms to maintain coverage continuity during communication losses.
- **Enhanced Fire Modeling:** Integrate dynamic wildfire simulations using cellular automata or agent-based modeling to capture spread behavior under varying environmental conditions.
- **Environmental Adaptability:** Introduce external variables such as wind, temperature gradients, and terrain elevation into the DSP-A model to improve practical reliability.
- **Human-in-the-Loop Integration:** Enable firefighter intervention by defining high-priority zones (e.g., power stations, water facilities) and incorporating ML-based predictive analysis for autonomous hotspot detection.

1.3 Paper : Tsunami: Scalable, Fault Tolerant Coverage Path Planning for UAV Swarms

1.3.1 Overview

Recent research addresses large-scale UAV swarm coordination as a *Coverage Path Planning (CPP)* problem, where the goal is to generate collision-free trajectories ensuring complete area exploration with minimal time and energy expenditure. CPP has been successfully applied across ground, aerial, and underwater domains; however, this study emphasizes the aerial case for UAVs operating under heterogeneous conditions.

The proposed framework, named **Tsunami**, introduces an online, fault-tolerant coverage algorithm inspired by the wavefront traversal method. Unlike conventional offline partitioning approaches, Tsunami dynamically allocates work among drones during runtime, enabling scalable and resilient operation. Prior work by Collins employed static Voronoi decompositions with auction-based cell allocation and local lawnmower navigation; Tsunami extends this concept by performing decomposition and redistribution online.

1.3.2 System Architecture and Workflow

Tsunami operates in two major phases: an *offline preparation phase* and a *runtime exploration phase.*

Offline Phase

During initialization, the environment is discretized into a grid of GPS waypoints. User-defined polygonal inputs specify the exploration region, restricted no-fly zones, and mission parameters such as altitude, velocity, and coverage resolution. A breadth-first wavefront expansion is then computed to establish traversal order among waypoints:

$$\Omega = \text{wavefront}(E, \text{NFZs})$$

where Ω represents the ordered waypoint set and NFZs denote no-fly zones. Simultaneously, a drone pool Δ is initialized to manage available UAVs, each maintaining state attributes such as position, energy, and operational status.

Runtime Phase

At runtime, Tsunami continuously dispatches drones until the environment is fully explored. Each UAV is categorized as:

- **Active:** currently executing a flight trajectory and performing sensing.
- **Idle:** in charging, maintenance, or non-operational mode.
- **Ready:** available for immediate task assignment.

For each ready UAV δ, the nearest unvisited waypoint ω_i is computed as:

$$\omega_i = \arg\min_{\omega \in \Omega} d(\delta_\omega, \omega)$$

After assignment, the waypoint is removed from Ω, and a spline-based trajectory T_δ is generated to ensure smooth, collision-free flight:

$$T_\delta = \text{buildSpline}(\delta_\omega, \omega_i, \Delta)$$

Simple altitude offsets provide inter-UAV collision avoidance in this implementation. Each UAV asynchronously executes its assigned path, collects sensory data, and returns corresponding image–GPS–ID tuples for post-processing.

Algorithmic Summary

[H] [1] TsunamiEnvironment, NFZs $\Omega \leftarrow$ WAVEFRONT(Environment, NFZs) $\Delta \leftarrow$ INITDRONEPOOL() imageMap $\leftarrow$ [ ] $|\Omega| > 0$ $\delta \in \Delta$ READY(δ) $\omega_i \leftarrow$ MINDISTANCE(Ω, δ_ω) REMOVE(ω_i, Ω) $T_\delta \leftarrow$ BUILDSPLINE($\delta_\omega, \omega_i, \Delta$) $\delta_\omega \leftarrow \omega_i$ SETSTATUS(δ, 'active') $I \leftarrow$ FLYASYNC(δ, T_δ) imageMap.append($[I, \delta.\text{gps}, \delta.\text{id}]$)

1.3.3 Fault Tolerance and Adaptability

Tsunami's online nature inherently supports fault-tolerant behavior:

- UAVs can seamlessly enter or exit the swarm, allowing flexible scaling.
- No fixed assumptions exist regarding energy capacity, communication reliability, or weather stability.
- The wavefront-based partitioning is recalculated in real time, maintaining load balance and continuity.

