



AFRL-RI-RS-TR-2021-168

QUALITY-OF-AUTONOMY-ORIENTED AND QOS-AWARE TRANSPORT AND ROUTING PROTOCOLS FOR CONTESTED AIRBORNE SWARM NETWORKS

SAN DIEGO STATE UNIVERSITY

SEPTEMBER 2021

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (<http://www.dtic.mil>).

AFRL-RI-RS-TR-2021-168 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /

ELIZABETH BENTLEY
Work Unit Manager

/ S /

GREGORY HADYNSKI
Assistant Technical Advisor,
Computing & Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) SEPTEMBER 2021		2. REPORT TYPE FINAL TECHNICAL REPORT		3. DATES COVERED (From - To) JUL 2018 - MAR 2021	
4. TITLE AND SUBTITLE Quality-of-Autonomy (QoA)-Oriented and QoS-Aware Transport and Routing Protocols for Contested Airborne Swarm Networks				5a. CONTRACT NUMBER N/A	
				5b. GRANT NUMBER FA8750-18-1-0023	
				5c. PROGRAM ELEMENT NUMBER 62788F	
6. AUTHOR(S) Sunil Kumar Fei Hu				5d. PROJECT NUMBER T2SI	
				5e. TASK NUMBER SD	
				5f. WORK UNIT NUMBER SU	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) San Diego State University San Diego CA 92182-1309				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/RITGB 525 Brooks Road Rome NY 13441-4505				10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/RI	
				11. SPONSOR/MONITOR'S REPORT NUMBER AFRL-RI-RS-TR-2021-168	
12. DISTRIBUTION AVAILABILITY STATEMENT Approved for Public Release; Distribution Unlimited; September 2021. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT The resource constraints and dynamic nature of the swarm bring critical challenges to the design of UAV routing protocols. To address the these issues, adaptive, low-complexity and low-overhead, distributed cross-layer routing and transport layer protocols are required. These protocols should also consider the quality of autonomy (QoA) status of the UAV network. In this project, we have developed three novel, adaptive routing protocols and two transport layer protocols for fully distributed UAV networks with different QoA levels.					
15. SUBJECT TERMS Cross-layer protocols, inter-swarm communication, intra-swarm communication, routing					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 136	19a. NAME OF RESPONSIBLE PERSON ELIZABETH BENTLEY
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U			19b. TELEPHONE NUMBER (Include area code) NA

TABLE OF CONTENTS

Sections	Page
List of Figures.....	iv
List of Tables.....	ix
1.0 SUMMARY	1
2.0 INTRODUCTION.....	4
3.0. DESIGN OF AN ADAPTIVE OLSR ROUTING PROTOCOL FOR UAV NETWORKS.....	6
3.1. Introduction	6
3.1.1. Need for a Cross-Layer Routing Protocol	6
3.1.2. Introduction to OLSR	6
3.2. Related Work on Finding Stable and Long-Lasting Routes.....	7
3.3. Methods, Assumptions, and Procedures.....	8
3.3.1. Description of Proposed Adaptive OLSR Protocol.....	8
3.4. Results and Discussion	18
4. 2D SKELETON ROUTING: MODEL, ALGORITHMS AND PROTOCOLS	19
4.1. Introduction	19
4.2. Related Work.....	21
4.3. Methods, Assumptions, and Procedures.....	24
4.3.1. System Model.....	24
4.3.2. Leaf-like Pipe Framework.....	26
4.3.3. Skeleton-based Routing.....	32
4.4. Results and Discussion	37
4.4.1. Swarm Construction	38
4.4.2. Establishment of Leaf-Like Routing Pipe	38
4.4.3. SSR Performance in Light External Traffic	39
4.4.4. SSR Performance when Network is Congested	41
4.4.5. SSR Performance with TSDP.....	43
4.4.6. Swarm Morphing.....	44
5. 3D SKELETON ROUTING: MODEL, ALGORITHMS AND PROTOCOLS	46
5.1. Introduction	46

5.2. Related Work.....	49
5.3. Methods, Assumptions, and Procedures.....	49
5.3.1. TSS Extraction Protocol.....	49
5.3.2. Bottleneck-aware Skeleton Routing.....	56
5.3.3. Swarm-Adaptive Transformative Routing.....	61
5.4. Results and Discussion.....	67
5.4.1. TSS Extraction.....	67
5.4.2. Performance of TSS-based Skeleton Routing.....	68
6. DRL-BASED ADAPTIVE TRANSPORT LAYER CONTROL SCHEME.....	73
6.1. Introduction.....	73
6.2. Related Work.....	74
6.3. Methods, Assumptions, and Procedures.....	75
6.3.1. Adaptive Network Coding for Congestion Control.....	75
6.3.2. DRL Aided Transport Layer Control.....	78
6.4. Results and Discussion.....	84
7. EMANE-BASED SIMULATION OF MULTIPATH BRIDGE ROUTING.....	89
7.1. Introduction.....	89
7.2. Related Work.....	90
7.3. Methods, Assumptions, and Procedures.....	91
7.3.1. Overview of Network Emulators: CORE and EMANE.....	91
7.3.2. Mobility Control and Progressive Bridge Establishment.....	91
7.3.3. Edmonds-Karp Algorithm for Multipath Routing.....	96
7.3.4. Design and Implementation of MP-UDP.....	99
7.4. Results and Discussion.....	103
7.4.1. Performance of MP-UDP Protocol.....	103
7.4.2. Performance of MP-UDP with Directional Antenna.....	107
7.4.3. MP-UDP with Omnidirectional Antenna for Mobile Nodes.....	109
8.0 CONCLUSIONS.....	114
9.0 REFERENCES.....	116
LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS.....	125

LIST OF FIGURES

Figure	Page
Figure 1. UAV Swarm Network in Civilian Applications	4
Figure 2. LLT Computation When Both UAVs fly in a Curve.....	11
Figure 3. LLT Computations When One UAV Flies in a Curve While the Other Moves in a Straight Direction at an Angle in the Range $[0, 2\pi]$ With Respect to the X-Axis	12
Figure 4. The Link Between a UAV Pair can Break (a) Without Any Trajectory Change (See Left Image) or (b) After Multiple Link Trajectory Changes (See Right Image)	13
Figure 5. The Channel Access Mechanism for a 3-Hop Topology Where Each Node is Equipped with an Omnidirectional Antenna and Source Node A Transmits Packets to Destination Node D Over the Route A-B-C-D in Over a Single Frequency Band. Here, the Source Node A Forwards its Packet to Node B in t_1 Time Slot. In t_2 and t_3 Time Slots, Node A shouldn't Communicate to Prevent Collisions at Node B. Therefore, Node A can Access the Channel Once in Every Three Time Slots.....	15
Figure 6. Modified Hello Packet used in Our Proposed Scheme, where the Red Fields have been Newly Added to the Standard OLSR Hello Packet. The Figure is Adapted from [19]	17
Figure 7. Modified TC Packet used in Our Proposed Scheme, where the Red Fields have been Newly Added to the Standard OLSR TC Packet. The Figure is Adapted from [19]	17
Figure 8. Different Swarm Architectures: (a) Swarm Cooperation, (b) Mission-Driven Formation	19
Figure 9. SSR Modules	26
Figure 10. Examples of Swarm Formations with Skeleton Consisting of 8 Bones.....	27
Figure 11. (a) Parent-Children Relation Between Different Node Types. Here, an Arrow Points from a Parent to its Children. (b) Priorities of Different Swarm Messages	27
Figure 12. Format of the Command Message	28
Figure 13. Comparison of Data Forwarding Paths in SSR and MAP	29
Figure 14. An Example of Assigning Geo-Addresses	30
Figure 15. An Example of SSR Routing Table	31
Figure 16. Finding the Hop-Count Distance-based on Geo-Address Information and Skeleton Structure	32
Figure 17. A Leaf-Like Pipe Composed of Inter-Connected Paths. Each Link is Associated with a Forwarding Probability	33
Figure 18. (a) Upper Bound on the Drop Probability in the First Step of TSDP, (b) Finding the Drop Probability of a Packet in the Second Step Based on its Probability of On-Time Delivery to the Destination.....	36
Figure 19. (a) Initial Positions of UAVs Randomly Distributed in a 10 km x 10 km Rectangular	

Area; (b) Square Formation Based on the Skeleton Concept; (c) Triangular Formation.....	37
Figure 20. Establishing a Leaf-Like Routing Pipe Based on the Propsoed Geometric Addressing: (a) A Large Leaf in Square Formation; (b) A Small Leaf in Square Formation; (c) A Small Leaf in Triangular Formation	38
Figure 21. Throughput Comparison for Different Forwarding Action Selection Schemes When Network Traffic is Light.....	39
Figure 22. Received Packet Delay Comparison for Different Forwarding Action Selection Schemes When Traffic is Light.....	40
Figure 23. Total Delay Comparison for Different Forwarding Action Selection Schemes When Traffic is Light.....	40
Figure 24. Throughput Comparison for Different Action Selection Schemes (with Congestion)	41
Figure 25. Delay Comaprison for Different Action Selection Schemes (with Congestion)	42
Figure 26. (a) External Load Distribution Due to Other Applications; (b) Load Distribution of SSR Traffic with Random Action Selection under Congestion; (c) Load Distribution of SSR Traffic for $\tau = 1$ under Congestion; (d) Load Distribution of SSR Traffic for $\tau = 0.2$ under Congestion; (e) Load Distribution of SSR Traffic for $\tau = 0.05$ Under Congestion; (f) Load Distribution of SSR Traffic for $\tau = 0.05$ without Congestion.....	43
Figure 27. Throughput Comparison with and without the Use of TSDP in Congested Scenario.	44
Figure 28. Delay Comparison with and without TSDP in Congested Scenario.....	44
Figure 29. Swarm Morphing from a Square to Triangular Formation and the Changes of the Leaf-Like Routing Pipe: (a) Initial Leaf-Like Pipe in the Square Formation; (b) A Snapshot of Swarm at the Middle of Morphing; (c) The End Pipe in the Triangular Formation. The Morphing Process Takes About 125 s.....	45
Figure 30. Morphing Distortion Measurement for the Scenario Shown in Figure 29	45
Figure 31. Nervous-System-Inspired Routing Structure	47
Figure 32. Need for Bottleneck-Aware Routing	48
Figure 33. (1) Coarse-to-Fine Surface Detection, (2) Imperfect Surface Detection	50
Figure 34. 3D Localization Result	51
Figure 35. Our TSS Extraction Scheme can Hadle a Faulty 3D Surface.....	52
Figure 36. Practical, Distributed Scheme to Build Network Skeleton.....	54
Figure 37. An Illustration of Building the Stemlets	56
Figure 38. (a) Geodesic Distance, (b) Injectivity Radius, (c) Find Bottleneck Nodes.....	57
Figure 39. Swarm Segmentation and Bottleneck Identification.....	57
Figure 40. (a) Mice Traffic (Command Data): The Lowest-Cost Inter-Region Path, (b) Inter- Region Routing: Cost Graph	58
Figure 41. Illustration of Multi-Region and Multi-Route for Elephant Traffic	59

Figure 42. (1) Build Interference-Free Multipaths in the Same Segment, (2) Use “Stemlets” to form Multipaths	60
Figure 43. The Concept of the Skeleton Prediction Zone (SPZ).....	62
Figure 44. Illustration of RH Cases (Based on Different Swarming Patterns)	63
Figure 45. Routing for Swarm Splitting.....	63
Figure 46. Routing When Swarms Merge.....	64
Figure 47. Routing for the Axle Expansion Case.....	64
Figure 48. Handle General Route Transformation Case: (1) Before Micro-Adjustment; (2) After Micro-Adjustment	65
Figure 49. Triangulation-based 3D Node Localization.....	67
Figure 50. (1) Large-Scale UAV Network; (2) Ground Truth of Surface Nodes; (3) Detected Surface	67
Figure 51. (1) Original 3D Shape; (2) L_1 -Medial Skeleton; (3) Bottleneck Detection	68
Figure 52. Throughput and Delay Performance for 1-3 Traces in Stable Swarms	69
Figure 53. Throughput and Delay Performance for 1-3 Traces in Dynamic Swarms.....	69
Figure 54. Throughput and Delay Performance for Smooth Swarm and Fixed Packet Generation Rate (= 700 packets/ s)	70
Figure 55. Throughput and Delay Performance for Dynamic Swarm and Fixed Packet Generation Rate (=700 packets/s)	71
Figure 56. Time Consumption Comparison among Routing Algorithms.....	72
Figure 57. Deep Reinforcement Learning Model for UAV Networks.....	74
Figure 58. Single-Path Communication	77
Figure 59. MPSB Communication	78
Figure 60. Architecture of Deep Learning Network	79
Figure 61. Feedforward Propagation using Symbol-to-Symbol Approach.....	80
Figure 62. DL Center Deployment.....	84
Figure 63. Data Flow Frameworks.....	85
Figure 64. Performance Comparison of the ABC-based Congestion Control Scheme for Single-Path (SP) Communication	87
Figure 65. Performance Comparison of ABC-based Congestion Control Scheme for MPSB Communication	88
Figure 66. An Example of an Inter-Swarm Communication Bridge	89
Figure 67. Tolerance Molecular Force Model for UAV Mobility Control	93
Figure 68. Interactions among Nodes based on Tolerance Molecular Force Model	93
Figure 69. Example of Obstacle Avoidance Mobility Control When Applying Tolerance	

Molecular Force Model	94
Figure 70. Progressive Communication Bridge Formation: First Bridge. (The Nodes for Bridge Formation should Arrive at their Expected Bridge Locations at Time = 45 sec)	95
Figure 71. Progressive Communication Bridge Formation: Second Bridge. (The Nodes for Bridge Formation should Arrive at their Expected Bridge Locations at Time = 90 sec)	95
Figure 72. The EKA Progress and EKA-Based Multipath Detection. (a) An Example of an EKA-Oriented Network Model; (b) EKA Progresses to the 1 st Hop Neighbours; (c) EKA Progresses to the Second Hop; (d) EKA Progresses to the Third Hop; (e) The First Path is Established; (f) Possible EKA Running Result with Three Paths; (g) An Example of an Ad-Hoc Network; (h) Apply Enhanced EKA on the Topology of (g); (i) Apply Enhanced EKA on (g), Here We Invoke the Timer on the Receiver Side after the First Path-Seeking Message Arrives	97
Figure 73. Adjusted EKA is Applied to Previous Network	98
Figure 74. MP-UDP Sublayer Design	100
Figure 75. MP-UDP Functions for Different Node Roles.....	101
Figure 76. Bridge Topology (2-Hop) Considered in Simulations.....	104
Figure 77. Bridge Topology (3-Hop) Considered in Simulations.....	104
Figure 78. MAC and Physical Layer Settings in CORE	105
Figure 79. Traffic Split When Applying MP-UDP in a 2-Hop Bridge. Here, the Red and Orange Curves are Overlapped with the Blue Curve	106
Figure 80. Throughput vs. Sending Rate for 2- and 3-Hop Topologies with Omnidirectional Antennas	107
Figure 81. 3-Hop, 2-Bridge Network with Directional Antennas	107
Figure 82. 3-Hop, 3-Bridge Network with Directional Antennas	108
Figure 83. Packet Loss Rate (PLR) vs. Data Sending Rate Channel Properties Provided by EMANE IEEE 802.11b	108
Figure 84. Throughput Performance of the MP-UDP Protocol for a Network with Directional Antennas	109
Figure 85. A Network Topology with Node Mobility	110
Figure 86. Throughput When Using UDP with OLSR Protocol.....	110
Figure 87. Throughput When Using MP-UDP	111
Figure 88. Simulation Scenarios for Overhead Analysis. Moving Range of Nodes are Highlighted by Red Rectangle Boxes	112
Figure 89. MP-UDP vs. General UCP and OLSR Throughput Tests	112
Figure 90. MP-UDP vs. General UDP and OLSR Overhead Ratio	113

LIST OF TABLES

Table	Page
Table 1. Simulation Parameters Used to Create Trajectories for UAV Swarm.....	9
Table 2. PDR Comparison Between Standard OLSR and Our Scheme	18
Table 3. Routing Protocols Comparison for UAV Network.....	23
Table 4. Simulation Parameters	37
Table 5. TSDP Simulation Parameters.....	44
Table 6. Simulation Parameters for Single-Path Communications.....	86
Table 7. Block Sizes of Transport Layer Control for Single Path Communication	87
Table 8. Block Size of Transport Layer Control for MPSB Communication.....	88
Table 9. Bridge Construction Algorithm	94
Table 10. Path-Seeking (PS) Packet Format	101
Table 11. Path-Seeking ACK Packet Format.....	102
Table 12. Multipath Forwarding Table (An Example)	102
Table 13. MP-UDP Packet Format	103
Table 14. Multi-Interface PS ACK Packet Format	109

1. SUMMARY

A swarm of unmanned aerial vehicles (UAVs) requires the transmission of mission-related data across the network. The resource constraints and dynamic nature of the swarm bring critical challenges to the design of UAV routing protocols. These challenges become even more difficult to address for a distributed, multi-hop network topology, which does not have a supervisory node for centralized control. Most of the conventional ad hoc routing schemes are not intelligent and cannot adapt to the dynamic nature of UAV swarming networks. On the other hand, some artificial intelligence (AI)-based routing schemes may consume significant computational resources in the UAVs.

To address the above issues, adaptive, low-complexity and low-overhead, distributed cross-layer routing and transport layer protocols are required. These protocols should also consider the quality of autonomy (QoA) status of the UAV network. In this project, we have developed three novel, adaptive routing protocols and two transport layer protocols for fully distributed UAV networks with different QoA levels.

The first routing scheme discussed in this report is an adaptive optimized link state routing (OLSR) protocol which (i) uses a novel multi-parametric (which are hop count, route lifetime, buffer occupancy and effective link capacity) route selection metric to select a robust, least-congested route, and (ii) proactively switches routes to prevent packet transmission over a short-lived, broken, or congested routes. Our scheme achieves a higher flow throughput and reduced routing computation overhead, compared to the standard OLSR protocol. Particularly our enhanced OLSR scheme can adapt to the QoA parameter (the swarm core stability) to find proper relay points and improve the throughput.

Next, our two-dimensional (2D) adaptive routing protocol, namely *skeleton-based swarm routing (SSR)*, is described. SSR exploits an intelligent online learning algorithm and the topology features of the mission-driven UAV swarm to distribute the traffic over optimal routes. The ‘QoA parameter swarm moving trend’ is used to identify the skeleton nodes. Here, the *skeleton* represents the most stable parts of the swarm formation. The SSR architecture consists of three modules: 1) A geometric addressing module, which assigns geometric coordinates to each node based on the swarm skeleton structure; 2) A leaf-like routing pipe which allows the selection of multiple candidate routes around the shortest path; 3) An intelligent low-complexity learning model which determines how to distribute the packets inside the routing pipe to achieve load-balanced, high-throughput transmissions. The proposed routing scheme can also facilitate the UAV formation construction and morphing. The simulation results show that the proposed SSR protocol can noticeably improve the network performance compared to the single path routing schemes, such as the ad-hoc on-demand distance vector (AODV) and link-quality and traffic-load aware optimized link state routing (LTA-OLSR) protocols.

Next, our extensions to the above 2D skeleton routing scheme to a 3D skeleton routing scheme are described. A challenging issue for a 3D UAV network is addressed in this section - how do we efficiently establish and maintain one or multiple routes among swarm *regions* (i.e., groups of UAVs), during the *dynamic* swarming process? Inspired by the human nervous system which can efficiently send brain signals to any tissue, we describe a novel 3D *transformative routing (3D-TR)* scheme, which consists of three novel designs: (1) *Proactive routing skeleton establishment*: Just like a human nervous system with a tree architecture, 3D-TR first identifies the *skeleton* of the UAV network, which has a *trunk-stem-stemlet (TSS)* structure. The *trunk* is in the relatively

stable area of the network (i.e., the central area), and *stems/stemlets* reside in the outer areas with more swarming movements. Such a TSS skeleton enables the easy establishment of the path(s) with low end-to-end delay and a high packet delivery rate. (2) *Reactive, transformative path selection*: The UAV swarming mission is considered as a smooth formation morphing process. The 3D-TR scheme can *smoothly* adapt to the formation changes through dynamic TSS adjustments. (3) *Bottleneck-aware route maintenance*: We propose an efficient scheme to recognize the *bottleneck* nodes in each path and build the diversified routing pipes with different thickness/stability levels. The proposed routing scheme is distributed (only uses *local* message exchanges among 1-hop neighbors) and does not require the GPS (global positioning system) positions of the UAVs. Extensive simulations are conducted to validate the performance of the proposed 3D-TR scheme during the dynamic UAV swarming process.

Next, an efficient transport layer protocol for UAV swarm network is described, which focuses on the congestion control. This is a challenging issue since many network factors can cause packet losses, such as dynamic swarming, high mobility, long-distance links (causing fading), frequent UAV tilting/vibrations, etc. These packet losses are interpreted as congestion events in traditional congestion control schemes, which simply retransmit the lost packets and, therefore, significantly reduce the data rate. In this project, a learning-based adaptive network coding scheme is proposed to handle the congestion events. In this scheme, a certain amount of redundancy codes is attached to the original data segments. When a loss is caused by random factors (such as radio interference), the lost segment is retrieved by the redundancy codes. On the other hand, when a loss is caused by serious congestion, the redundancy may not be enough to retrieve all the lost segments. Therefore, the sender will retransmit the lost segments and decrease the sending rate. The main contribution of our protocol design is to avoid coarse and inaccurate network coding parameter adjustments. Instead, we use a deep reinforcement learning (DRL) algorithm to determine the optimal coding rate of the network coding scheme. The past network *state-action* pairs (i.e., network conditions and the corresponding optimal coding rates) are collected and used in the DRL model as the training inputs. After the DRL network is trained offline, it can be executed in real time in the online testing phase. Our experimental results show that DRL-based network coding provides the improved quality of service (QoS) performance compared with TCP and general network coding schemes.

Finally, a protocol to support the intra-swarm and inter-swarm communication in the UAV swarm network is described. We have implemented a *bridge routing* structure to establish the route(s) for efficient inter-swarm communications. The bridge routing scheme is a progressive process with two functions: UAV formation control and data transmissions. Based on the bridge routing topology, we have implemented the multipath user datagram protocol (MP-UDP), as a sub-layer between the transport and application layers. Compared with the multipath transmission control protocol (MP-TCP), MP-UDP is more suitable to the dynamic network topologies since it does not use the high-overhead sliding window control. We have used the emulator tools, the Extendable Mobile Ad-hoc Network Emulator (EMANE) and Common Open Research Emulator (CORE), to implement and study the QoS performance of MP-UDP and bridge routing in UAV swarm networks. Our results demonstrate the improved throughput performance for inter-swarm communications.

In the next chapter, an introduction of our novel routing and transport layer protocols is provided, including a review of existing literature. The routing and transport layer protocols developed in

this project are described in Chapter 3, followed by their performance analysis in Chapter 4. Conclusions are given in Chapter 5.

This project has achieved fruitful outcomes: one invention disclosure, three accepted journal papers, one manuscript under review and two manuscripts under preparation. The detailed list is provided below.

Invention Disclosure:

1. Fei Hu, Qian Mao, Lei Hu, Sunil Kumar, *Intelligent High-Speed Unmanned Vehicle Communications via Bio-Inspired Multi-Beam Pipe Transmission*, March 2017.

Journal Papers:

1. L. Zhang, F. Hu, Z. Chu, E. Bentley, and S. Kumar, *3D Transformative Routing for UAV Swarming Networks: A Skeleton-Guided, GPS-Free Approach*, IEEE Transactions on Vehicular Technology, 2021.
2. N. Toorchi, F. Hu, E. Bentley, and S. Kumar, *Skeleton-Based Swarm Routing (SSR): Intelligent Smooth Routing for Dynamic UAV Networks*, IEEE ACCESS, Vol. 9, Dec. 2020, pp. 1286 – 1303.
3. Z. Chu, F. Hu, E. Bentley, and S. Kumar, *Model and Simulations of Multipath Bridge Routing for Inter-Swarm UAV Communications in EMANE/CORE*, International J. Modeling and Simulation, 2021.
4. Q. Mao, L. Zhang, F. Hu, E. Bentley, and S. Kumar, *Deep Reinforcement Learning (DRL)-Based Adaptive Transport Layer Control in UAV Swarm Networks*, under review.
5. S. Garg, A. Ihler and S. Kumar, *An Adaptive OLSR Routing Protocol for UAV Networks*, under preparation.
6. S. Devaraju, A. Ihler and S. Kumar, *A Connectivity Oriented Pheromone based UAV Mobility Model*, under preparation.

2. INTRODUCTION

Airborne networks (AN) play an important role in the next generation air transportation system, which can self-configure to provide a seamless, low cost and secure connectivity in real-time to a multitude of devices with stringent and dynamic quality of service (QoS) requirements, whenever and wherever needed [1]-[3]. Networks of autonomous UAVs are emerging as an essential component of AN. Examples of the applications of these UAV networks in civilian and military sectors are environmental sensing and disaster management, traffic and urban monitoring, search and rescue operations, patrolling, and relaying networks (see Figure 1) [1][4]-[6].



Figure 1. UAV Swarm Network in Civilian Applications

The main characteristics of UAV networks are high node density and fast mobility which result in a dynamic topology with high co-channel interference and frequent link disruptions. This leads to packet transmissions over a congested and/or broken route, which degrades the flow QoS. To address it, periodic topology change notifications are broadcasted in the network to select an alternative route, which increases the control and computational overheads [5][7]-[11], and energy consumption which can result in a node failure [5][11][12]. This reduces channel utilization, flow reliability, and increases latency. Moreover, an increase in the flow data rate and/or number of simultaneous flows further increases the congestion in the network [9][11][12], which makes seamless network connectivity a challenging task. To address the above-mentioned issues, an adaptive, low-complexity and low-overhead routing protocol is required which can select an optimal route for communication and change it preemptively to reduce the packet drop, and support different traffic flows of varying data rates, latency, and flow priority. In addition to the above-mentioned distinct network characteristics, these networks also accommodate application-specific mission requirements, which makes them significantly different from traditional wireless networks [13].

Many network protocols in literature consider a centralized control through one or more supervisory nodes to address these challenges. However, the networks used in the hostile environment are typically fully decentralized due to the dynamic nature of their mission. The centralized network protocols are not suitable for fully distributed networks because (i) they incur a large overhead and delay in frequently collecting traffic demands and statistics of all nodes in

the network in order to provide a feasible packet transmission schedule, (ii) their solutions become obsolete whenever topology and/or QoS requirements (e.g., flow priority and source-destination pairs) change, which requires retransmission of traffic demands and statistics of each node as discussed in the previous point, (iii) they cannot scale with the network size, and (iv) are not reliable because failure of central node(s) can disrupt the network operation.

These protocols should also consider the QoA of the UAV network. Our routing layer designs have considered the following QoA performance metrics:

- (1) Swarm changing rate (SCR): SCR is a critical QoA metric because it determines the protocol adaptivity. The UAV network may form different network topologies based on mission requirements. If the swarm shape changes, the routing protocol should adjust the path if local repair is impossible. A higher SCR requires a more robust protocol. Our routing schemes (including enhanced OLSR and skeleton routing protocols) can adapt to the high-SCR networks.
- (2) Swarm moving trend (SMT): This metric represents the moving direction/speed of the entire swarm or one or more regions. A well-designed routing protocol should establish the path by following such a trend. Our 2D/3D skeleton routing schemes can extract the SMT patterns and build a path along the trend.
- (3) Swarm core stability (SCT): SCT measures how stable the core part of the network is. Typically, when a UAV network changes formation, it tries to keep the core part in a stable status and only move the marginal nodes to form a new shape. Such a core stability reduces unnecessary node movements and node confliction rate. Our skeleton routing scheme can extract the stable core skeleton and use it to establish a new path.

Note that QoA and QoS can be tightly coupled with each other to achieve high-throughput and low-delay routing. For example, when the QoA parameter SCR is high, the QoS metric (path throughput) will be low. However, a well-designed routing scheme can minimize the impact of QoA on QoS.

To meet the challenges discussed above, we have developed three novel, adaptive routing protocols and two transport layer protocols for fully distributed UAV networks with different QoA levels. These protocols consider the features of the application as well as the physical layers, whenever needed. A swarm of low SWaP (size, weight, and power) fixed-wing UAVs is considered [2][6]. Each UAV is assumed to know its GPS location, to cooperate with its 1-hop neighbors, to not be able to make sharp turns which results in a smooth trajectory, to have the collision avoidance capability, and to be half-duplex.

3. DESIGN OF AN ADAPTIVE OLSR ROUTING PROTOCOL FOR UAV NETWORKS

3.1 Introduction

The routing protocols can be categorized into geographic, reactive, and proactive routing protocols [7][12][14]. In geographic routing protocols (e.g., aeronautical routing protocol [15], greedy perimeter stateless routing protocol [16]), location servers store the updated GPS locations of all nodes in a fixed geographical region, which a source node queries to obtain the recent location of its destination node. However, a separate routing scheme (other than geographic routing) is required for the location servers to maintain the updated location information. Furthermore, geographic routing protocols are not suited for a distributed network with no central or regional servers.

Reactive routing protocol (e.g., AODV) [17], dynamic source routing [18]) discovers routes on-demand for a source-destination pair. These schemes have a smaller signaling traffic but a large route discovery delay when the route to destination is not available. Therefore, frequent link breaks in a highly mobile airborne network can significantly increase the route rediscovery delay. In addition, these schemes do not scale well as the route failure probability increases with the network size and mobility [5].

In the proactive routing protocol (e.g., OLSR) [19], destination-sequence distance-vector [20]), control messages are periodically exchanged among the nodes in the network, which incurs a large signaling overhead in a dense network. However, each node keeps the updated information for all nodes in the network and can immediately find a route to a destination node when needed, without any route discovery delay. Hence, it is suitable for dynamic topologies where source-destination pairs change with time.

3.1.1 Need for a Cross-Layer Routing Protocol

Since the application-specific requirements, such as traffic type (e.g., audio, video, statistics and sensor values of a node) and priority, generation rate, volume, reliability and latency tolerance, can vary due to the dynamic nature of the mission (which leads to varying QoS constraints), the protocol design in UAV networks becomes very challenging [13]. To address it, efficient cross-layer routing and MAC protocols are needed to find optimal route(s) for packet transfer such that the channel utilization is maximized [13]. However, the cross-layer protocols available in literature mostly consider a centralized network architecture [13], as discussed in Section 1.

Since OLSR is a widely used proactive routing protocol in the literature and is also well-suited for distributed networks, we design an adaptive, QoS-aware, cross-layer OLSR protocol for a distributed UAV network in this project. Our aim is to make the proposed protocol low complexity and low overhead for addressing the unique application-specific requirements as well as the challenges posed by dynamic topologies of swarms of UAVs.

3.1.2 Introduction to OLSR

The OLSR protocol uses two types of control packets - Hello and TC (topology control). Each node includes its 1-hop neighbor nodes in its Hello packet which is broadcasted periodically after

a Hello interval (the default value is 2s). Hello packets are used to construct the 1- and 2-hop neighbor sets (i.e., $N^1(X)$ and $N^2(X)$, respectively) at a node X .

Traditional link state routing uses classical flooding where each node forwards the control packet, which results in a huge control overhead and increases the packet collision probability. Whereas each node in OLSR finds the smallest subset of its $N^1(X)$ (which is called MPR (multi-point relay) set) to cover all the nodes in its $N^2(X)$ and includes it in its Hello packet to inform its $N^1(X)$. A node also maintains an MPR selector set to store those 1-hop neighbors which have included it in their MPR set. Only MPR nodes forward the control packets of its MPR selector node(s) in OLSR, which reduces the total number of control packet forwards and, thereby, the control overhead. Note that Hello packets are not forwarded in OLSR.

Each MPR node includes its MPR selector nodes in its TC packet which is broadcasted periodically after TC interval (default value is 5 s). A node uses TC packets to construct its *Topology Set* which stores information of the links between an MPR node and its MPR selectors. Note that the *Topology Set* of a node represents its current knowledge of the network. Each node in OLSR uses Dijkstra algorithm on the graph built using its *Topology Set* to find a shortest-hop route to the destination node.

3.2 Related Work on Finding Stable and Long-Lasting Routes

In a mobile network, the network topology changes with time which makes the identification of unbroken routes a challenging task [21]-[23]. In the absence of the topology change updates, a transmitter node retransmits a unicast packet seven times in the CSMA/CA (carrier sense multiple access with collision avoidance) MAC protocol before recognizing a link break, which reduces the channel utilization and increases the queuing delay for the remaining packets at that node [22]. Frequent topology changes can increase congestion in the network [22] which results in the packet drop due to buffer overflow and degrades the QoS of the flow [21].

To prevent the packet transmission over broken routes, each node in [22]-[23] includes the GPS locations of itself and its 1-hop neighbor nodes in its control messages to create a cartography of the network at each node. Based on these locations, the source node selects a route such that links along it do not break before the reception of new control messages. However, the control packet lengths in [22]-[23] increases significantly in a dense network which results in a higher control overhead and packet collision probability. In addition, periodic reconstruction of the cartography increases the computational overhead at each node which reduces its residual battery life.

The shortest-hop routes generally select the edge nodes. Consequently, the signal strength at the receiver node is minimized, which increases the packet loss ratio and reduces the PDR (packet delivery ratio) of the flow. Therefore, the routing scheme proposed in [24] differentiates links based on their RSSI (received signal strength indicator) values using Chebyshev inequality and prefers links with a lower variance in RSSI values for route selection. Since interference from neighbor nodes increases significantly in a dense network which causes inaccurate computation of RSSI values [25], it is ill-suited for the robust route selection.

A mobility-aware route selection scheme is proposed in [21] where the link stability is determined by the variance in the distance values (computed using GPS) of a UAV pair. UAV pairs with a smaller variance in distance values are expected to remain in the range of each other for a longer duration due to their similar mobility patterns. However, [21] fails to select links where UAVs come closer because of the high variance in their monotonically decreasing distance values.

An SVM (support vector machine)-based link failure prediction scheme is proposed in [26] which uses the SNR (signal to noise ratio) to classify the link stability. It reroutes the traffic when a link fails at an intermediate node. However, the route rediscovery at intermediate node can result in a longer *hop-count* route, which increases the latency and route failure probability [27].

To reduce the control overhead, each node in [28] uses the local information of its 1-hop neighborhood to select the best next-hop node for traffic forwarding. However, schemes like [28] fail to find globally optimal routes. The scheme discussed in [29] uses the distribution of the past *LLT* (link lifetime - the duration for which two nodes remained connected) and the current link age (i.e., the duration since the node pair is connected till the current timestamp) to estimate the *residual LLT* (i.e., the duration after which the link between the node pair would expire from the current timestamp). However, [29] does not consider the effect of a trajectory change which can result in an inaccurate *LLT* computation because the trajectory change can cause nodes to either come closer or move away [27].

A mathematical formulation is proposed in [30]-[31] to compute *LLT* for a node pair using their speed, directions of movement and current coordinates. However, these formulations are limited to ground vehicles and cannot be used for the airborne node where the direction of movement continuously changes due to the smooth mobility on a curve.

3.3 Methods, Assumptions, and Procedures

3.3.1 Description of Proposed Adaptive OLSR Protocol

We use a multi-parameter route selection metric which includes the *Hop Count*, *Route Lifetime (RLT)*, *Buffer Occupancy (BO)*, and *Link Capacity*. Here *RLT* represents the duration for which the route exists and is computed based on the link lifetimes (*LLT*). The *BO* of a node is the packet count in its queue. The *Link capacity* is the maximum number of packets that a link can service per second. Although many schemes, such as [10][11][29][32][33], use a combination of *RLT*, *BO* and *Hop Count* metrics for the route selection, none of them use *Link Capacity*. In addition, our route selection formulation is different from these schemes. The computation of these route selection metric parameters and the route selection metric are discussed below.

A. Airborne and UAV Mobility Models

The performance of a routing protocol partially depends on the node distribution in the network and connectivity among them [6][34]. The node connectivity depends on the node trajectory, and thereby, on the underlying mobility model [6][34]. Therefore, designing and evaluating useful routing protocols for airborne networks require mobility models which can produce realistic node movements [1][6][34][35]. Note that the words *Node* and *UAV* are used interchangeably in this document from here onward.

Several mobility models, including random walk, random waypoint, and Manhattan grid, have been proposed for MANETs in the literature [1][4][36]. However, they assume node movement in 2D space; hence, they cannot be used in airborne and UAV networks [1][3]-[6][36]-[40]. Further, a fixed-wing UAV follows a smooth trajectory due to its aerodynamic constraints [1][3][4] and cannot abruptly change its direction as its movement depends on its previous locations, speed and heading direction [3][34][41]. Therefore, a time-based mobility model is preferred for airborne networks where the node movement is controlled by mathematical equations [1][35][36][41]. The

paparazzi model [37], smooth-turn (ST) model [3], semi-random circular movement (SRCM) [38], and Gauss-Markov model [39][40] are time-based mobility models discussed in the literature.

In this report, we use ST mobility model in which each node independently selects a center and radius based on its history and rotates around the center in the clockwise (or counterclockwise) direction for a randomly selected duration, called *Wait Time*, which results in a smooth realistic trajectory [1][3]. Other time-based mobility models are ill-suited because they generate limited mobility patterns. For example, node movement in the paparazzi model is limited to eight possible trajectories [1][4]; nodes rotate around a static common center in SRCM model [1][4]; and Gauss-Markov model cannot reproduce typical UAV turns [1][4].

The above-mentioned mobility models use either a reflection boundary model in which a UAV makes a sharp 180° turn at the boundary or wrap-around boundary model which causes sudden node appearances and disappearances [36]. To address this issue, we use a buffer boundary model [36], which forces a UAV to select its new direction such that it does not go out of the boundary, while complying with its aerodynamics. Possible directions for a fixed-wing UAV to fly are clockwise, counterclockwise, and straight. Note that a UAV does not change its direction from clockwise to counterclockwise and vice-versa without flying in the straight direction for at least a small duration, to maintain its stability [34].

In addition, the turn radius R of a UAV depends on its velocity V as, $R = \frac{V}{\omega}$, where ω is the angular velocity. The *Wait Time* duration for which a node maintains its current trajectory is uniformly selected from the range $[minWaitTime, maxWaitTime]$, which corresponds to the minimum and maximum durations for which the node stays on its current trajectory.

B. Trajectory Generation

In our simulation, 50 UAVs are randomly placed in the simulation area. They move under the ST mobility model for 2 hours in each run at a constant speed of 70 km/H. Each UAV randomly selects its center, radius and *Wait Time* as discussed above. The parameters used to generate UAV trajectories are given in Table 1.

Table 1. Simulation Parameters Used to Create Trajectories for UAV Swarm

Simulation Area	8x8 km ²
Total Simulation Runs	100
Each Simulation Run (UAV Mobility Duration)	2 hours
Number of UAVs	50
UAV Speed	70 km/H
Minimum Turn Radius	1 km
$[minWaitTime, maxWaitTime]$	$[1, 3]$ min
Hello Message Interval	2 s
UAV Transmission Range	1 km

C. Computation of Link Lifetime (LLT)

We assume that UAVs cooperate with each other and have collision avoidance capabilities. Each UAV includes its trajectory information (GPS location, movement state (i.e., clockwise, counterclockwise, or straight), current center, radius, and slope (if UAV is moving in a straight direction)) in its Hello message which is periodically broadcast to its 1-hop neighbors. If a UAV does not know its trajectory, its center, radius, and movement state can be computed by using its three consecutive GPS locations, and its speed can be computed using the distance travelled (or angular displacement) during a Hello interval when the UAV goes straight (or travels on a curved trajectory).

When the current location and trajectory details of a UAV pair are known, the corresponding *LLT* is computed using the following steps:

Step 1: Find the coordinates of the future location for both UAVs at time t .

Step 2: The link between a UAV pair breaks when the distance between them exceeds the node transmission range. Hence, an equation with one unknown variable t is computed using,

$$\text{Distance between UAVs at time } t > \text{Transmission Range.}$$

Step 3: Compute root(s) of the above equation.

Step 4: Select the root which best approximates the equation of Step 2.

Note that two UAVs establish a link at the *Link Establishment Time* when they exchange their Hello packets for the first time. The link between the UAV pair terminates when they move out of each other's transmission range. This time is called *Link Termination Time*. Therefore, *LLT* is computed as,

$$LLT = \text{Link Termination Time} - \text{Link Establishment Time} \quad (1)$$

Based on the possible movement states for a UAV as discussed above, the following three cases are possible for a UAV pair. The *LLT* computation for each case is different as discussed below.

Case A. Both UAVs fly in a curve

For a UAV₁ and UAV₂ pair in Figure 2, assume their current GPS locations are $(X_0^{(1)}, Y_0^{(1)})$ and $(X_0^{(2)}, Y_0^{(2)})$, the current trajectory centers are (Cx_1, Cy_1) and (Cx_2, Cy_2) , the current radii are R_1 and R_2 , the velocities are V_1 and V_2 , and the movement directions are Dir_1 and Dir_2 , respectively. Here, Dir is -1 for clockwise direction and +1 for counterclockwise direction.