Unlike offline decomposition methods that require a dedicated cleanup phase to reassign missed regions, Tsunami's dynamic allocation ensures continuous redistributive coverage without global recomputation.

1.3.4 Experimental Evaluation

Extensive simulation and limited real-world trials validate Tsunami's scalability and robustness:

- **Simulated Tests:** Up to 50 UAVs achieved near-linear scaling in computation time with a 1.6× improvement in coverage efficiency over standard methods.
- **Physical Deployment:** Field trials using 10 drones reported 95% coverage success and maintained 87% efficiency despite induced 30% drone failures.
- **Performance Gains:** Crop-scouting simulations demonstrated 1.9× improvement under fault conditions compared to classical Voronoi–auction decompositions.

1.3.5 Discussion and Advantages

Tsunami delivers a unified framework for UAV swarm coordination that balances:

- **Scalability:** Linear computational scaling with swarm size.
- **Reliability:** Graceful degradation under partial failures.
- **Consistency:** Reduced collision frequency due to wavefront ordering.

By partitioning environments online, Tsunami ensures even workload distribution, improved trajectory predictability, and reduced communication overhead, making it suitable for agricultural surveying, disaster assessment, and large-area reconnaissance.

1.3.6 Limitations and Future Work

While Tsunami offers strong practical advantages, it remains constrained by several factors:

- Lack of in-depth real-world validation under wind disturbances, GPS drift, and intermittent communication.
- Simplified altitude-based avoidance; advanced ORCA or MPC methods can enhance energy-efficient 3D coordination.
- Absence of real-time onboard mosaicking and analytics integration for mission-time decision support.

Future extensions may focus on:

- Integrating heterogeneous UAV types with varying payload and endurance.
- Adaptive replanning for dynamic environments and no-fly zone updates.
- Embedding ML-based path optimization for energy-aware and context-sensitive tasking.
- Scaling experiments to swarms exceeding 100 UAVs for multi-hectare operations.

Chapter 2

BATMAN: Better Approach To Mobile Ad-hoc Networking

2.1 Important Concepts Required

2.1.1 Mobile Adhoc Network (MANET)

A **Mobile Adhoc Network (MANET)** is a self-configuring, infrastructure-less network of mobile devices connected wirelessly. Each node in a MANET acts as both a host and a router, forwarding packets for other nodes. Since the topology of a MANET changes frequently due to node mobility, routing protocols in MANETs must be adaptive, decentralized, and capable of handling dynamic network conditions.

2.1.2 Optimized Link State Routing (OLSR)

The **Optimized Link State Routing (OLSR)** protocol is a proactive routing protocol designed for MANETs. It continuously exchanges topology information to maintain updated routes to all nodes. OLSR reduces overhead by using *Multipoint Relays (MPRs)*—a set of nodes responsible for broadcasting control messages. While OLSR ensures low-latency route availability, it incurs higher control message overhead in highly dynamic or dense networks.

2.1.3 Multihop Communication

Multihop communication refers to the process where data is transmitted across multiple intermediate nodes before reaching its final destination. In MANETs, due to limited transmission range and node mobility, direct communication between all nodes is often not possible. Multihop routing ensures that packets can traverse the network efficiently even when nodes are separated by several hops, thereby improving connectivity and network scalability.

2.2 Introduction

The **BATMAN** protocol (Better Approach To Mobile Ad-hoc Networking) is a routing protocol designed for multi-hop, mobile ad-hoc (mesh) networks. It was developed by the Freifunk community beginning around 2006 as a response to limitations in existing routing protocols such as OLSR. Its goal is to provide a robust, decentralized method of maintaining routes in networks with frequently changing topology, avoiding excessive control-traffic, and enabling scalable, resilient mesh behavior.

There are two major variants of BATMAN in active use:

- **batmand** – operates at Layer 3, exchanging packets (Originator Messages etc.) using the IP-based stack.
- **batman-adv** (advanced) – operates at OSI Layer 2, routing (or more accurately bridging) Ethernet frames. It provides a virtual network interface that makes all participating nodes appear as if they are directly connected by a virtual switch.