The angular velocity for the UAV pair is computed as, $\omega_1 = \frac{V_1 * Dir_1}{R_1}$ and $\omega_2 = \frac{V_2 * Dir_2}{R_2}$. The initial displacement at the link establishment time for each UAV is computed as, $\theta_1 = \tan^{-1} \left(\frac{(Y_0^{(1)} - Cy_1)}{(X_0^{(1)} - Cx_1)} \right)$ and $\theta_2 = \tan^{-1} \left(\frac{(Y_0^{(2)} - Cy_2)}{(X_0^{(2)} - Cx_2)} \right)$. Hence, the future location coordinates at time t for both UAVs are,

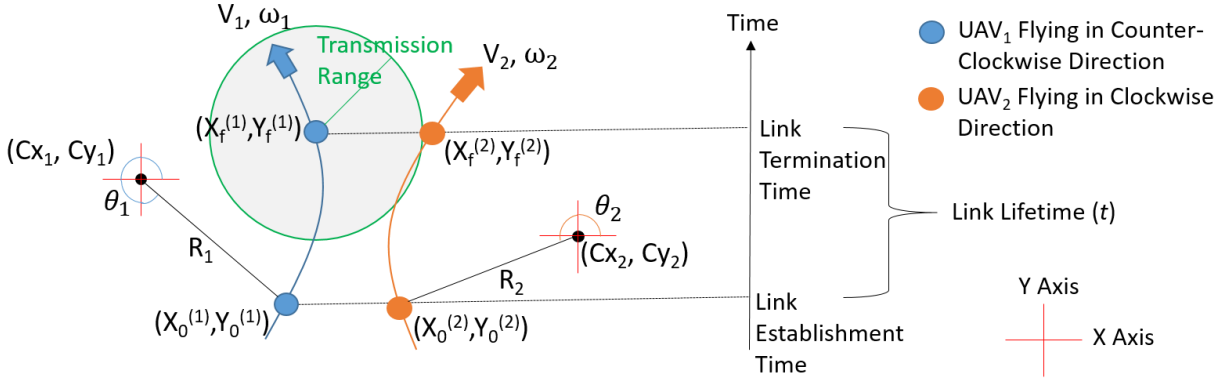


Figure 2. LLT Computation When Both UAVs Fly in a Curve

$$\begin{aligned} (X_f^{(1)}, Y_f^{(1)}) &= (Cx_1 + R_1 \cos(\theta_1 + \omega_1 t), Cy_1 + R_1 \sin(\theta_1 + \omega_1 t)) \text{ for UAV}_1, \text{ and} \\ (X_f^{(2)}, Y_f^{(2)}) &= (Cx_2 + R_2 \cos(\theta_2 + \omega_2 t), Cy_2 + R_2 \sin(\theta_2 + \omega_2 t)) \text{ for UAV}_2. \end{aligned}$$

The Euclidean distance between the UAV pair is called *Link Distance*, which is computed at time t as,

$$\text{Link Distance} = \left((X_f^{(1)} - X_f^{(2)})^2 + (Y_f^{(1)} - Y_f^{(2)})^2 \right)^{\frac{1}{2}} \quad (2)$$

Upon plugging the values of future coordinates of both UAVs, the right-hand side (RHS) of Eqn. (2) changes to,

$$\left((Cx_1 + R_1 \cos(\theta_1 + \omega_1 t) - Cx_2 - R_2 \cos(\theta_2 + \omega_2 t))^2 + (Cy_1 + R_1 \sin(\theta_1 + \omega_1 t) - Cy_2 - R_2 \sin(\theta_2 + \omega_2 t))^2 \right)^{\frac{1}{2}},$$

It can be further simplified to,

$$\begin{aligned} & \left((Cx_1 - Cx_2)^2 + (Cy_1 - Cy_2)^2 + R_1^2 + R_2^2 \right. \\ & \quad \left. - 2R_1R_2(\cos(\theta_1 + \omega_1 t - \theta_2 - \omega_2 t)) \right. \\ & \quad \left. + 2R_1((Cx_1 - Cx_2) \cos(\theta_1 + \omega_1 t) + (Cy_1 - Cy_2) \sin(\theta_1 + \omega_1 t)) \right. \\ & \quad \left. - 2R_2((Cx_1 - Cx_2) \cos(\theta_2 + \omega_2 t) + (Cy_1 - Cy_2) \sin(\theta_2 + \omega_2 t)) \right)^{\frac{1}{2}} \end{aligned} \quad (3)$$

The link between a UAV pair breaks when their *Link Distance* $\geq$ *Transmission Range*. If $a = \text{abs}(Cx_1 - Cx_2)$, $\text{sign}_1 = \frac{(Cx_1 - Cx_2)}{a}$, and $b = \text{abs}(Cy_1 - Cy_2)$, $\text{sign}_2 = \frac{(Cy_1 - Cy_2)}{b}$, we can use $\sin \alpha = \frac{b}{\sqrt{a^2 + b^2}}$, $\cos \alpha = \frac{a}{\sqrt{a^2 + b^2}}$ and $\alpha = \cos^{-1} \frac{a}{\sqrt{a^2 + b^2}}$ to simplify Eqn. (3) to,

$$\begin{aligned} \text{Link Distance} = & \left(a^2 + b^2 + R_1^2 + R_2^2 - 2R_1R_2(\cos(\theta_1 + \omega_1 t - \theta_2 - \omega_2 t)) \right. \\ & \left. + 2(\text{sign}_1)R_1\sqrt{a^2 + b^2} \cos(\theta_1 + \omega_1 t - (\text{sign}_1 * \text{sign}_2) \alpha) \right. \\ & \left. - 2(\text{sign}_1)R_2\sqrt{a^2 + b^2} \cos(\theta_2 + \omega_2 t - (\text{sign}_1 * \text{sign}_2) \alpha) \right)^{\frac{1}{2}} \end{aligned} \quad (4)$$

Case B. One UAV flies in a curve and the other UAV moves in a straight direction at any angle in the range $[0, 2\pi]$ with respect to the X axis.

Assume the current GPS locations for the pair UAV_{straight} - UAV_{curve} in Figure 3 are $(X_0^{(straight)}, Y_0^{(straight)})$ and $(X_0^{(curve)}, Y_0^{(curve)})$. For the UAV flying in a straight direction, let the speed be $V_{straight}$ and the slope is $m_{straight}$. For the UAV flying in a curve, the current trajectory center is (Cx_{curve}, Cy_{curve}) , radius is R_{curve} , velocity is V_{curve} , angular velocity is ω_{curve} , initial displacement at the time of link establishment is θ_{curve} , and movement direction is Dir_{curve} . Here, Dir_{curve} is -1 for clockwise direction and +1 for counterclockwise direction.

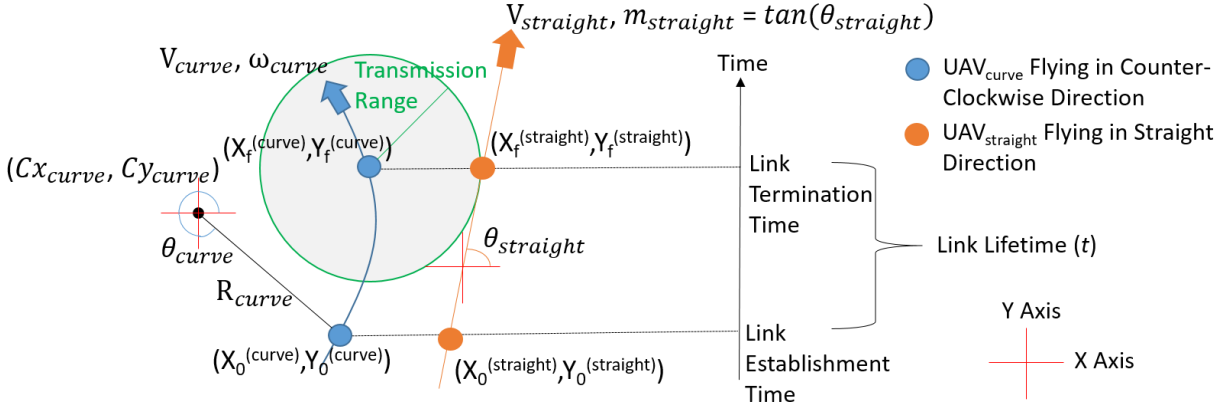


Figure 3. LLT Computation When One UAV Flies in a Curve While the Other UAV Moves in a Straight Direction at an Angle in the Range $[0, 2\pi]$ With Respect to the X-Axis

Using the approach discussed for the case A, the polynomial equation for this case is,

Link Distance =

$$\left[\begin{aligned} & a^2 + b^2 + R_{curve}^2 + (V_{straight} * t)^2 \\ & + 2(sign_1)R_{curve}\sqrt{a^2 + b^2} * (\cos(\theta_{curve} + \omega_{curve}t - (sign_1 * sign_2) \alpha)) \\ & - 2R_{curve} * V_{straight} * t * \cos(\theta_{curve} + \omega_{curve} * t - \tan^{-1}(m_{straight})) \\ & - 2V_{straight} * t * \cos(\tan^{-1}(m_{straight}) - (sign_1 * sign_2) \alpha) \end{aligned} \right]^{\frac{1}{2}} \quad (5)$$

where $a = \text{abs}(Cx_{curve} - X_0^{(straight)})$,

$$sign_1 = \frac{(Cx_{curve} - X_0^{(straight)})}{a},$$

$$b = \text{abs}(Cy_{curve} - Y_0^{(straight)}),$$

$$sign_2 = \frac{(Cy_{curve} - Y_0^{(straight)})}{b}, \text{ and}$$

$$\alpha = \cos^{-1} \frac{a}{\sqrt{a^2 + b^2}}.$$

Case C. When both UAVs fly in a straight direction at random angles in range $[0, 2\pi]$ with respect to the X axis.

For the UAV pair, assume the current GPS locations are $(X_0^{(1)}, Y_0^{(1)})$ and $(X_0^{(2)}, Y_0^{(2)})$, slopes are m_1 and m_2 , and velocities are V_1 and V_2 , respectively. Using the approach discussed in case A, the polynomial equation obtained is,

Link Distance =

$$\left[\begin{aligned} & t^2(V_1^2 + V_2^2 - 2V_1V_2 \cos(\tan^{-1}(m_1) - \tan^{-1}(m_2))) \\ & + t * \left(\begin{aligned} & 2V_1 \left((X_0^{(1)} - X_0^{(2)}) \cos(\tan^{-1}(m_1)) + (Y_0^{(1)} - Y_0^{(2)}) \sin(\tan^{-1}(m_1)) \right) \\ & - 2V_2 \left((X_0^{(1)} - X_0^{(2)}) * \cos(\tan^{-1}(m_2)) + (Y_0^{(1)} - Y_0^{(2)}) \sin(\tan^{-1}(m_2)) \right) \end{aligned} \right) \\ & + (X_0^{(1)} - X_0^{(2)})^2 + (Y_0^{(1)} - Y_0^{(2)})^2 \end{aligned} \right]^{\frac{1}{2}} \quad (6)$$

Note that we have expanded the trigonometric functions up to 12 steps using Taylor's expansion in Cases A and B to get their close approximation. Then the roots of the equation are computed from which an appropriate root is selected that approximates the polynomial equation the best. Note that this approach can be easily extended for elliptical UAV trajectories by using the minor and major axis of the elliptical trajectory.

C.1 Discussion on Random UAV Trajectory Change

During the lifetime of a link, any UAV of a given UAV pair can randomly experience a *Trajectory Change*. For example, both UAVs in Figure 4 change their respective trajectory once before the link termination (see 1st and 2nd *Link Trajectory Change* in Figure 4(b)).

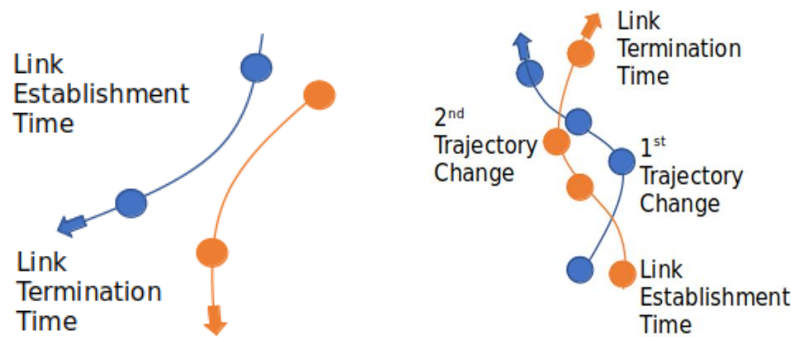


Figure 4. The Link Between a UAV Pair Can Break (a) Without Any Trajectory Change (See Left Image) or (b) After Multiple *Link Trajectory Changes* (See Right Image)

A link between two UAVs breaks after l th *Trajectory Change*, where l is a positive integer. Note that it is not possible to reliably predict the complete *LLT*, if the future trajectory changes are unknown to the UAVs [27]. The *LLT* value is reevaluated when either of the UAVs changes its

trajectory before the link break. Hence, a UAV pair calculates its *LLT* when the link is established or UAV trajectory changes.

D. Computation of *RLT*

A node computes *LLT* for its each 1-hop neighbor node. Since *LLT* changes with time, our scheme uses *residual LLT* for the route selection, which is computed by using Eqn. (7) below. In our scheme, each node in the network knows the updated *LLT* of all the links because each MPR node includes the *residual LLT* for each of its MPR selector node in its TC packet which is broadcast in the entire network periodically.

$$\text{Residual LLT} = \text{Time at the LLT Computation} + \text{LLT value} - \text{Current Time} \quad (7)$$

Therefore, *RLT* of each route *R* is computed as,

$$RLT(R) = \min(\text{Residual LLT of all Links in Route } R) \quad (8)$$

Routes with a high *RLT* are preferred in our scheme.

E. Use of *BO* and Link Capacity in Route Selection

A packet is stored in the queue at a node in the first-in, first-out order and its queuing delay is the duration it waits in the queue before its transmission to the next node or destination. The queuing delay at a node increases with its *BO* value, which can lead to a higher packet drop due to the expiration of a packet's TTL value. In addition, since each node has a fixed queue size, forwarding packets to a node with a high *BO* can result in packet drop due to queue overflow. Therefore, intermediate nodes with a low *BO* are preferred in the route selection.

Total buffer occupancy of a route *R* is computed as,

$$BO(R) = \sum_{node \in R} BO_i \quad (9)$$

The *Link Capacity* of a link is the number of packets a node-pair can service per second. Since the channel is used for the control packet transmission as well, the *Effective Link capacity* depends on the Channel Utilization Factor (see Eqn. 10). It can be computed for the communication in an ideal condition where the packet is not dropped due to channel noise, interference, link break, or queue overflow by using Eqn. (11).

$$\text{Channel Utilization Factor} = \frac{\text{Duration of the Channel Occupancy for Data Transmission}}{\text{Total Channel Occupancy Duration}} \quad (10)$$

$$\text{Effective Link Capacity} = \frac{\text{Channel Access Frequency} * \text{Channel Capacity (in bits/sec)}}{\text{Channel Utilization Factor} * \text{Packet Size (in bits)}} \quad (11)$$

Here, *Channel Access Frequency* is the duration after which a node can access the channel in an ideal condition. Its value is 1/3 for omnidirectional communication because a node can access a channel once in every 3 time slots (see Figure 5). Similarly, it is 1/2 in directional communication.

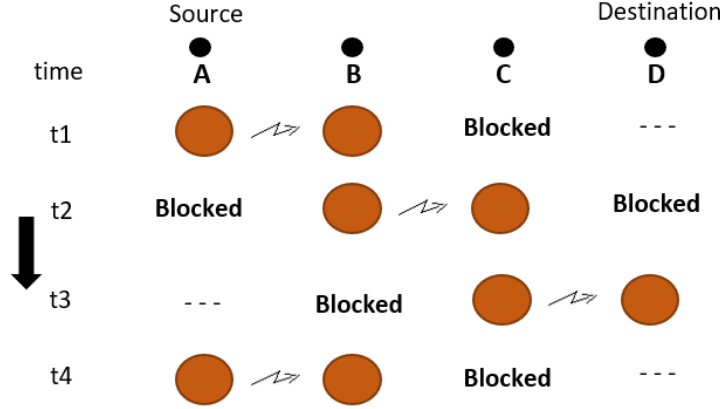


Figure 5. The Channel Access Mechanism for a 3-Hop Topology Where Each Node is Equipped with an Omnidirectional Antenna and Source Node A Transmits Packets to Destination Node D Over the Route A-B-C-D in a Single Frequency Band. Here, the Source Node A Forwards its Packet to Node B in t1 Time Slot. In t2 and t3 Time Slots, Node A should not Communicate to Prevent Collisions at Node B. Therefore, Node A can Access the Channel Once in Every Three Time Slots

F. Route Selection in Our Scheme

In our scheme, *Topology Set* is modified to store the information of known links, their *residual LLT* values, and *BO* of nodes on those links. Each node uses the breadth first search (BFS) algorithm to find all candidate routes to the destination node. The cost for route R is computed as,

$$Route\ Cost\ (R) = w_1 \frac{Hop\ Count_R^{(Exponent)}}{minHC} + w_2 \frac{maxRLT}{RLT_R} + w_3 \frac{BO_R}{Effective\ Link\ Capacity} \quad (12)$$

Here, w_1, w_2, w_3 are the weights for the metrics, and $\sum_i w_i = 1$.

$$Exponent = \begin{cases} 1, & n \leq 1 \\ n \log n, & n > 1 \end{cases}$$

n = shortest *Hop Count* between the source and intermediate node,

$$minHC = \min_{R \in Candidate\ Routes} Hop\ Count_R,$$

$$maxRLT = \max_{R \in Candidate\ Routes} RLT_R.$$

Here, we use normalized values of *Hop Count*, *RLT* and *BO* metrics because they have different ranges. A node selects a globally optimum route with the minimum *Route Cost* in our scheme. This cost metric prefers a route with (i) a shorter *hop-count* which reduces the route failure probability [27], (ii) a longer *Route Lifetime* which reduces expensive route recalculations and conserves energy [29], and (iii) low congestion which reduces the queuing delay and improves the PDR [10][33].

F.1 Route Reevaluation

In OLSR, a node reevaluates its routing table whenever it receives a new control message. Note that the number of routing table reevaluations increases with the network density. Since route computation is expensive, we want to minimize it to conserve node energy and reduce disruption to the ongoing flow.

However, the route(s) are computed only for the *active source-destination pair(s)* in our scheme, where a node which generates data traffic is called an *active source*, and the node for which data is generated is called the *active destination*. Once a route R is selected, a node uses it for the RLT duration. However, if a new entry is created in the *Topology Set* or an updated *residual LLT* value is received for any link on route R which is lower than the current $RLT(R)$, the RLT is updated by using Eqn. (8). Then the existing routes are reevaluated by using Eqn. (12).

In our scheme, a node updates the next-hop address of the packets waiting in its queue when it selects a new route R as shown below. It prevents packet transmission over an outdated route which improves PDR of the flow.

For each packet P in the queue at a node

If (destination address of packet P == destination address in route R)

next-hop address of packet P = next-hop address in route R

End If

End For

Our proposed multi-metric route selection scheme reduces the route computation overhead which conserves the node battery. Note that the time complexity for the Dijkstra and BFS algorithms are $O(E \cdot \log(V))$ and $O(V+E)$, respectively, for a graph with V vertices (nodes) and E edges (links).

G. Control Packet Structure

The modified Hello and TC packet structures used in our scheme are shown in Figures 6 and 7, respectively. Note that the fields added in our proposed scheme are shown in red.

In Figure 6, the *HTime* field specifies the time for the next Hello packet transmission, the *Willingness* field specifies node's ability to carry and forward traffic for other nodes [19]. The remaining black fields are for each 1-hop neighbor node N . The *link Code* specifies the link type (i.e., symmetric, asymmetric, or lost) between the Hello originator node and node N , and the neighbor type (i.e., symmetric, not a neighbor, MPR neighbor) of node N [19]. A node can have multiple interfaces but only one main address in standard OLSR. *Link Message Size* field is used to find the number of interface addresses for node N by using Eqn. (13), which are then obtained from the subsequent *Neighbor Interface Address* field(s) [19]. Note that each node has only one interface in our scheme.

$$\text{No. of Interface Addresses (Node } N) = \frac{\text{Link Message Size Value (in bytes)}}{\text{size (Neighbor Interface Address (in bytes))}} \quad (13)$$

In our scheme, each node includes its *BO* value (in *Originator Buffer* field), trajectory information, such as its current state, GPS location (i.e., coordinate X , Y , Z values), center

coordinates (X, Y, Z values), radius or slope, and the *residual LLT* and *BO* values for each of its 1-hop neighbor node in its Hello packet (see red fields in Figure 6).

An MPR increments its *ANSN* (advertised neighbor sequence number) field whenever it detects a change in its advertised neighbor set and includes the main address of its each MPR selector node in the *Advertised Neighbor Main Address* field of its TC message (see Figure 7). The *ANSN* field allows a receiver node to determine whether the information of the received TC message is more recent than what it already has [19]. In our scheme, an MPR node also includes the *residual LLT* and *BO* values for its MPR selector nodes in the *Residual LLT* and *Buffer* fields, respectively.

Htime (1 Byte)	Willingness (1 Byte)	Originator Buffer (1 Byte)	State (2 Bits)		
Coordinate X (2 Bytes)		Coordinate Y (2 Bytes)		Coordinate Z (2 Bytes)	
Center Coordinate X (2 Bytes)		Center Coordinate Y (2 Bytes)		Center Coordinate Z (2 Bytes)	
Radius or Slope (2 Bytes)					
Link Code (1 Byte)	Link Message Size (2 Bytes)				
Neighbor Interface Address (4 Bytes)				Residual LLT (1 Byte)	Buffer (1 Byte)
Link Code (1 Byte)	Link Message Size (2 Bytes)				
Neighbor Interface Address (4 Bytes)				Residual LLT (1 Byte)	Buffer (1 Byte)
...					
(etc.)					

Figure 6. Modified Hello Packet Used in Our Proposed Scheme, where the Red Fields are Newly Added to the Standard OLSR Hello Packet. The Figure is Adapted from [19]

ANSN (2 Bytes)	Originator Buffer (1 Byte)	
Advertised Neighbor Main Address (4 bytes)		Residual LLT (1 Byte) Buffer (1 Byte)
Advertised Neighbor Main Address (4 bytes)		Residual LLT (1 Byte) Buffer (1 Byte)
...		
(etc.)		

Figure 7. Modified TC Packet Used in Our Proposed Scheme, where the Red Fields are Newly Added to the Standard OLSR TC Packet. The Figure is Adapted from [19]

3.4 Results and Discussion

Simulations are run in the widely used discrete event network simulator ns-3 (version 3.29) for a mobile network topology consisting of 30 nodes. Each node is equipped with an omnidirectional antenna with 1 Km transmission range, has a queue to store 1000 packets, and moves randomly at 20 m/s speed under the ST mobility model. The network size is 8x8 km². The channel capacity is

11 Mbps and the data rate of the flow is 2 Mbps. The packet size is 1 Kbytes with 10 s TTL. The CSMA/CA MAC protocol is used to access the channel.

We have compared standard OLSR and our scheme with different metric weights. Results are obtained for 10 runs where a source-destination pair is randomly selected for each run. Each simulation is run for 100 s, and the data generation starts at 25 s. The evaluation metric used for the comparison is PDR, which is computed as,

$$PDR \text{ (in \%)} = \frac{\text{Total Packets Received at the Destination Node}}{\text{Total Packets Generated at the Source Node}} * 100 \quad (47)$$

The results are shown in Table 2. The flow PDR decreases with the increase in route length. We observe a significant improvement in PDR in our scheme.

Table 2. Average PDR Comparison Between Standard OLSR and Our Scheme

Scheme	Avg. PDR
OLSR	37%
Our Scheme ($w_1=1, w_2=w_3=0$)	47%
Our Scheme ($w_1=w_2=0.5, w_3=0$)	55%
Our Scheme ($w_1=w_3=0.5, w_2=0$)	51.2%
Our Scheme ($w_1=w_2=w_3=0.3$)	56.4%

We also observed a huge decrease in the average number of routing table computations from 22,421 in standard OLSR to 848 in our scheme, which can increase the battery life of a node and, thereby, the network lifetime.

4. 2D SKELETON ROUTING PROTOCOL

4.1 Introduction

UAV networks often need to transmit data (e.g., high-resolution surveillance videos) among UAVs or to the control station. Hence, establishing reliable end-to-end paths among UAVs is critical for many applications, which demand high QoS. For effective coordination and collaboration, the UAVs usually communicate in an ad hoc fashion. A subset of UAVs may link up with the control station. Based on the degree of coordination among UAVs, the network can have different application architectures. For instance, the swarm cooperation architecture (Figure 8(a)) requires a lower coordination among UAVs to accomplish tasks such as the target search. On the other hand, a physically linked architecture (e.g., lifting and transportation of objects) and mission-driven (formation-based) architecture (Figure 8(b)) require a higher degree of coordination and collaboration [21][42]. This research focuses on *mission-driven* swarm networks where a certain network structure is desired. Based on the mission requirements, UAVs are often deployed in a certain spatial formation and the whole network topology may change from one formation to another (called *swarm morphing* hereafter). The formation control can be done through different methods such as leader-follower [43]. Since UAVs have limited resources in terms of power, memory, computations, etc., the low complexity protocols should be designed for swarm networks.

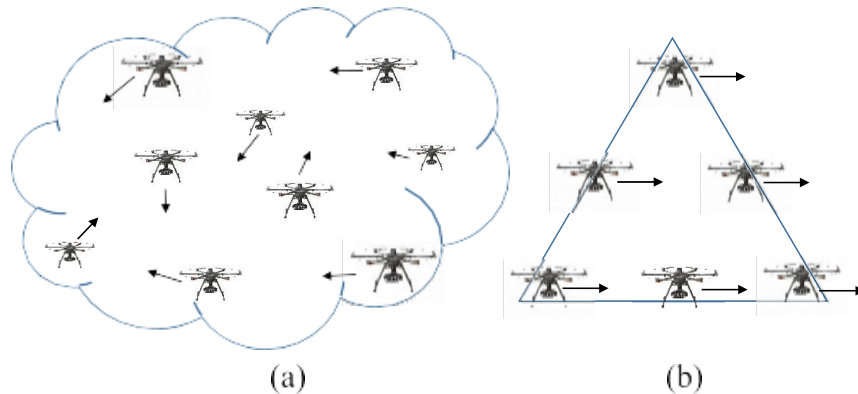


Figure 8. Different Swarm Architectures: (a) Swarm Cooperation, (b) Mission-Driven Formation

In this project, an intelligent, high-throughput routing scheme is designed that adapts to the dynamics of swarm formation. In general, the proposed scheme is distributed and can be used to improve different QoS metrics (such as throughput, delay, load-balancing, etc.) as well as the network lifetime. It exploits the swarm structure (formation) to reduce the complexity of smart routing. Many existing routing schemes typically search the shortest path and cannot adapt to the network dynamics. On the other hand, the centralized solutions and routing schemes, which rely on the frequent update of link-state information to construct the topology database, introduce high complexity and overhead, which is not suitable for resource-limited swarm networks.

To add intelligence and adaptivity to network applications, artificial intelligence (AI) techniques, such as deep learning (DL) [44] and deep reinforcement learning (DRL) [45], are becoming increasingly popular. Despite the effectiveness of these techniques, many AI schemes may not be

practically applicable to the resource-limited UAV swarm networks in which powerful computing platforms are not accessible and energy consumption is a critical issue.

The routing scheme proposed here uses a special *geometric addressing* system to identify the roles and location of different nodes based on the swarm structure, to reduce the routing overhead and latency. In fact, mission-driven swarm networks, which follow a specific formation, can be represented by a framework, called a ***skeleton*** in this project. Here, the skeleton refers to the main structure of the swarm which consists of several relatively stable nodes, called bones. The nodes located in the outer area of the swarm typically have higher mobility than the core/inner area nodes. In terms of the node addressing model, here the term “geometric” is preferred to “geographic”, since the geographic coordinates of the nodes may not be accessible. Note that the geometric address represents the area where the node is located in.

The proposed routing protocol is *hybrid*: (1) *Geometric forwarding*: Packets are forwarded to the area that the node is expected to reach based on a greedy forwarding scheme. (2) *Reactive search*: A local search is conducted to find the destination’s location. Here it is assumed that the UAV formation’s shape changes smoothly based on the mission requirements. We call such a process as ***swarm morphing***, like the concept of image/polygon morphing in the field of computer vision.

Particularly, a distributed dynamic-programming-based online routing scheme, called ***skeleton-based swarm routing (SSR)***, is proposed which uses a leaf-like routing *pipe* to transmit the packets through the nodes that experience less of a traffic load (thus have a lower probability of getting congested) and hence, improves QoS. The proposed routing scheme is flexible and can be used to improve other networking metrics, such as network lifetime, when QoS requirements are not tight. To the best of our knowledge, it is the first work that benefits from the geometric addressing derived from the UAV swarm structure to improve the routing procedure.

Our SSR protocol adapts to QoA parameters in the following intelligent ways: (1) The swarm evolution pattern (SEP) is a critical QoA parameter. In this 2D scheme, we use a new geometric addressing model to represent such a skeleton structure; (2) Our routing scheme can also capture the swarm evolution trend. The trend-based route can adapt to the swarm movement direction and avoids frequent rerouting.

The main contributions of our routing design are as follows:

- *Swarm skeleton-based geometric addressing*: A novel geometric addressing model is designed based on the swarm skeleton structure, which represents the UAV’s approximate location. A formation morphing strategy is implemented to guide the node into the position with the minimum impact on its geometric address when the topology of the entire network changes.
- *Adaptive pipe routing*: A leaf-like routing pipe is constructed according to the addressing model. The pipe serves as the main framework of the routing scheme and can adapt to the changes of the network skeleton structure.
- *Dynamical-programming-based route optimization*: A novel distributed, low-cost, intelligent routing protocol is proposed to achieve the high-throughput and load-balanced data forwarding inside the pipe.

4.2 Related Work

A UAV swarm network has specific characteristics, in terms of mobility patterns, computational power, energy consumption and radio propagation, which makes it different from other types of ad hoc networks. Due to the increasing popularity, many studies have been conducted on the suitable protocols and the corresponding challenges of such networks.

In general, the routing schemes can be classified into five categories: 1) *Reactive routing*: These schemes, such as Ad-hoc On-demand Distance Vector (AODV) [17], are on-demand and mainly designed for mobile ad-hoc networks. For route discovery, they rely on flooding control messages throughout the network which results in extra overhead and latency. 2) *Proactive routing*: These protocols are based on routing tables that are regularly updated (even when there is no data to transmit) and hence are faster. However, frequent updating of the link-state information throughout the network causes high overhead [21]. 3) *Geographic/geometric routing*: These schemes are sometimes considered as proactive routing since they do not perform the initial route discovery phase. However, they do not need the periodic update of the routing tables and the information about the entire network link states. Instead, they depend on the geographic location of the nodes for greedy (distance-based) forwarding. As the geographic locations of nodes may not be always known, the recent routing designs on this topic focus on virtual coordinates and naming approaches [46]–[48]. 4) *Store-carry-forward*: This category is more applicable to sparse and mobile networks where nodes can move to deliver data packets. This solution is mainly suitable for centralized and delay tolerant applications [21]. 5) *Hybrid*: Hybrid routing protocols combine the attributes of the other categories to better adjust to the network features [49].

The integration of software defined networking (SDN) with UAV swarm network is investigated in [50] where the communication and routing policies are managed by the SDN controller. A QoS-based disjoint multipath routing scheme based on a distributed SDN architecture is studied in [51], which can exclude energy-exhausted UAVs and reselect new RF links, if some links are broken. In [52], a centralized traffic-differentiated routing protocol is proposed for an SDN-based hierarchical flying ad hoc network (FANET) architecture aiming to meet specific QoS requirements, where each UAV cluster is controlled by an upper stationary airship. A centralized SDN-based topology and routing management scheme is introduced in [53], where the controller positions the relay nodes to optimize the link availability and constructs the routing table for each node based on the length of the links. The deployment of SDN requires the collection of network information and the use of specific network infrastructure.

An autonomous flocking control scheme is proposed in [54] to maintain the hierarchical network structure. Due to the high cost of deploying GPS and the possibility of losing GPS signals, it uses the received signal strength (RSS) to estimate the distance. The work in [55] solves the problem of communication and control of a triangular swarm of three cellular-connected UAVs and mathematically derives the reliability of the wireless system in terms of meeting the delay requirements. The effect of UAVs sharing the same spectrum with the uplink of cellular users is investigated in [56]. The result shows that the presence of UAV links may slightly degrade the cellular users' uplink performance. The quality of UAV links as well as the user links degrades as UAVs fly higher, due to the possibility of larger line-of-sight interference.

Some recent studies on swarm networks have focused on new routing schemes since the existing ones for ad hoc or sensor networks may not be sufficiently mobility-adaptive,

communication and computationally efficient, or supportive of UAV- to-ground-station communications. It is mentioned in [57] that nodes can estimate the time and energy consumption for data transmission in each path by accessing the position information. They construct a weighted directed graph for UAV cluster architecture. Based on the graph analysis, the optimal relay path can be found through the Bellman-Ford algorithm. An adaptive scheme is proposed in [58] which dynamically adjusts the HELLO packet interval and the timeout timer based on the swarm mission information and the network condition, to minimize the energy consumption in FANET routing schemes.

An enhanced version of AODV, called robust and adaptive reliable predictive (RARP) scheme [59], is proposed for UAV networks which combines the omnidirectional and directional transmissions, and uses a modified RREQ format that includes the sender trajectory information (in 3D), minimum expected connection time of the path and maximum of nodes' failure probability. The destination waits for a specific time to receive several RREQs and selects the path based on a utility function which is a weighted sum of the metrics in RREQ and the hop count to the destination. A PSO-GLFR protocol is proposed in [60], which improves greedy forwarding routing (GFR) using particle swarm optimization (PSO) and limited flooding. Besides the distance factor, the PSO-GLFR protocol considers the number of neighbors and the deflection angle to find the next forwarding relay.

Some studies have extended OLSR [61] to adapt it for a FANET. To address the high mobility of UAVs, in [62] it weights the expected transmission count (ETX) metric based on the relative speed between the nodes using the GPS information. In [21], a mobility and load aware OLSR (ML-OLSR) is proposed which assigns a stability degree to the links based on the statistical information of the distance. Moreover, a load factor is calculated using the buffer load of the node and its neighbors, which avoids selecting the congested paths in the path selection phase. In a similar approach, a link-quality and traffic-load aware OLSR (LTA-OLSR) protocol is proposed in [24], in which the statistical information of the received signal strength is used to find the link quality. The traffic load of each UAV is obtained using the node's buffer load and the channel utilization (which is an indication of neighbors' traffic loads). Despite the effectiveness of such proactive schemes, the need for frequently updating the topology state introduces high overhead, and the computational complexity of the path selection algorithm may be a hindrance, especially in large networks of battery-powered UAVs [21].

In swarm networks with sparse density, the store-carry-forward technique is a popular solution that benefits from the node mobility to forward packets to suitable nodes. The authors in [63] propose the location-aided delay tolerant routing (LADTR) protocol, which combines geographic forwarding with a store-carry-forward strategy to improve the availability of paths between searching UAVs in post-disaster operations. The GPS information is used to estimate the future locations of UAVs.

In [64] it used the queue backlog information along with the geographic information of the nodes to select the next forwarding relay in the store-carry-forward method. Moreover, the Raptor Q code technique is used to further reduce the packet loss rate. Although these routing schemes improve the delivery ratio, they are more suitable for delay-tolerant applications due to the latency caused by the store and carry procedure.

A lightweight stochastic packet forwarding scheme is proposed in [65], where the forwarding probability to each neighbor is found using the link throughput and expiration time. These two

metrics are estimated based on geographic coordination and moving information propagated via HELLO packets. The stochastic forwarding can distribute the packets among several forwarders based on their weights and hence, reduces the interference among nodes compared to single path routing. Distributing packets over a “pipe” (instead of a routing path) is another approach to mitigate the inter-node interference, which is investigated in [66] using a multi-beam directional antenna (MBDA). However, the use of MBDA may be costly in large-scale UAVs networks.

With the popularity of online learning and AI algorithms, some researchers have used such techniques in swarm networks. For example, in [67] an online reinforcement learning (RL) scheme is used for transferring time-insensitive packets in sparse networks, where UAVs help to transfer data. Inspired by geographical routing, it defines a reward function and learns whether to relay data among UAVs or move UAVs to a new location. Although several studies have been conducted to improve the convergence speed of RL algorithms [68], they still face the convergence problem in a large state space. The use of DRL has shown more promising results for large state and action sets. The deep Q-learning technique is used in [69] in an airborne network composed of powerful airplanes at the higher layer and high-density UAV swarm at the lower layer. The scheme adjusts the locations of some UAVs to make up for the broken RF links. However, due to its high computational complexity, DRL may not be applicable in resource-constrained UAV networks. Table 3 compares some of the major routing protocols, in terms of routing category, the metrics being used, load balancing, communication overhead and computation complexity.

Table 3. Routing Protocols Comparison for UAV Network

Reference	Routing strategy	Metrics	Load bal.	comm. overh.	compu. compl.
STFANET [15]	centralized SDN	length of each link	×	High	High
RARP [21]	reactive	# hop, failure prob., connection time	×	High	Low
PSO-GLFR [22]	geographic, reactive	distance, # neighbors, deflection angle	×	Low	High
P-OLSR [24]	proactive	relative speed-weighted ETX	×	High	High
ML-OLSR [25]	proactive	stability degree of node, buffer load	✓	High	High
LTA-OLSR [26]	proactive	link quality, traffic load	✓	High	High
LADTAR [27]	geographic, SCF	position information	×	Low	High
FGQPA [28]	SCF	queue backlog, position information	✓	High	High
SPA [29]	stochastic forward.	link throughput, link expiration time	×	Low	Low

SCF: store-carry-forward.

The scheme proposed in this research is distributed and avoids the overhead and complexity of the centralized solutions such as SDN and AI-based approaches, while adapting to the network conditions. It uses geometric forwarding and hence, does not rely on the geographic coordinates (which may be sometimes unavailable). To the best of our knowledge, it is the first work that benefits from the geometric addressing derived from the UAV swarm structure. Compared to [65], SSR forwards data to only a subset of neighbors in the direction of the destination (according to the geometric addressing) instead of all the neighbors, so that the data is not diverted from the desired trend in large networks. SSR does not perform route discovery and thus, avoids the latency and overhead of flooding RREQs. Moreover, it dispatches data through a routing pipe which significantly improves the throughput. Like [21][24], it is a load-balanced routing scheme, but does not require frequent updates of the link state information required by proactive approaches. However, it requires the distribution of the updated geometric address table, which is not very frequent in the formation-based UAV network, considered in this project.

4.3 Methods, Assumptions, and Procedures

4.3.1 System Model

In this section, a swarm network model is described and the system assumptions for the morphing and routing schemes are provided. The UAVs are interchangeably referred to as the network “nodes” in the report.

A. Network Assumptions

A. Node Resources: As the GPS data may not be available at some nodes, the proposed scheme does not rely on the exact geographic location information. To stay in a proper relative position, the nodes estimate the angle and distance of their neighbors by using inexpensive equipment, such as a compass and sensors. The inter-node distance can also be estimated via RSS-based methods. Nodes have limited power and computational resources and are thus not able to execute complex algorithms. Simple omnidirectional antennas are used here, and nodes time clocks are synchronized.

B. Swarm Topology: Swarm nodes fly in a 2D environment. One of the nodes is assumed to be the swarm leader, which is preselected and fixed in this project. The leader can access larger resources. However, to make the scheme applicable to networks with limited resources, it is assumed that only the leader has access to the mission commands. The leader may decide about the swarm topology itself or receive the command from another entity, e.g., the control station.

The information about the new structure can be represented as the basic information about the skeleton, such as the length and angle of the bones and is transmitted by the leader to the nearby bone nodes. The swarm acts in a distributed manner and each node guides its child node to the proper position (parent-child formation control strategy). Hence, the need for the geographic information (e.g., GPS data) and the trajectory of all the swarm nodes is eliminated. This reduces the complexity of the formation control algorithm, especially in large swarms and improves the scalability and the flexibility of the network.

C. Node Mobility: The UAVs are assumed to move at a speed of 10-50m/s. In mission-driven swarm networks, the UAVs cooperate with each other to fulfill a mission. However, the

mission-driven formation does not mean that all the nodes have a predetermined trajectory, and all the locations are already known. Nodes can still move freely in their proximity, but they need to maintain the overall topology and, therefore, an approximate swarm “framework” should be kept (called a skeleton in this project). When a new swarm mission command is received, nodes may move together towards a target area or locally to properly fill a region. They may also move to help with the route establishment. Hence, the network dynamics can be described by a topology prediction model.

D. Tables Maintained in Nodes: In addition to the general neighbor table that records the 1-hop neighbors’ information (such as their IDs, distances, etc.), each node also maintains a geo-address table that contains all nodes’ ID and their geo-addresses, as well as the timestamp when the geo-address is updated.

The geo-address table is distributed throughout the network (by the leader) when the swarm formation is constructed and may be updated if there are major changes in the swarm (e.g., after each shape morphs). Meanwhile, each node may also update the geo-address extracted from a packet’s header, if the corresponding timestamp is newer than the one saved in the table. Nodes also maintain a routing table for one or more destinations, which includes the next-hop relay nodes (called *potential forwarders* or *PF*) and the cost of the path to the destination, initiated by each *PF* (denoted as Q_j).

B. SSR Components

Figure 9 shows a multi-tier architecture of the proposed SSR scheme. At the very bottom layer, the geometric addressing system provides the approximate location of each node. The formation information, such as the length and the angle of the bones, is distributed through the skeleton by the leader via mission command messages, helping to construct and morph the formation. The geo-address table, containing the nodes’ geometric addresses, is constructed based on the skeleton structure and a parent-child relation model among the nodes. Using this table and some basic information on the structure (can also be extracted from the table), the nodes construct their routing tables which include the set of next-hop forwarders *PF* and their qualification values Q_j . Forwarding to a set of next-hop nodes leads to a leaf-like routing pipe between a source-destination pair.

SSR is a distributed scheme. To make its forwarding decisions, a node needs information only from the nodes in the *PF* set, i.e., their Q values, instead of all the pipe nodes or the whole network. Therefore, the SSR protocol has a lower complexity compared to the conventional RL-based schemes or proactive link-state routing schemes. The Q_j values can be derived based on a weighted combination of some metrics, such as queue status (including service delay, queue length, etc.), channel condition, the node’s remaining power, link stability or expiration time, etc. When the data packets are dispatched through the pipe, nodes gradually update their value and the forwarding probability to the *PF* set, leading to the selection of high-quality paths.

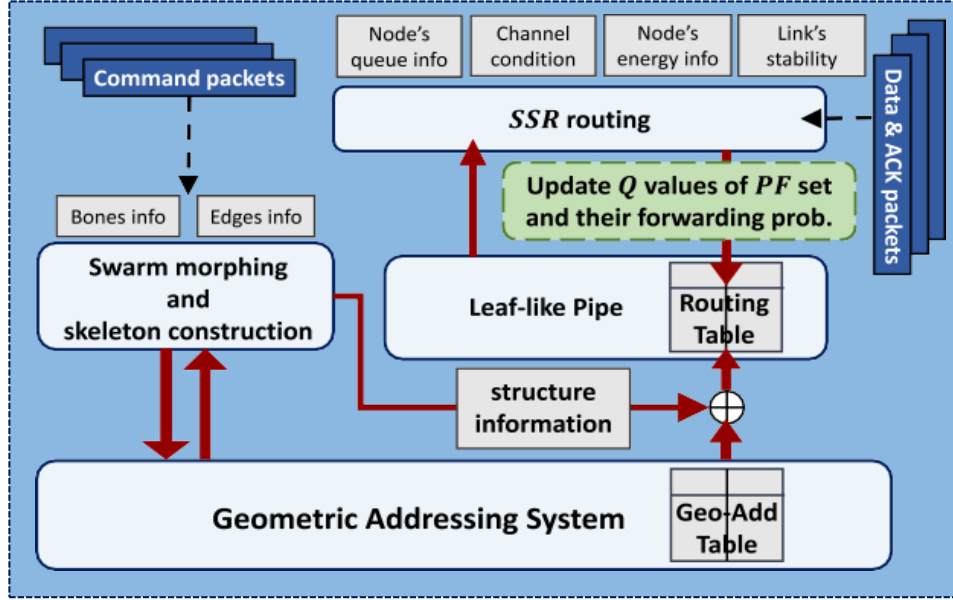


Figure 9. SSR Modules

4.3.2 Leaf-Like Pipe Framework

To speed up the data forwarding process and reduce the route discovery overhead, the SSR protocol uses the greedy forwarding concept by exploiting the swarm structure. Based on the approximate geometric position of the destination, a relay node transmits the packets to a set of its neighbors called *potential forwarders* that are in the direction to the destination based on their geo-addresses.

A. Swarm Skeleton Structure

The skeleton is used as a reference to facilitate the geometric addressing and shape morphing. A similar concept has been previously used for static sensor network [46], where the extracted skeleton was the medial axle of the network shape. In our case, due to the mission-guided UAV movement, the skeleton structure is already known that can guide the swarm to form a new shape.

Inspired by the study in [70], a star-shaped skeleton with the swarm leader as its root is used, as shown in Figure 10. The reasons behind choosing a star skeleton are threefold: 1) it helps propagate the leader's commands uniformly and quickly throughout the network, 2) it helps to guide the nodes to build a new formation in a distributed manner during the swarm morphing process, 3) it provides a 2D coordinate system suitable for the geo-addressing framework.

The skeleton consists of branches (Figure 10), which form the skeleton bone and are not necessary arranged in a line. The use of “elbow” nodes enables the skeleton to represent more complex shapes and divide them into approximately equal-area pieces. In general, the nodes are classified into four main categories: 1) leader or root l ; 2) skeleton node set S that consists of bone nodes $B_i \in S, i = 1, \dots, k$, where k is the number of skeleton bones; 3) edge nodes located on the border of the swarm, and 4) regular nodes. Each bone can have up to e elbows.

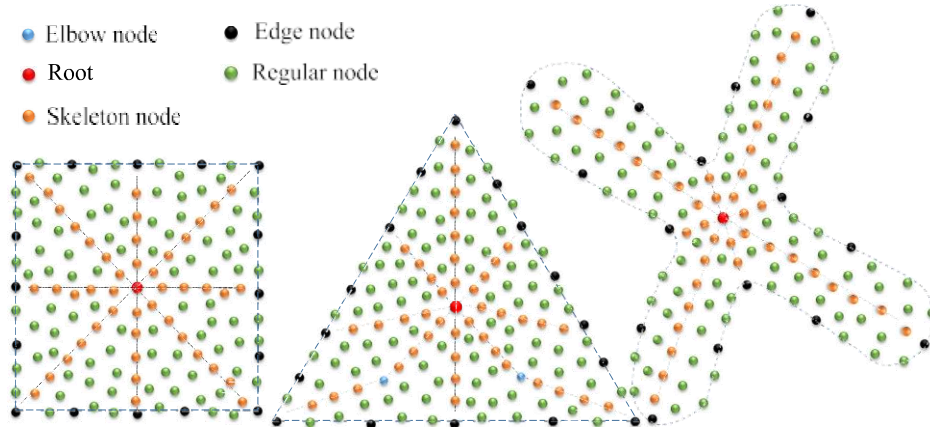


Figure 10. Examples of Swarm Formations with Skeleton Consisting of Eight Bones

The skeleton nodes can guide other nodes to form a desired swarm shape. To manage the formation in a distributed manner, nodes follow a hierarchical parent-children structure as shown in Figure 11(a). The leader l transmits the abstract information on the desired swarm structure to the nearby bone nodes and guides them to reach the desired positions. Then this information is passed through the bones and each bone node guides the next bone node (its child) until the complete framework is constructed (parent-child formation control strategy). Transmitting this geometric structure information only to the skeleton nodes instead of the entire network, reduces the communication overhead.

The network can first make a rough representation of the desired shape by constructing the skeleton and some main edges. Then other (regular) nodes can be placed, starting from the inner layers. Some UAVs, called free nodes, may move towards inner parts to fill any vacant area there. A node can leave its parent node if it receives a positioning command with a higher priority.

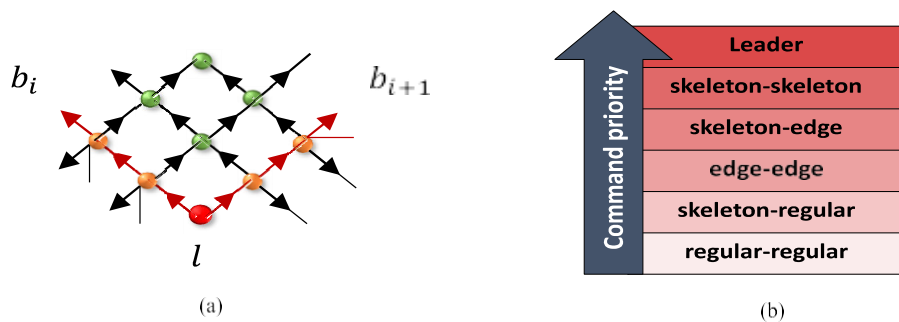


Figure 11. (a) Parent-Children Relation Between Different Node Types. Here, an Arrow Points from a Parent to its Children. (b) Priorities of Different Swarm Messages

B. Swarm Messages

Swarm messages have different priorities as illustrated in Figure 11(b). Here the “A-B” command means that a node with role “A” commands another node to accept the role “B”. For

instance, if a node has already been accepted as a skeleton node (by receiving a leader or skeleton-skeleton message), it will discard any skeleton-regular or regular-regular messages. Among all command messages, the leader and skeleton-skeleton messages have different formats as they contain the information on the skeleton's shape, such as the angles and the lengths of the bones. The swarm messages are briefly described below.

B.1 Skeleton Messages

Every bone node that receives a skeleton command message records all the information about the skeleton, and updates its position based on its assigned bone index. Then it passes the message to the next bone node. Figure 12 illustrates the command message format. Suppose the number of bones is 8, then the skeleton information can be fit into a 72-Byte (72 B) packet, containing 8 B information for each bone and an additional 8 B for the packet type and sender and receiver information.

In Figure 12, the 4-bit *Num* field specifies the total number of bones k ($k \leq 16$) in the skeleton. The *bIdx* contains the *boneindex* that the sender is in, where $1 \leq bIdx \leq k$.

SenderBoneIdx contains the bone index of the sender. Initially, the root sets it to zero, and it increases by one when passing through each bone node. The *BoneLength_i* represents the number of nodes in bone i (denoted as B_i). By assigning 10 bits to it, B_i can reach 1024, which represents a large swarm network. The 11-bit *BoneAngle_i* field specifies the initial angle of the bone i , in terms of a pre-defined reference line such as the East or North direction. The *ElbowIndex_i* and *ElbowAngle_i* represent the bone index and the angle of the elbow node in i^{th} bone, respectively. *ElbowIndex_i* 0 means that i^{th} bone does not have any elbow.

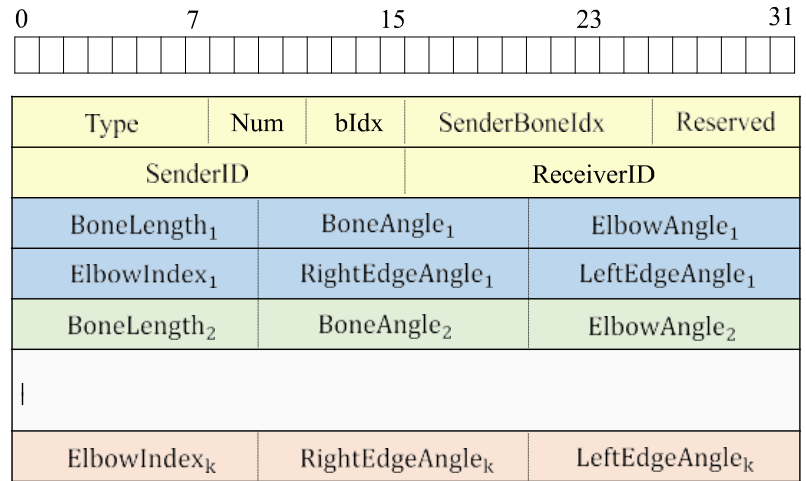


Figure 12. Format of the Command Message

Upon receiving a command message, the node first checks *bIdx* and increases it by one. Then it finds its own bone index by increasing *SenderBoneIdx* field and compares it with *ElbowIndex_i*. If the node is located after an elbow node, it takes the corresponding *ElbowAngle_i* as its bone parent;

otherwise, it adjusts its position corresponding to $BoneAngle_i$'s value. If the node's bone index is equal to the value of $BoneLength_i$, it is located at the end of bone i . The end bone node guides the left- and right-edge nodes to reach their desired positions, based on the fields $LeftEdgeAngle_i$ and $RightEdgeAngle_i$, respectively. Then every edge node that has been adjusted in the determined location guides the next edgenode (by sending an edge-edge command message).

B.2 Edge and Regular Command Messages

These messages do not contain much information about the skeleton. They only tell about the new position of the receiver node. Besides common information on packet type and sender/receiver IDs, an edge-edge command includes the direction and distance that the receiver needs to travel, to reach the desired location in the swarm edge.

C. Skeleton Based Geometric Addressing

The geometric addressing of each node is determined based on its distance to the adjacent bones. Here the number of hops is used as the distance to a node since the exact geographic location of the nodes may not be accessible. Still nodes are distinguished by their unique IDs. If the destination has recently moved in the region, a local search is performed to find the new geometric position (geo-address) of the destination.

The proposed addressing procedure is inspired by the naming method of the medial axis-based naming and routing protocol (MAP) [46]. However, the addressing procedure proposed in this project overcomes three drawbacks of the MAP scheme: (1) MAP may get too far from the shortest path when it first moves along the latitude and then along the longitude, as shown in Figure 13. The proposed scheme moves along the diameter direction to reduce the path length. (2) Unlike MAP, the proposed scheme does not require that every node has a complete knowledge of the skeleton. Note that frequently updating the skeleton information and flooding it all through the network introduces a high overhead. (3) MAP-based routing cannot achieve load balancing. For instance, two sources with the same height h will transmit their data along the same h -latitude route, which may lead to a congested path. In contrast, the SSR-based leaf-like routing pipe (shown in Figure 13(b)) can easily achieve load balancing by using different paths.

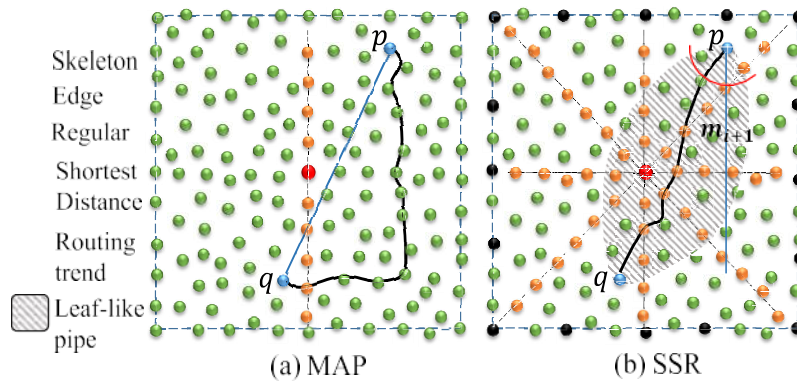


Figure 13. Comparison of Data Forwarding Paths in SSR and MAP

A given node g is addressed using the tuple $G(g) = (R_i, n_i, n_{i+1}, h_i, h_{i+1})$. Here R_i is the index of the region surrounded by two bones b_i and b_{i+1} . The number of skeleton bones, k , stays constant during the swarming process. These bones are indexed in order, but their sizes and shapes may change. n_i and n_{i+1} are the indices (hop counts) of the nodes located in bones b_i and b_{i+1} , respectively. These are the roots of the shortest-path trees from the bones to g . h_i and h_{i+1} are the hop-count distances of g from n_i and n_{i+1} , respectively.

They also reflect the depth of the entire tree. If g is a bone node, then the geo-address can be simplified as $G(g) = (R_i, n_i, 0, 0, 0)$, where n_i is the index of the bone node. The leader's geo-address is a tuple of zeros.

By using such an addressing system, a 2D virtual coordination system is generated, where the skeleton bones are located in the coordination axis. Figure 14 illustrates an example of geometric addressing for a given node g . Considering $i=2$ (i.e., the node is located in the region $R_i=2$, which is surrounded by b_2 and b_3), the address of node g is $G(g) = (2, 4, 2, 2, 3)$ in this example.

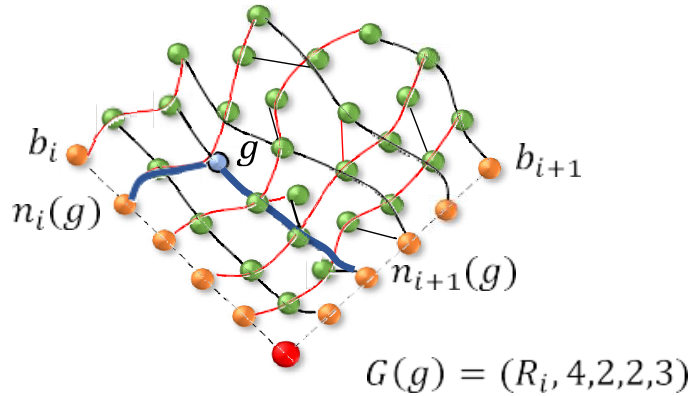


Figure 14. An Example of Assigning Geo-Addresses

D. Leaf Establishment and Potential Forwarders

Like the AODV scheme, every node maintains a routing table for one or more destinations (denoted as q_i). However, there are some differences between SSR and AODV routing tables. Figure 15 illustrates the structure of a typical routing table in SSR. Since there is no route discovery phase and consequently no RREQ and RREP packets, the “destination sequence number” and “hop count” fields are not used. Another major difference is that the “next hop” field is now a multi-input field that can record up to J ($= 4$ in this example) potential forwarders (i.e., the next-hop relay nodes, denoted as $u_j \subseteq PF(q_i)$) to reach the destination. This routing table has another field that records the cost of the path (denoted as Q_j) initiated by each potential forwarder. It is used to calculate the forwarding probability for different next-hop nodes.

After knowing the geo-address of the destination q , the source node p starts data transmission. To not divert the packets much from the shortest path between nodes p and q , the routing scheme dispatches the packets inside a leaf-like pipe around the shortest path, shown by the shaded area in Figure 13(b). Upon receiving the data, the relay node g keeps an entry in its routing table

for destination q , like the case shown in Figure 15. The node finds the PF set, and distributes the received packets among them, based on the forwarding probability (to be explained later).

Dst-ID	Geo-Add	Update Time	$PF_v(q)$	Q-value
q_1	$G(q_1)$	t_1	u_1	Q_1
			u_2	Q_2
			u_3	Q_3
			u_4	Q_4
q_2	$G(q_2)$	t_2	u_1	Q_1
			u_2	Q_2
			u_3	Q_3
			u_4	Q_4

Figure 15. An Example of SSR Routing Table

Each node that receives a packet addressed to the destination q estimates the direction of q , called *trend*, and then selects the set of next relay nodes with equal or shorter distance to q . It is worth mentioning again that by distance we mean hop-count distance based on the geo-address. By comparing the first element of their geo-addresses, the node will know that it is in the same or different region as q . If the node g and destination node q (with $G(g) = (R_i, n_i, n_{i+1}, h_i, h_{i+1})$ and $G(q) = (R_j, n_j, n_{j+1}, h_j, h_{j+1})$ are in the same region R , the hop-count distance corresponding to the bone b_i can be found as follows:

$$d(g, q) = \sqrt{(n_i - n_j)^2 + (h_i - h_j)^2} \quad (14)$$

Note that (14) can be rewritten with respect to bone b_{i+1} as well. Estimating the distance of nodes located in different regions may need a more complex calculation. However, a good estimation can be obtained using trigonometry (illustrated in Figure 16). The distance $d(g; q)$ can be found through the following equations:

$$\alpha = \sum_{k=i+1}^{j-1} \alpha_k + \tan^{-1} \frac{h_j}{n_j} + \tan^{-1} \frac{h_{i+1}}{n_{i+1}} \quad (15)$$

$$d(g, q) = \sqrt{n_{i+1}^2 + h_{i+1}^2 + n_j^2 + h_j^2 - 2\sqrt{n_{i+1}^2 + h_{i+1}^2}\sqrt{n_j^2 + h_j^2} \cos \alpha} \quad (16)$$

where α_k is the angle between the skeleton bones located in the areas between the R_i and R_j regions. If the node does not know the precise values of α_k , it can use an approximate value; $\alpha_k = 360/k$. The above equations are derived by assuming that the node estimates a *trend* in a clockwise direction around the root by comparing R_i and R_j . If the trend is anti-clockwise, the equations should be updated by substituting (h_j, n_j) with (h_{j+1}, n_{j+1}) , and (h_{i+1}, n_{i+1}) with (h_j, n_j) , respectively. To make the search of PF easier, the node g can first find the closest bone node in the trend (m_{i+1}

in Figure 16), and then use (14) to locally compare the distance of the neighbors to m_{i+1} . Here m_{i+1} can be found by using the equations below:

$$\gamma = \sin^{-1} \frac{\sqrt{n_j^2 + h_j^2} \sin \alpha}{d(g, q)}, \quad \beta = 180 - (\gamma + \tan^{-1} \frac{h_{i+1}}{n_{i+1}}) \quad (17)$$

$$m_{i+1} = \left\lfloor \frac{\sin \gamma \sqrt{n_{i+1}^2 + h_{i+1}^2}}{\sin \beta} \right\rfloor \quad (18)$$

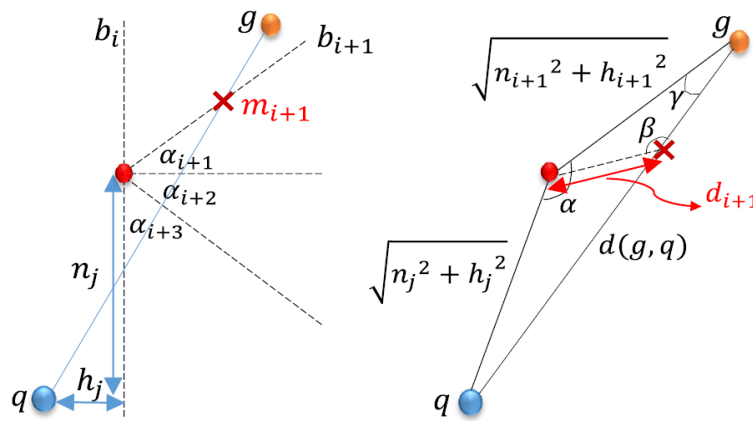


Figure 16. Finding the Hop-Count Distance-based on Geo-Address Information and Skeleton Structure

4.3.3 Skeleton-based Routing

The SSR scheme aims to achieve an intelligent swarm-adaptive, load-balanced, and high-throughput routing. It builds a leaf-like routing pipe (from the source to the destination) composed of interconnected paths, as illustrated in Figure 17. Based on the feedback received from the potential forwarders, nodes gradually adjust the frequency of data transmission through these paths.

When a source node p decides to send packets to a destination q , it first looks for a matching entry in its routing table and extracts $G(q)$ from the geo-address table. If there is a corresponding forwarder set PF that is up to date (SSR can find this by comparing $G(q)$ and its timestamp with those in the routing table), it starts to dispatch packets among $PF(q)$ based on the corresponding Q -values. Otherwise, it first updates $PF(q)$ and their corresponding Q -values.

Besides the source and destination IDs, the data packet's header also includes $G(p)$ and $G(q)$ (as well as the timestamp) when the source has updated $G(q)$. When a relay node receives a packet, it updates $G(p)$ in its geo-address table. If the node has a newer $G(q)$ compared to the one in the packet, it inserts the updated one in the packet header and notifies the source about the changes in $G(q)$. In general, every node that hears a new geo-address will update its table. Upon significant

changes in the geo-address of the node or destination q , the set of next-hop nodes, $PF(q)$, is updated and the Q -values of newly added nodes is initialized.

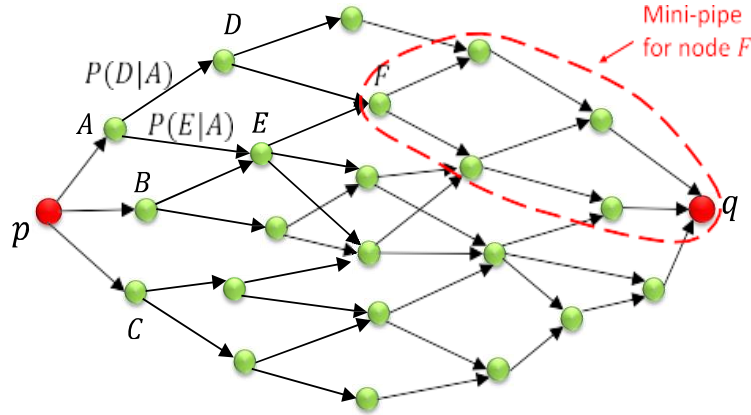


Figure 17. A Leaf-Like Pipe Composed of Inter-Connected Paths. Each Link is Associated with a Forwarding Probability

A. Building Skeleton Routing via a Distributed Stochastic Dynamic Programming Approach

The mesh architecture shown in Figure 17 inspires us to formulate the problem of skeleton-based routing as a stochastic shortest-path problem [71], which can be solved using a dynamic programming approach [72]. A cost (or reward) is assigned to each link inside the leaf-like pipe, and the cumulative cost is calculated in a backward recursion fashion. However, the cost values are not known in advance, and are subject to the changes due to the dynamic nature of the swarm network. Hence, an online dynamic programming-based approach is used which can adapt quickly to the changes of the network conditions. It has two main advantages:

- 1) It is solved in a distributed manner. Instead of having a central point which gathers all the state information frequently and solves the whole problem, every node calculates its own value.
- 2) Nodes gradually update their forwarding probabilities based on the values of their forwarding nodes and hence, the scheme will learn how to dispatch the data packets inside the pipe to get the best performance with a low computation complexity.

To dispatch the cumulative cost (or reward) through the path in the backward manner, one can take advantage of the ACK messages that are popularly used in routing protocols. A mini pipe is initiated from each relay node g to the destination q , as shown in Figure 17. The “expected” cost (or reward) of the sending packets from the node g through the mini pipe, is denoted as $V(g)$, which can be piggybacked onto g ’s ACK messages. Upon receiving it, the sender updates the Q -value that corresponds to the forwarder node g in the routing table.

The problem can be modeled as a Markov decision process (MDP) in which every node is a state, $s \in S = N$ (N is the set of the nodes in the pipe), with the action space $A \triangleq PF(s)$ and the next state $s' = a(s)$. Here the transition probability function is the probability of choosing a possible forwarder, i.e., $P(s' | s, a(s)) = P(a(s) | s)$. Then the expected cost (value) at each node v can be found by using the Bellman equation [72]:

$$V(v) = \sum_{u \in PF(v)} [c_{vu} + V(u)]P(u|v) \quad (19)$$

where c_{vu} is the immediate cost of transferring packets from v to u , and $V(u)$ is the cost value of node u . $P(u|v)$ is the probability of forwarding packets to node u , where

$$\sum_{u \in PF(v)} P(u|v) = 1$$

Initially, when a new forwarder is added to the pipe, its associated cost is set to the initial value V_0 which gets updated as the packets pass through the node. Considering α_Q as the learning rate, the Q -value in the routing table can be updated by using the following equation:

$$Q(u) = (1 - \alpha_Q)Q(u) + \alpha_Q V(u) \quad (20)$$

B. Packet Forwarding Probability

A node $u \in PF(v)$ may become the next forwarder based on a probability distribution function. There are different action-selection strategies in the reinforcement learning (RL)-based schemes, such as random action-selection and ε -greedy approaches. Here, the Boltzmann approach is adopted which selects the actions based on a probability distribution function, such that the more rewarding actions are selected with higher probabilities (more frequently).

The major advantage of the Boltzmann approach over the ε -greedy approach is that the non-optimal actions are not selected with equal probability. Instead, they are taken with a frequency corresponding to their estimated rewards. Hence, more rewards can be gathered in the exploration phase. In this approach, the probability function is represented by the Soft-max function (or normalized exponential function) with the temperature parameter τ . Here, the proper choice of τ is important. The higher values lead to almost equiprobable actions as in a random approach, while the lower values can make a big difference in the action selection probability [73].

Other exploration approaches such as Bayesian neural networks or deep RL (DRL) might work better but introduce a high computation complexity (thus not suitable to UAVs networks). Thus, the Boltzmann action-selection approach is adopted where the probability of choosing node u as the next forwarder (i.e., forwarding probability) can be found through the following equation [74]:

$$P(u|v) = \frac{e^{Q(u)/\tau}}{\sum_{y \in PF(v)} e^{Q(y)/\tau}} \quad (21)$$

C. Cost Function

The definition of c_{vu} is application-specific and depends on QoS requirements. It can be a weighted combination of several metrics. For instance, video streaming demands a high data rate and very low latency. Control packets hold sensitive information and hence require timely delivery. Some IPv4 applications and offline transfer of texts or documents can be classified as the best-effort services.

For the best-effort traffic, the data packets can be routed through the paths with a higher remaining energy in the intermediate nodes, although they may not provide low-latency

communications. For instance, the work in [75] has proposed an energy-aware reward function which, with respect to our problem, can be represented as follows:

$$r_{vu}^{EA} = \text{sigm}(\Delta E(u) \times \beta_e) \quad (22)$$

where $\Delta E(u)$ is the difference between the energy of the node v and the next forwarder node u , and β_e is an energy difference scaling factor. The Sigmoid function rescales the energy difference with $[-1, 1]$.

To make SSR support high-throughput and low-latency communication for high-priority traffic, the per-hop service delay is defined as the cost, i.e., the time duration from the moment the packet enters the queue to the moment it is successfully delivered, denoted by c^{SD} . It represents the queue length and the channel utilization (i.e., the medium access delay in contention-based protocols) and hence, leads to a load-balanced solution [21][24]. It also indirectly talks about other situations such as RF interference, channel collisions and link handoff. If a node is experiencing high interference or poor SINR (due to the long distance), the packet drop rate increases, which leads to an increase in packet delivery delay. A node can find the expected service delay through the statistics of the past experiences.

D. Local Search

If the destination q moves away from the expected location specified in the packet header, a node that expects q to be in its neighborhood (by comparing their geometric addresses) may not be able to find it. In this case, the node initiates a local search to find the current address of q by broadcasting an *address request* (AREQ) packet. AREQ contains the initiator's ID and address, q 's ID and its previous address, and the timestamp. Any node with a newer address should reply and every node hearing a newer address updates its geo-address table.

In fact, a newer address represents q 's trajectory towards the new position. The initiator is responsible for gathering the addresses and sending an update on the latest address to the source. AREQ flooding only occurs in a small region (R_i) in the skeleton-based routing protocol.

E. Two-Step Discarding Policy (TSDP)

For service delay as the cost function, $V(v)$ in (19) represents the expected latency before reaching the destination. It can also be used for active queue management that aims at relieving the network congestion or reducing the end-to-end latency by discarding some packets in the queue before it gets full. Tail drop and random early detection (RED) are examples of widely used discarding approaches (RFC 2309 and 7567). However, for delay-sensitive applications with large queue sizes, it is preferable to use a discarding policy that drops packets based on the priority levels and the probability of on-time delivery to the destination. Inspired by the weighted random early detection (WRED) approach [76], here a two-step discarding policy (TSDP) is proposed that operates based on the service delay.

TSDP consists of two steps:

1. Finding P_d^{ub} , the upper bound of the packet discard probability. As illustrated in Figure 18(a), this value is low (close to 0) for low latency queues but increases linearly as the

service delay rises until reaching P_d^{max} . One can determine how to discard the packets with different traffic types through different configurations of P_d^{max} , SD_{min} and SD_{max} . The calculated P_d^{ub} will be used in the second step.

2. In this step, the discarding probability of packets (in the same traffic type) is differentiated by estimating the possibility of timely delivery. Assume that t_l is the packet lifetime, t_{vq} is the expected latency from node v to destination q , and t_{pkt} is the packet age. Then the *survival ratio* of packet pkt entering node v 's queue can be defined as follows:

$$SR(pkt, v) = \frac{\max(t_l - t_{pkt} - t_{vq}, 0)}{t_l} \quad (23)$$

When the packet is new and a low-latency route is estimated, the survival ratio increases, which leads to a decrease in P_d , as shown in Figure 18(b). However, a low $SR(pkt)$ indicates that the packet has a small chance of arriving at the destination in time and hence, this packet is dropped with a higher probability.

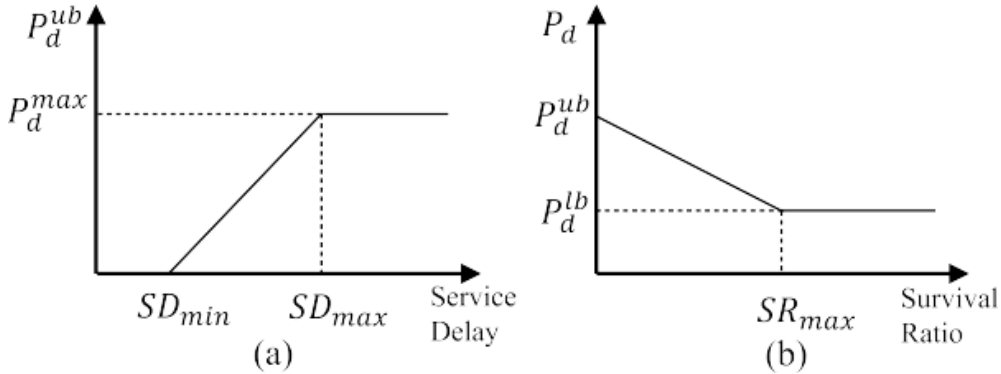


Figure 18. (a) Upper Bound of Drop Probability in the First Step of TSDP, (b) Finding the Drop Probability of a Packet in the Second Step Based on its Probability of On-Time Delivery to Destination

P_d^{lb} is a fraction of P_d^{ub} , i.e., $P_d^{lb} = \alpha P_d^{ub}$. If a node does not have enough information to estimate t_{vq} , α can be set to 1 (i.e., the survival ratio is not counted). In general, t_{vq} can be estimated through the recursive equation (19), i.e., it is equal to $V(v)$ if c_{vu} is the per-hop service delay. Each node v in the pipe can evaluate the service delay to the next forwarder u , calculate t_{vq} and send it to its upstream node.

F. Resilience to Intentional Interference

SSR can be combined with the detection schemes for intentional interference to detour the data around the interfered areas. If UAVs can detect the intentional interference and scoring themselves based on the probability of being interfered (similar to [66]), this score can be reflected in the Q value of nodes. Hence, in (21), the probability of forwarding to a potential forwarder u $PF(v)$ increases, if the mini-pipe initiated from u has a lower chance of being involved in intentional interference activities (see Figure 17). In this case, SSR can route the packets through the

“safe” areas in the pipe. Please note that boundary nodes, i.e., the nodes close to the interfered area, are more vulnerable to intentional interference soon due to mobility. Thus, they should be assigned a low score.

4.4 Results and Discussion

In this section, the performance of the proposed UAV swarm morphing and SSR routing scheme is evaluated. A wireless swarm networking platform is developed in Matlab, where the nodes can morph from one shape into another and exchange packets through IEEE 802.11 channel access model. The simulation parameters are summarized in Table 4.

Table 4. Simulation Parameters

Parameter	Value
Max UAV Speed	50 m/s
Radio transmission range	1 km
Minimum distance between nodes	500 m
Number of nodes	137
Number of skeleton bones	8
Data packet size	2 kB
ACK packet size	14 Bytes
Max link rate	5 Mbps
Packet lifetime	5 s
Channel access CW	15
Source node's CW	7
Initial node's forwarding cost V_o	0

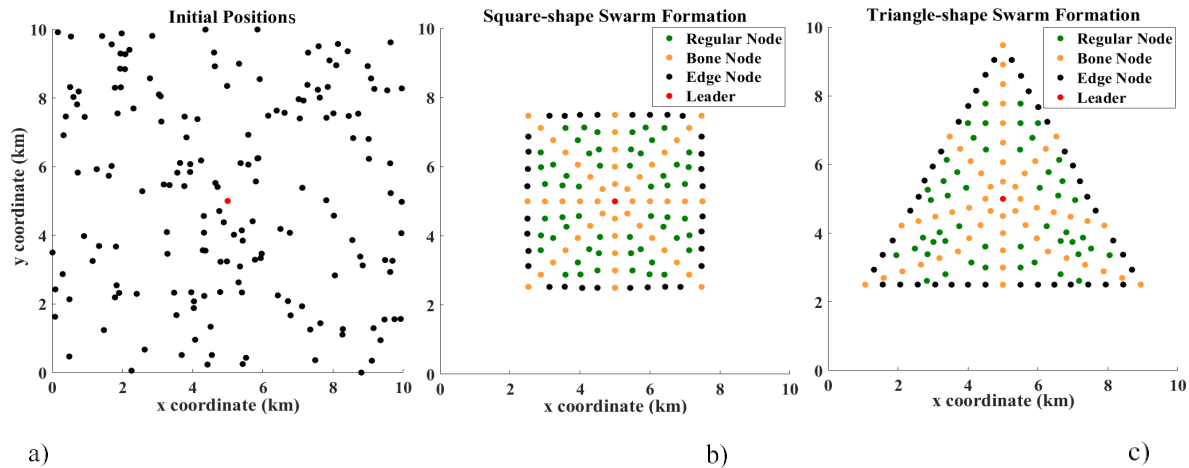


Figure 19. a) Initial Positions of UAVs Randomly Distributed in a 10 km x 10 km Rectangular Area; b) Square Formation Based on the Skeleton Concept; c) Triangular Formation

4.4.1 Swarm Construction

The swarm formation of two example shapes (square and triangular) is examined using the procedures explained before. To construct the swarm formation from randomly distributed UAVs within a 10 km x 10 km rectangular area, as depicted in Figure 19(a), the leader in the center propagates command messages to the nearby nodes and guides them to move towards the desired locations to form the first layer. Then, the skeleton nodes construct the core of swarm based on the information in the command message (Figure 12).

The final shapes of the square and triangle formations are illustrated in Figures 19(b) and (c), respectively. The results validate the ability of the simulator in terms of using the skeleton nodes to facilitate the swarm formation.

4.4.2 Establishment of Leaf-Like Routing Pipe

In this part, several scenarios with different source-destination pairs are considered, and the ability of the proposed model to construct leaf-like routing pipes based on the skeleton infrastructure and geometric addressing is evaluated. Figure 20(a) illustrates the pipe constructed between two relatively faraway nodes (in two opposite regions in a square formation). The pipe starts from the source at the top, passes through several regions and gets merged again near the destination. Figure 20(b) illustrates a smaller pipe from the leader to destination.

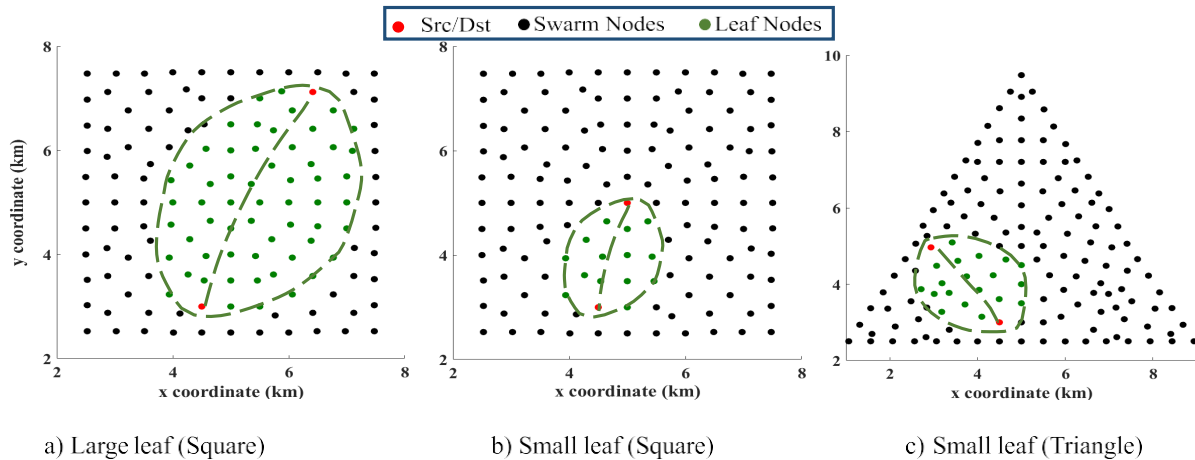


Figure 20. Establishing a Leaf-Like Routing Pipe Based on the Proposed Geometric Addressing: a) A Large Leaf in Square Formation; b) A Small Leaf in Square Formation; c) A Small Leaf in Triangular Formation

As seen in Figure 20(a)(b), if the distance between the source and destination is longer, the formed leaf could become wider and allows the establishment of more backup paths in the pipe. A wider pipe also leads to a higher throughput and better traffic balancing performance since more non-interfering paths can be built. Figure 20(c) depicts another example of the pipe within a triangle formation. The pipe may not be like a leaf when the inner-bones' angles are different

or the bone elbows are presented. However, this is not the main concern if a group of nodes can be found to build the optimal routes among them.

4.4.3 SSR Performance in Light External Traffic

In this section, performance of the SSR scheme is examined when the external (or background) network traffic is light and has a lower priority than the flow under investigation and is generated by other applications. Only one source and destination pair are active in the routing pipe. Figure 21 compares the (normalized) throughput of the SSR protocol with that of the AODV scheme, for the formation and the routing pipe illustrated in Figure 20(a). Moreover, the impact of different values of τ (the thermal factor in Boltzman model) on the routing performance is investigated. Here, τ is a hyper-parameter and determines the sensitivity of action selection in Markov decision model (smaller τ means higher sensitivity). In this figure, *SSR-random* represents the case when the data forwarders (*PF*) are chosen randomly.