2.3 Operational Principles

2.3.1 Originator Messages (OGMs) and Neighbor Discovery

Each node in the network periodically broadcasts an *Originator Message* (OGM) to its immediate neighbors to announce its presence. Neighbors receive OGMs, evaluate link qualities, and forward (rebroadcast) the messages onward. Over time, each node maintains a view of which of its direct neighbors provides the best path (in terms of link quality, reliability) to each originator. Each node thus builds a routing table where, for each known destination (originator), it only needs to store the single-hop neighbor that is the best next hop. There is no necessity for global topology knowledge at every node.

2.3.2 Link Quality and Metrics

BATMAN uses metrics such as "Transmission Quality" (TQ) to measure how good a link between two neighbors is. OGMs include fields for metrics so that receiving nodes can rate paths via different neighbors and pick the best. Asymmetric links are handled carefully by considering both forward and reverse qualities.

2.3.3 batman-adv and Layer 2 Routing

In batman-adv, routing is done at the Ethernet (layer 2) level. The protocol emulates a virtual switch across all nodes. This means that higher-layer protocols (IPv4, IPv6, DHCP etc.) operate unchanged, as if the mesh network is a single large broadcast domain. This simplifies integration and client roaming, especially when nodes do not yet have IP addresses.

2.3.4 Flooding, Propagation, and Message Overhead

OGMs are flooded through the network (subject to certain optimizations, such as discarding clearly suboptimal OGMs) to maintain reachability information. The amount of overhead scales with the number of nodes and the frequency of OGMs. Therefore, there is a trade-off between fresh, accurate routing information and overhead.

2.4 Advantages and Use Cases

- **Decentralization:** No single point of failure; each node only needs local neighbor info rather than full topology. This supports robustness in dynamic, mobile environments.
- **Layer 2 Transparency (batman-adv):** Transparent to IP/Layer 3 protocols; clients can roam without IP changes; works even if nodes lack IP addresses initially.
- **Handling Asymmetric Links:** BATMAN includes mechanisms to evaluate link directionality and quality from both ends, which helps in realistic wireless environments with asymmetries.
- **Low Overhead Compared to Link-State Protocols:** Because nodes do not need to maintain full global topology, and do not calculate large routing tables as in OLSR, the control traffic and computation demands are reduced.
- **Real-World Deployment:** Used by mesh networks (e.g. Freifunk), and included in Linux kernels, making it mature and tested.

2.5 Limitations and Challenges

- **Scalability of Flooding:** Periodic OGMs will still incur substantial overhead in large or dense networks. As the number of nodes increases, bandwidth consumed by broadcasts can impact performance.
- **Storage for Routing Tables:** Although each node only stores next-hop neighbor for each originator, in large-scale networks, the number of originators (i.e. other nodes) can become large, causing memory/storage concerns.

- **Delay and Freshness Trade-offs:** If OGMs are infrequent, paths may become stale; if too frequent, overhead increases. Balancing update intervals is non-trivial.
- **Collision and Broadcast Storms:** Wireless mesh environments are prone to interference, collisions, and broadcast high traffic, especially in dense topologies.
- **Mobility Dynamics:** High speed mobility may lead to frequent link breakages; the protocol needs to quickly adapt, which is difficult without high message rates.

2.6 Potential Improvements and Extensions

- **Adaptive Originator Message Interval:** Dynamically adjusting OGM broadcast frequency based on mobility, link stability, or node degree could reduce overhead while maintaining fresh routes.
- **Hierarchical or Clustered Routing:** Organizing nodes into clusters to limit the scope of OGMs or aggregate routing info may improve scalability.
- **Energy-Aware Metrics:** Incorporating node energy, battery life, and link stability into route choice could be beneficial in battery-constrained networks.
- **Integrating with Other Mesh / CPP Algorithms:** For UAV swarms or mobile robots, BATMAN could be combined with coverage path planning algorithms like DSP or Tsunami to offer robust communication backbone.
- **Enhanced 3D / Heterogeneous Scenarios:** The protocol's behavior under heterogeneous node capabilities (different antennas, mobility, etc.) or in 3D aerial/underground mesh should be further evaluated.
- **Security and Trust:** Extensions for authentication, resilience to misbehaving nodes, and preventing routing attacks (e.g. false originators) are important.

Chapter 3

Understanding Communication

3.1 MAC and IP Addresses

A **MAC (Media Access Control) address** is a unique 48-bit identifier assigned to a network interface card (NIC). It operates at the **Data Link Layer (Layer 2)** of the OSI model and is essential for local communication between nodes. MAC addresses facilitate neighbor discovery, frame forwarding, and link-level identification in networks without centralized infrastructure. In decentralized networks, such as MANETs, MAC addresses are used to maintain routing tables and ensure proper packet delivery over multiple hops.