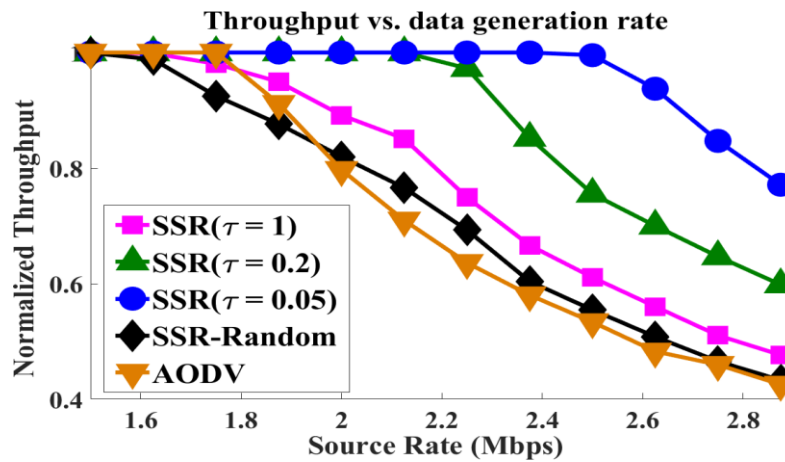


Figure 21. Throughput Comparison for Different Forwarding Action Selection Schemes When Network Traffic is Light

As shown in Figure 21, all schemes work well for a low data generation rate. When the source rate increases, SSR with $\tau = 0.05$ still shows a satisfactory performance while others have degrading throughput. The reason is that SSR with $\tau = 0.05$ distributes packets to the paths that have a lower RF interference with each other. However, when the source data generation rate exceeds 2.5 Mbps, SSR with $\tau = 0.05$ also exhibits throughput degradation. The main reason for the throughput drop in high data rates is that the source node cannot flush out all the generated packets in a timely manner due to the congested channel.

Uniformly distributing packets among the forwarders in the *SSR-random* scheme may not be a good solution since the inner nodes will suffer a stronger interference from nearby nodes. The result shows that *SSR-random* and SSR with $\tau = 1$ exhibit a similar performance. The throughput of AODV, on the other hand, falls more rapidly since it uses only a single path at the center of the pipe.

Figure 22 compares the delay of packets received at the destination for the above-mentioned scenarios. The result reveals that a higher τ causes longer delay since the service time grows in queues. The AODV's expected delay is lower than the other schemes at higher data generation rates as it passes the packets through the shortest path in the middle of the pipe. However, a large portion of the packets are dropped on their way to the destination.

To account for the dropped packets in the delay performance analysis, Figure 23 illustrates the *total delay* as the expected delay of all packets, where the delay associated with a dropped packet is the packet lifetime. Based on the figure, the AODV delay is similar to that of SSR with $\tau = 1$ and SSR-random, while SSR with $\tau = 0.05$ has the best performance in terms of routing the data through faster paths. The *total delay* is used in delay analysis hereafter.

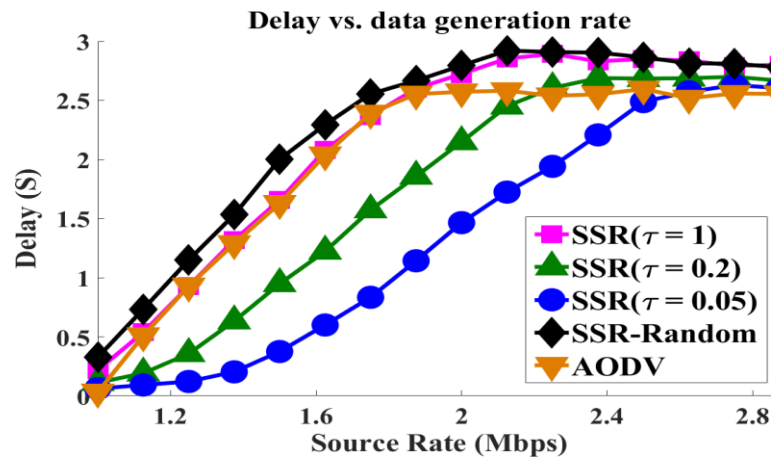


Figure 22. Received Packet Delay Comparison for Different Forwarding Action Selection Schemes When Traffic is Light

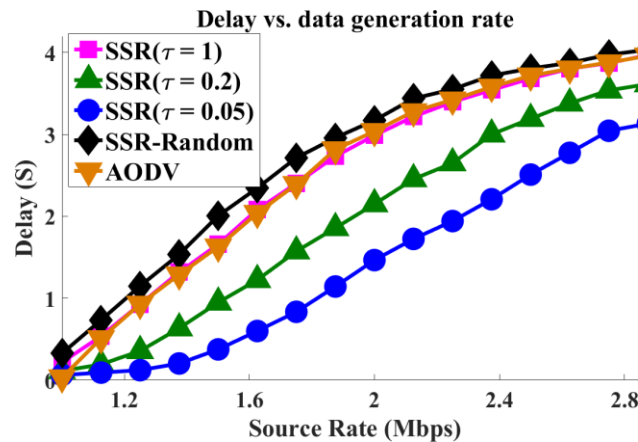


Figure 23. Total Delay Comparison for Different Forwarding Action Selection Schemes When Traffic is Light

4.4.4 SSR Performance when the Network is Congested

Next, the performance of *SSR* in terms of its load-balancing capability is investigated under network congestion, for the network topology shown in Figure 20(a). The external traffic could cause certain congestion around the leader node, as depicted in Figure 26(a).

The throughput and delay for several configurations are evaluated in Figures 24 and 25, respectively. Again, *SSR* shows a better performance when using smaller values of τ , as the probability of choosing the optimal actions in Markov model increases. Though $\tau = 1$ performs poorly, it is still slightly better than the random selection of the forwarders. Higher value of τ leads to almost equal probability of actions in Markov model, and hence, the result is closer to that of *SSR-Random*.

The *SSR* performance is compared with load-aware proactive routing schemes, ML-OLSR [21] and LTA-OLSR [24]. Both the ML-OLSR and LTA-OLSR find the best route to the destination which experiences the lowest traffic load. Averaging over the buffer load and channel occupation, LTA-OLSR performance is comparable to *SSR* with $\tau = 0.05$ for smaller data generation rates, as shown in Figure 24. However, when the source rate rises, LTA-OLSR performance degrades due to the inter-node interference in single path routing, and *SSR* ($\tau = 0.05$) outperforms it as it dispatches the data packets through a routing pipe.

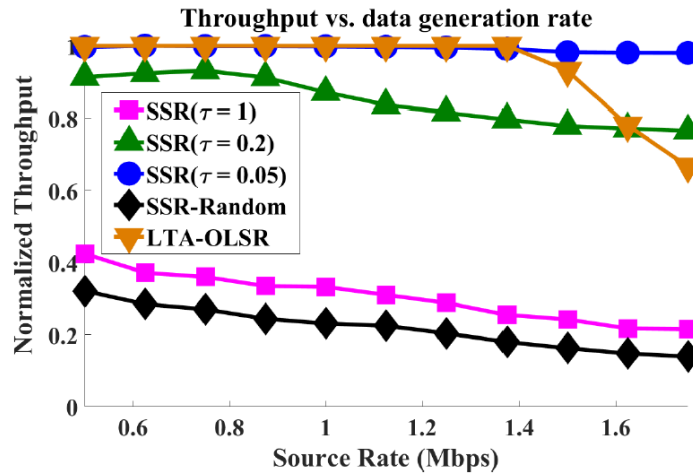


Figure 24. Throughput Comparison for Different Action Selection Schemes (with Congestion)

As shown in Figure 25, the packet delay in *SSR-Random* and $\tau = 1$ may not have an increasing trend when the source rate increases. In Figures 26(b) and (c), some packets may be dropped in the path, which helps to mitigate the congestion and reduces the delay for the rest of the traffic. This fact is also mentioned in [66]. Due to the use of the routing pipe, *SSR* with $\tau = 0.05$ can balance the traffic load and decrease the delay more efficiently compared to LTA-OLSR which uses a single-path routing.

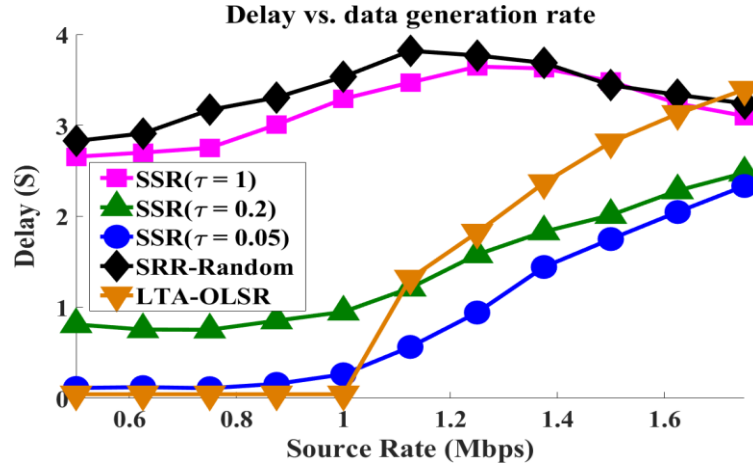


Figure 25. Delay Comparison for Different Action Selection Schemes (with Congestion)

Figure 26 compares the load distribution of *SSR* traffic across the entire leaf-like pipe, for the topology shown in Figure 20(a). Here the routing traffic is almost evenly distributed in *SSR-Random* scheme or when using $\tau = 1$, regardless of the external traffic. However, the traffic load is heavier in the start section of the pipe, but it decreases when it travels through the pipe, because some packets are dropped before reaching the destination. In these cases, *SSR* ignores the location of the congested areas and therefore cannot avoid them. On the contrary, the traffic associated with smaller values of τ can successfully get away from the congested area (see Figures 26(d), (e)).

Figure 26(f) reveals that when there is no congestion, *SSR* can evenly distribute traffic in the entire pipe, and the center of the pipe experiences slightly less traffic than the outer areas (thus the nodes in the center experience less interference). Here the destination receives more packets from the outer areas than the center.

Although small τ increases the throughput by selecting the optimal Markov actions more frequently, it also reduces the possibility of exploring other actions. Hence the routing scheme may not be able to quickly adapt to the changes in network conditions. Thus, selection of a suitable value of τ is a tradeoff between performance and learning speed in a dynamic swarm environment. In the experiments, using $\tau = 0.05$ achieves an optimal routing performance.

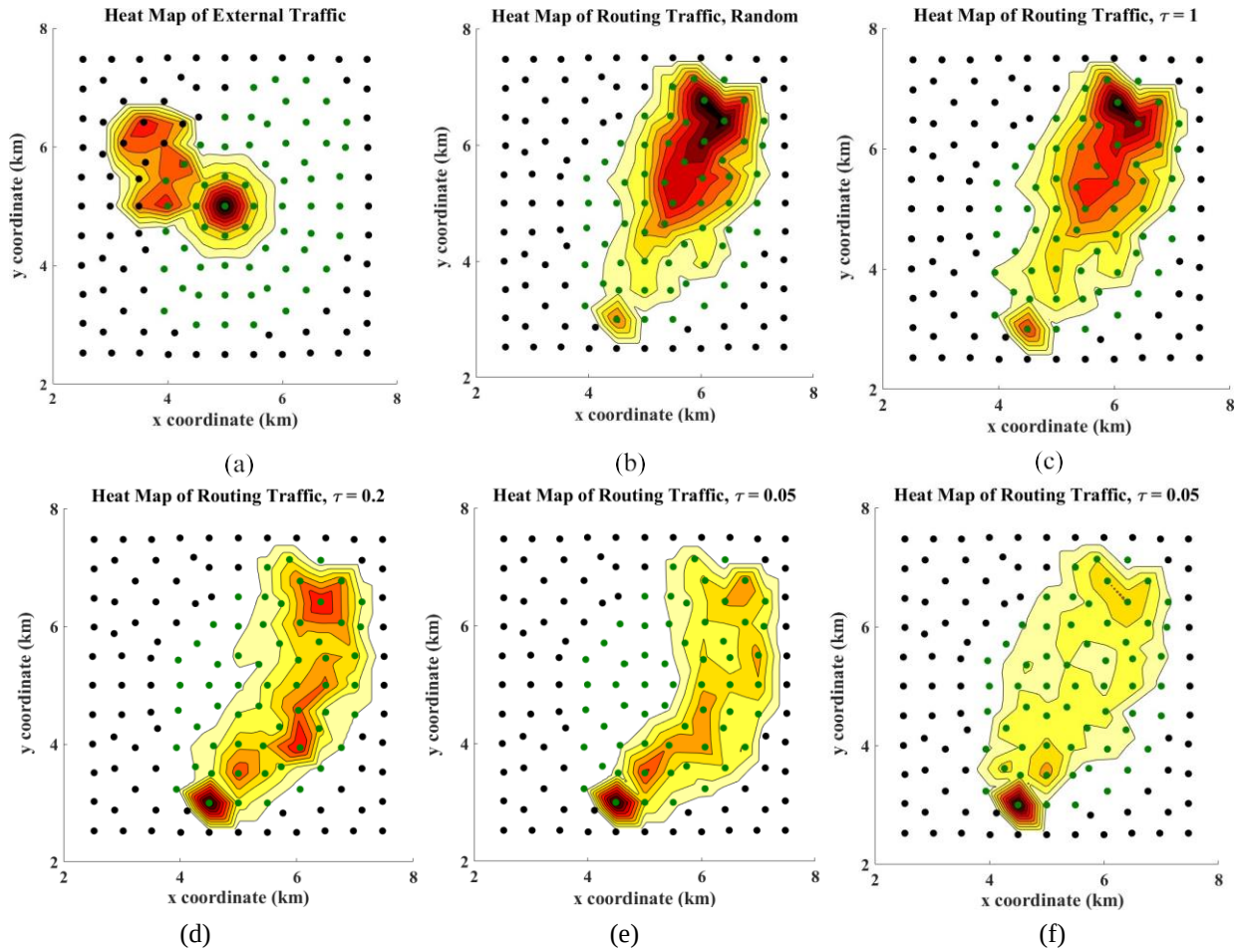


Figure 26. (a) External Load Distribution Due to Other Applications; (b) Load Distribution of SSR Traffic with Random Action Selection under Congestion; (c) Load Distribution of SSR Traffic for $\tau = 1$ under Congestion; (d) Load Distribution of SSR Traffic for $\tau = 0.2$ under Congestion; (e) Load Distribution of SSR Traffic for $\tau = 0.05$ under Congestion; (f) Load Distribution of SSR traffic for $\tau = 0.05$, without Congestion

4.4.5 SSR Performance with TSDP

Figures 27 and 28 illustrate the effect of TSDP on the SSR throughput and delay, for the congestion scenario shown in Figure 26(a). Table 5 summarizes the TSDP parameters for the SSR and the low-priority external traffic. In Figure 27, TSDP does not noticeably affect the performance when the throughput is high. However, when the queues get congested and the throughput degrades at higher source rates, TSDP improves the performance by filtering out part of the low-priority traffic as well as packets having a low chance of reaching the destination in time. Therefore, the competition for the channel access is moderate and more high-priority packets can reach the destination. Figure 28 reveals that TSDP may slightly increase the expected delay of the received packets, because more packets can reach the destination before expiration.

Table 5. TSDP Simulation Parameters

Parameters	SSR Traffic	External Traffic
p_d^{max}	0.4	0.7
SD_{min}	0.25 s	0.5 s
SD_{max}	1 s	1 s
SR_{max}	0.5	N/A
α	0.25	1

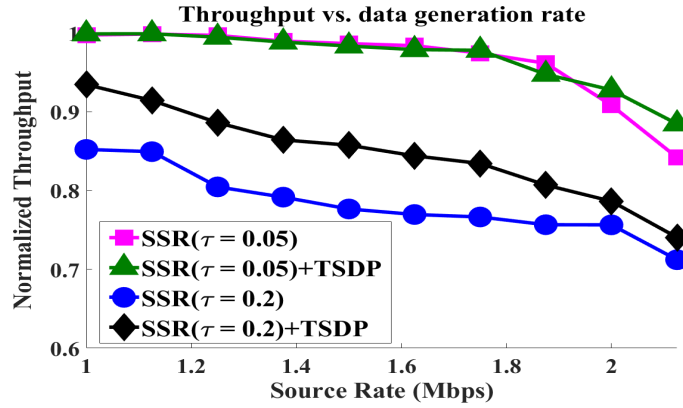


Figure 27. Throughput Comparison with and without TSDP in Congested Scenario

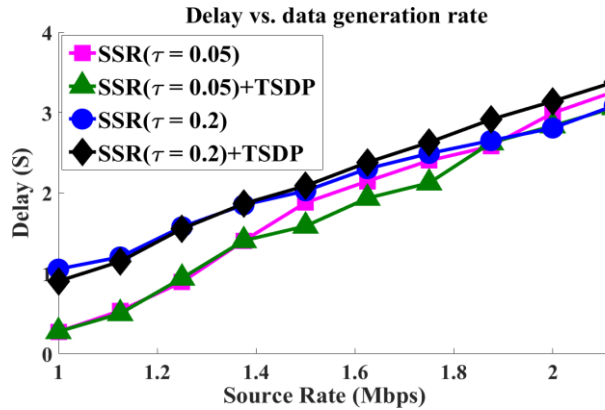


Figure 28. Delay Comparison with and without TSDP in Congested Scenario

4.4.6 Swarm Morphing

In this section, the performance of the SSR scheme is evaluated when the swarm morphs from one shape to another. Here, morphing from the square formation to triangular shape takes around 125 s. Figure 29 shows the morphing results at three different time instances.

During morphing, the geometric addresses of some nodes may change, and some other nodes may join or leave the pipe. This may also lead to an undesired change in the PF list. Generally,

minimizing the topology distortion (in terms of RF connectivity) during morphing is preferred. We thus define the *morphing distortion* as the difference (in terms of the hop-count distance) between the previous and new geometric address.

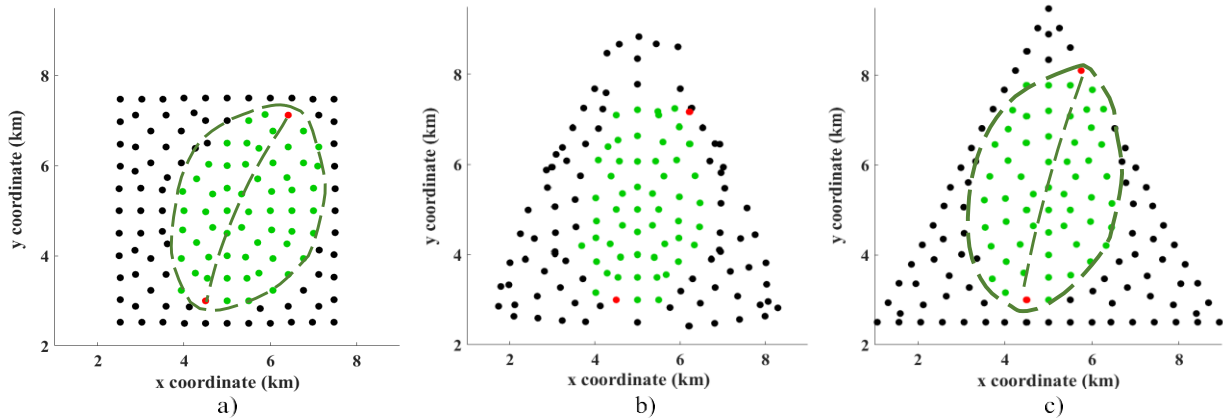


Figure 29. Swarm Morphing from a Square to Triangular Formation and Changes of the Leaf-Like Routing Pipe: (a) Initial Leaf-Like Pipe in the Square Formation; (b) A Snapshot of Swarm at the Middle of Morphing; (c) End Pipe in the Triangular Formation.

Figure 30 shows that about 58% of the nodes moved only 1-hop away during morphing, and only 6% of them moved more than 3-hops away. This implies that due to localized movement of nodes, most paths inside the pipe still have good RF connectivity. Most displacements occur in the outer areas of swarm, and the inner nodes can still maintain their established links. Moreover, the nodes do not leave their region R_i during morphing which significantly helps the local search.

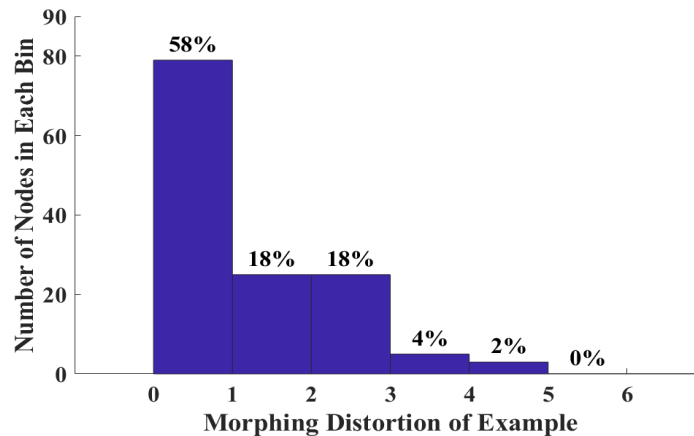


Figure 30. Morphing Distortion Measurement for the Scenario Shown in Figure 29

5. 3D SKELETON ROUTING PROTOCOL

5.1 Introduction

This scheme addresses a challenging routing problem for UAV swarm networks: How do we guarantee that any two swarming nodes, which may be located in different swarm regions (a region is a group of UAVs with the same surveillance task), can find one or more optimal inter-region routes which meet the following eight conditions: (1) It *supports 3D routing*, (2) It *can adapt to the dynamic swarming topology*, (3) It is *executed in a distributed manner* without global network control, (4) It has *low overhead and complexity* since each UAV has limited communication and computational resources, (5) It can operate in a *GPS-free environment*, (6) It meets the QoS requirements, (7) It is *easy to maintain/manage* and has minimum re-routing operations, and (8) It is *scalable* to large UAV networks.

Conventional routing schemes for general mobile ad hoc networks (MANETs) can either be proactive (i.e., an end-to-end path is always available) [77], or reactive (i.e., a path is searched on-demand) [78], or hybrid (i.e., maintain partial route information and search for the rest to build a complete path) [79]. However, none of the conventional schemes can meet all the above eight requirements. Particularly, they cannot adapt to the dynamic swarming process while maintaining a satisfactory QoS performance. Most of them blindly broadcast control messages to search for a possible route. Such a blind search can introduce a high protocol overhead for large-scale UAV networks. *Geometry* routing was proposed in [80] to utilize the network geometry features to find a short path. However, they often require the knowledge of GPS positions or at least some landmark locations. In recent years, a few 3D routing studies [81][82] have been conducted. However, they often involve high-cost 3D geometry analysis. Most importantly, they are not suitable to multi-region, dynamic swarming networks. We propose a new routing scheme to overcome all these drawbacks.

Although dynamic UAV swarming presents some challenges to the routing protocol design, it also provides three unique features which can be used to facilitate route search:

1. *Relative topology stability*: Once a swarm network is formed, it performs certain surveillance tasks for at least a short time ($>5s$). This implies that a certain amount of time is available to perform some basic, *proactive* route search operations, such as finding the 3D surface contour (i.e., the nodes located in the outer surface of the 3D network).
2. *The rule-of-thumb of minimizing node position changes during a mission*: To avoid UAV collisions and reduce the energy consumption, the swarming algorithms [83] often use efficient trajectory control schemes to minimize the movement of most nodes. Generally, the center of the network has less node movement than the outer areas. The *minimum movement* rule-of-thumb implies that a relatively more stable network region (typically the central region) can be used to deliver data via a more stable route. The routes in the outer network areas are less stable due to more drastic position changes.
3. *Smooth topology transformation*: In most cases, the UAV swarming process experiences a 'smooth' topology change. A practical surveillance application often knows the swarm shape to be formed, and each UAV is guided to move along a certain trajectory. Such a *smooth* topology change enables us to create an elegant route transform scheme, which can smoothly adjust the route (partially or entirely) to match the network topology changes.

Based on the above three UAV swarming features, we propose a new routing scheme, called **3D transformative routing (3D-TR)**, which can decompose a big topology change into a series of smooth, continuous micro-adjustments, thus minimizing the packet loss rate (due to path disruptions). Our 3D-TR concept is inspired by the human nervous system (Figure 31(1)). The human brain uses the most efficient way (i.e., with the minimum signal delivery delay) to send commands to each muscle/tissue. The trunk-stem-stemlet (TSS)-based nervous system structure plays a critical role here: (1) The *trunk* is located in the most stable part of the system (i.e., the spine). It delivers the brain signals to most body parts; (2) A few major nerves around the trunk (i.e., *stems*) help relay the signals to four limbs; (3) Many thin nerves (i.e., *stemlets*) are linked to the stems and distribute the signals to each small tissue.

We propose to use the TSS structure for designing a routing protocol in the UAV network (Figure 31(2)), since it matches the three swarming features very well. The proposed 3D-TR protocol has the following three advantages:

First, it provides a fast way to transmit data between two nodes that may be in different swarm regions. Instead of blindly broadcasting route search messages, 3D-TR allows a node to first reach the closest stemlet (within only a few hops away), and then go along the TSS to reach any node in different regions. If the destination is in the same region (i.e., share the same stem), a simple intra-region path can be used without going through other stems.

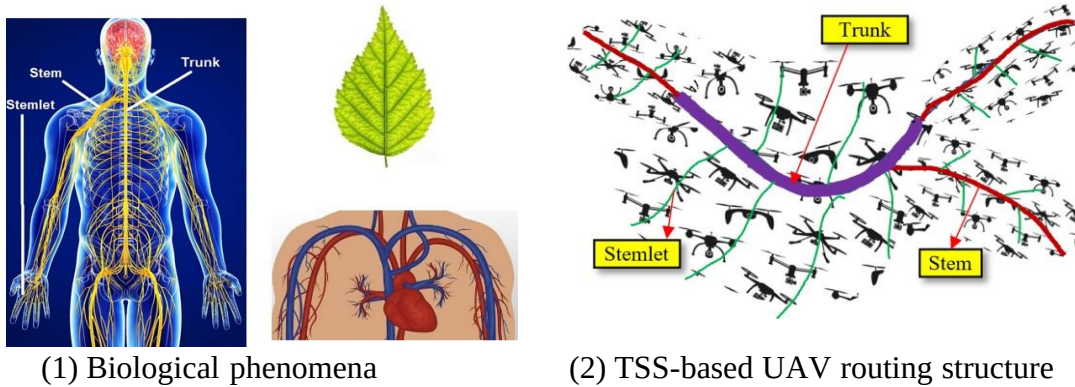


Figure 31. Nervous-System-Inspired Routing Structure

Second, we extract the *trunk* from the relatively stable swarm region (i.e., central area). The skeleton axes of other swarm regions (i.e., *stems*) can also be extracted. The nodes in the skeleton have higher stability than the nodes in the outer surface.

Third, when the UAV network formation changes, the entire swarming procedure can be decomposed into multiple small phases, and the micro-adjustments of the skeleton-based route sections can be performed in each phase. By *integrating routing topology control with skeleton adjustments in each swarming phase*, we avoid the conventional high-overhead route search in a large-scale 3D UAV network.

Despite the attractive characteristics of TSS-based routing, it is challenging to design a *practical* 3D-TR protocol that meets the aforementioned eight routing requirements. For example, given only the local network information (i.e., the distances to each neighbor of a UAV), how do we use a distributed, GPS-free approach to quickly identify the skeleton nodes? Moreover, bottleneck nodes might exist among network regions. As shown in Figure 32, from source A to destination B, the shortest path is a straight line. However, if *big data* needs to be transmitted, such a path needs to be extended to multiple paths. Obviously, due to the existence of a bottleneck in the straight-line path, a better routing strategy would be to go through another swarm region that allows the establishment of multiple paths while avoiding the bottleneck nodes.

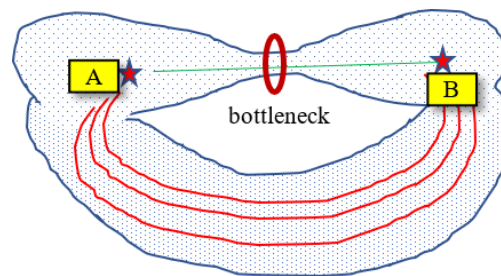


Figure 32. Need for Bottleneck-Aware Routing

Contributions: The novelty of our 3D skeleton routing is three-fold:

1. *A low-overhead, distributed, incremental method to extract the TSS nodes (i.e., skeletons):* Given only the basic information (such as connectivity and approximate distances to 1-hop neighbors), our 3D-TR scheme can recognize the outer surface (called *the boundary*) of a 3D swarm network and assigns virtual coordinates to those boundary nodes. The entire boundary is divided into different network sections. Then an *incremental, on-line* algorithm is executed among each section of boundary nodes to quickly identify the medial axle (i.e., skeleton) of each section. These axle segments are connected to form the complete TSS.
2. *Bottleneck-aware 3D multipath route ‘highway’ maintenance:* A cylinder-like routing pipe with different thickness levels will be established to deliver the high data rate traffic. The bottleneck locations will be identified to avoid network congestion. A minimum cost 3D multipath search will be used to find the best paths across different skeleton segments.
3. *A smooth route transform to adapt to the dynamic network topology:* To ensure that the routes match the changes in the network topology, our scheme performs a smooth ‘path handoff’ to minimize the packet loss rate. Multiple path adjustment scenarios will be considered, such as skeleton axle’s shifting, expansion, condense, rotation, etc.

The 3D-TR scheme considers the QoA parameters and is robust to swarm changes. Particularly, the skeleton nodes are typically located in the most stable regions of the network. The QoA metric – core stability level, can be used to measure the topology stability during swarm shape changes. A higher core stability indicates a longer lifetime of the established skeleton framework. 3D-TR is also robust to swarm evolutions by switching between the skeleton nodes and backup nodes.

5.2 Related Work

UAV swarming networks and protocols: Many UAV swarm establishment algorithms have been proposed in the literature. For example, an adaptive and reliable network structure which could maintain swarm connectivity was proposed in [83]. The work reported in [84] tackled the problem of UAV swarm formation and proposed a bandwidth-efficient multi-robot coordination algorithm. A survey regarding UAV swarm control was given in [85]. All the above UAV swarm control algorithms aim to maintain a stable swarm shape with high-speed communications. However, none of them provided an elastic routing scheme suitable to the dynamic UAV swarming process, and they cannot deliver smooth traffic flows over low-cost, 3D-oriented, GPS-free, proactive paths.

Skeleton extraction algorithms: Some skeleton extraction algorithms for 2D or 3D geometry shapes were discussed from a computer graphics viewpoint in [86][87]. However, they have not been applied to wireless network applications due to the difficulty of obtaining the global geometry knowledge in a distributed UAV network. A pioneering work on network skeleton extraction was presented in [88]. Unlike traditional schemes which assume the knowledge of complete boundaries, its algorithm only uses the connectivity information. A bottleneck detection in 3D sensor network was suggested in [89]. In our work, we will utilize the geometry models in [88][89] to come up with a practical, comprehensive skeleton routing protocol.

3D routing schemes: The boundary detection is the first step of 3D localization and routing. A precise boundary/surface detection algorithm was proposed in [90]. End-to-end routing for 3D MANETs has attracted some attention. For example, geographic routing for 3D surface [91] [92] can partition the whole network via 3D restricted Delaunay triangulation. By identifying intersecting triangles and edges, efficient routes are constructed on the surfaces by bypassing the local minimums. The concept of Bubble routing was proposed in [93] to guarantee the packet delivery success. *The main drawbacks of conventional 3D routing include two aspects:* First, they did not use a low-overhead route adjustment to adapt to the dynamic swarming network. Because a UAV swarm has a ‘smooth’ topology transform, it is unnecessary to make drastic changes for the entire end-to-end path. Second, those schemes did not utilize different mobility features between central and outer areas in a UAV network. They simply used the consistent path establishment and maintenance scheme. Our 3D-TR scheme differentiates the mobility patterns between central and outer nodes and identifies the ‘trunk’ located in the core area, and then only searches the shortest paths to find the closest ‘stemlets’. Such a TSS-based routing scheme has much lower protocol overhead than the conventional methods.

5.3 Methods, Assumptions, and Procedures

5.3.1 TSS Extraction Protocol

In this section, we describe a practical skeleton (i.e., TSS) extraction protocol from a 3D UAV network based on surface node localization and computer graphics algorithms.

A. GPS-Free, Distributed 3D Surface Localization

A.1 Identification of Boundary Nodes

It is important to detect the nodes located in the *outermost network surface* (called *the boundary*) since they represent the approximate 3D shape of the UAV network. Due to the rule-of-thumb of minimum node movement, the boundary nodes typically change their locations with a larger scope than the inner nodes. These boundary nodes can be used to determine the skeleton (i.e., the TSS nodes around the medial axes of each swarm region). Since no GPS information is available, the boundary detection algorithm relies on the local *connectivity* information among nodes, which can be easily obtained via any of the existing neighbor discovery schemes, such as [94].

There are only a few studies on 3D boundary detection in wireless networks. As an example, CABET [95] assumed that a boundary node has less neighbors than an inner node. However, this assumption is not valid in UAV networks without an even node density. A more effective approach is to use the model of *triangular tetrahedral geometry* [96], which is a distributed approach and can start from any node (called the first landmark node). The starter node broadcasts “landmark search” messages to find its k -hop nodes. All non-landmark nodes get associated with the closest landmark nodes. Virtual edges are added to interconnect all the landmark nodes to form a 3D Voronoi mesh structure, which consists of a set of tetrahedra without crossing edges and faces.

Definition 1 (3D boundary): The boundary of a UAV swarm network consists of all tetrahedra faces, each of which has only ONE neighboring tetrahedron, which is called an internal tetrahedron since it is inside the boundary surface. Each *internal* tetrahedron, is always shared by TWO neighboring tetrahedra.

Next, we need to ‘seal’ the whole surface which separates the boundary and internal nodes, by using the Dijkstra’s algorithm [97]. Note that the boundary surface detection needs *continuous refinement* since some true surface nodes may not be detected by the algorithm in one round.

The boundary landmark expansion procedure [96] can be used to find more surface nodes in each iteration (Figure 33). In this project, we do not run through the entire refinement process as in [96], since such a refinement is more time consuming. Moreover, our 3D localization scheme does not need such a high-resolution result. Therefore, we only run a small number of iterations (<5) to find enough boundary nodes. The stopping point of the iterations can be controlled by a simple index, i.e., *average distance* (d_{ave}) between two boundary nodes. If the effective transmission (Tx) range of a node is d_0 , the boundary search process is iterated if d_{ave} does not exceed d_0 (i.e., two neighboring nodes can communicate with each other). Note that d_{ave} can be easily obtained by exchanging the received signal strength (RSS) levels among the neighboring boundary nodes [97].

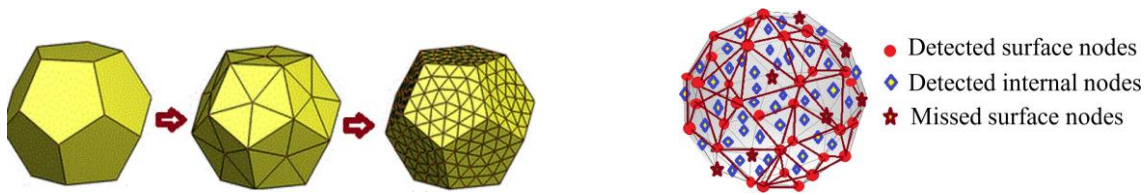


Figure 33. (1) Coarse-to-Fine Surface Detection, (2) Imperfect Surface Detection

Unlike [96], our surface building process can stop in a relatively short time (with low communication overhead), and the generated surface has a reasonably good sealing level to hold all the internal nodes. There could still be a small number of undetected surface nodes. Later, we will show that even with an *incomplete* boundary surface, our algorithm can extract the skeleton from a 3D UAV network.

A.2 Surface Node Localization

Our scheme does not need the exact GPS locations. However, to perform skeleton routing, it is necessary to assign each node a *relative* space position (also called a *virtual coordinate*). Given the simple distance estimation between surface nodes, we can use the 3D localization algorithm to find the approximate relative positions of each node. Once we assign a reference coordinate to one of the nodes, other nodes' coordinates can be easily determined (Figure 34).

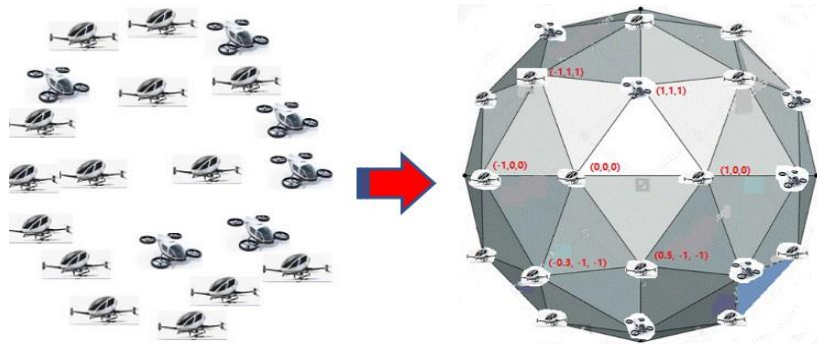


Figure 34. 3D Localization Result

With UAV altitude information, as well as the estimated distances between surface nodes, we could use a distributed algorithm *cut-and-sew* [98], to achieve 3D surface localization. Its basic idea is to first find the projected point locations in the X-Y plane for a 3D node, and then apply a low-cost 2D localization method (such as distributed multidimensional scaling (MDS) [99]) to determine its X-Y coordinates, which are combined with the *altitude* (Z) to form the 3D coordinate (X, Y, Z).

B. TSS Establishment for Incomplete 3D Surface

B.1 Skeleton Extraction

Existing 3D network surface detection and localization methods [98] are mostly based on computer vision algorithms. However, so far there are no effective methods that can quickly and accurately detect the TSS nodes from a 3D surface network. The main challenges include the following four aspects:

(1) *Missing values*: Due to time and memory limits, the above surface node detection may not be complete (see the stars in Figure 33(2)), which means that the skeleton must be detected even with incomplete surface. (2) *Noisy values*: Due to the distance estimation inaccuracy, each identified location is often a noisy value, which can affect the detection of actual skeleton axes.

(3) *Erroneous values*: Due to the approximation nature of the above triangulation model, a small number of internal nodes may be mistakenly identified as the surface nodes. Thus, the 3D surface has ‘outlier’ nodes. A good TSS extraction algorithm should be able to overcome the effect of these outliers. (4) *Distributed, incremental computations*: When we convert the computer vision algorithms to practical network protocols, all the computations must be performed in a distributed manner, i.e., each node conducts *local* computations without a global control. Moreover, the computation must be incremental, that is, it starts from a small region, and gradually handles new sections, and eventually combines all the detected TSS segments into the complete skeleton.

The network medial axle construction was studied in [100]. However, it does not overcome the *first* three issues mentioned above. In this project, we propose a novel TSS extraction protocol to address all the four challenges. Our scheme can correctly recognize the skeletons under a noisy, incomplete, and outlier 3D surface (see Figure 35).

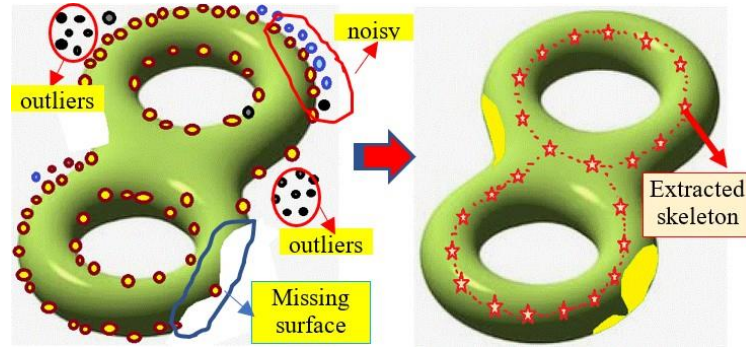


Figure 35. Our TSS Extraction Scheme Can Handle a Faulty 3D Surface

The L_1 -medial model [101] has been successfully used in computer vision to robustly find the low-dimensional skeleton tree for an imperfectly scanned 3D image. However, no study is performed so far, to apply it for wireless network applications. Here we improve such an algorithm and come up with a practical protocol for the route establishment in the 3D UAV network.

Definition 2 (L_1 -medial skeleton): Consider a group of 3D surface points $P = \{p_j\}$ (each coordinate p_j has reference X, Y, Z values). The L_1 -medial skeleton for such a 3D surface consists of an optimal set of projected skeleton points $S = \{s_i\}$, $i \in I$, which meet the conditional regularization as follows:

$$\underset{s}{\operatorname{argmin}} \sum_{i \in I} \sum_{j \in J} D(s_i - p_j) W[D(s_i - p_j)] + R(S) \quad (24)$$

Here, $D(s_i - p_j)$ is the Euclidian distance between the skeleton and boundary nodes. Obviously, the skeleton axle points have the average minimum distance to all neighboring boundary nodes; $W[\cdot]$ is the weight determined by a decaying function. Its value quickly decreases when leaving the skeleton neighborhood. $R(\cdot)$ regulates the distribution of skeleton points to make sure that all skeleton nodes are evenly distributed in the medial axes. As a matter of fact, without $R(x)$, the skeleton points become very sparse because L_1 -algorithm tends to accumulate the local medial centers into point clusters.

Eq. (24) simply uses a set of raw points for skeleton calculations. This is different from the existing skeleton extraction methods [102], which require the points to be complete, watertight, or represented by fine tessellations. In our scheme, the points are allowed to be unevenly distributed, noisy, incomplete, and with outliers.

Let ρ be a constant that represents the radius of the supporting local region around the medial axle, then we can define the weight function $W[.]$ as follows:

$$W[d] = e^{-(d^2)/(\frac{\rho}{2})^2} \quad (25)$$

The other advantage of using the L_1 -based method is that we can start with a small section of boundary points and find their projections in the center of the skeleton and then gradually extend it to other sections of boundary nodes.

To calculate the boundary nodes' projections onto the medial axles, a natural choice is to use weighted *principal component analysis* (PCA), which aims to project all nodes onto the same axle. Evenly distributed nodes are more suited for PCA calculations. Such nodes can be easily achieved through the following method: Starting from any node (it is better to start with the node with the least X or Y values, such as the left-most node), it sends messages (with the packet hop-ID field set to 0 in the beginning) to other nodes. Each time a node receives the message (it will discard any duplicate messages), it adds 1 to the hop-ID field. Only the nodes with hop-ID = $3 \cdot h$ will be selected for PCA calculations (note that we can change $3 \cdot h$ to $2 \cdot h$, $4 \cdot h$, etc., depending on the boundary node distribution density). The selected PCA node n_i uses the distances to its neighbor nodes (within 3 hops) to calculate the 3×3 covariance matrix as,

$$C_i = \sum_{j \in \text{node } i's \text{ neighborhood}} W[D(n_i, n_j)] (n_i - n_j)^T (n_i - n_j) \quad (26)$$

Definition 3 (Skeleton Direction Index): Suppose the obtained eigenvalues are $\{\lambda_{i1}, \lambda_{i2}, \lambda_{i3}\}$, where $\lambda_{i1} \leq \lambda_{i2} \leq \lambda_{i3}$. Thus λ_{i3} reflects the direction of the PCA projection line. We define the *directionality* index in its neighborhood (within 3 hops) as follows:

$$\gamma(n_i) = \lambda_{i3} / \sum_{j=1}^3 \lambda_{ij} \quad (27)$$

We can then define the regularization function as:

$$R(S) = \beta_i \sum_{j \in \text{node } i's \text{ neighborhood}} \frac{W[D(n_i, n_j)]}{\gamma(n_i) D(n_i, n_j)} \quad (28)$$

Here β_i is the adjustable balancing parameter for node i to avoid over regularization [100][103].

B.2 Network Protocol Implementation

We can utilize the incremental nature of L_1 scheme, starting from the node with the least X value (all in virtual coordinates), and evenly divide the whole swarm area into m bands (Figure 36). The value of m depends on the desired resolution of skeleton. Here we select m such that each band has approximately 3 hops of nodes. In each segment, the node closest to the middle position of each band broadcasts a neighborhood search message to identify the nodes located in its band.

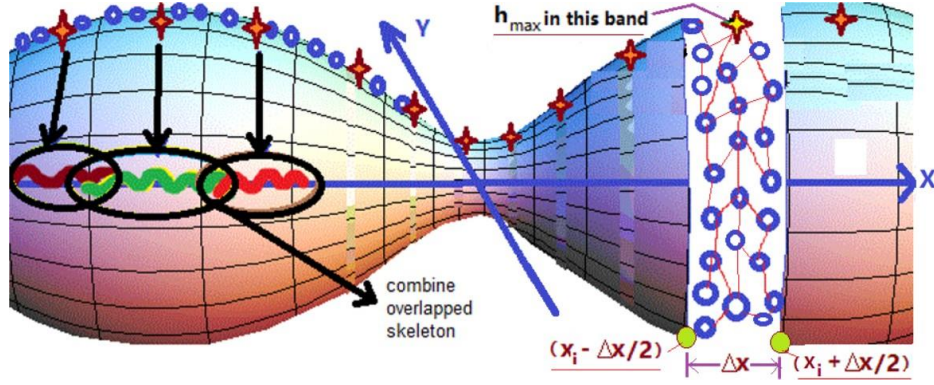


Figure 36. Practical, Distributed Scheme to Build Network Skeleton

Note that our protocol does not need an exhaustive search of all nodes in the band since the L_1 -based scheme can overcome the shortcoming of an *incomplete* boundary surface. *Protocol 1* shows our proposed skeleton extraction scheme.

Protocol 1 can be applied to both the central swarm region and other major regions. The skeleton of the central region is called a *trunk*, and the skeletons of other major regions are called *stems*. In the next section, we explain how to further find the *stemlets* by building branches starting from each trunk or stem node (thus trunk and stem nodes become the *roots* of the stemlet branches).

After we have obtained the skeleton's *trunk* and *stem* information by using *Protocol 1* (see below), we can calculate the thickness levels in different skeleton positions.

Protocol 1: Identification of Routing Skeleton

Input: Given parameters: (1) All detected boundary surface nodes' IDs via triangular tetrahedral method; (2) The virtual coordinates of all surface nodes via 3D localization; (3) Altitudes of boundary nodes via equipped barometers; (4) The number of bands (m).

Output: All nodes closest to the skeleton axle positions.

(a) Starting Phase (network division to form bands):

1) Assume the entire network has a virtual coordinate (in X direction) as $[X_{min}, X_{max}]$ (this info. can be obtained by using simple coordinate broadcasting during the 3D localization process).

2) Each node determines its band ID by comparing its X value to the segment interval: $(X_{min} + \Delta * b)$, here $\Delta = (X_{max} - X_{min}) / m$. And b is the band ID.

3) Each node broadcasts its band ID to only 3-hop away nodes to tell them its band location.

(b) Neighborhood Determination Phase (find "band members"):

1) The boundary nodes with band ID b that are closest to the location $[X_{min} + \Delta, b + 0.5\Delta]$, (i.e., the middle of each band, in terms of the X direction), broadcast a message *MID_BAND_b* around. If multiple nodes send out such messages, the node with the largest node ID will win.

2) The winner mid-band node broadcasts a message *BAND MEMBER* to up to 3-hop away nodes.

3) A node (with the same band ID) receiving such a message, responds back to the winner node with its node ID and virtual coordinate. The winner node records this information in a table.

(c) PCA-based skeleton position search:

1) The above winner node in each band uses all ‘band members’ information to run PCA based on eq. (24), to calculate the projection to the skeleton direction/axis. These skeleton positions are recorded as $\{p_s\}$, $s = 0, 1, 2 \dots$

2) The nodes near the skeleton axis (i.e., their distances $d_i = |p_i - p_s| < d_0$) are chosen as the trunk nodes in that band.

(d) Skeleton segment combination:

1) Starting from the leftmost winner, the winners exchange their Trunk node list T_{bs} , where b is band ID and s is the sequence number in that band for this trunk node. All skeleton nodes are linked together to form the complete Trunk.

(e) Stemlet forming and Thickness Index calculation:

1) Select trunk/stem nodes evenly (depending on the density of stemlets we want to build) as the roots of all stemlet branches.

2) Evenly select some nodes in each branch as stemlet nodes.

3) Calculate the hop distance from a skeleton node (could be trunk/stem/stemlet nodes) to its closest surface/boundary nodes; the average hop distance is called *thickness index*.

Definition 4 (Thickness Index): If a skeleton node, with node ID = s and band ID = b , takes on an average h hops to reach the closest k boundary nodes, then we define the *thickness index* in that particular skeleton position as: $\Omega(s, b) = h$, or write $\Omega(x, y, z) = h$ (here (x, y, z) is the virtual coordinate).

The skeleton node s can broadcast message THICK_TEST to try to reach some boundary nodes. This message can be forwarded along the stemlets. Obviously, the larger the thickness index of a skeleton node is, the thicker the 3D shape near that skeleton node is, and more paths could be built around that skeleton node.

Definition 5 (Stability Index): Given the initial swarm formation and desired formation, we can find the relative movement distance (Δd_i) of each skeleton node i . The largest movement is assumed to be Δd_{max} . Suppose the whole formation change must be finished within time Δt , then the individual stability index of each skeleton node is defined as $\Theta = (\Delta d_i / \Delta d_{max})$; and the whole skeleton’s stability index is the average of all nodes’ Θ .

For route selection, the skeleton nodes with a higher stability index are preferred since their radio links will not be easily broken. Typically, the trunk nodes have the highest stability index.

B.3 Stemlet Forming Process

The above L_1 -based protocol can find all the skeleton nodes. The *Trunk* section is typically located in the center of the network and has the largest average *thickness index* and *stability index*. All other skeleton sections are called *stems*. Below we explain the stemlet search process.

For each stem or trunk, we start from one side (typically from the node with the smallest X or Y value), and evenly sample the whole stem or trunk. If some swarm regions have higher density

levels, it is necessary to build more stemlets in those regions. The sampling interval (i.e., every how many hops) is inversely proportional to the density of nodes in the region with the stem. The method of identifying different swarm regions will be described later.

Those selected stem or trunk nodes become the ‘roots’ of the stemlets. Then each root simply broadcasts the message STEMLET BUILD to nearby nodes. This message is discarded if it is received by a *stem* node or when it is a duplicate message. When forwarding the message, the protocol targets the neighbors in the perpendicular direction to the *skeleton axle orientation* (SXO). Since each node is equipped with an inexpensive compass, a root can easily determine the SXO by observing the compass angle when it tries to talk with a neighboring stem node.

Once the STEMLET BUILD message reaches a surface node (we pick up the one which receives the message first), that surface node immediately announces the END FOUND message to nearby 3-hop nodes to prevent further broadcast. That message will also be sent back to the root along the reverse path. Then the entire stemlet’s members are determined (Figure 37). Each stemlet has an ID, which is the same as the root node’s ID.

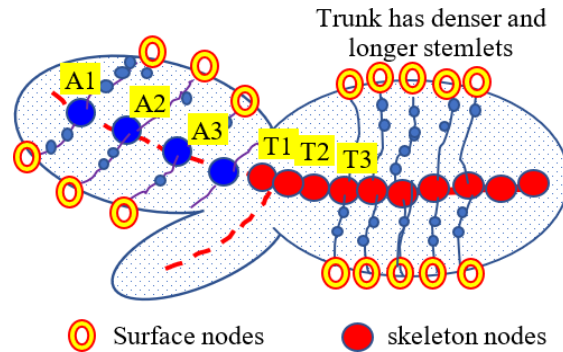


Figure 37. An Illustration of Building the Stemlets

When any node wants to send data, it first searches the nearest stemlet members. In a high-density region, a node may need to search within a few hops of scope to reach a stemlet node. After reaching the stemlet, the data is forwarded towards the root node (located in the stem).

5.3.2 Bottleneck-Aware Skeleton Routing

Since the UAVs may form different swarm regions, the above skeleton extraction algorithm should be applied to each region. In this section, we will describe a distributed method to quickly identify different regions. Once the regions are determined, we can easily distinguish between the *inter-region* routing (which crosses different *stems*) or *intra-region* routing, which occurs in the same region without crossing any *skeleton joint* (i.e., a *bottleneck* position). Note that the swarm shape can become ‘thin’ in a bottleneck, which may cause traffic congestion.

A. Swarm Network Segmentation and Bottleneck Identification

The region segmentation is based on the concept of *injectivity radius* (IR) [104], which uses the geodesic distance (GD) to find the shortest, along-the-curved-surface distance between any two points (Figure 38(a)).

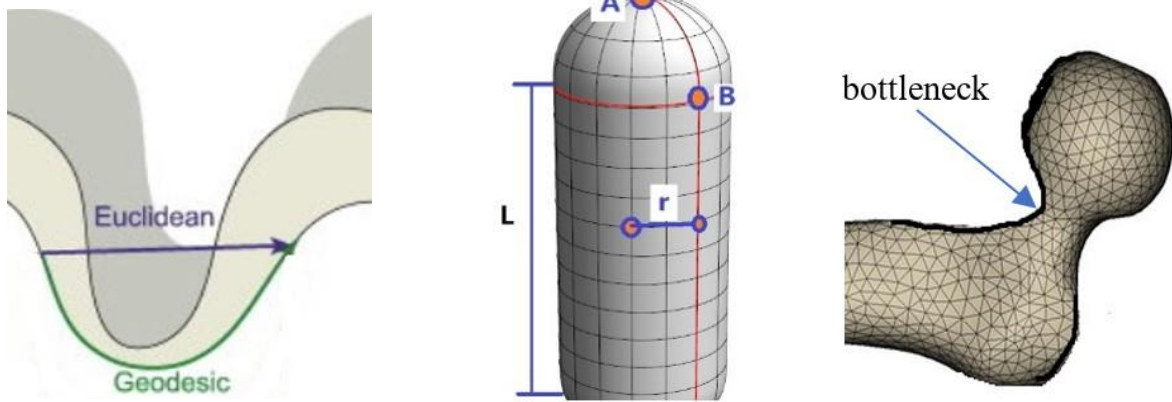


Figure 38. (a) Geodesic Distance, (b) Injectivity Radius, (c) Find Bottleneck Nodes

The IR at a point p of a Riemannian manifold is the largest radius for which the exponential map at p is a diffeomorphism. The IR of a Riemannian manifold is the infimum of the IRs at all points. As shown in Figure 38, IR at the tip point A is $D(A_to_B) + L$. We use a distributed local approach to find the bottleneck nodes (Figure 38(c)) which have smaller IRs than those of nearby points. This method utilizes the detected tetrahedron faces to calculate the approximate IR (either measured by the hop-distance or virtual-coordinate-based distance) for any surface node. Then all the surface nodes with similar IR levels form a segment, and the bottlenecks exist among neighboring segments (Figure 39).

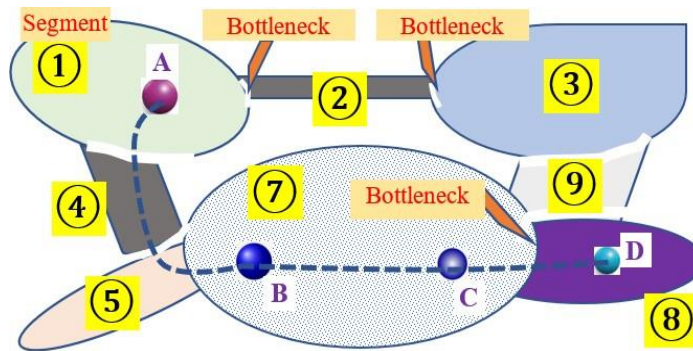


Figure 39. Swarm Segmentation and Bottleneck Identification

A simple method to find separate network regions is discussed in [100]. However, it involves high algorithmic overhead when analyzing the 3D connectivity map to recognize different groups. In this project, we propose a more efficient method that consists of three steps: (1) *Sample trunk/stem points*: Recall that we have built the skeleton, i.e., TSS segments. We can evenly sample each TSS segment (but ignore stemlets), such as selecting a skeleton point every 2 hops. Note that a smaller sampling distance provides a higher resolution of bottleneck identification. These sampled skeleton nodes serve as the *roots*. (2) *Measure stemlets' hop distances*: When the stemlet keeps 'growing' and eventually reaches the boundary surface, we use the total stemlet length (in hop distance) as the *pipe radius* (i.e., thickness level) at that root position. We define the 'concave part' as the network region where the thickness level in that location is suddenly getting lower than

a threshold but then quickly increases above the threshold. (3) *IR-based bottleneck detection*: The IR-based bottleneck detection scheme [100] will then be applied only in these concave parts. This approach avoids blind global search of bottlenecks (thus reduces the protocol overhead). Once the bottlenecks are detected, the segments (i.e., regions) can be easily identified (they are the parts between bottlenecks).

Figure 39 shows a toy example of segment and bottleneck identification. After we clearly distinguish all the small regions, we can classify the routing problems between any two nodes which may be in different regions as *intra-region* and *inter-region* routing issues:

Intra-region routing: Both source and destination UAVs are in the same region, such as the route from B to C. There is no bottleneck in the route. Therefore, we can build one or multiple paths along the TSS direction.

Inter-region routing: This refers to the route establishment across different swarm regions. The path(s) may pass through one or more bottlenecks. If we can avoid bottlenecks, less congestion events will occur.

B. Dijkstra-Based Inter-Region Routing for High-Priority Mice Traffic

In practical UAV networks, the swarming control commands often need to be sent from a leader node to other nodes. Such traffic has the highest priority, but it does not occupy much bandwidth. We call this *mice* traffic, which needs to select the best path along the TSS skeleton across different swarm regions. Here we assume the source and destination nodes belong to different regions (i.e., inter-region routing) since intra-region routing is easy to handle.

We prefer the path with the lowest end-to-end delay (Figure 40(a)) since the commands need to be quickly delivered. Therefore, the hop distance is a critical routing metric. Additionally, each region that mice traffic needs to go through should have a high *stability index*. There could be other routing metrics to be considered, such as the number of bottlenecks on the route, the *thickness index* of each bottleneck/TSS section, etc. But since mice traffic does not need a high sending rate, these factors may not be critical.

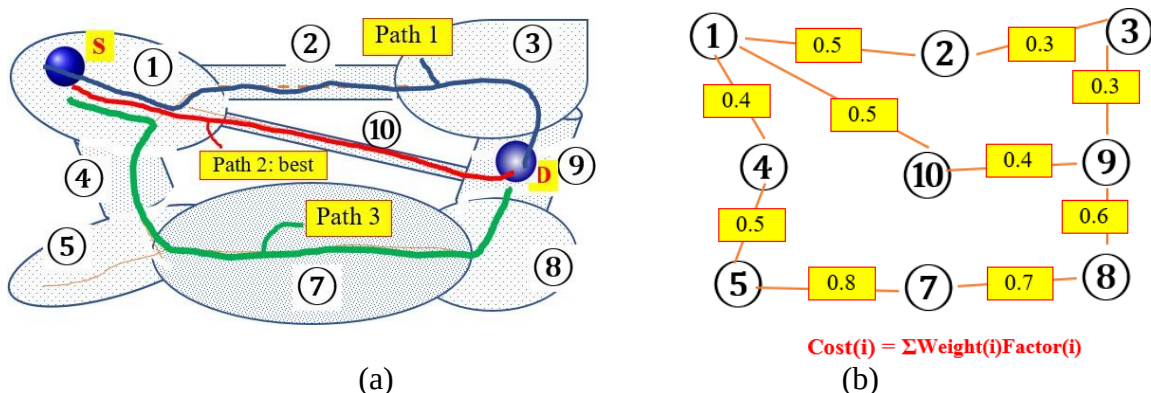


Figure 40. (a) Mice Traffic (Command Data): The Lowest-Cost Inter-Region Path, (b) Inter-Region Routing: Cost Graph

To quickly search an inter-region route, we utilize some available results (such as the detected segments/bottlenecks, stability index, and hop distance) from the source to destination (along TSS nodes), to construct the network cost graph with inter-region communication cost as the link weight(see Figure 40(b)). We then apply Dijkstra's algorithm to find the best path between any two regions. The source packets just need to reach the closest stemlet, and then go along the TSS skeleton to pass through different regions, and finally reach the destination.

To construct the Dijkstra's topology as shown in Figure 40(b), we quantitatively define the weight (i.e., the inter-region communication cost) as follows:

$$W_{ij} = \|\alpha \cdot H_{ij} + \beta\| \text{ Ave}(S_i, S_j), \text{ where } \alpha + \beta = 1 \quad (29)$$

Here W_{ij} is the total cost from region i to j ; H_{ij} is the total skeleton hop distance when crossing regions i and j ; $\text{Ave}(S_i, S_j)$ is the average stability index for all the skeleton nodes in regions i and j .

C. Multipath Inter-Region Routing for Elephant Traffic

Another typical traffic type is *elephant traffic* (such as image/video data) recorded by UAVs. A UAV node may need to send such data to the leader UAV (or a UAV center region), which determines the new coverage allocation plan among UAVs or a new swarm formation. Though such data has a lower priority than the above command data, it needs a higher sending rate to achieve the real-time monitoring. To meet the high bandwidth requirement of *elephant traffic*, we establish a *multi-region, multipath (MRMP)* routing structure, as shown in Figure 41. It consists of two steps:

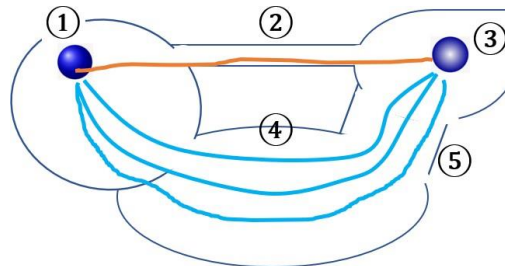


Figure 41. Illustration of Multi-Region and Multi-Route for Elephant Traffic

Step 1: Multi-region (MR) determination: First, based on all the identified swarm regions, we need to find the feasible region-to-region passing sequences, which ensure that a source region can reach a destination region with a low cost. We may need to find multiple source-to-destination *region sequences* to prepare for ‘Step 2 – multipath determination’.

If we imagine each region as a ‘virtual node’, we may use the route-seeking protocols used in multipath AODV [105] or multipath Dijkstra algorithms [106] to find feasible multi-region sequences. For example, if we adopt the multipath AODV scheme, each time we forward the RELAY REGION message in any relay region, we record all the neighboring regions that could help forward the messages. Finally, we start from the destination region, and go in the reverse direction along the recorded regions to let the source region know about all the feasible multi-region sequences. As an example, in Figure 40, from the source node's region (ID=1) to the

destination's region (ID=9), we find three feasible region sequences: 1-2- 3-9, 1-10-9, or 1-4-5-7-8-9.

Step 2: Multipath (MP) determination: We may establish one or multiple paths for each region sequence, depending on the bottlenecks. Here, the narrowest bottleneck determines the strength of the whole region sequence. For example, among the above identified three sequences, although 1-10-9 has the shortest hop-length, its bottleneck is too narrow to allow more than one route.

For any feasible region sequence, we use its regions' TSS nodes as the basis, and build multiple paths parallel to the TSS axes. Here we propose the two following methods to build multiple paths: concentric pipe construction and stemlet-to-stemlet relay (Figure 42).

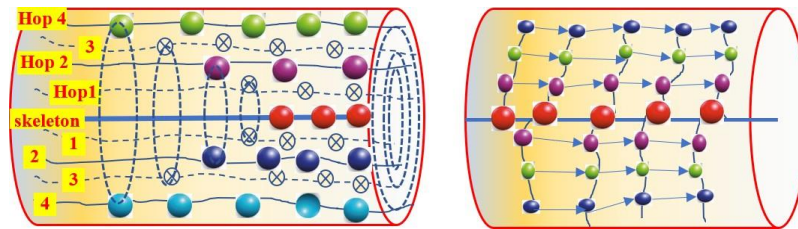


Figure 42. (1) Build Interference-Free Multipaths in the Same Segment, (2) Use “Stemlets” to Form Multipaths

(1) *Concentric pipe construction:* Since this is a 3D network, the nodes in the skeleton (trunk or stems) can simply broadcast *HOP Distance d* messages to look for a path that is d hops away from the skeleton axle (Figure 42(1)). To reduce the path-to-path interference, it may look for paths that are at least 2 hops away from each other. For example, Hop 1 nodes will be skipped, and Hop 2 nodes could form a path. Similarly, Hop 3 nodes can be skipped, and Hop 4 nodes form another path.

(2) *Stemlet-to-stemlet relaying:* Another way to form multiple paths is to utilize the stemlet nodes to directly form multihop routes without the involvement of skeleton nodes. As shown in Figure 42(2), the nodes with the same hop distance to their roots can form a path.

Protocol 2 below shows the process of MRMP protocol.

Protocol 2: Construction of MRMP

Input: Skeleton (trunk/stem/stemlet) node IDs and their virtual coordinates; Required no. of paths with QoS parameters.

Output: Multi-region, Multipath route (with all node IDs in those paths).

MR Phase: /* find region-to-region route sequence between source and destination node's regions */

- 1) Define the cost as the function of delay, bottleneck, data rate, and region node density.
- 2) The cost between two regions is the weighted sum of multiple factors (such as bottleneck thickness).

- 3) Assume each region is a 'virtual node', we then construct the cost graph.
- 4) Run multipath Dijkstra algorithm to find the top 3 (or more) feasible region-to-region sequences.
- 5) Mark out each sequence's narrowest bottleneck (thickness), sequence length (estimated total delay), etc.

MP phase: /* Construct one or multiple paths in each region-to-region sequence. */

- 1) **Concentric pipe construction:** Evenly choose roots in the skeleton. Each root broadcasts HOP ID to nearby nodes (towards the surface). The value in HOP ID represents the distance to the root. The nodes in that HOP ID will set up path(s) among them. All those nodes with the same HOP ID are in the same surface. They may form multiple paths.
- 2) **Stemlet-to-stemlet relaying:** Each root broadcasts the message HOP ID along the stemlets. One node in a stemlet finds its next relay node in another stemlet with the same HOP ID.
- 3) **Determine # of paths in each region sequence:** Find the narrowest place across the regions. The maximum number of paths depends on the thickness level in the narrowest place.

5.3.3 Swarm-Adaptive Transformative Routing

In the TSS-based routing scheme, it is critical to prepare for the skeleton position changes since some links between TSS nodes may be broken due to UAV movement during the swarming process. This section describes the TSS-based route adjustment strategies.

A. Online Regression-Based Learning for Skeleton Distortion Prediction

To achieve *smooth* communication, we can prepare backup nodes if we know that a particular TSS link will be broken based on the mobility prediction results. In our scheme, a TSS node maintains two relay links at the same time: one is the original link along the TSS direction, and the other is the backup link provided by the backup nodes. The backup nodes are close to the TSS nodes (at most 1- or 2-hops away).

To identify the backup nodes, we need to predict the next-phase locations for the nodes near the TSS axle. First, we propose the concept of the *skeleton prediction zone* (SPZ) (Figure 43). The SPZ members consist of all TSS nodes and their 1-or 2-hop neighbors. The SPZ prediction algorithm does not need to be executed in all TSS nodes. Instead, we can evenly select some TSS nodes, such as every 2 hops in TSS direction. Each selected node maintains the virtual coordinates of its SPZ neighbors (within 2 hops).

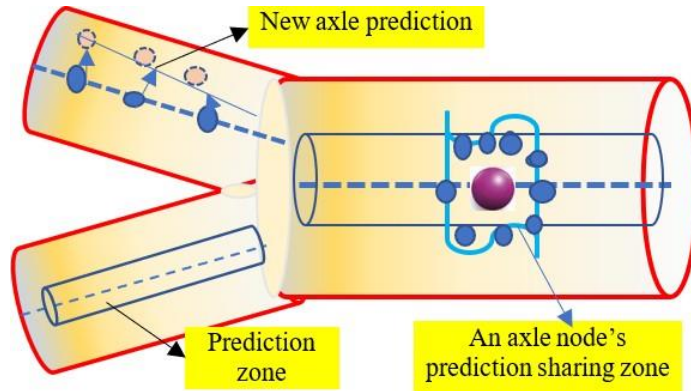


Figure 43. The Concept of the Skeleton Prediction Zone (SPZ)

Each selected TSS node will be responsible for the trajectory prediction of its SPZ nodes. This is a high dimensional prediction problem. We propose to predict the SPZ's new locations accurately and quickly via the Hidden Markov Model (HMM) algorithm [107]. Note that the prediction variable could use either the *polar coordinate system* (consisting of compass angle and movement distance) or virtual coordinate (i.e., Cartesian coordinate system) changes. In our previous work, we successfully used HMM to predict the aircraft trajectory [108]. Here we use both coordinate systems.

B. Smooth 'Route Handoff' Under Different Formation- Changing Scenarios

After we apply the HMM-based prediction algorithm, we know the new locations of the nodes within the SPZ. If a link involves the node that moves out of SPZ, we perform the *route handoff (RH)*. One benefit of using TSS-based routing is that we can perform RH locally near the broken link(s). Here we first consider six typical UAV formation transform cases during the swarming process. Then we consider a general swarming case (i.e., 7th case).

Case 1: Flat shift (i.e., the whole TSS axle shifts to a new level): This is a typical swarming behavior. A particular region needs to move to another area for new coverage. Thus, its TSS axle shifts flatly, see Figure 44(1). From the Cartesian coordinate system perspective, the skeleton nodes experience the similar virtual coordinate changes (Δx , Δy , Δz). If we use HMM-based trajectory prediction, we can obtain the new virtual coordinates of the TSS nodes. In this case, we do not need to perform *RH* since the original RF links will not be broken after the shift.

Case 2: Axle rotation (Figure 44(2)): Another common swarm adjustment is to simply rotate a local region. From the polar coordinate system perspective, if we choose a TSS node in the bottleneck position as the original point of the polar coordinate system (say, node A in Figure 44(2)), then the whole region's skeleton simply rotates by a certain angle θ . Here, each skeleton node keeps the same *hop distance* to node A. For the rotation case, the RF link connectivity is well maintained since the relative distance between any two neighboring skeleton nodes does not change.

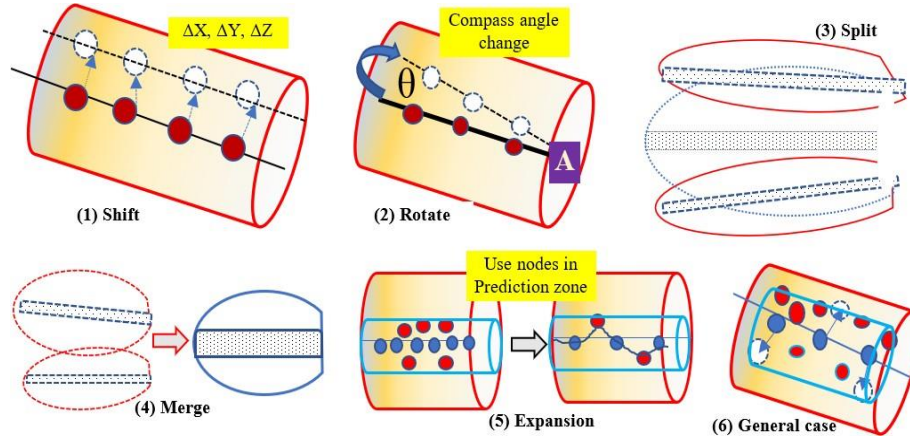


Figure 44. Illustration of RH Cases (Based on Different Swarming Patterns)

Case 3: Region split (Figure 44(3)): In this case, the original skeleton axle will totally disappear, and two new TSS axles will be formed. Our routing strategy in the swarm-splitting case is to build the backup paths (called *side paths*) across some stemlet nodes. As shown in Figure 45, besides the original path along the TSS (before split, marked as ① in Figure 45, called the *main path*), we also establish two backup paths across some stemlet nodes. When the region is split into two parts, their new skeleton axles (②, ③) can temporarily be the side paths parallel to the main path ①. Each path consists of the stemlet nodes with the same hop distance to the *roots* in the main path ①.

By using the trajectory prediction results, we can build the above two side paths before the swarm split. The packets will go to two side paths (instead of the main path) during swarm splitting. Once the packets reach the end of the side paths (i.e., the bottleneck plane), they will be aggregated into one traffic flow in the next region's TSS skeleton.

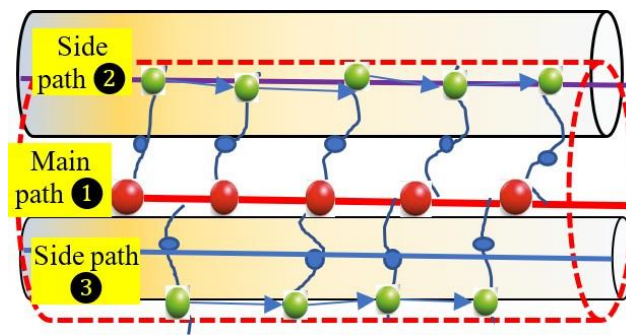


Figure 45. Routing for Swarm Splitting

Case 4: Region merge (Figure 44(4)): This case is opposite to the above split case. According to the HMM-based trajectory prediction result, we will know which *side path* in Region 1 (here it is Figure 46 Axle ①) will be merged with which *side path* in Region 2 (here it is Figure 46 Axle ②). Each side path consists of a series of stemlet nodes with the same hop distance to their roots. These two side paths will be merged into the *main path* in the merged region. During swarming,

whenever we have data going through the axle ①, those packets will go through the detour along the side path ①; likewise, data going to axle ② will go to side path ② instead. After the merge process is finished, the packets in those two side paths are aggregated into one traffic flow in the new route (i.e., the merged axle).

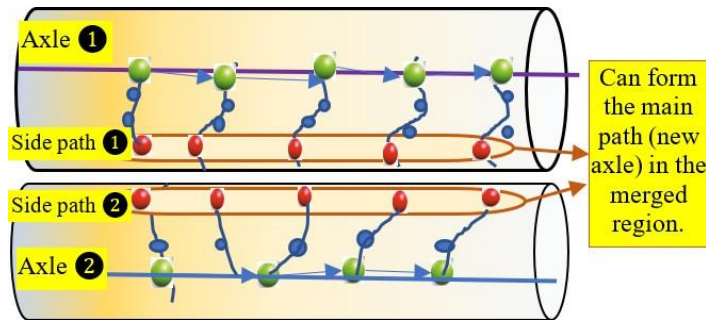


Figure 46. Routing When Swarms Merge

Case 5: Axle expansion (Figure 44(5)): In many UAV swarming applications, with *the same total number* of nodes, the entire network may need to cover a larger surveillance area. Thus, the entire network gets sparser. When two neighboring TSS nodes get too far away from each other, the link between them is broken.

Our solution to the above issue is to prepare backup nodes for the links that are predicted to be broken. Each TSS node (for example, the sender shown in Figure 47) maintains a table called the *Backup Nodes List* (BNL), which is the list of all nearby 1- or 2-hop neighbors. Once the prediction tells that its downstream link will be broken, it can immediately switch to a BNL node (here it is node A in Figure 47) which has the following feature: It has a good, received signal strength (RSS), in terms of both the current TSS node (i.e., the sender) and the next TSS node (i.e., the receiver) that will move away from the current position.

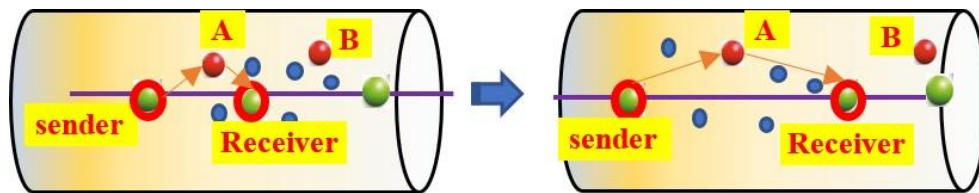


Figure 47. Routing for the Axle Expansion Case

Case 6: Axle compression: This is the inverse case of Case 5. Since the RF distance gets smaller, the RSS becomes higher. Thus, we do not need the help of any backup nodes. If the network gets too dense, we can skip some relay nodes in the skeleton, if the RSS is still satisfactory to each TSS node.

Case 7: General case (Figure 44(6)): When the UAV swarming behavior does not belong to any of the above categories, we call it a general case. Here, we propose the concept of a *route*

backup zone (RBZ). As shown in Figure 48, RBZ consists of all 1- or 2-hop away nodes near the TSS.

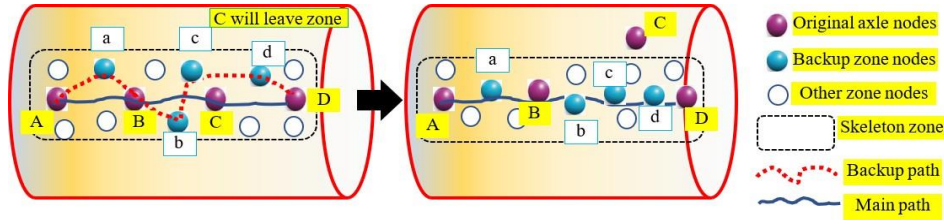


Figure 48. Handle General Route Transformation Case: (1) Before Micro-Adjustment; (2) After Micro-Adjustment

The RBZ nodes with the following three features are marked as “Backup Candidates” (BC):

- (1) Each BC will move to a position that is very close to the skeleton.
- (2) The BC has the satisfactory RSS performance to the closest two TSS nodes, denoted as A and B (before swarm change). Suppose the minimum RSS value is ρ , then $RSS(A \rightarrow BC) > \rho$, $RSS(BC \rightarrow B) > \rho$.
- (3) The BC has a small RSS variance, that is, its communication quality does not fluctuate much. Suppose a path needs to pass through the axle $A \rightarrow D$, we will maintain at least one more route (called *backup path*) which consists of both TSS nodes and BC nodes.

Protocol 3 below shows the operations of RH process.

Protocol 3: Route Handoff (RH) Protocol

Input: (1) Source (S), destination (D), and original route. (2) (pre- swarming) 3D localization and skeleton node determination; (3) (Prediction of post-swarming locations) Apply HMM to estimate the new locations of skeleton nodes; (4) (Back-up nodes): maintain the nearby backup nodes which could compensate for the possibly broken links.

Output: The route performs the handoff to the new calculated S-D path with updated skeleton section(s).

Case 1: Shift (the entire skeleton section is flat shifted to a new location).

- 1) The 3D coordinates of all skeleton nodes simply perform the same degree of change.
- 2) Then S and D find the closest stemlets to reach the stem or trunk nodes of the new skeleton.

Case 2: Rotate (the whole skeleton axis is rotated by a certain degree)

- 1) Use the polar coordinate system to define the predicted new locations of skeleton nodes.
- 2) Each skeleton node keeps its original distance (d) to the rotation axle, and only changes the angle.

- 3) S finds the closest stemlet nodes to reach the new skeleton.

Case 3: Split (a big swarm region is split into two parts)

- 1) The original stem nodes send SPLIT HALF messages to nearby stemlet nodes.
- 2) If a stemlet node is in the center position between the rootnode (i.e., stem node) and the surface node, it replies to the root with a NEW AXLE YES message. The roots confirm each selected stemlet node.
- 3) All stemlet nodes selected by the roots form a multi-hop path, which serves as the new skeleton of the split region.

Case 4: Merge (two swarm regions merge into one)

- 1) Stem nodes send a NEW STEM message to the surface nodes which will be merged with another region's surface.
- 2) The surface nodes of two regions which receive NEW STEM send back STEM REPLY to stem nodes.
- 3) Some of the surface nodes which are located near the new skeleton axle are evenly selected as the new stem nodes.

Case 5: Expand (the original swarm region expands its size evenly)

- 1) If it is an even expansion towards all directions, the stem axle stays the same; however, the node density becomes sparser. Therefore, the original stem nodes simply send out EXPAND messages to neighbors.
- 2) The nodes which receive the EXPAND from the two closest stem nodes (it can use signal strength to find the closest stem nodes), will join between these two stem nodes and become new stem nodes.
- 3) The newly joined stem nodes send out NEW STEM messages to all stem nodes.

Case 6: Compression (the whole swarm region evenly compresses its size towards the skeleton)

- 1) The skeleton nodes send out a RSS DETECT message to each other.
- 2) If a stem node finds it is too close to its upstream and downstream nodes, it sends out a STEM QUIT message.

Case 7: General case (cannot be classified into any of the above 6 cases)

- 1) Each stem node maintains a table on 1- or 2-hop neighbors located in the Backup Zone (via the HMM-based trajectory prediction algorithm).
- 2) During swarming, if a stem node finds its RSS is getting weak, it sends a NEW STEM message to Backup Zone.
- 3) A node closest to the 'weak' stem node sends back STEM REPLY to join the stem.
- 4) An original stem node which detects that it is leaving both upstream and downstream stem nodes, will send a STEM LEAVE message to other stem nodes to claim that it is not part of the stem anymore.

5.4 Results and Discussion

5.4.1 TSS Extraction

We have built the discrete-event and time-based simulators in C++ and MATLAB to validate the TSS-based 3D routingscheme. We first used C++-based package/tools to verify the efficiency of boundary and bottleneck detection algorithms. For a 3D UAV swarm network, as shown in Figure 49, we can simply use the RF connectivity information to determine the boundary nodes.

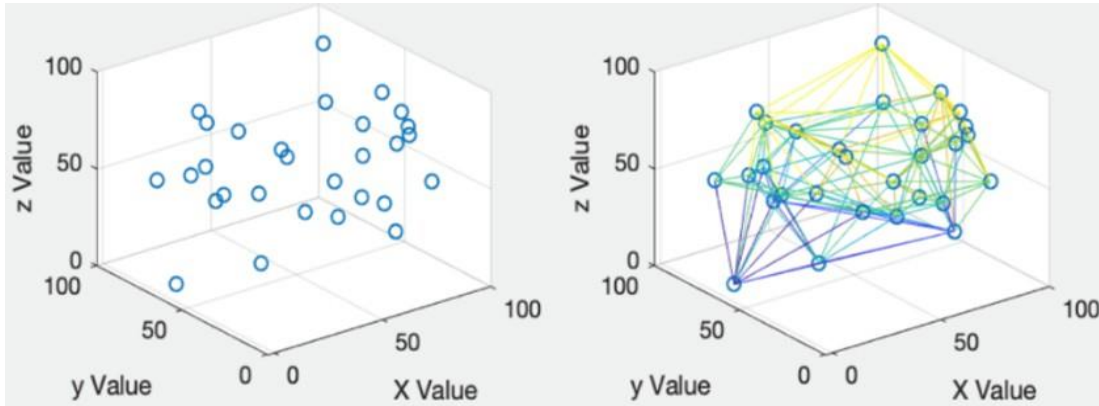


Figure 49. Triangulation-based 3D Node Localization

Figure 50(1) shows a large-scale UAV network (>100). The middle figure (Figure 50(2)) shows the ground truth of the surface nodes, and Figure 50(3) shows the result from 3D boundary detection scheme. As we can see, even without GPS information, the algorithm can distinguish between the boundary and inner nodes. There are slight differences on the surface nodes between two swarm regions. This issue can be addressed by using the bottleneck determination scheme. Only the surface nodes are needed for TSS extraction. These surface nodes are assigned virtual coordinates for the determination of skeleton.