An **IP (Internet Protocol) address** functions at the **Network Layer (Layer 3)** and provides logical addressing for end-to-end communication across different networks. IPv4 addresses consist of 32 bits, whereas IPv6 uses 128 bits for larger address space. In mobile networks, nodes often require dynamic IP allocation because devices join or leave frequently, and topology changes occur continuously. The combination of MAC and IP addresses ensures that packets can be correctly routed from source to destination while identifying neighboring nodes for link-level forwarding.

3.2 OSI Layers

The **Open Systems Interconnection (OSI)** model divides network communication into seven layers:

1. **Physical Layer:** Responsible for transmitting raw bits over the communication medium, including wireless radio signals.
2. **Data Link Layer:** Manages MAC addresses, framing, and error detection for reliable local communication.
3. **Network Layer:** Handles logical addressing, routing, and path determination between nodes, typically using IP addresses.
4. **Transport Layer:** Provides end-to-end communication with protocols such as TCP (reliable) or UDP (connectionless).
5. **Session Layer:** Maintains sessions or connections between applications.
6. **Presentation Layer:** Ensures data is in the correct format, supporting encryption, compression, and translation.
7. **Application Layer:** Interfaces with end-user applications and services, like web browsers or email clients.

For wireless ad hoc communication, the Data Link and Network layers are particularly important. Protocols like BATMAN and OLSR rely on Layer 2 neighbor discovery and Layer 3 routing mechanisms to maintain connectivity and adapt to network topology changes dynamically.

3.3 MANET and WANET

A **Mobile Adhoc Network (MANET)** is a self-configuring network of mobile nodes that communicate wirelessly without centralized infrastructure. Nodes act as both hosts and routers, forwarding packets for other nodes. MANETs are characterized by:

- Dynamic topologies due to node mobility.
- Limited wireless bandwidth and variable link quality.
- Decentralized management, requiring adaptive routing protocols.

Wireless Adhoc Networks (WANETs) include all types of decentralized wireless networks, such as MANETs, Vehicular Adhoc Networks (VANETs), and wireless mesh networks. WANETs may consist of both mobile and static nodes, providing self-healing and self-configuring network capabilities. These networks extend coverage and enable communication in areas lacking traditional infrastructure.

3.4 Routers and Access Points

A **Router** is a networking device that operates primarily at Layer 3 (Network Layer) of the OSI model. It forwards packets between different networks based on IP addresses, determines optimal paths for data, and performs functions such as Network Address Translation (NAT) and firewall filtering.

An **Access Point (AP)** operates at Layer 2 (Data Link Layer) and bridges wireless devices to a wired network. It broadcasts SSIDs, handles authentication and encryption (e.g., WPA2/WPA3), and extends wireless coverage. Unlike fixed networks, in MANETs, nodes can function as both routers and access points dynamically, allowing peer-to-peer communication without centralized control.

3.5 UDP vs TCP

Transmission Control Protocol (TCP) and **User Datagram Protocol (UDP)** are core Transport Layer protocols:

- **TCP:** Connection-oriented, ensures reliable delivery with error checking, acknowledgments, and retransmissions. Suitable for applications where data integrity is critical but adds overhead and latency.
- **UDP:** Connectionless, provides no delivery guarantees or ordering, resulting in low-latency communication. Ideal for real-time applications, streaming, and control messages in MANETs where network conditions change rapidly.

Chapter 4

Simulation of Drone Communication using UDP Sockets and Wifi-Direct

4.1 Simulation

4.1.1 Overview

The code uses python library named **socket** which is helpful in establishing connections.

4.1.2 System Architecture

The simulation uses the following components:

- **UDP Communication:** Each drone binds to a specific IP address and port combination, allowing it to send and receive messages over the UDP protocol. UDP is chosen for its low-latency, connectionless communication, mimicking fast broadcast or peer-to-peer drone messaging.
- **Graphical User Interface:** A Tkinter-based GUI displays sent and received messages, along with an input field for manually sending messages. This interface provides visual feedback, allowing users to verify communication and observe network behavior.
- **Threading:** Separate threads handle receiving messages and sending periodic messages to prevent blocking the GUI. This ensures that the interface remains responsive while background network communication continues.
- **Logging and Timestamps:** All messages are timestamped, providing a sequential record of communication. Automated messages are color-coded differently from manual messages and received messages to improve readability.

4.1.3 Operation

Upon starting the simulation, each drone is assigned a local IP address and port, as well as a target remote IP and port corresponding to the other drone. The simulation performs the following operations:

1. **Listening:** Each drone continuously listens for incoming UDP messages on its local port. Messages from the remote drone are decoded and displayed in the GUI with a timestamp.
2. **Automated Sending:** Each drone periodically sends messages to its peer, simulating continuous data exchange, such as telemetry or status updates.
3. **Manual Sending:** Users can type messages into an input field and send them immediately, simulating manual commands or urgent messages in the field.
4. **Real-Time Logging:** Messages are displayed in a scrollable text area, with automatic scrolling and color-coded categorization for easy identification.

This setup enables users to visualize bidirectional communication between drones and test various message patterns without the need for real network hardware.

4.1.4 Results

- **Single-Machine Simulation:** Two instances of the drone program were run on the same laptop using different local ports. The drones were able to exchange messages continuously, both automatically and via manual input, with all messages properly logged and timestamped in the GUI. This validated the correctness of the UDP-based communication logic and the responsiveness of the GUI interface.
- **Local Network Test:** Two laptops connected to the same Wi-Fi network were configured to run the drone programs. Each drone successfully sent and received messages over the network using the assigned local IP addresses and ports. The communication was stable, and messages were delivered in near real-time, demonstrating that the system can handle real network interfaces under ideal LAN conditions.

4.1.5 Limitations

While this simulation provides a useful testing environment, several limitations prevent its direct application in real-world drone operations:

- **Network Assumptions:** The current simulation assumes each drone has an assigned IP address (loopback 127.0.0.1 in testing), which is unrealistic in environments like forests, open fields, or disaster zones without network infrastructure.
- **UDP Reliability:** UDP is inherently unreliable, meaning messages can be lost or arrive out of order. In a real-world drone network, packet loss may affect coordination and safety.
- **Single-Machine Limitation:** The simulation runs on a single machine, which simplifies communication. In physical deployments, drones require wireless transceivers and robust handling of interference, distance, and obstacles.
- **No Dynamic Network Discovery:** Real-world drones cannot rely on pre-assigned IPs and ports. They need mechanisms for ad-hoc discovery, routing, and addressing without centralized infrastructure.
- **No Environmental Considerations:** Factors like signal attenuation, multipath interference, or collision avoidance are not modeled.

4.2 Wi-Fi Direct for Drone Communication

4.2.1 Introduction

Wi-Fi Direct is a peer-to-peer wireless networking technology that allows devices to connect directly without relying on traditional network infrastructure such as routers or access points. It establishes a network in which one device acts as the *Group Owner* (GO), functioning similarly to a soft access point, while other devices connect as clients. The GO manages IP address assignment, connection handling, and overall network coordination, enabling simple direct communication between devices.

4.2.2 Wi-Fi Direct for Drone-to-Drone Communication

Wi-Fi Direct can theoretically facilitate direct communication between drones over UDP without requiring a router or network infrastructure. Each drone would either act as the GO or connect as a client. On capable hardware, like Linux-based Jetson boards (Nano, Xavier NX, Orin Nano) with Wi-Fi dongles supporting P2P mode (e.g., Atheros AR9271), drones can run the full networking stack, handle GO negotiation, and execute UDP communication applications.

GO Setup and Negotiation

During network formation, a *Group Owner* is elected through a negotiation process. Each device provides a *GO Intent* value (0-15), with the higher value device becoming the GO. The GO then runs a soft AP, and clients connect to it. This process is managed in software by tools like `wpa_supplicant` and can be monitored using `wpa_cli`, Wireshark, or `iw`.

4.2.3 Limitations for Multi-Drone Swarms

While Wi-Fi Direct supports direct communication, it exhibits several limitations for dynamic drone swarms:

- **Single-Hop Communication:** Only the GO can directly communicate with clients; client-to-client communication requires traffic to route through the GO, limiting flexibility.
- **Dynamic Topology Challenges:** Changing the GO or reconfiguring the group requires a complete group reformation, which is slow and disrupts ongoing communications.
- **Scalability Constraints:** Only one GO per group exists, which restricts fully distributed communication among multiple drones.