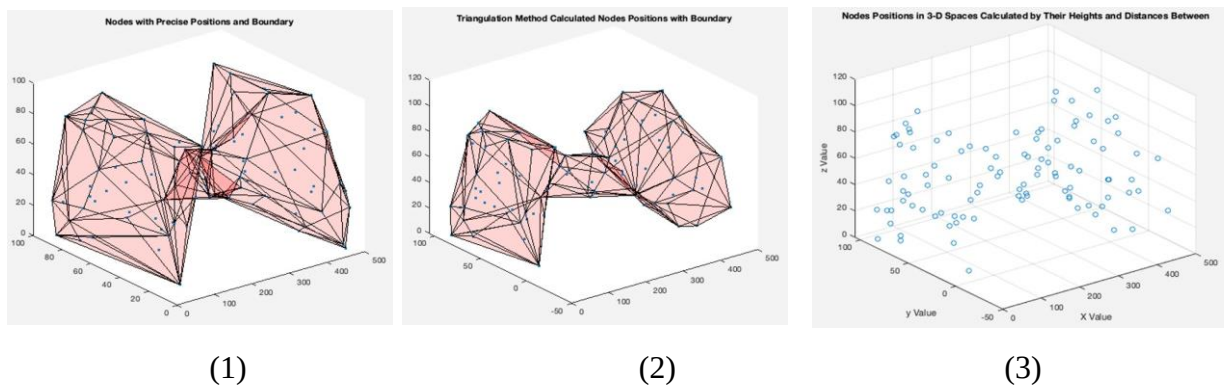


Figure 50. (1) Large-Scale UAV Network; (2) Ground Truth of Surface Nodes; (3) Detected Surface

Then we use the L_1 -medial skeleton identification scheme to extract the TSS nodes. As explained before, our scheme can tolerate missing nodes, outliers, location noise, etc. In this simulation, we have built a network with an irregular shape, shown in Figure 51(1). We have used different colors to show the connections between different regions. For example, the yellow tunnel links two remote regions. As we can see from Figure 51(2), the L_1 -medial algorithm can successfully capture the skeletons (trunk or stems) of each region in a large UAV network. Moreover, our IR-based algorithm can identify the interfaces between regions, i.e., bottlenecks, as shown in Figure 51(3).

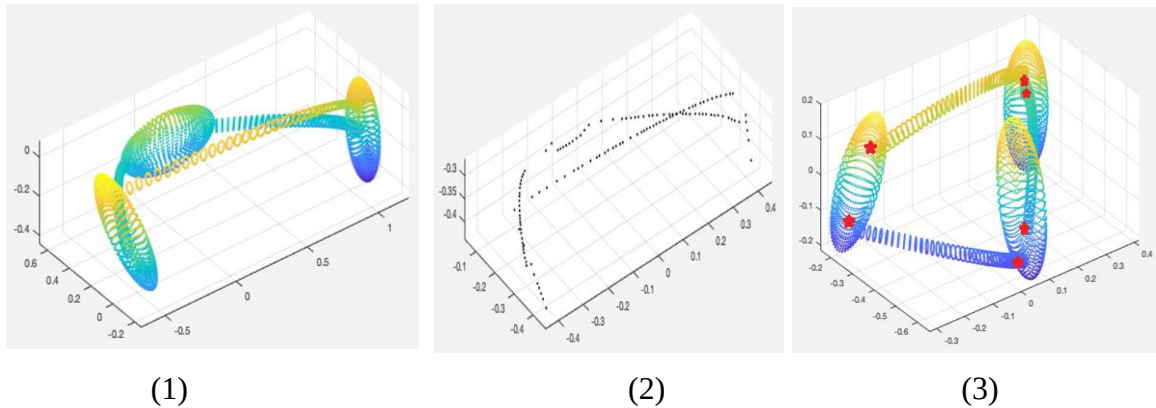


Figure 51. (1) Original 3D Shape; (2) L_1 -Medial Skeleton; (3) Bottleneck Detection

5.4.2 Performance of TSS-based Skeleton Routing

A. Event-based Simulation

We have built both event- and time-based simulations based on the standard UAV mobility model [34] to verify the performance of our 3D-TR routing scheme. The event-based engine makes the simulator respond to special network events (such as packet generations or swarming behaviors). Multiple threads can be executed in parallel. The time-based engine uses the time as the x-axis and observes how the network evolves into a stable status.

Assume that all nodes are randomly deployed in an area of 10 km x 10 km. The average link distance is 1 km, and the radio transmission range is less than 2 km. The average link rate is 10 Mbps and the packet size is 1500 bytes. We observe the end-to-end throughput and delay in different network scenarios (static or swarming).

Multiple paths can be established between a source and destination across different swarm regions. As explained before, the number of paths depends on the narrowest bottleneck location. Because multiple paths may cross entirely different swarm regions, there is very little interference between them. But there could be interference among the paths that pass through the same region sequence.

Figure 52 shows the throughput and delay performance in a UAV network without dynamic swarming or when the swarm reaches a stable state (i.e., without big formation changes). Here three traces mean that three different paths are established across different regions (no two traces belong to the same region sequence). As we can see from Figure 52(b), the delay of the 3-trace

case is much lower than the other two cases. On the other hand, the 1-trace case reaches the stable state when the source rate is only 200 packets/sec. In Figure 52(a), the throughput of the 3-trace route is almost triple that of the 1-trace route (after reaching saturation).

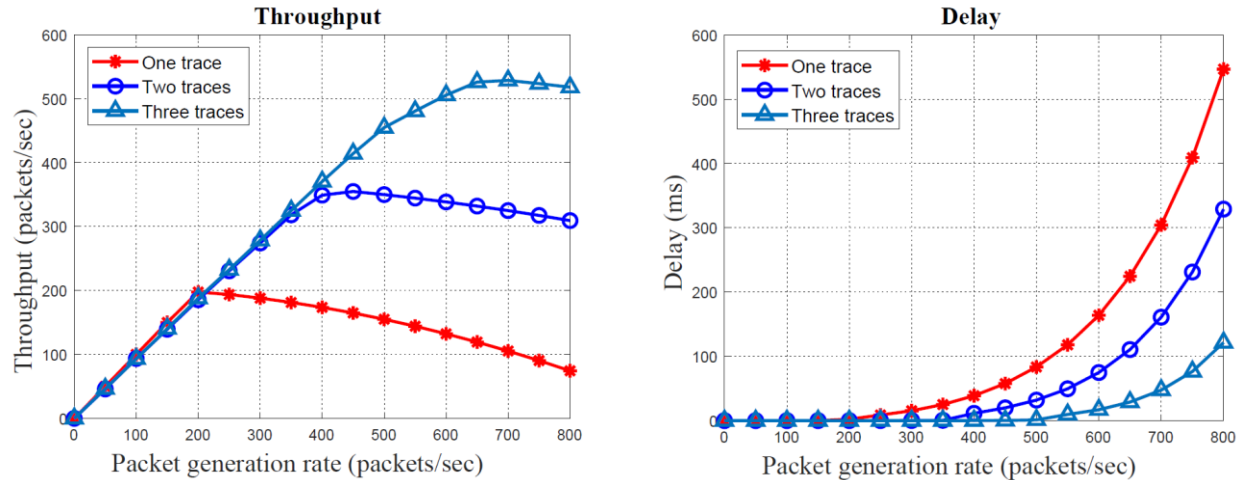


Figure 52. Throughput and Delay Performance for 1-3 Traces in Stable Swarms

Figure 53 shows the throughput and delay performance during *dynamic swarming*. The fluctuations in the curves represent the impact of swarming behavior. The temporary delay increase, or throughput decrease are all due to the in-progress swarming behaviors. Such QoS fluctuations can be reduced if we perform smooth adjustments of the formations.

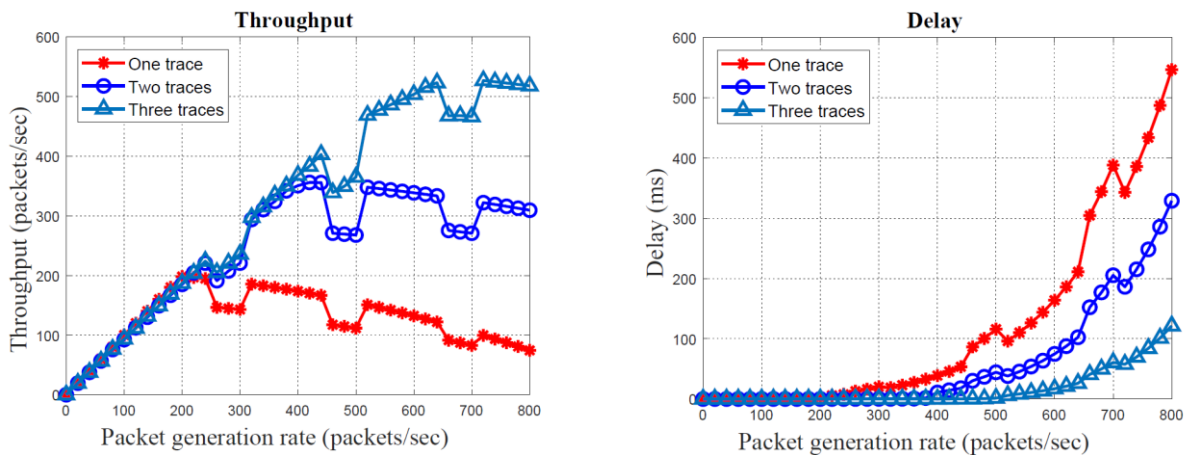


Figure 53. Throughput and Delay Performance for 1-3 Traces in Dynamic Swarms

We observe that the multi-region, multipath (MRMP) solution (i.e., the 3-trace case) provides a much better throughput and delay performance than the 1-trace case. Because our routing scheme can automatically establish multiple paths if the bottleneck place is wide enough between regions, the elephant traffic can be sent through inter-swarm routes.

B. Time-based Emulation

The above simulations helped us to observe the network's traffic handling capability by increasing the packet generation rate. We also study the network's QoS performance with time for a fixed data transmission rate, by using the time-based (instead of event-based) simulation engine.

In a practical UAV network, UAVs can maintain a relatively stable topology. Each UAV has a small mobility around the desired monitoring locations, or it could receive a new swarm command and perform very smooth location changes. For such stable cases, the throughput and delay performance smoothly reach a stable status (see Figure 54). Here the MPMR (3-trace) always has better performance than the 1-trace case.

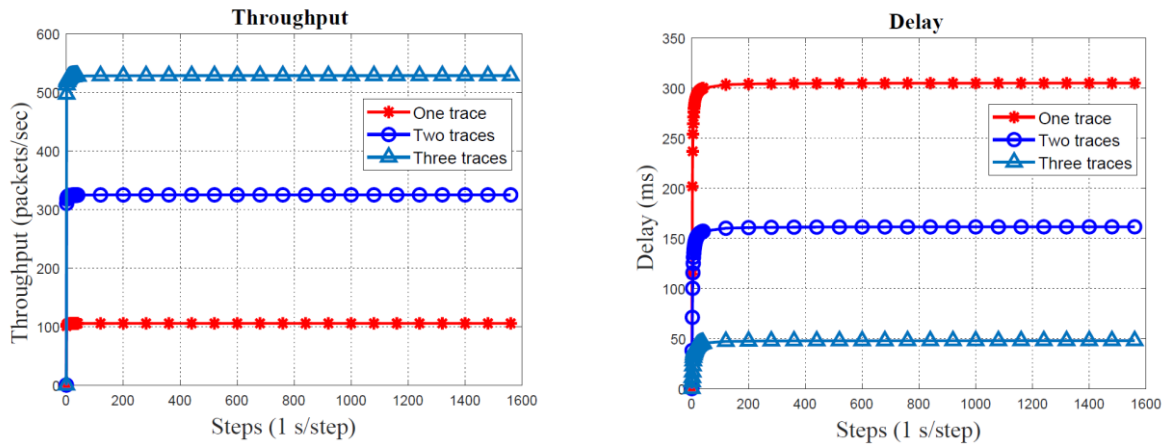


Figure 54. Throughput and Delay Performance for Smooth Swarm and Fixed Packet Generation Rate (=700 packets/s)

Then we observe the QoS trends for big swarm changes. We can see from Figure 55 that our 3D-TR routing scheme can tolerate dynamic swarming behaviors. For the first 200 seconds, the UAVs maintain a relatively constant formation. At the time of 230 sec, 450 sec, 860 sec and 1100 sec, the UAVs experience more dynamic swarming and then resume normal stable swarming after a certain time. The throughput and delay for each case get worse during drastic swarming, but then recover to a normal performance. More paths bring less performance fluctuation compared with less paths. For example, at 230 sec, the largest throughput change (measured in percentage) for the 3-trace case is only 24% while that of the 1-trace case is 50%.

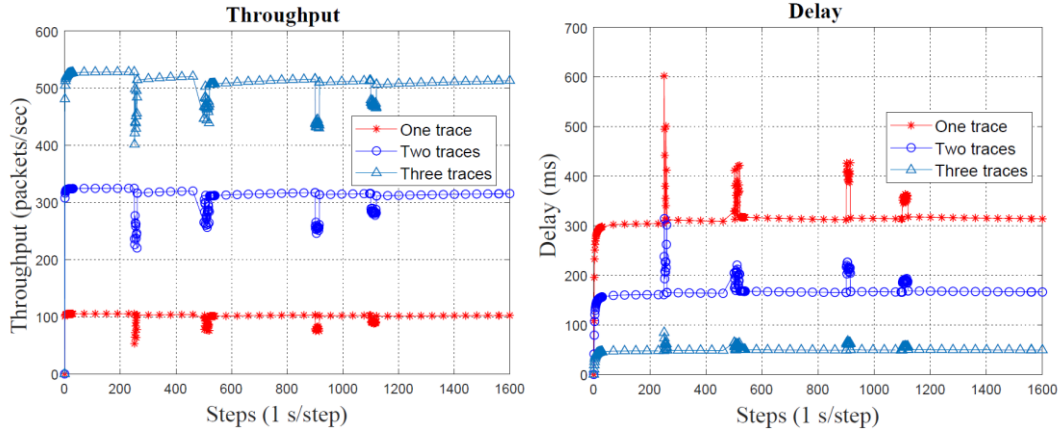


Figure 55. Throughput and Delay Performance for Dynamic Swarm and Fixed Packet Generation Rate (=700 packets/s)

C. Comparison of Algorithms

It is known that the Dijkstra algorithm is good for Single Source Shortest Path searching (SSSP) using a greedy strategy for a weighted dense graph with non-negative edge path costs [109]. The Shortest Path Faster Algorithm (SPFA), the improvement of the Bellman-Ford algorithm, computes SSSP in a weighted directed graph where some of the edge weights may be negative [110]. The Floyd Warshall algorithm is designed for All Pair Shortest Path (APSP) problems for a weighted graph with positive or negative edge weights (but no negative cycles) [111]. Note that the *Heap+Dijkstra* method [112] which was implemented through the priority queue of the C++ Standard Template Library (STL), has been used here.

As can be seen from Figure 56, the time consumption of the *Heap+Dijkstra* algorithm is the lowest for all the six experimental scenarios. In each testing case, the time consumed by Floyd Warshall algorithm is higher than that of the *Heap+Dijkstra* algorithm, but their performances are still comparable with each other. The SPFA method has the largest time cost among the three algorithms for all the six cases. Since the 3D boundary recognition and inter-region route searching processes can be viewed as SSSP, the *Heap+Dijkstra* algorithm is chosen in this project.

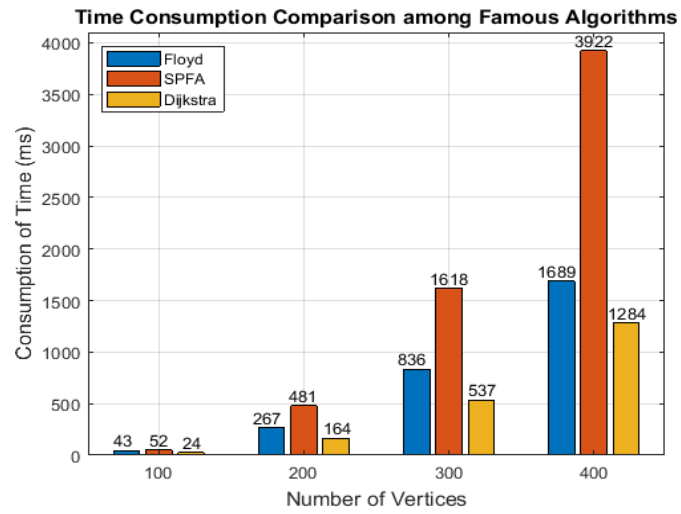


Figure 56. Time Consumption Comparison among Routing Algorithms

6. DRL-BASED ADAPTIVE TRANSPORT LAYER CONTROL SCHEME

6.1 Introduction

The transport layer needs to handle network congestion and provide end-to-end transmission reliability. The most popular transport layer protocol is TCP, which is not optimal for wireless networks that have more dynamic link conditions than wired networks. In UAV networks many factors may cause packet losses, which frequently trigger TCP's sliding window shrinking algorithm and thus decrease the network throughput. For example, the network shape may change (this process is called *swarming*), based on certain mission and coverage requirements. During the swarming process, some links may lose radio signals due to UAV mobility. Also, a neighboring UAV may cause radio interference with the current communication pair if they are too close to each other. The above factors could cause bit errors and those damaged packets will be dropped by the receiver. However, such a packet loss does not indicate true network congestion and should not trigger the shrinking of the TCP window size.

To differentiate packet loss caused by bit errors and by congestion, network coding has been used [113]. In this project, we target a large-scale, dynamic network and use learning models to provide redundancy to the original message using network coding, so that packet loss caused by bit errors will be retrieved by redundant information while packet loss caused by congestion will be retrieved by retransmission. A deep reinforcement learning (DRL) model is used to accurately determine the amount of redundancy according to the wireless network's condition. DRL is superior to general machine learning (ML) schemes or conventional neural networks, due to its capability of handling many input data samples without the need of pre-extracting the data features. DRL is also advantageous over general deep learning (DL) algorithms because it not only learns the system *states* but also produces the most rewarding *actions* by using the Markov decision model.

There are three entities in a DRL network: *agent*, *interpreter*, and *environment*. In our model, some parameters that reflect USN conditions are collected and sent to the recurrent neural network (RNN) as the *environment* parameters. Those parameters are then sent to the *interpreter*, which implements the neural network operations, obtains the results from the output layer, and calculates the reward/penalty values. Those values are finally sent to the *agent* which takes actions such as adjusting the network coding parameters. Thus, fine resolution congestion control can be achieved to maintain a high network throughput. The structure of the proposed DRL based congestion control scheme is shown in Figure 57.

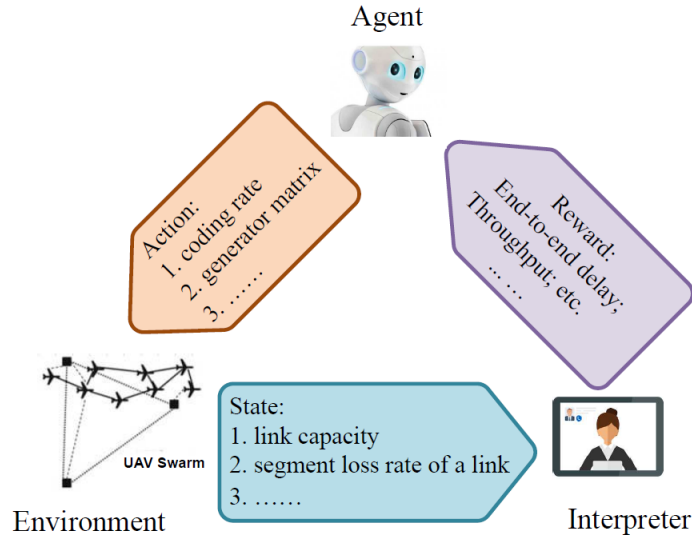


Figure 57. Deep Reinforcement Learning Model for UAV Networks

The contributions of our work are two-fold:

1. *DRL-based wireless network parameter adjustment for transport layer control*: A proper DRL model structure for the practical transport layer control is defined. The state-action pairs are mapped to the input/output layers of the DRL model. Due to the difficulty of obtaining a large amount of training data, we have synthesized the training samples that cover various dynamic topologies. The detailed DRL parameters/settings that reflect practical transport control requirements are defined. This is the first study that applies DRL algorithm for network coding-based transport control. The DRL-based transport layer control achieves the two functions of the transport layer – end-to-end reliability control (i.e., achieving reliable packet delivery over dynamic UAV links) and congestion control (i.e., relieving the queue overflow).
2. *Routing-topology-aware congestion control*: The second contribution of our work is that we have investigated the congestion control strategies and performance for various network routing topologies. We target two routing scenarios, i.e., Single Path (SP), and Multipath, single beam (MPSB). SP and MPSB nodes could use omnidirectional antennas.

6.2 Related Work

Humans can learn from their experience and make optimal decisions. When a new problem occurs, the human brain first searches the experience database to try to find the similar scenarios in the past and then uses the best strategy to solve the problem. Once a problem is solved, the brain can use this new case to update its experience database to reinforce future learning capabilities. The DRL model is inspired by the human brain's working patterns and has achieved significant success in various scientific and engineering applications [114]– [115].

Particularly, DRL has been used for improving the wireless communication performance. For instance, physical layer performance can be significantly improved by using DRL for interference alignment [116], jamming resistance [117], modulation classification [118], error correction codes

[119], etc. In the data link layer, DRL is used for spectrum allocation [120], traffic prediction [121], link evaluation [122], etc. Additionally, DRL helps to enhance the performance in the routing layer, such as delay reduction [123], optimal path search [124][125], etc. Using DRL, different traffic types can be identified in a more accurate manner, so that the malicious traffic on the Internet can be detected [126]. DRL is also used in network intrusion detection and has higher detection accuracy than traditional methods [127][128].

In the recent years, some studies have suggested the use of DRL for transport layer control. For example, Jay et al. [129] proposed a DRL-based congestion control scheme, which mapped the observed network statistics (e.g., throughput and latency) to a certain sending rate. An enhanced TCP protocol called QTCP was proposed in [130], which integrated reinforcement-based Q-learning with TCP. QTCP allows the sender to learn the optimal congestion control policy in an online manner, therefore improving the throughput. A reactive adaptive congestion control method was suggested in [131], which uses online reinforcement learning to maintain an optimal congestion window size, and the weights of the DL network can be initialized either randomly or based on the previous training results.

All the above DRL-based congestion control schemes were based on simple extensions of traditional TCP and focused on the optimization of the congestion window size. When the packet loss occurs, the sender regards it as a sign of congestion and drastically decreases the congestion window so that the congested nodes have enough time to clear up their queues.

To avoid an aggressive window size reduction caused by TCP in UAV swarm networks, we proposed the concept of *adaptive batch coding* (ABC) to adaptively respond to the packet loss events in [113]. In the ABC scheme, the packet losses are categorized as *random loss* and *congestion loss*. The original packet flow is first divided into batches, each of which is encoded by using a network coding model with controllable code redundancy levels. If a random loss occurs, the lost packets can be retrieved by the network decoding algorithm with the use of redundant codes. However, if a congestion loss occurs, the redundancy may not be sufficient to retrieve all the lost segments. In this situation, a negative acknowledgement (NACK) is sent to the sender, which then retransmits the lost segments and decreases the sending rate.

However, due to many dynamic factors (such as varying channel conditions, UAV swarming behaviors, end-to-end route changes, node mobility, etc.), it is challenging to determine exactly what redundancy level should be added to each coding batch so that the above two types of packet losses can be handled efficiently. In [113], the optimal coding rate was determined by using a simulation-based trial-and-error method (instead of using math models). In this method, by observing the communication performance under various network conditions, we may estimate the packet loss rate and then choose a suitable coding rate that is large enough to retrieve the lost packets. However, such a method is not accurate since many factors can contribute to the packet loss, and the loss rate may vary with time.

6.3 Methods, Assumptions, and Procedures

6.3.1 Adaptive Network Coding for Congestion Control

Congestion control is the main task of the transport layer. To achieve this goal, TCP uses an ACK to notify the sender about the possible congestion events. Each segment has a field, called a segment ID. The receiver sends back ACKs to indicate the reception of the segments. If the receiver

does not receive the expected segment within a certain time, duplicate ACKs are sent back to the sender. The three duplicate ACKs are treated as a sign of congestion by the sender. In this situation two actions may be taken: (1) The sender retransmits the lost segments; (2) The sender reduces the sending rate by half.

However, this strategy may not work well in the USNs, where the wireless links are often unstable due to the channel noise, node mobility, interference, etc. We define the segment loss caused by those factors as *random loss*. Since the random loss has nothing to do with the network congestion, the TCP unnecessarily reduces the sending rate.

To solve the above issue, we proposed an ABC scheme [113] for transport layer control. In this scheme, k original segments are bundled as a batch and are encoded as below:

$$C = G \cdot M = \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1k} \\ g_{21} & g_{22} & \cdots & g_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ g_{n1} & g_{n2} & \cdots & g_{nk} \end{bmatrix} \cdot \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ m_k \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \quad (30)$$

where m_i ($i=1, 2, \dots, k$) represents the i^{th} original segment in a batch, c_i ($i=1, 2, \dots, k$) represents the i^{th} encoded segment, and G is the generator matrix with a size of $n \times k$ ($n > k$).

Note that the size of the original segment is the same as that of the encoded segment. After encoding some redundancy is added to a batch to resist the random loss. According to information theory, the tolerable loss threshold ρ for a decoding algorithm is determined by the generator matrix G . After encoding, the source node sends all the encoded segments, $c_1, c_2, \dots, c_n$, to the destination. Some encoded segments may get lost during transmission, due to either network congestion or some random factors. Once the receiver has collected the encoded segments, it starts to decode them and tries to retrieve the original segment, $m_1, m_2, \dots, m_k$. If the number of the lost segments $\leq \rho$, all the lost segments can be fully retrieved through the decoding process. When the number of the lost segments $> \rho$, some lost segments cannot be retrieved. This case is regarded as congestion.

The ABC algorithm can be implemented over various network topologies. In this work, we will enhance the performance of the ABC algorithm for the following three USN routing topologies.

A. Single-Path (SP) Communication

In this case, there is a single path between the source and destination, as shown in Figure 58. Here the nodes may be equipped with omnidirectional or directional antennas. The use of omnidirectional antennas could cause higher interference between neighboring links/paths and thus the SP has lower throughput than the path using directional antennas. The *multihop* data relay increases the probability of random loss, which makes the ABC-based transport control more efficient than TCP. Therefore, in the SP case the source just encodes the segments using the ABC scheme, and the destination retrieves the original segments via a decoding algorithm and then delivers them to the application layer.

Figure 58 shows a SP scenario with the use of ABC. There are five segments (see the green bars) in each batch at the source. After encoding, one redundant segment (the purple bar) is added. Therefore, the source sends out a total of six segments to the next hop. Each hop forwards all the

six segments of the batch. Assume that one segment is lost when it is forwarded from hop 1 to 2. Therefore, the destination node only receives five segments, among which four are the original ones and one is the redundant one. If the number of the lost segments is less than the decoding threshold (which can be determined through the empirical data), the destination is able to retrieve all the five original segments by using the decoding algorithm. However, if the number of the lost segments is more than the decoding threshold, the destination node cannot retrieve all the original segments. In this situation, the destination will send a negative ACK (NACK) to the source and requests it to retransmit the lost segments. As a result, the source also reduces the data sending rate.

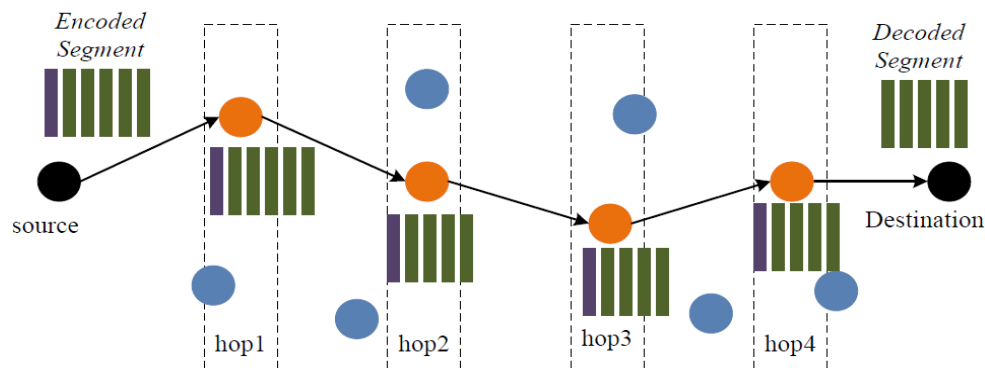


Figure 58. Single-Path Communication

B. Multipath, Single-Beam (MPSB) Communications

In this case (see Figure 59), multiple disjoint paths are established between a given source and destination node pair. To further reduce the interference between these paths, we assume that each relay node is equipped with a directional antenna with a steered beam. If the adjacent disjoint paths have a good separation between them, a simple omnidirectional antenna can be used instead of a directional antenna.

A general multipath transport layer model, called MP-TCP [132], is used, in which all the paths use an independent, TCP-based congestion control scheme. In MP-TCP the source and destination nodes are assumed to be equipped with multiple antennas. As a result, the source and destination nodes can communicate with three nodes simultaneously in Figure 59.

The ABC-based transport control can be used in the MPSB case so that the congestion and reliability issues of all the paths can be addressed together. Figure 59 demonstrates the ABC-based transport control model in MPSB scenario. The source encodes the original segment, adds one redundant segment, and sends a total of six segments over three paths. Assume that one packet is lost when it is relayed from hop 2 to 3 in the top path. Therefore, only five segments reach the destination. If the destination fails to retrieve all the original segments, a NACK is sent to the source. In this situation, the source will pack all the lost segments of the batch into a new retransmission batch and send them through the three paths. As a result, the sending rate is also reduced.

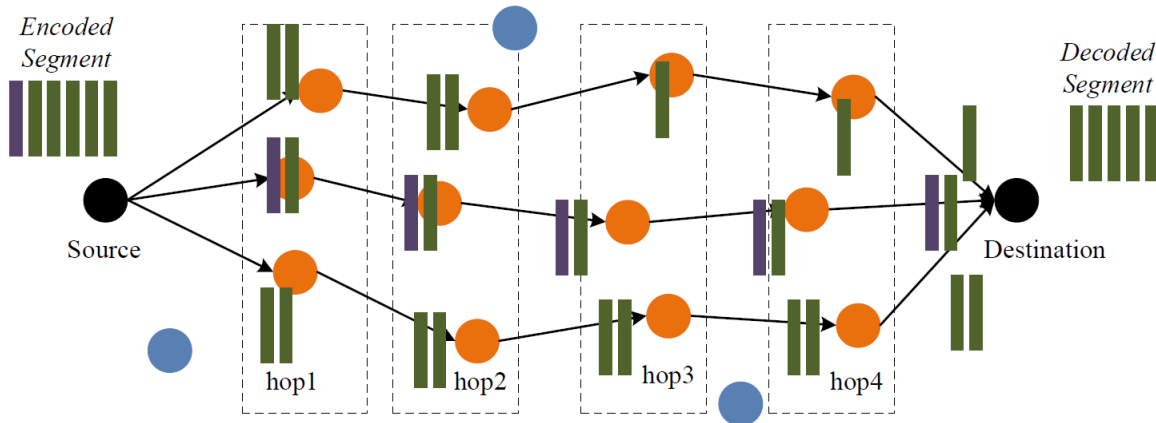


Figure 59. MPSB Communication

6.3.2 DRL Aided Transport Layer Control

The design of the *generator matrix* in the ABC scheme is critical since it determines the code redundancy level. An optimal design of the generator matrix should consider the network conditions, including the topology, packet loss rate, link quality, etc. These parameters are impacted by various factors, such as node mobility, interference, channel fading, etc. Traditional methods use either empirical data or simulation-based tests to determine the appropriate network coding parameters. For example, in our previous work [113], the ABC-based transport layer control was simulated for different network conditions. Then the batch coding parameters that provided the best communication performance were selected for different network conditions.

However, in the simulation-based method, it is impossible to simulate all the communication conditions and select the best coding parameters that are suitable to all the scenarios. Thus, the coding parameters obtained through the simulations may not be the global optima. To overcome this issue, we use the DL network to find the optimal coding scheme for transport layer control.

A. Architecture Design for the DRL Network

In general ML algorithms, the raw data needs to be preprocessed for three purposes: 1) The features of the object are carefully selected so that the ML algorithms can easily recognize different patterns, 2) the features with different scales are normalized to the same range, and 3) the bias of the feature data is minimized before they are given to the input layer of the ML models. Typically, the preprocessing of the raw data is implemented manually, which is an inefficient and unreliable procedure.

Unlike ML, the DL network uses multiple layers composed of neural nodes to achieve accurate pattern recognition. DRL is a Markov extension of a DL model. It uses state-action pairs to train the DL network and integrates a Markov decision model to determine the best action that brings the largest long-term reward (here the network throughput is the reward).

In our DRL model, the weights of the connections between the neural nodes are adjusted through a gradient-based backpropagation process. Meanwhile, a regression model is adopted, the number of neurons in the input layer is equal to the number of the condition parameters of the wireless

network, there is only one neuron in the output layer, which predicts the optimal segment number in a coding block (block size), and there are five hidden layers between the input and output layers, as shown in Figure 60. We assume that (1) there are l_i neurons in the input layer. Each neuron is denoted as N_k^i ($k=1, 2, \dots, l_i$), (2) there are l_o neurons in the output layer, and each neuron is denoted as N_k^o ($k=1, 2, \dots, l_o$), (3) there are m hidden layers in the middle, and the k^{th} neuron of the j^{th} hidden layer is denoted as N_k^j ($j=1, 2, \dots, m, k=1, 2, \dots, l_j$), and (4) the weight of the connection between the j^{th} neuron of the k^{th} hidden layer and the i^{th} neuron of the previous layer is denoted as ω_{ij}^k ($k=0$ means the link connecting a neuron of the output layer with that of the last hidden layer).

To determine the architecture of the DL network, the following parameters and functions have to be properly defined: (1) The optimization target of the DL network, i.e., the definition of the output layer; (2) To achieve the optimization results in the output layer, specific data samples and formats should be provided, i.e., the definition of the input layer; (3) The number of the hidden layers and the activity function used by the hidden layer nodes; and (4) The loss function. These four aspects are discussed below.

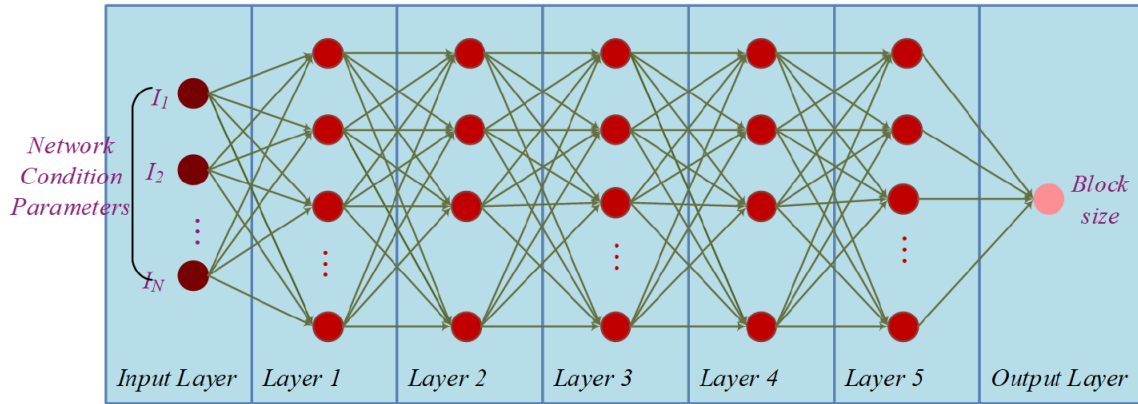


Figure 60. Architecture of Deep Learning Network

B. Feedforward Propagation

We denote the transmission function of the i th hidden layer as F_i ($i=1, 2, \dots, m$), and the transmission function of the output layer as F_o . Then, the feedforward propagation of the entire DL network can be represented as

$$Y = F_o (F_n \cdots (F_2 (F_1 (X)))) \quad (31)$$

where X is the input of the DL network and Y is the output.

For the j^{th} node of layer i , N_j^i , its output z_j^i is calculated as:

$$z_j^i = \sigma \left((W^{i-1})^T Z^{i-1} + b_j^i \right) \quad (32)$$

where W^{i-1} is the weight vector from all the neurons of layer $i-1$ to neuron N_j^i , Z^{i-1} is the output vector of layer $i-1$, b_j^i is the bias of node N_j^i , and σ is an activation function. The activation function

can be determined either by training or experience. Popular choices include rectified linear units (ReLU), logistic Sigmoid, and hyperbolic tangent. In our model, ReLU is selected as the activation function, i.e., $\sigma(d) = \max(0, d)$.

The derivative of the rectified linear function is always 1 when the unit is active (i.e., a positive input). Since the output of the ReLU with the negative input is always zero, a drawback of the ReLU is that it cannot use gradient-based algorithm to handle the samples that cause negative inputs to the ReLU. To solve this problem, some enhanced ReLU models may be used such as absolute value rectification, leaking ReLU, parametric ReLU, etc.

We use f to represent the operations of a single node. In a multilayer DL network, every node yields an output based on multiple inputs from its previous layer. Therefore,

$$f(Z^{i-1}) = \sigma((W^{i-1})^T Z^{i-1} + b^i) \quad (33)$$

Figure 61 exemplifies a feedforward propagation of Figure 60, which is from the first input element x_1 to the first output element y_1 , going through the top route shown in Figure 60. In Figure 61, z_1^i represents the output of the first node of the i^{th} hidden layer, N_1^i ; f_i represents the operations implemented by node N_1^i ($i=1, 2, \dots, m$), and f_o represents the operation implemented by the output layer neurons. In Figure 61, only one input for each node is marked out. However, each node may take multiple outputs produced by the previous layer as its input.

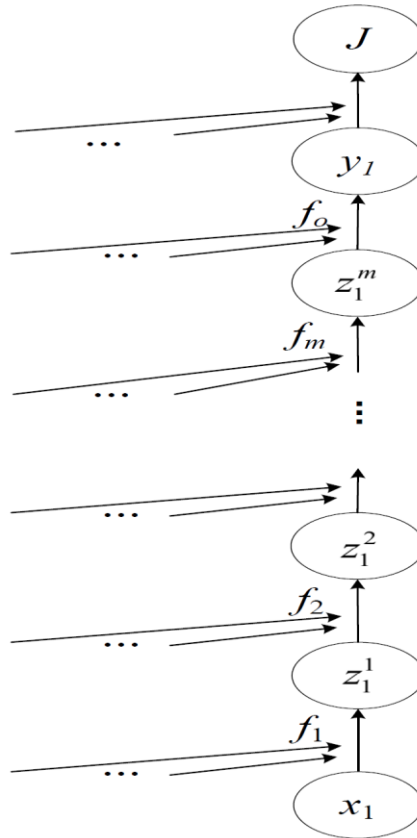


Figure 61. Feedforward Propagation Using Symbol-to-Symbol Approach

Since our goal is to use the DL network to optimize the batch coding scheme for congestion control, the inputs of the DL network should be the network state parameters, and the outputs are the optimized ABC coding parameters. Therefore, first the network environment parameters are collected and sent to the DL network as inputs. Some features are preliminarily abstracted by the input layer. These features serve as the inputs of the next layer. Such an operation is iteratively implemented in layer by layer, and each layer extracts more intrinsic features towards the final output, i.e., the ABC coding parameters. Finally, the output layer provides the optimal batch coding parameters.

C. Output Layer

The output of the DL network is the optimal network coding parameters. In the ABC-based congestion control, one of the most important parameters is the coding rate r where $r = k/n$, k is the number of the original segments in a batch, and n is the number of the encoded segments in a batch.

In USN, the channel noise causes random segment losses, while network congestion may cause many segment losses. Therefore, in our proposed ABC-based transport control, the coding rate should be determined in such a way that only the random loss is healed by redundant codes. When the congestion occurs, the coding rate should not be set to a large number since it will make the congestion worse and thus the amount of redundancy codes cannot retrieve all the lost segments. In this situation, the loss segments can be retrieved via retransmissions and the sending rate is reduced, so that the congested nodes have enough time to clear up their queues.

To reduce the protocol overhead of the transport layer, we proposed a systematic code for the batch coding with the following feature [113]:

$$C = G \cdot M = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 1 & 1 & \cdots & 1 \end{bmatrix}_{(k+1) \times k} \cdot \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ \vdots \\ m_k \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{(k+1)} \end{bmatrix} \quad (34)$$

Here $c_i = m_i$ for $i = 1, 2, \dots, k$, and the redundant segment is:

$$c_{k+1} = m_1 \oplus m_2 \oplus \cdots \oplus m_k \quad (35)$$

where $\oplus$ is the bitwise exclusive OR operation. In this case, the coding rate is $r = k/(k+1)$. Therefore, the coding rate can be controlled by adjusting the block size k . Hence, we will use the block size as the output of the DL network. By optimizing the block size, an appropriate number of redundant codes can be added to each batch.

The input of the output layer is the output of the last hidden layer. The structure of each node in the output layer is like the one shown in Figure 61, except the choice of the activation function. If the regression model is chosen, the linear unit can be used as the activation function for the output layer, i.e.,

$$y = W_n^T Z_n + b_o \quad (36)$$

where W_n is the weights from the last hidden layer to the output layer, Z_n is the output of the last hidden layer, and b_o is the bias of the output layer. In this situation, the output of the DL network, y , is the scalar that predicts the value of the block size for the ABC scheme.

Alternatively, a classification model can also be adopted for the output layer computation. In this case, the Softmax function may be a good option for the activation function of the output layer, which predicts the Multinoulli output distributions [44]. Assume that there are total t classes. The Softmax function is composed of two parts. First, a linear layer is used to obtain the unnormalized log probabilities, i.e.,

$$Y' = W_n^T Z_n + b_o, \text{ where } Y' = [y'_1, y'_2, \dots, y'_t]^T$$

Then, the Softmax output is:

$$Y = [y_1, y_2, \dots, y_t]^T$$

It is achieved by the normalization upon Y' , that is, [44]

$$y_i = \text{softmax}(y'_i) = \frac{\exp(y'_i)}{\sum_{j=1}^t \exp(y'_j)} \quad (37)$$

Apparently, the sum of all the Softmax units is equal to 1. Assume that in the proposed ABC scheme, the optimal block size varies from β_1 to β_t . To predict which block size is the optimal value in the DL network, we use a total of t neural nodes in the output layer, each of which represents a specific block size. The DL network can predict the probability for each block size at the output layer, and the block size with the highest probability is the optimal solution for the current network conditions, which have been represented as the input values to the DL network.

D. Input Layer

The input of the DL networks should use the raw data (i.e., collected network parameters) that can contribute to the computation of the optimal network coding parameters. Since the optimization target is the block size k , the input layer data can be the following two categories:

1) Link capacity: From the source to destination the links of the path may have different transmission rates due to node mobility, channel noise, etc. If a single path (SP) is built, the link with the minimum rate is the bottleneck of the entire path. If a multipath topology is used, the total throughput of the pipe is more dynamic than the SP case since different links/paths have time-varying traffic loads. Therefore, it is beneficial to use the DL network to analyze the capacity of all the links in a multipath pipe to find the optimal block size.

2) Packet loss rate (PLR): Due to the instability of the channel quality, the packets may get lost in each link. There may be different PLRs for each link. Since the proposed batch coding scheme aims to resist random loss caused by the channel noise and interference, the PLR of each link directly impacts the selection of the optimal coding rate.

Even though the link capacity and PLR may change with time, it may not be necessary to change coding parameters too frequently since this can increase the computational complexity and congestion control overhead. From the detection of congestion event to the complete congestion relief, it may take a few hundred milliseconds (ms) to a few seconds in a large UAV network (> 50 nodes). Assume that the proposed DL-aided coding scheme changes the coding rate every T

second ($0.2 < T < 2$). The DL center (which can be the sender or a dedicated control node) collects and analyzes the network conditions within each T -second time window and then seeks an optimized coding rate for the batch coding control in the sender.

Assume that the number of stages (i.e., hops or columns) from the source to the destination is D , and the total number of links for all the paths is F_i for stage i . At time t ($t \leq T$), for the j^{th} link of stage i , denoted as l_{ij} , ($i = 1, 2, \dots, D$; $j = 1, 2, \dots, F_i$), the link capacity is denoted as c_{ij}^t and the PLR is denoted as e_{ij}^t . Then, all the link capacities and loss rates during a time window of T seconds can be represented as an environment vector (which will be sent to the input layer of the DL model):

$$E = [c_{11}^1, e_{11}^1, c_{12}^1, e_{12}^1, \dots, c_{DF_D}^1, e_{DF_D}^1, c_{11}^2, e_{11}^2, c_{12}^2, e_{12}^2, \dots, c_{DF_D}^2, e_{DF_D}^2, \dots, c_{11}^T, e_{11}^T, c_{12}^T, e_{12}^T, \dots, c_{DF_D}^T, e_{DF_D}^T] \quad (38)$$

The communication scenarios are simulated for different block sizes, and the block size yielding the highest throughput, denoted as B_{opti}^t , is chosen as the optimal block size. Through this approach, we can obtain different training samples for the DL network. For each sample, the vector represented in the above equation is the feature (which will serve as the input of the DL network), and B_{opti}^t serves as the label (which is the output of the DL network). This procedure iterates for some rounds until enough samples are collected to train the DL network.

In our scheme the DL input is the network environment vector E . Therefore,

$$\begin{aligned} d_{0,1} &= c_{11}^1, d_{0,2} = e_{11}^1, d_{0,3} = c_{12}^1, \\ d_{0,4} &= e_{12}^1, \dots, d_{0,l-1} = e_{DF_D}^T, d_{0,l} = e_{DF_D}^T \end{aligned} \quad (39)$$

Using the input data, the weights, and bias of the DL network, the optimal ABC coding parameter can be predicted by the output layer.

E. Deployment of DRL Computation Node

To implement the USN performance optimization by using the DRL model, certain computations are needed, including the collection of training data, the training of a neural network, the determination of Markov actions by using the trained DL network, etc.

The source node may be chosen to run the DRL algorithm for two reasons: First, the source can collect all the network performance parameters through ACKs. Secondly, it is the source that implements the batch encoding algorithm. Therefore, it is a natural choice for the source to solve the coding parameters. Because the time-consuming DRL training phase can be executed offline, the sender just needs to run the testing phase, which can be performed in real time.

Another DRL execution location could be another wireless node in the network, as shown in Figure 62. We call such a node as DL-Aided Control Center (DLACC). DLACC may communicate with other nodes through a common control channel (CCC) if it does not want to interrupt the normal data transmissions. It collects training data samples, trains the DRL network, monitors the exchange of control messages via the CCC, optimizes the coding parameters, and sends the optimal coding parameters to the source during each communication session with specific network conditions (such as routing topologies, noise/fading models, link rates, etc.).

After the training phase is finished, the DLACC stores all the state-action pairs, i.e., network conditions vs. ABC coding parameters. For a new communication session, the DLACC collects the communication parameters of the session. Assume that the environment vector collected by the DLACC is as follows:

$$E_p = [c_{11,p}^1, e_{11,p}^1, c_{12,p}^1, e_{12,p}^1, \dots, c_{DFD,p}^1, e_{DFD,p}^1, c_{11,p}^2, e_{11,p}^2, c_{12,p}^2, e_{12,p}^2, \dots, c_{DFD,p}^2, e_{DFD,p}^2, \dots, c_{11,p}^T, e_{11,p}^T, c_{12,p}^T, e_{12,p}^T, \dots, c_{DFD,p}^T, e_{DFD,p}^T] \quad (40)$$

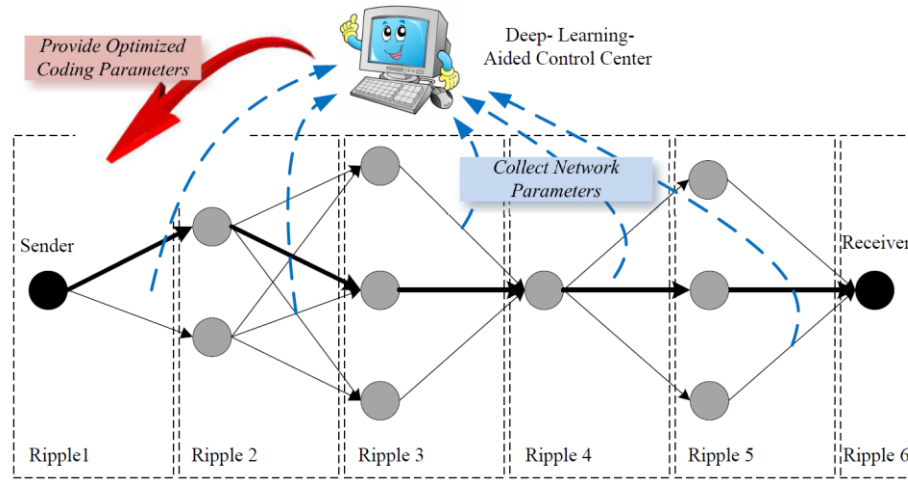


Figure 62. DL Center Deployment

Then, the DLACC searches its database and calculates the mean squared error (MSE) between E_p and the stored data:

$$\begin{aligned} MSE &= \frac{1}{n} \sum_{j=1}^n (E_{p,j} - E_{i,j})^2 \\ &= \frac{1}{n} [(c_{11,p}^1 - c_{11,i}^1)^2 + (e_{11,p}^1 - e_{11,i}^1)^2 + \dots + (e_{DFD,p}^T - e_{DFD,i}^T)^2] \end{aligned} \quad (41)$$

where E_i is the i^{th} sample of the stored state-action pairs ($i=1,2, \dots, I$), where I is the number of pairs in the database), and n is the size of the environment vector. The sample that yields the minimum MSE will be selected, and the corresponding coding scheme will be used for the ABC-based congestion control. This process is implemented in each time window. The length of the time window T may be chosen based on the variation frequency of the wireless network states, such as mobility mode changes, interference conditions, etc.

6.4 Results and Discussion

We have used Matlab to build a simulator. In the simulator, the network parameters (i.e., the link capacity and the PLR for each link) are generated for each link for T seconds (a time window).

Note that since we are optimizing the performances on the transport layer, the simulator does not handle physics layer parameters directly.

We have simulated the communication process using the ABC-based transport control with a specific block size, varying from B_{min} to B_{max} . The block size yielding the highest throughput (or the lowest end-to-end delay), denoted as B_{opti} , is chosen as the optimal block size. Thus, a training sample $[E_p, B_{opti}]$ can be obtained, where E_p is the input of the DRL network. This process is repeated until enough training data samples are collected. Once the DRL network is trained, it can be used to find the optimal block size. The data flow of the training process and the optimization process is shown in Figure 63.

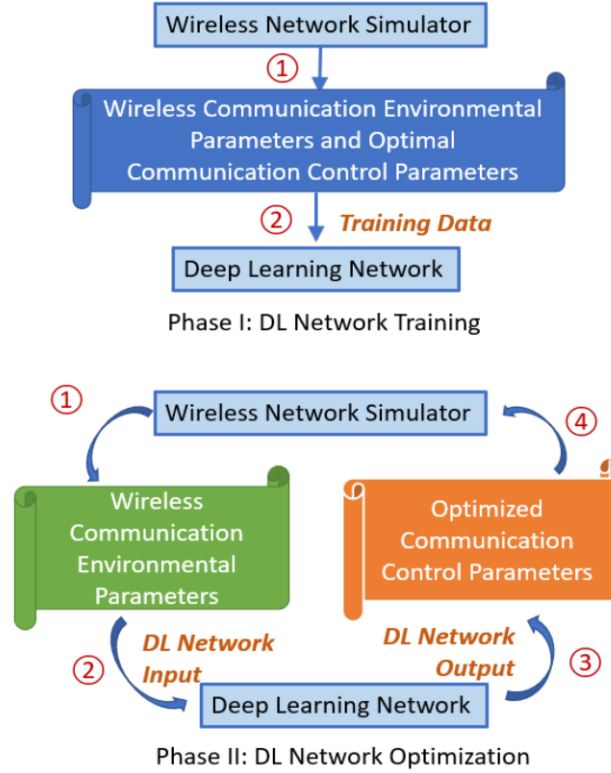


Figure 63. Data Flow Frameworks

In the following simulations, the communication scenarios with single-path (SP) or multipath (MP) routing topologies are investigated. Assume that the stage (i.e., hop) number from the source to the destination is D . For each link (denoted as l_{ij} , at time t both the link capacity c_{ij}^t and PLR e_{ij}^t are random numbers with the following probability distributions, respectively,

$$c_{ij}^t = \frac{1}{2}c_m\mathcal{N}(u, \sigma^2)$$

$$e_{ij}^t = e_m(a\mathcal{R} + 1) \quad (48)$$

where c_{ij}^t follows the normal distribution; R is a random number varying from 0 to 1, and a is a constant. Using these functions, the link capacity is a random number with a normal distribution, and the PLR of the link is also a random number that is α times larger than the constant number e_m . The networking parameters used for all the simulations are listed in Table 6.

Table 6. Simulation Parameters for Single-Path Communications

Notation	Parameters	Values
u	Mean of the normal distribution of link capacity (eq. (22))	1
s	Standard deviation of the normal distribution for the link capacity (eq. (22))	0.1
e_m	Parameter in the formula of packet loss rate (eq. (23))	10^{-4}
a	Parameter in the formula of packet loss rate (eq. (23))	19
T_s	Simulation time for one session	40 seconds
L	Segment size	2000 bytes
B_{min}	Lower bound of the block size of the adaptive batch coding	6
B_{max}	Upper bound of the block size of the adaptive batch coding	30

We have implemented the ABC scheme for congestion control. For each channel condition set E_p , the following three methods are used to determine the coding parameters:

- **Method 1:** Traditional ABC scheme. In our previous work [113], we used simulations to find a good block size to encode the original batches. The block size is fixed for each second. For example, with the network parameters shown as Table 6, we found that when the block size was 10, the wireless network yielded the best throughput performance.
- **Method 2:** Proposed DRL model. Using the trained DRL model proposed in this report, we can find the optimal block size. Since the DRL network requires certain computation overhead (especially in the training phase), it is difficult to generate the optimal block size in a very small-time window. However, if the same block size is used for a long time without any updates under the time-varying wireless network conditions, the network cannot efficiently relieve the congestion. Therefore, the frequency of block size updates (i.e., the control time window) needs to be carefully chosen. In the following simulations, the control window is 40 s. Within each time window, we collect the network parameters, send them to the DRL model, and compute the optimal block size. Using the generated block size, the original batches are encoded through the ABC scheme.
- **Method 3:** Ground-truth values. For each given channel condition, we simulate the UAV communication with block sizes varying from 7 to 30. The block size yielding the highest throughput is chosen as the optimal block size, which is used to encode the original batches. The size found in this optimal method is the ground-truth value for the best block sizes.

In the following, we compare the network performance with the use of the above three methods. First, we investigate the communication performances of the SP scenario. Table 7 shows the block size used by each method in 10 simulation sessions. We observe that the traditional ABC scheme does not consider the channel variations and uses the same block size. The proposed DRL-based

scheme can find the optimal block sizes in each control window. These block sizes are very close to the ground-truth block sizes. This indicates the accuracy of our proposed DRL-based scheme.

Table 7. Block Sizes of Transport Layer Control for Single Path Communication

Session ID	1	2	3	4	5	6	7	8	9	10
Traditional	10	10	10	10	10	10	10	10	10	10
DRL-Based	19	17	17	11	15	17	13	11	12	18
Ground-Truth	18	17	17	12	14	20	13	13	12	16

Figure 64 shows the performance comparison in terms of the throughput and end-to-end delay for the SP topology. Here ten communication sessions are simulated, and the block sizes shown in Table 7 are used for each method. The simulation results show that the throughput yielded by the DRL-based method is higher than that of the traditional ABC method and is close to the result from the ground-truth block sizes. The end-to-end delay yielded by the DRL-based method is generally lower than that of the traditional method. However, for a few cases, the end-to-end delay of the DRL-based method is higher than that of the traditional method. This is because the throughput alone was used for searching the optimal block sizes during the training. However, the end-to-end delay might not be optimal when the throughput is maximized.

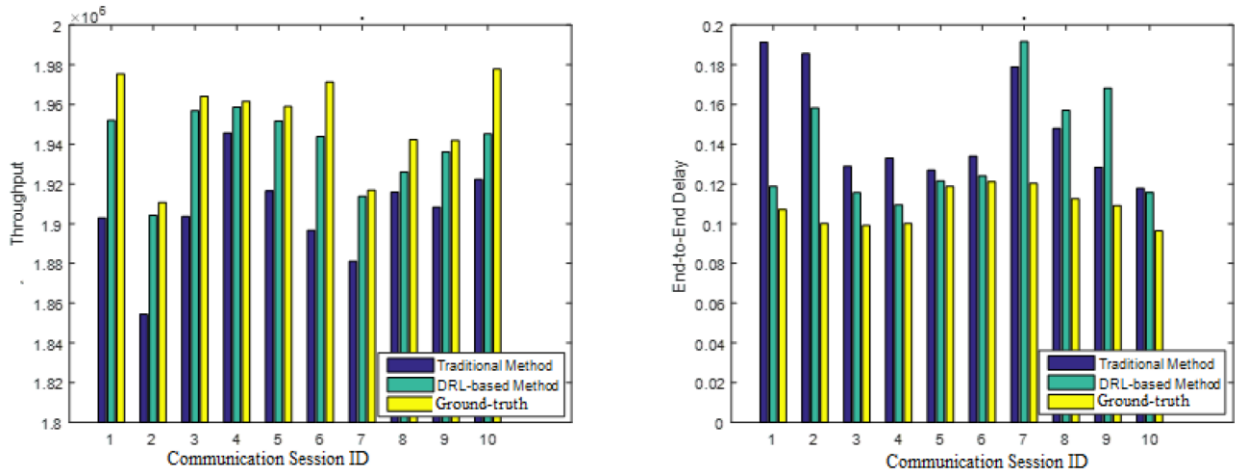


Figure 64. Performance Comparison of the ABC-based Congestion Control Scheme for Single-Path (SP) Communication

Next, we investigate the MPSB routing scheme. Assume that the total number of links at stage i is F_i . Let us denote the j^{th} link of stage i as l_{ij} . In multipath communications, the main path is assumed to have better communication performances. Therefore, if a link belongs to the main path, the link capacity is set to a constant value (2 Mbit/s) and the PLR of that link is also set to a constant number (10^{-4}). If the link l_{ij} is a side-path link, both the link capacity and PLR are random numbers with the probability distributions. Therefore, the capacity of the side path is a random number with a normal distribution and the mean of 1 Mbits/s, and the PLR of the side path is a random number, which in our experiments is set to be 19 times greater than the PLR of the main-path links. To fully

leverage the link capacity, we have conducted many simulation tests for MPSB scenarios and found that the average network throughput could get closer to the maximum aggregated link capacity if the data sending rate in the source is set to 3.4 Mbits/s. When the source rate is higher than this number, significant packet loss events could happen due to frequent congestion events. Other simulation parameters, such as segment size and simulation time, for the MPSB communications are given in Table 6.

Table 8 shows the block sizes used by the traditional ABC, DRL-based ABC, and optimal ABC (ground-truth result) schemes for the MPSB scenario. Similarly, to the SP case, the traditional ABC does not change the block size under different channel conditions, while the DRL-based ABC computes the optimal block size, which is close to the ground-truth value.

Table 8. Block Size of Transport Layer Control for MPSB Communication

Session ID	1	2	3	4	5	6	7	8	9	10
Traditional	10	10	10	10	10	10	10	10	10	10
DRL-Based	18	13	13	12	15	13	22	13	17	15
Ground-Truth	17	13	13	12	15	13	23	13	15	16

Figure 65 shows performance of the traditional ABC, DRL-based ABC, and optimal ABC (ground-truth) schemes, in terms of throughput and end-to-end delay. Here, the DRL-based ABC scheme yields higher throughput and lower end-to-end delays than the traditional ABC scheme. Compared to Figure 64, the DRL-based ABC shows more advantages in the MPSB case. This is because the topology and channel conditions of MPSB communication are more dynamic compared to SP communication. Using DRL to find the optimal block size is thus more beneficial.

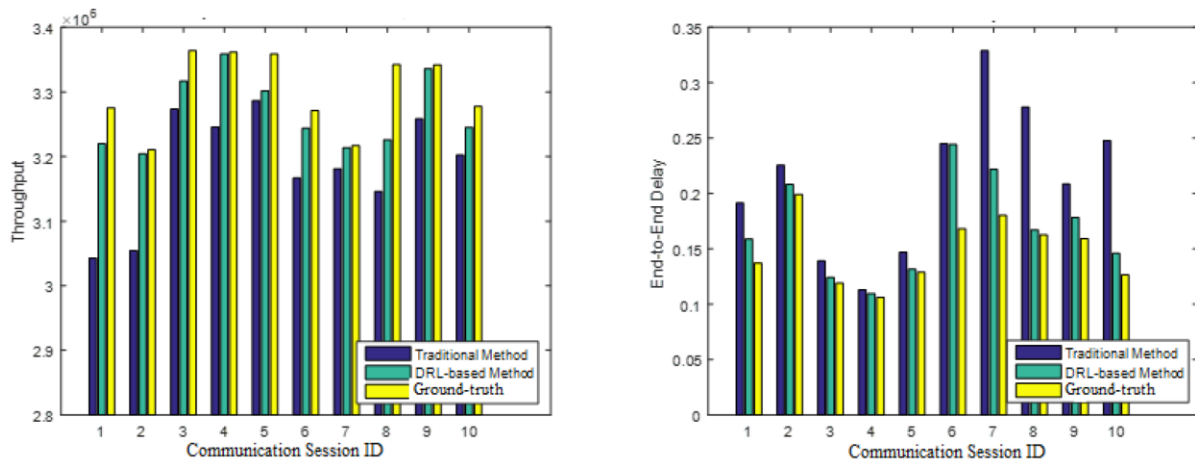


Figure 65. Performance Comparison of ABC-based Congestion Control Scheme for MPSB Communication

7. EMANE-BASED SIMULATION OF MULTIPATH BRIDGE ROUTING

7.1 Introduction

The UAV networks have a highly dynamic topology and can form multiple swarms (see Figure 66). We assume that some backup UAVs are available that can be used for route establishment between the swarms. Our goal is to build a *bridge-like routing topology* for high throughput inter-swarm communication.

Besides the above route establishment issue in the network layer, the end-to-end traffic flow control in the transport layer is also critical to address the traffic flow smoothness and reliability issues. The TCP and UDP based transport protocol, used for the end-to-end flow control in a single path, cannot handle the parallel paths very well in the bridge topology shown in Figure 66. To increase the data transmission rate and fully utilize the network resources, the MP-UDP is desirable. Unlike the MP-TCP protocol [132][133], the MP-UDP is more suitable for the transmission of a large amount of multimedia traffic (such as video/image data) since it does not use frequent packet retransmissions and complex sliding window control algorithm.

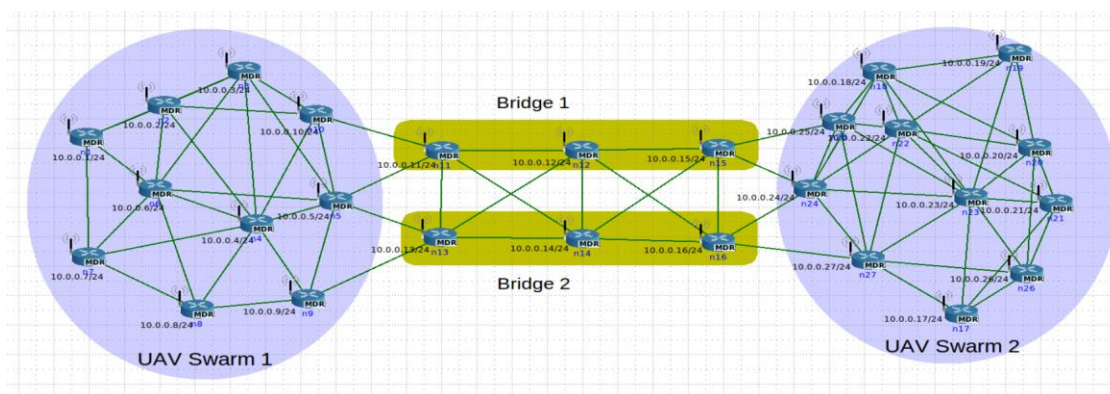


Figure 66. An Example of an Inter-Swarm Communication Bridge

To avoid the protocol interface incompatibility issues, we use the basic UDP protocol functions and add a sublayer between the transport and application layers. This sublayer performs the MP-UDP functions above the multi-bridge routing protocol of the UAV swarm network. The basic procedure of building a MP-UDP connection is as follows: First, the sender sends out the path seeking (PS) messages. This is a flooding process with a loop avoidance strategy. Next, the destination node collects all path information, and calculates the best path by using the Edmonds-Karp algorithm [134]. After that, the destination sends back a Path Seeking Acknowledgement (PS ACK) message through conventional UDP packets. Finally, the sender separates its data into multiple traffic flows and sends them to the client through different paths. The following sections describe the details of MP-UDP implementation.

The CORE with EMANE is used to verify the MP-UDP performance. EMANE operates in the physical and data link layers only, and the higher layers are added by building extra modules in CORE. The CORE emulator is suitable for the UAV network due to its flexible modules on node mobility and the support for large-scale wireless networks [135][136].

7.2 Related Work

The highly dynamic nature of UAV network topologies makes the MANET communication protocol development very challenging. Most of the existing protocols address the transport layer congestion control and network layer routing protocol separately. From the perspective of network layer, several major routing protocols with different versions have been introduced and analyzed. The proactive routing protocol, Optimized Link State Routing Protocol (OLSR), and reactive protocol, Ad-hoc On-demand Distance Vector (AODV), are two representative schemes. Most studies such as Ouacha et al. [137] and Romanik et al. [138] focus on the enhancement of previous protocols. The performance metrics to estimate the path cost and make routing decision are the core topics in these enhanced routing schemes. From the perspective of the transport layer, congestion control is important. For example, Wang et al. [139] discussed the modifications of TCP congestion control by modifying its slow start phase. Other schemes enhance the congestion window control [140] to eliminate unexpected slowdown of TCP sending rate.

However, focusing on an individual communication layer cannot achieve efficient ad-hoc communication. For example, a link/path quality-based routing protocol (such as OLSR) may provide stable traffic path inside a single swarm. However, it will create significant protocol overhead as the network size increases (e.g., when multiple swarms exist in the network). Trushna et al. [141] compared multiple versions of OLSR and found an exponential increase in the routing overhead when the swarm size increases. On the other hand, improving the transport layer alone may not be able to fully explore system resources. For example, general ad-hoc network protocol only returns a single traffic path to the upper transport layer. A sudden path failure may cause a traffic flow to be cutoff and finding a new path may introduce delay.

The UAV ad-hoc network communication protocol design should follow two rules: (i) Different communication schemes should be applied for different circumstances. This means that intra- and inter-swarm communications should use different schemes; (ii) Cross-layer design is needed to reach a system-wide optimum result. Working on a single network layer can only provide a sub-optimum system solution. In this project, we focus on a light-weight inter-swarm cross-layer communication protocol design, called MP-UDP.

Cui et al. [142] provided a brief survey, which discussed ‘many-to-many’ swarm communication. In their scheme, two representatives/managers were pre-selected to build a communication bridge in between two swarms. These two representatives/managers have strong long-range antennas, which make them stay within the signal range of each other. However, no communication protocol is provided in [142]. In our proposed scheme, there is no need for the representative nodes to have strong antennas. This is critical to MANET, since there is no guarantee that the representative nodes can stay in the network all the time.

In terms of cross-layer MANET protocol design, Yin et al. discussed a modification for both routing protocol and transport layer congestion control [143]. In their schemes, OLSR was modified to provide multipath information for the transport layer and network coding was used to eliminate the TCP congestion window control issue. However, a proactive protocol quickly increases transmission overhead as the network size increases. This may lead to poor performance for large swarm size. In contrast, our design is a light-weight reactive protocol.

7.3 Methods, Assumptions, and Procedures

7.3.1 Overview of Network Emulators: CORE and EMANE

CORE and EMANE form an emulation bundle and have been used for studying the performance of wireless networks. CORE focuses on the routing and higher layers, whereas EMANE provides real-time modelling for the lower layers (physical and data link layers). Several schemes in the literature have used this emulation bundle for studying the Wi-Fi communication security issues [144], mobile micro clouds [145], and serverless group chat [146].

A. Common Open Research Emulator (CORE)

CORE runs in real-time and provides a virtual environment for each node. In other words, each node operates like a real machine with Linux running in the background. This provides the ability for each simulated machine to run with real applications and protocols [135].

Particularly the CORE framework uses Linux namespaces to create node containers and virtual interfaces are used to link these nodes together. Each single machine or namespace has its own process environment and private network stack.

With a private network stack, the network protocols and applications can be configured in each node. To setup the protocols for each virtual machine, the concept of services is used in CORE to specify the processes or scripts. Multiple software plugins can provide services. For example, the Quagga routing suite can support routing protocols, such as OSPFv2. EMANE is also a plugin in this case and can provide multiple transport services such as IEEE802.11b.

B. Extendable Mobile Ad-hoc Network Emulator (EMANE)

EMANE is a real-time modeling and emulation tool for mobile networks. EMANE as an individual emulator has been integrated with CORE [136]. When EMANE works with CORE, it provides the physical and link layer functions for real-time emulation. EMANE uses the graphs with the probability of reception vs. signal to interference and noise ratio (SINR) to demonstrate wireless channel properties. Its physical models help to calculate packet loss rate at each wireless interface/channel.

EMANE provides four categories of radio models - the RF Pipe, IEEE802.11(b, a/g, and b/g), TDMA, and LTE. The IEEE802.11b (DSS only) model is utilized in this project. Further discussion about the wireless channel model is provided in later sections.

7.3.2 Mobility Control and Progressive Bridge Establishment

The UAV network protocols need to adapt to the time-varying swarm topologies without sacrificing QoS performance. The communication protocols and UAV movement control (to form certain swarm topologies) can be mutually helpful. On one hand, the communication protocols help with the swarming process by telling each node the best movement trajectory to form a desired network topology. On the other hand, a proper swarm topology can also help improve the communication performance. For example, a mesh-like network topology can have a lower end-to-end routing delay than a simple chain-like multihop structure. This section focuses on how a proper UAV movement control model can help to establish bridge route(s) and achieve satisfactory

QoS performance.

A. Tolerance Molecular Force Model

A basic movement control algorithm is needed to help UAVs avoid obstacles and body collisions. Some solutions have been proposed to control the formation and swarming of UAVs. Most schemes are based on the Gilles Labonte model [147], which used the model of molecular forces to control the node-to-node distance. The UAVs attract each other when they are far away and push their neighbors away when they get too close. Such a model can generate complex swarm shapes that are like the chemical compounds.

In this project, we also use the molecular force model to control the UAV's movements and enhance it by adding certain system redundancy. We call it Tolerance Molecular Force Model. The following equation shows the mathematical model that we have used for the control of attractive/repulsive 'forces' among UAVs:

$$F = \begin{cases} C * \left[\left(\frac{a}{r} \right)^\alpha - \left(\frac{b}{r} \right)^\beta \right] & ; r < r_1, \text{ or } r > r_2 \\ 0 & ; r_1 \leq r \leq r_2 \end{cases} \quad (42)$$

$$\text{where } \begin{cases} r_1 = \left(\frac{a^\alpha}{b^\beta} \right)^{\frac{1}{\alpha-\beta}} - \frac{\text{tolerance_distance}}{2} \\ r_2 = \left(\frac{a^\alpha}{b^\beta} \right)^{\frac{1}{\alpha-\beta}} + \frac{\text{tolerance_distance}}{2} \end{cases}$$

The above equation has only one input variable r , which is the distance between two UAVs. The function will generate a curve shown in Figure 67. Parameters α, β, a, b, C and $\text{tolerance_distance}$ are defined based on specific application requirements. These parameters control the shape of the curve. Increasing a or α will increase the slope for the positive value part of the curve. In contrast, increasing b or β value will increase the curve's tail size, and further create lower minimum value. The C value is simply a constant multiplier. The variable 'tolerance_distance' decides values for r_1 and r_2 , where r_1 and r_2 give the lower and upper bound for the curve of Figure 67, respectively. The expression for both r values contain a term $\left(\frac{a^\alpha}{b^\beta} \right)^{\frac{1}{\alpha-\beta}}$, which is the r 's value for standard molecular force model with a value equal to zero. The final model will generate a twin cycle forces field shown in Figure 68. The center node will generate a force field with a twin cycle shape. Any node within less than r_1 will experience a repulsive force $f_2 = F(r)$, and the nodes located at a distance larger than r_2 will experience an attractive force $f_1 = F(r)$.

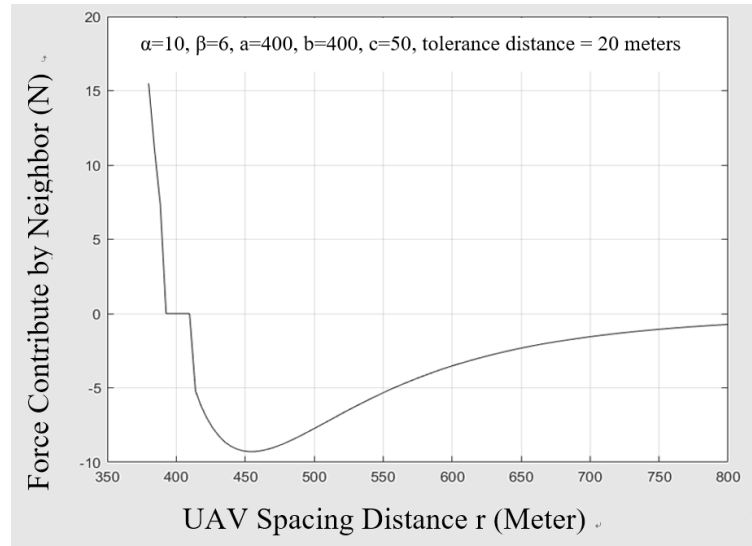


Figure 67. Tolerance Molecular Force Model for UAV Mobility Control

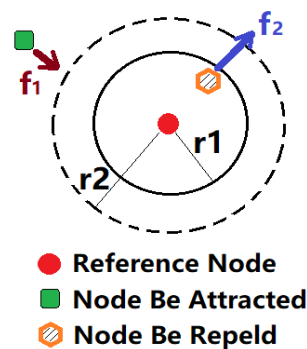


Figure 68. Interactions among Nodes based on Tolerance Molecular Force Model

Based on this model, the collision can be avoided during UAV movement. Each single UAV can also stay within the network during movement. Our mobility control algorithm makes the following assumptions: (i) The destination UAV (target) only generates attractive forces for UAVs, and (ii) obstacles only generate repulsive forces for UAVs.

Figure 69 shows a typical application of our model. The swarm departs from the lower left conner, bounds together, bypasses obstacles, and finally reaches the target (destination).

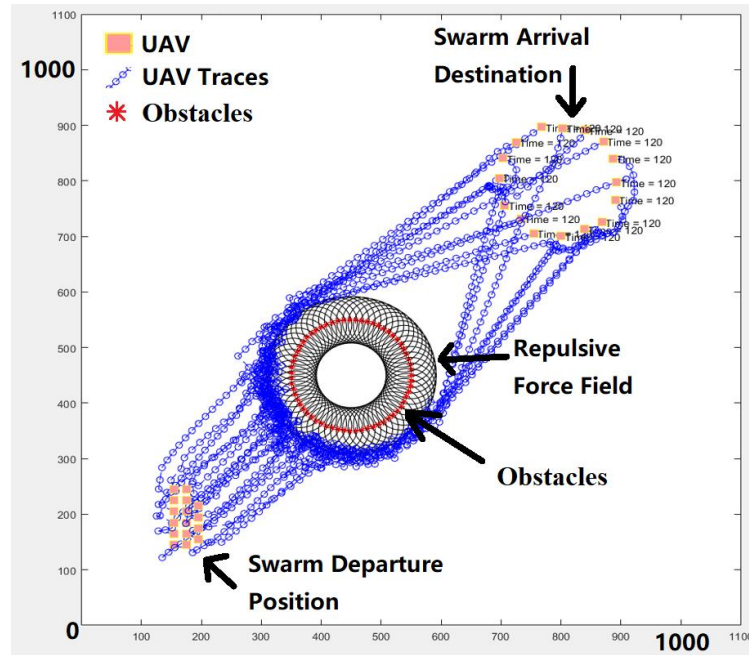


Figure 69. Example of Obstacle Avoidance Mobility Control When Applying Tolerance Molecular Force Model

B. Progressive Bridge Formation

To build the communication bridges shown in Figure 66, we have used the bridge construction process shown in Table 9. Figures 70 and 71 show the one-bridge and two-bridge scenarios for progressive bridge establishment. Here, the UAVs located on the swarm boundary are marked with red stars. The yellow nodes do not belong to any swarm since they are reserved for building communication bridges between swarms. The blue lines show the mobility traces of the reserved nodes when they move to different positions for forming the bridge(s).

Table 9. Bridge Construction Algorithm

- Each swarm detects its swarm boundary and finds out boundary nodes.
- Find the available shortest path between two swarms. Select boundary nodes to be bridge starting and ending nodes.
- Calculate bridge starting/ending nodes' positions based on RF signal range.
- if (there are enough UAVs left in the pool of reserved UAVs), then:
 - Split the required number of reserved UAVs and apply Tolerance Molecular Force Mode to perform mobility control. Use UAVs to construct the communication bridge.
 - Mark out the current bridge positions as occupied and unavailable.
 - Wait for a short delay $\Delta\sigma$ to avoid aircraft route congestion.
 - Go back to Line 2.

Figures 70 and 71 show the process of building the first and second bridge, respectively. When needed, more bridges can be built by using a similar procedure.

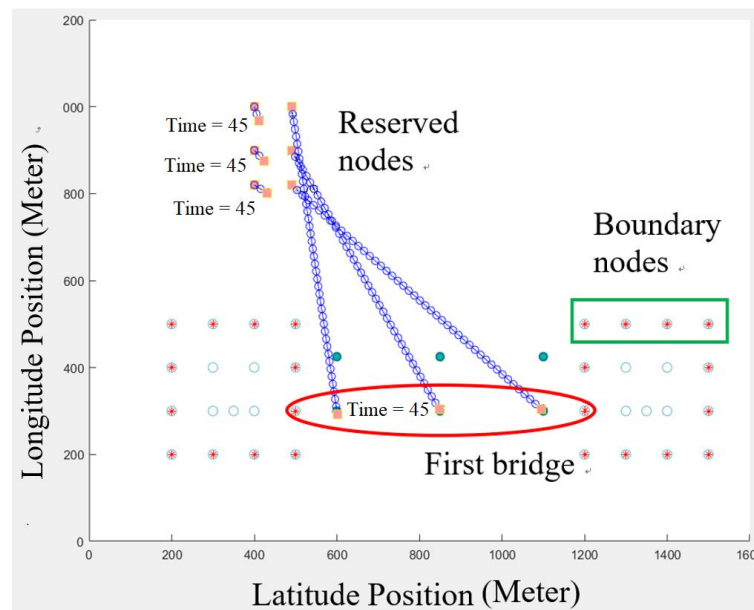


Figure 70. Progressive Communication Bridge Formation: First Bridge (The Nodes for Bridge Formation should Arrive at their Expected Bridge Locations at Time = 45 sec)

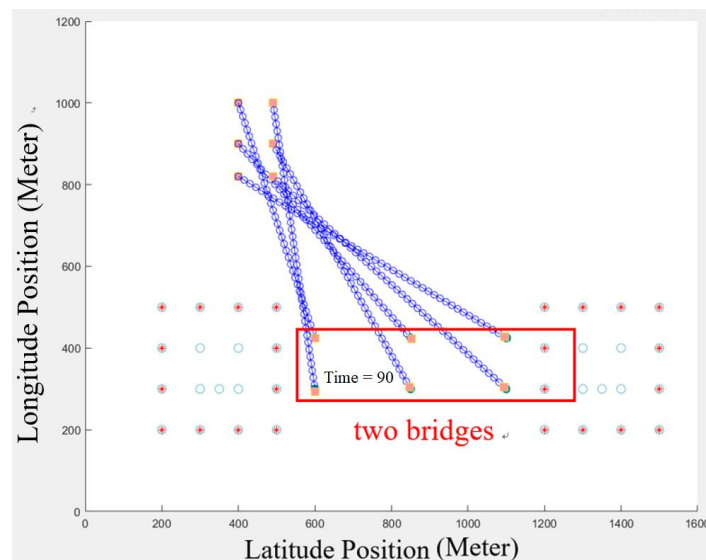


Figure 71. Progressive Communication Bridge Formation: Second Bridge (The Nodes for Bridge Formation should Arrive at their Expected Bridge Locations at Time = 90 sec)

After the first bridge is built, the protocol will continuously check whether the bridge meets the QoS requirements. If not, more bridges will be formed. In contrast, the teardown of one or more bridges may be performed if the network does not need to deliver heavy traffic. Note that the usage

of less bridges helps to reduce the route maintenance overhead.

The benefit of building communication bridges progressively, instead of building all bridges at the same time, is that it allows us to gradually add or remove the data paths based on the throughput requirements. Thus, any unused reserved UAVs can be used in other missions, as needed.

7.3.3 Edmonds-Karp Algorithm for Multipath Routing

This section describes the establishment of multiple paths based on the Edmonds-Karp Algorithm (EKA) [134], which is an implementation of the Ford-Fulkerson method [148] to compute the maximum flow amount in a network. EKA helps to quickly find out the maximum throughput that the entire network or a specific path can provide. Multiple paths may be used if a single path cannot meet the application's throughput requirement.

In this project, we have enhanced the EKA to build a multipath forwarding table. After that, the MP-UDP distributes the user traffic to multiple paths based on each path's capacity.

A. EKA Working Principle

Here we briefly explain the operating principle of the original EKA. Figure 72(a) shows an example of an EKA-oriented network model. Here, the left-most node is the source (or server) and the right-most node is the destination (or client) node. The division numbers indicate the channel usage ratio. For example, '0/2' indicates that the maximum data rate through this channel is 2 Mbps but zero data is flowing through it.

The algorithm starts from the source and executes the EKA on each node until reaching the destination. Figures 72(b)-(i) show the process of implementing EKA.