4.3 Conclusion

Although Wi-Fi Direct enables peer-to-peer communication without a traditional network, it is largely unsuitable for dynamic, multi-drone scenarios. The single-hop nature, the need for GO negotiation and network reformation, and lack of efficient multi-hop routing make it impractical for real-time drone swarm coordination in environments where drones constantly move and change roles.

Chapter 5

Establishing an Ad-Hoc Network Between Two Devices

5.1 IBSS (Ad-Hoc) Mode

The Independent Basic Service Set (IBSS) is a technical Wi-Fi mode that enables peer-to-peer communication without a central Access Point (AP). In IBSS, each node manages its own beaconing, timing synchronization, and link establishment, allowing devices to directly exchange frames using the IEEE 802.11 MAC protocol. Nodes communicate over a shared wireless channel, and multi-hop routing protocols (e.g., OLSR, BATMAN) can be implemented on top to enable data forwarding across the network. IBSS supports dynamic topology changes, but requires manual or protocol-driven IP configuration for higher layer networking.

5.2 Overview of Steps

The process involves several key steps:

1. Preparing the wireless interface.
2. Configuring the interface to IBSS (ad-hoc) mode.
3. Setting network parameters such as SSID, channel, and IP addresses.
4. Verifying connectivity via tools like `ping` and `netcat`.

5.3 Detailed Procedure

5.3.1 Wireless Interface Preparation

Initially, we inspected the wireless hardware using:

- `iw dev`: Lists all wireless interfaces and their current configuration.
- `iw list`: Displays the capabilities of the wireless device, including support for IBSS mode.
- `ip link set <interface> down/up`: Brings the wireless interface down or up to apply changes.

Before making changes, we stopped the NetworkManager service (`sudo systemctl stop NetworkManager`) to avoid conflicts during manual configuration.

5.3.2 Setting IBSS Mode and Ad-Hoc Parameters

Once the interface was ready, we changed it to ad-hoc mode using:

- `sudo iw dev <interface> set type ibss`: Configures the interface to IBSS mode.
- `sudo iwconfig <interface> mode ad-hoc`: Legacy command confirming ad-hoc mode.
- `sudo iwconfig <interface> essid "ADHOC"`: Sets the network name (SSID) for the ad-hoc network.
- `sudo iwconfig <interface> channel 6`: Configures both devices to communicate on the same channel.

5.3.3 IP Configuration

Each device was manually assigned a static IP within the same subnet to enable communication:

- `sudo ifconfig <interface> 192.168.10.1 netmask 255.255.255.0 up`: Assigns IP and netmask to the first device.
- The second device was assigned 192.168.10.2 in a similar manner.

This ensures both devices can exchange packets using UDP or TCP protocols without DHCP or central routing.

5.3.4 Testing Connectivity

Connectivity was verified using basic network utilities:

- `ping 192.168.10.2`: Confirms that the second device is reachable.
- `nc 192.168.10.2 12345`: Tests TCP connections to a specific port.
- `scp <file> user@192.168.10.2`: Used to transfer files between devices over the ad-hoc network.
- `nc -l 12345 > file.pdf`: Listens on a port to receive files from the other device.

k without any external network infrastructure.

5.4 Results

We were successful in establishing communication between two devices using the ad-hoc (IBSS) network. UDP messaging and file transfer tests confirmed reliable peer-to-peer connectivity over the assigned IP addresses. Specifically, we tested:

1. Communication between two laptops connected via Wi-Fi in IBSS mode.
2. Data exchange using UDP sockets and TCP-based `netcat` commands.

Attempts to extend the setup to three devices were unsuccessful due to port conflicts and network management issues. We plan to retry the multi-device setup using different ports for each pair of devices to avoid conflicts and achieve stable multi-node communication.

5.5 Discussion and Limitations

- IBSS mode is single-hop by default. For multi-hop communication, additional routing protocols like OLSR or BATMAN must be implemented.
- Manual IP configuration is needed for each node, which may not scale well for large dynamic networks.
- Network performance (throughput, latency) is highly dependent on hardware capabilities and wireless channel conditions.