At the very beginning (Figure 72(b)), the source sends out *Path-Seeking* messages through all possible paths to reach the destination. After the messages reach the first-hop nodes, they are forwarded to the second-hop nodes (marked as violet in Figure 72(c)). This forwarding process continues, until at least one message reaches the destination (Figure 72(d)). The first received message tells the client the fastest path. The green nodes in Figure 72(e) represent the first path found by EKA. Furthermore, the message also has the information about the capacity that can be provided by the path. In this example, the maximum capacity of the first path is 1 since its minimum channel capacity is 1.

Before building the second path, another process needs to be performed, i.e., recalculating the remaining capacity of each channel. In this example (Figure 72(e)), we assume that all the channels between green nodes consume 1 unit of channel capacity, and therefore are marked as '1/1', indicating that these channels have been fully utilized and will not be considered for further path-seeking process.

Based on the calculated channel capacities, the sender sends out the *Path-Seeking* messages again. The hop-to-hop message forwarding process is performed until no more messages reach the client. Each time the channel capacity is recalculated and the channels with zero capacity are avoided. Figure 72(f) shows a possible result with three paths.

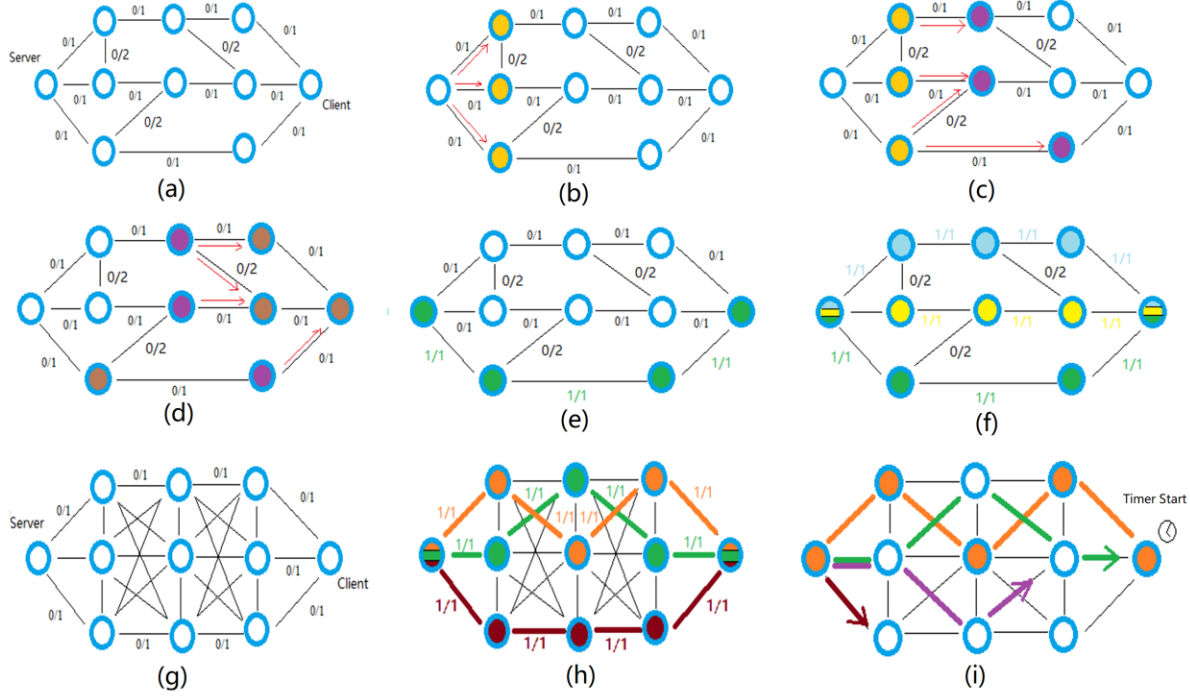


Figure 72. The EKA Progress and EKA-Based Multipath Detection. (a) An Example of an EKA-Oriented Network Model; (b) EKA Progresses to the 1st Hop Neighbours; (c) EKA Progresses to the Second Hop; (d) EKA Progresses to the Third Hop; (e) The First Path is Established; (f) Possible EKA Running Result with Three Paths; (g) An Example of an Ad-Hoc Network; (h) Apply Enhanced EKA on the Topology of (g); (i) Apply Enhanced EKA on (g), Here We Invoke the Timer on the Receiver Side after the First Path-Seeking Message Arrives

Equation 43 below shows that the running time is related to each path's discovery time $|E|$ and the maximum saturated edge-to-source distance $|V|$. Here, the time complexity is independent of the maximum network capacity.

$$T \sim O(|V| \cdot |E|^2) \quad (43)$$

B. Enhanced EKA in Ad-Hoc Networks

In this project, we have enhanced EKA to make it more compatible with ad-hoc networks. Figure 72(g) shows a case of an ad-hoc network.

The maximum capacity for each end-to-end path is assumed to be 1. We first apply the general EKA (without enhancement) for the route search. A three-path case is shown in Figure 72(h), which is generated by applying the EKA algorithm to the network topology shown in Figure 72(g). By using the basic EKA, we can find that the total capacity is 3, and the traffic is proportionally distributed among the three paths.

Such a method may help us to find multiple paths for the MP-UDP scheme with high throughput. However, general EKA has high path-seeking overhead since the routes are found iteratively and the sender floods messages to the whole network in each round. Each subsequent round takes

longer time than the previous round. Meanwhile we also wish to utilize the remaining ‘0/1’ links (i.e., without any traffic load) to increase the communication redundancy. Our *enhanced EKA* solves the above issues.

The goal of bridge routing is to proportionally distribute the traffic on each path (based on respective path’s capacity) to achieve the maximum path redundancy, while decreasing the path-seeking time. Assume that the capacity of each channel is 1 initially. Each EKA iteration tries to find the shortest path (with the minimum number of hops) to the destination, since the number of hops has a big impact on the path delay and packet loss rates [149]. The destination starts a timer after it receives the first *Path-Seeking* message. Figure 72(i) shows this process. Here, the destination has received multiple *Path-Seeking* messages before the timer expires. Hence, multiple paths are detected at the same time. After the timer expires, the client only keeps the shortest-hop path(s). Figure 73 shows the message forwarding trajectories for the above case based on the *enhanced EKA*. The enhanced EKA therefore converts the multiple iteration process into a one-round process, i.e., the source sends messages to all possible paths, and the receiver collects messages at the same time to find out the winner paths. The biggest benefit of utilizing this method is the reduction in path-seeking time.

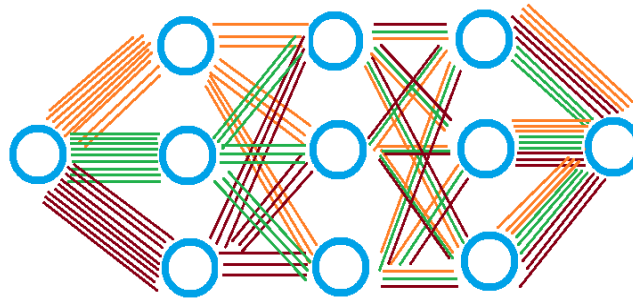


Figure 73. Adjusted EKA is Applied to Previous Network

Equation 44 below shows the runtime for the enhanced EKA, which is only related to the longest time ($|V|$) of seeking a path. Our lightweight *enhanced EKA* is more suitable for UAV networks which are highly dynamic and have unstable links.

$$T \sim O(|V|) \quad (44)$$

Equation (45) below gives a detailed computation time for MP-UDP to search traffic paths. Here n is the minimum hop number from the traffic sender to receiver and μ is the average delay for a PS packet to propagate from one node to a neighboring node. The μ value is affected by the propagation delay as well as the buffer queueing delay. Δt is the duration of the timer to be set up at the receiver node when it receives the first PS packet from a traffic sender. The computational time is proportional to the minimum number of hops from the sender to receiver node, which reflects the correctness of Equation (44).

$$T = 2n * \mu + \Delta t \quad (45)$$

Another benefit of the *enhanced EKA* is the improvement of routing redundancy. A conventional transport layer protocol uses only one path, and the loss of any node in that path causes the whole connection to be lost. Since the proposed EKA-based method can efficiently find multiple paths,

a link break in a path does not shut down the end-to-end transport layer connection. Compared with traditional EKA which finds the paths with maximum possible throughput between two nodes, the *enhanced EKA* provides a methodology to find high redundancy paths for a UAV network.

C. Probability of Detecting All Possible Paths

There is no guarantee that all the neighbors of a PS-forwarding node can receive the PS packets due to the possible packet loss events in each wireless link. If the PLR of each hop is γ and there are n hops between the source (server) and the receiver (client), Equation (46) below shows that the client has a probability of η to successfully receive such a PS packet.

$$\eta = (1 - \gamma)^n \quad (46)$$

The value of η drops quickly as the number of hops increases.

7.3.4 Design and Implementation of MP-UDP

To implement the above enhanced EKA in UAV networks, a MP-UDP cross-layer design is introduced, by building an MP-UDP sublayer between the transport and application layers. In addition, the MP-UDP also operates on the network layer for route selection. Figure 74 shows the cross-layer design architecture and operation process. The MP-UDP protocol uses three steps to transmit packets through different paths: *First*, the protocol finds out multiple paths to the destination by using the enhanced EKA. The PS packets are sent out by the sender. Upon receiving the PS messages, the receiver applies the *enhanced EKA* to calculate multiple paths. Then, the receiver sends back an ACK packet which contains all filtered path information. In short, this process creates a multipath forwarding table, which is different from the traditional routing table. This table is sent to the upper layer. In the *second step*, the MP-UDP allocates traffic to different routes based on the route quality (such as the average link capacity). The sender adds header and other information to each packet. *Finally*, the MP-UDP packets are sent on different paths.

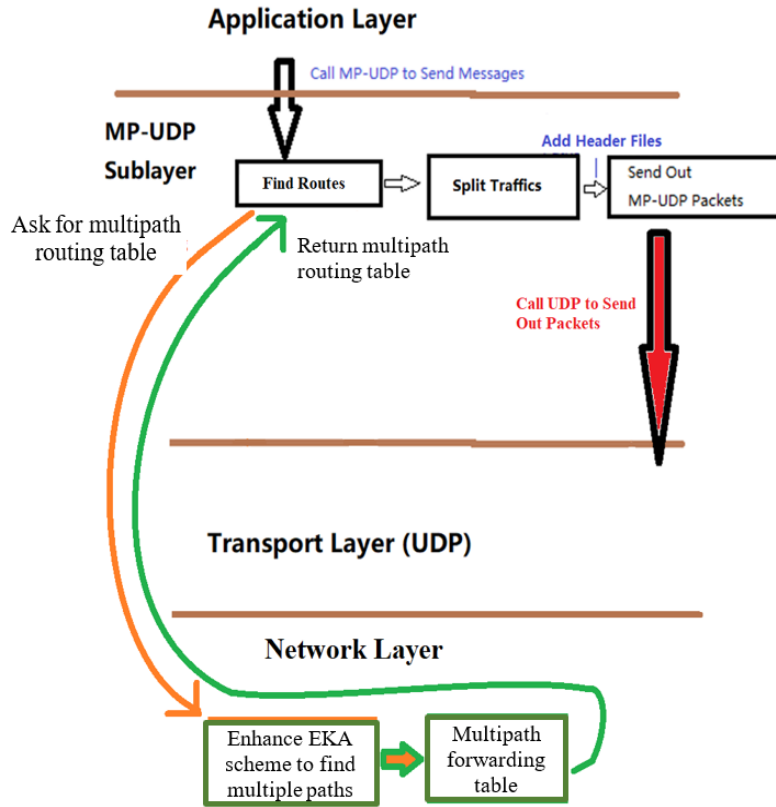


Figure 74. MP-UDP Sublayer Design

The following section describes how the sublayer is built and operates during data transmissions.

A. Nodes with Different Roles

Figure 75 shows the detailed functions for different node types. The nodes can be classified based on their roles in packet transmission: sender (or server), intermediate (relay) nodes, and receiver (or client). Each role performs different operations for the received MP-UDP packets.

In the beginning, the application layer calls MP-UDP functions to send out packets at the sender side. The application needs to provide the source IP, destination IP, local IP, as well as actual data to be sent out. Then, the application layer data units are passed to the MP-UDP sublayer.

The *sender* (server) *node* performs four operations in sequence: (1) Propagate the PS messages; (2) Wait for PS ACK packets; (3) Extract routing information from the PS ACK packet; (4) Add headers to packets and send out MP-UDP packets through multiple paths.

The *intermediate nodes* then forward both PS packets and MP-UDP packets to the next hop. When it receives a PS packet, it will check the previous packet's path and then decide whether a routing loop occurs. If it does not occur, the node adds its own IP address to the PS packet. Similarly, during the MP-UDP transmission period, each intermediate node checks the packet's path. Then, it selects the next node and forwards the packet to it.

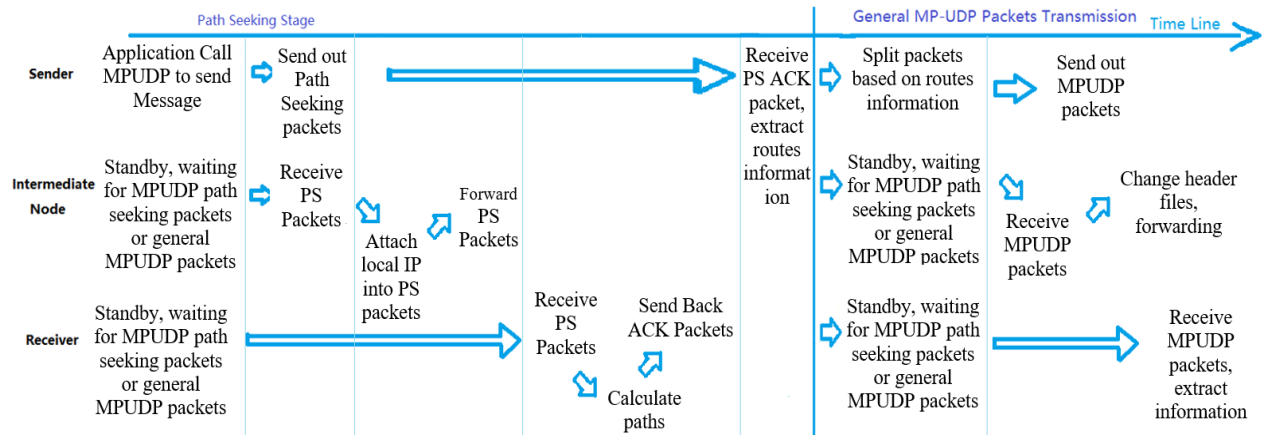


Figure 75. MP-UDP Functions for Different Node Roles

The *receiver* (or client) *node* performs different operations in path-seeking and MP-UDP transmission phases. During the path-seeking phase, it collects PS packets from different paths, and then applies *enhanced EKA* to calculate the best path(s). Then the PS ACK message containing path information, is sent back to the server. In the MP-UDP transmission phase, the receiver (client) opens a socket and receives the MP-UDP packets.

B. Path-Seeking

The first step of the MP-UDP protocol is to try to find the routes through the enhanced EKA. Whenever the MP-UDP sublayer receives a packet, it first compares the source IP, local IP, and destination IP. This process helps to determine the role of the local node. If the local IP is the same as the source IP, the local node is the source (server). The source node then propagates the PS messages. The message structure is shown in Table 10. It begins with a PS message check mark, which indicates whether it is a MP-UDP-oriented PS message. There are also other fields, such as the source IP, destination IP, loop prevention counter, and intermediate node IP. The *loop prevent counter* counts down each time the packet reaches a new node. It helps to prevent infinite flooding loops. Finally, we will have all intermediate nodes' IP addresses. The source node does not need to add its IP to this list, since the source IP was already included in the previous message. Then, the sender waits for the response from the client/destination and checks the received ACK to extract the multipath information.

Table 10. Path-Seeking (PS) Packet Format

PS Message Header/Check Mark	Server IP	Client IP	Loop Prevent Counter	Intermediate Node IP
------------------------------	-----------	-----------	----------------------	----------------------

The receiver keeps the socket open for a short time period (Δt) after it receives the PS packets from the network. After the receiver collects PS packets from all possible paths, it applies *enhanced EKA* to remove unnecessary paths. The selected paths have the least number of hops.

This process can screen out the paths with high delay and low throughput. After finding the filtered paths, the receiver sends back a PS ACK packet to the sender.

The PS ACK packet structure is shown in Table 11. It contains the attributes of the multipath forwarding table with the basic structure as shown in Table 12. The table provides the forwarding path information with all the intermediate nodes' IP addresses in each path. Those IPs are used to forward the MP-UDP packets with *hop number index* (HNI), which can be found by comparing its own IP address and IPs listed in the path. HNI will be discussed in Part C below.

To send the forwarding table back to the source node, the receiver uses the first path shown in Table 12 to send the PS ACK packet in the reverse direction. In this example, it follows the path 10.0.0.6 -> 10.0.0.4 -> 10.0.0.3 -> 10.0.0.1. Each intermediate node checks that path to find out whether it is in the IP sequence. If so, it will forward the packets to the next relay node based on the sequence.

Table 11. Path-Seeking ACK Packet Format

First Path	Second Path	Third Path	...	...	
Sender's IP	1st Intermediate Node IP	2nd Intermediate Node IP	...	Last Intermediate Node IP	Receiver's IP

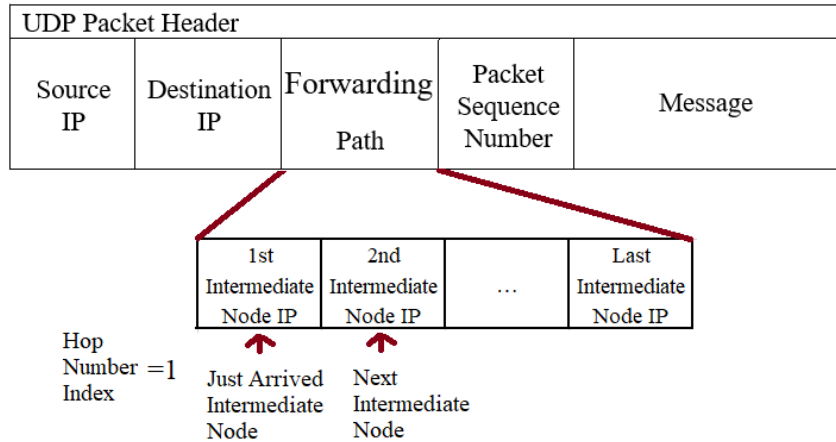
Table 12. Multipath Forwarding Table (An Example)

Path Number	Sender IP	InterNode 1	InterNode 2	Receiver IP
Path #0	10.0.0.1	10.0.0.3	10.0.0.4	10.0.0.6
Path #1	10.0.0.1	10.0.0.2	10.0.0.4	10.0.0.6
Path #2	10.0.0.1	10.0.0.3	10.0.0.5	10.0.0.6
Path #3	10.0.0.1	10.0.0.2	10.0.0.5	10.0.0.6

C. Sending and Forwarding the MP-UDP Packets

After the server receives MP-UDP path information from the PS ACK packet, it checks the available paths and then transmits the data packets from the application layer to different paths. The MP-UDP data packets are created, with the structure shown in Table 12. Here, the HNI represents the hop ID that the packet has reached. In other words, the position of the current relay node's IP in the *forwarding path* goes to the HNI value. For example, an HNI value of 1 tells that the packet reaches the first intermediate (relay) node. The next intermediate (relay) node's IP is the 2nd IP indicated in the field of *forwarding path* (see Table 13). Therefore, a relay node knows how to forward the MP-UDP packets to the next hop based on HNI. Every time the MP-UDP packet reaches a new relay node, the HNI value is increased by 1. The *packet sequence number* helps to combine all the MP-UDP packets and rebuild the messages at the client side. The *Message* field is the payload to be delivered to the application layer.

Table 13. MP-UDP Packet Format



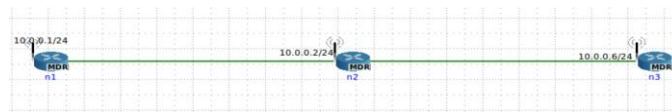
In summary, MP-UDP is a light-weight protocol for the communications between two UAV swarms. Each swarm can select an ambassador node that is responsible for the aggregation of all packets in its swarm and then utilizes the bridges for inter-swarm communications. The client and server shown in Figure 72(g) are the examples of ambassadors. The proportionally distributed traffic flows can fully utilize all the bridge links between the two ambassadors.

7.4 Results and Discussion

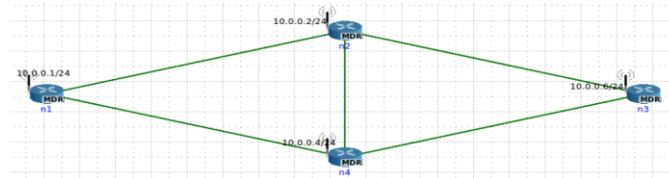
7.4.1 Performance of MP-UDP Protocol

In our implementation, CORE cooperates with EMANE to run physical and MAC layers. We have implemented the entire MP-UDP sublayer as a module to CORE. A UDP traffic generator is specially designed, which can create messages and directly call the MP-UDP functions to search optimal paths, split the traffic, and finally transmit data packets on different paths. The 2-hop and 3-hop network topologies with 1~3 bridges are built and tested in Figures 76 and 77, respectively.

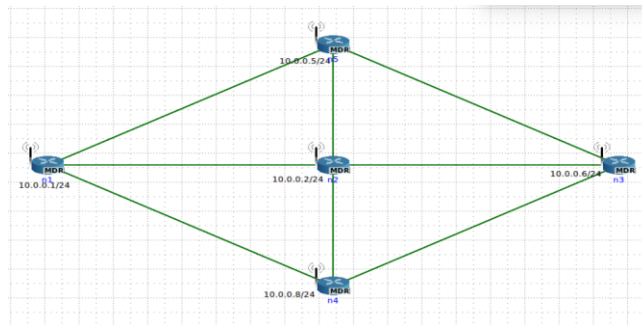
The traffic is generated at the source node (with IP 10.0.0.1, located at far left in Figures 76 and 77) which is sent to the destination node (with IP 10.0.0.6, located at far right). The horizontal distance between two adjacent nodes is 400 meters, and the corresponding vertical distance is 100 meters. IEEE 802.11b with Ad-Hoc mode default settings is chosen as the physical and MAC layer. The omnidirectional antennas are used in these figures.



(1). 2-Hop, 1-Bridge Topology



(2). 2-Hop, 2-Bridge Topology

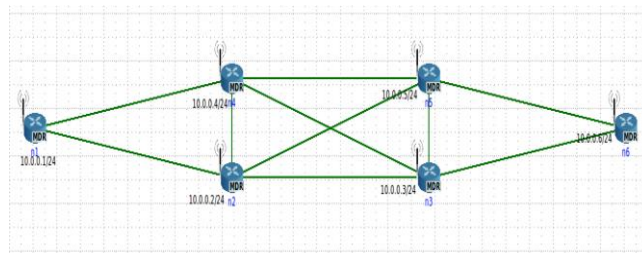


(3). 2-Hop, 3-Bridge Topology

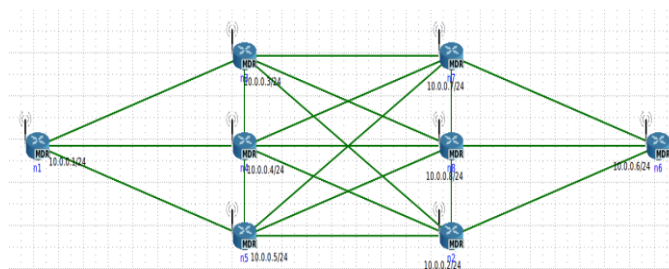
Figure 76. Bridge Topology (2-Hop) Considered in Simulations



(1). 3-Hop, 1-Bridge Topology



(2). 3-Hop, 2-Bridge Topology



(3). 3-Hop, 3-Bridge Topology

Figure 77. Bridge Topology (3-Hop) Considered in Simulations

The MAC and physical layer configurations in CORE are shown in Figure 78, where the unicast channel data rate is 11 Mbps and most parameters use default settings. Figure 78(2) shows the properties of omnidirectional antenna.

Figure 78. (1) 802.11 MAC Parameter Settings, (2) 802.11 Physical Layer Parameters Settings

The EMANE plugin libraries automatically calculate the received signal strength (RSS) at each node, and then use a built-in lookup table to obtain the corresponding PLR. With these MAC and physical layer settings, the maximum transmission range of a node is 700 meters.

The two important QoS parameters that we study are throughput and PLR. Note that the delay is highly related to the number of hops.

A. Throughput Performance

Our MP-UDP scheme aims to achieve the maximum throughput. The *enhanced EKA* can find the bottleneck link for each path. The MP-UDP protocol can fully utilize the capacity of each link by properly splitting the traffic flows for each path.

Figure 79 shows the traffic received at the client from each of the three routes for the topology shown in Figure 76(3). Here, the server uses approximately 2 Mbps of data sending rate and an even split of traffic is achieved by the MP-UDP protocol.

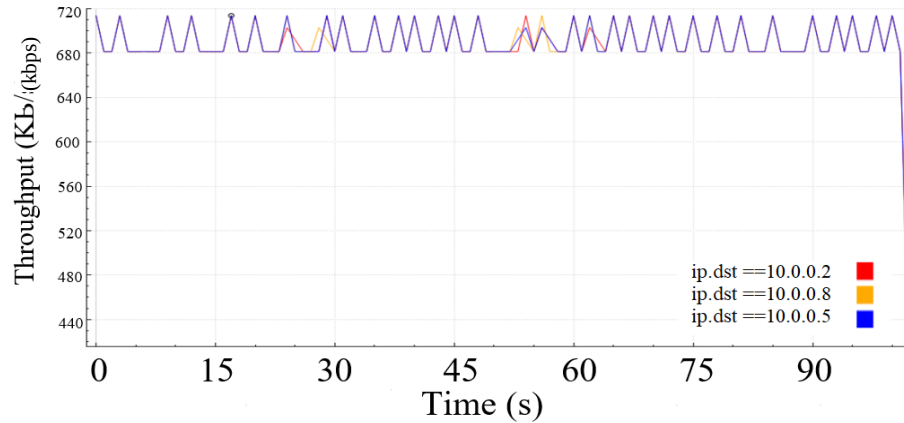


Figure 79. Traffic Split When Applying MP-UDP in a 2-Hop bridge. Here, the Red and Orange Curves are Overlapped with the Blue Curve

Figure 80 shows the throughput results for the network topologies shown in Figures 76 and 77. The dark blue, orange, and grey lines show the throughput for the 2-hop topology shown in Figure 76 (1), (2), and (3), respectively. Further, the light blue, green and yellow lines correspond to throughput for the 3-hop topology shown in Figure 77 (1), (2), and (3), respectively. In Figures 76(1) and 77(1), the conventional UDP protocol (instead of MP-UDP) is applied, and only single path is established, whereas the 2-bridge and 3-bridge topologies in Figures 76(2), 76(3), 77(2) and 77(3) enable the use of MP-UDP.

For the 2-hop network topology shown in Figure 76, the throughput increases with the data sending rate initially. However, the throughput for a 1-bridge topology (where UDP is applied) is saturated at 2.06 Mbps when the data sending rate reaches around 2.10 Mbps; the throughput decreases when the data sending rate is increased further. In contrast, the maximum throughput for 2- and 3-bridge topologies (where MP-UDP is applied) is 2.60 Mbps and 2.68 Mbps, respectively, when the data sending rate is 2.85 Mbps. Note that the 3-bridge topology achieves only a marginally higher throughput compared to the 2-bridge topology because the channel capacity has been fully utilized and the nodes on the neighboring paths interfere with each other. For the 3-hop topology shown in Figure 77, the throughput for 1-bridge topology (where UDP is applied) is saturated at 1.41 Mbps when the data sending rate reaches around 1.8 Mbps; the throughput decreases when the data sending rate is increased further. In contrast, the maximum throughput for 2- and 3-bridge topologies (where MP-UDP is applied) is 1.8 Mbps for the data sending rate of 3 Mbps. Note that the 2-bridge and 3-bridge topologies achieve almost the same throughput. As mentioned above, this is mainly because the channel capacity has been fully utilized and the nodes on the neighboring paths interfere with each other.

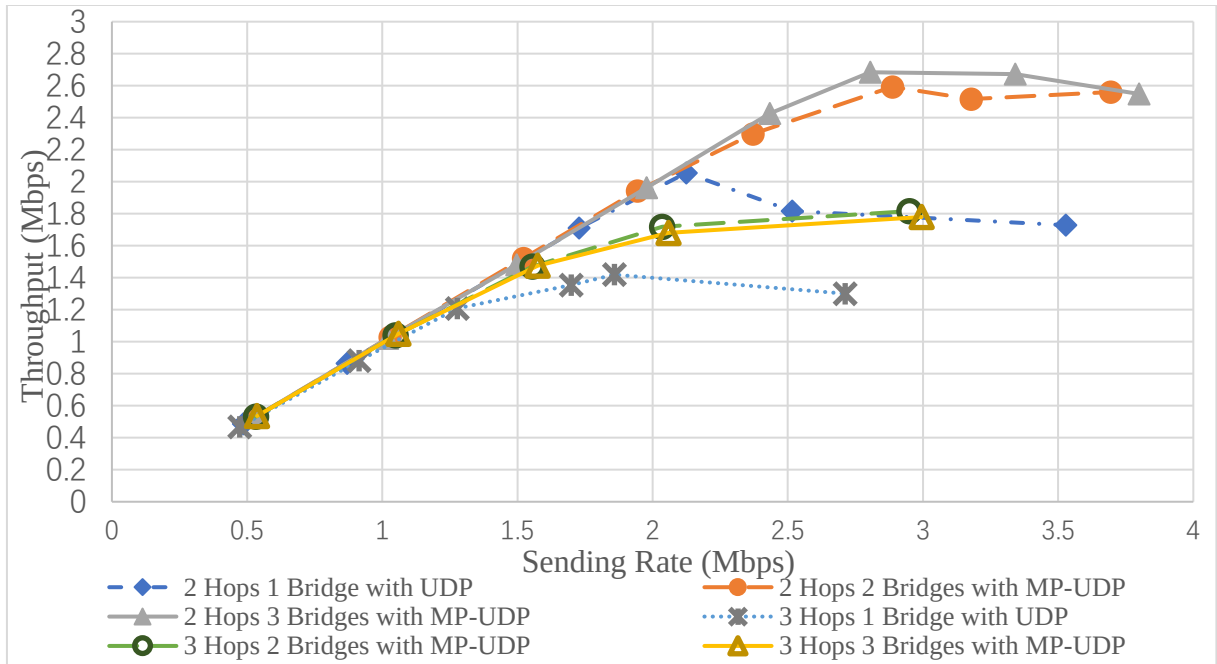


Figure 80. Throughput vs. Sending Rate for 2- and 3-Hop Topologies with Omnidirectional Antenna

7.4.2 Performance of MP-UDP with Directional Antenna

The previous tests were all based on omnidirectional antennas, where the throughput reaches its limit when the wireless channel cannot send more packets due to interference. That is the reason why 2- and 3-bridge routes did not provide significant throughput gain. To fully utilize the advantages of the proposed MP-UDP protocol, we use directional antennas in the network, which are known to provide the spatial reuse and reduced interference [150]. Figures 81 and 82 show the modified network topologies as compared to those in Figures 77(2) and 77(3). The nodes in these two networks have multiple interfaces with distinct IPs for each interface.

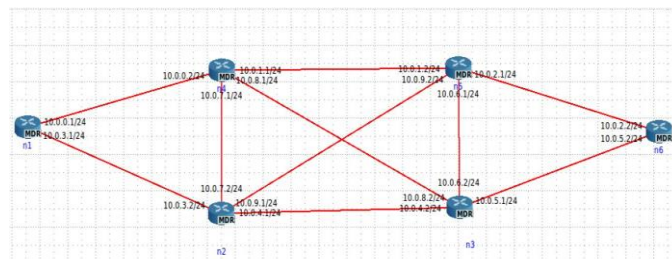


Figure 81. 3-Hop, 2-Bridge Network with Directional Antennas

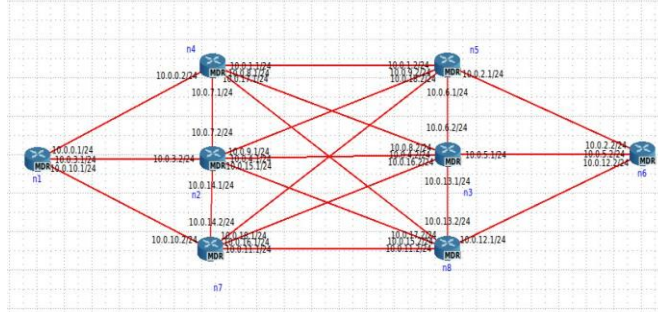


Figure 82. 3-Hop, 3-Bridge Network with Directional Antennas

Due to the complexity of implementing the directional antenna patterns in EMANE, an alternative method is used below, wherein we have plugged in the wireless channel properties into wired channel. To simulate the wireless channels for a directional antenna, the IEEE 802.11b channel properties are measured in CORE with EMANE plugin libraries for the physical and data link layers. Figure 83 shows the PLR and data sending rate relationship, measured by placing two nodes 400 meters apart horizontally (same as the horizontal interval distance used in the remaining tests in this project). The PLRs of each wireless link for different data rates are measured in the emulator. The PLR in Figure 83 increases with an increasing data sending rate due to the traffic congestion. Note that placing these nodes 200 meters apart also provides the same result. This fits the wireless channel properties provided by EMANE physical layer IEEE 802.11b model. If the data sending rate is specified, the PLR can be found from Figure 83. Therefore, a wireless channel can be replaced by a wired channel with manually setup channel properties for different data rates.

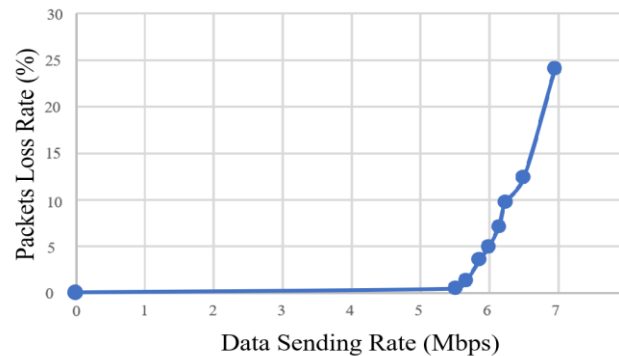


Figure 83. PLR vs. Data Sending Rate Properties

The MP-UDP protocol has been enhanced to support more radio interfaces in a single node. Typically, the PS messages are flooded through all the interfaces of a node. In addition, each radio interface of the directional antenna in a node needs to find its pair (reception) directional antenna in another node. Here, we assume that the antenna pairing information is known. Another significant adjustment is the packet format of the PS ACK message, shown in Table 14. Compared with Table 11, this format indicates which interface of a node should handle the MP-UDP packets.

The current implementation of MP-UDP uses the built-in routing protocol in CORE, called OSPFv3-MDR, which is an extension of OSPFv3 to support wireless networks. OSPFv3-MDR

runs in the background and maintains the *OSPF routing table* that is used by the destination node to send back the multipath forwarding table to the source node.

Table 14. Multi-Interface PS ACK Packet Format

Sender	First Node		Second Node		...		Destination
Sender Interface IP	Receiver Interface IP	Sender Interface IP	Receiver Interface IP	Sender Interface IP		...	Receiver Interface IP

Figure 84 shows the 3-hop directional network simulation results. With the use of directional antennas, the network can reach a maximum of 10.8 Mbps and 15.7 Mbps throughput for the 2- and 3-bridge topologies, respectively.

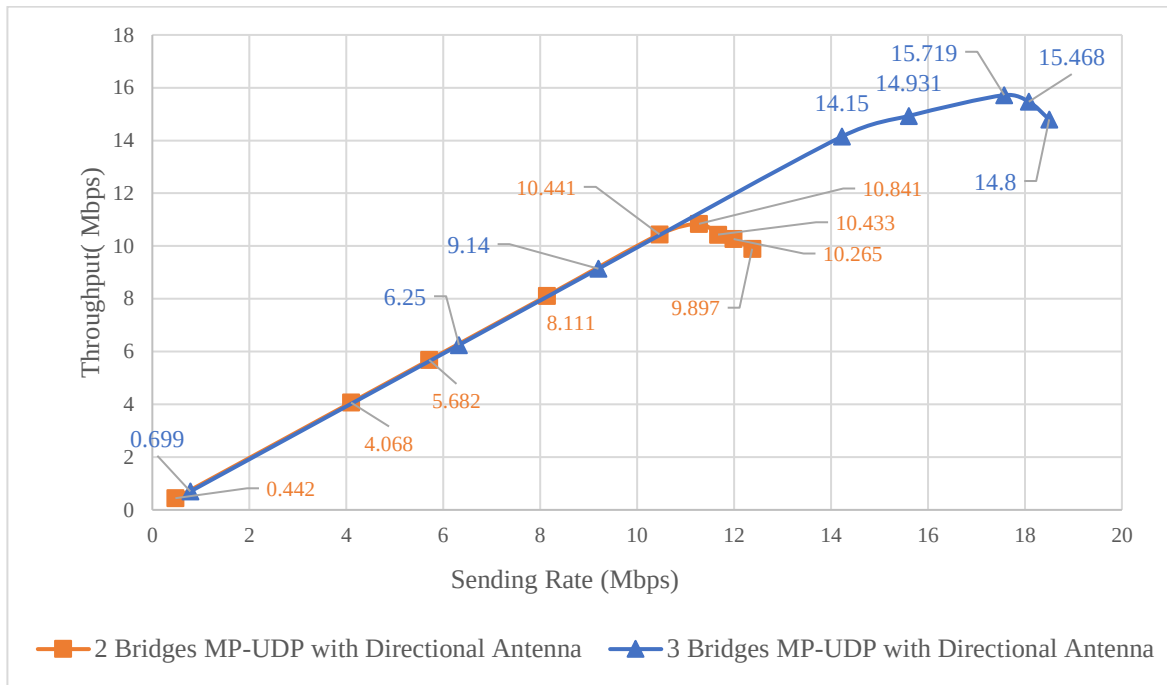


Figure 84. Throughput Performance of MP-UDP with Directional Antennas

7.4.3 MP-UDP with Omnidirectional Antenna for Mobile Nodes

In this section, we demonstrate that our proposed MP-UDP scheme can provide better throughput compared with conventional UDP for mobile nodes.

Figure 85 shows a simple scenario, where node *n1* (with IP 10.0.0.1) first moves down vertically from its initial position at (200, 500) to (200, 900). Then it moves up vertically to (200, 100) and

then finally to its initial position at (200,500). The node speed is 30 meters/s. During this movement, node *n1* keeps sending data packets to node *n4* at 500 Kbps. The IEEE 802.11b settings with omnidirectional antennas are used here.

Figure 86 shows the throughput when *conventional UDP* is applied with the optimized link state routing (OLSR) protocol. The throughput drops to zero for the time instants when no path is available from node *n1* to *n4* (i.e., when node *n1* moves totally out of the signal range of its intermediate nodes *n2* and *n3*). Here, the convergence speed of OLSR protocol determines the time duration with zero throughput.

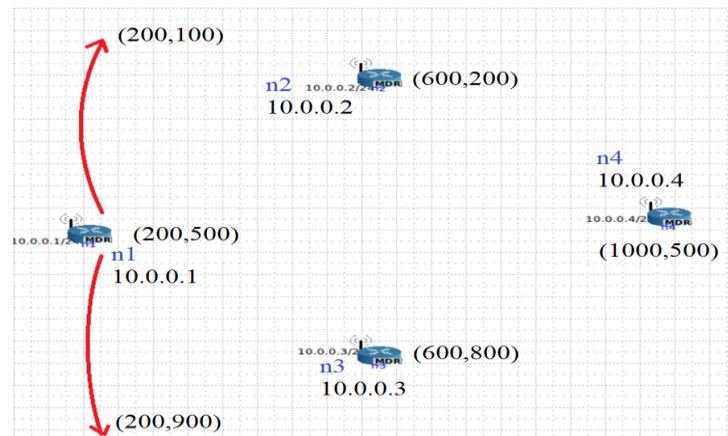


Figure 85. A Network Topology with Node Mobility

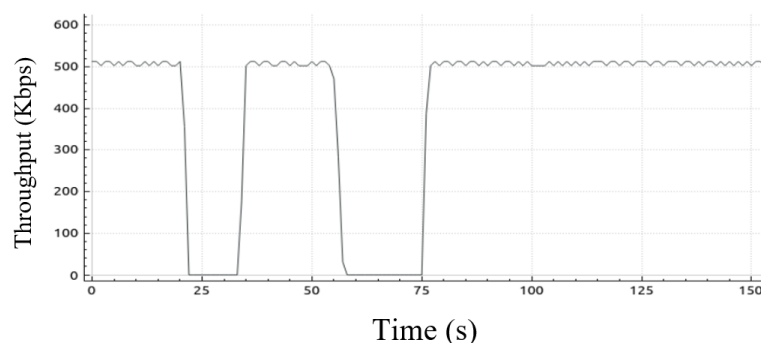


Figure 86. Throughput When Using UDP with OLSR Protocol

Figure 87 shows the throughput when the MP-UDP protocol is used. Note that the MP-UDP also works at the network layer which utilizes the Enhanced EKA to find multipath forwarding table. The black line shows the total throughput, whereas violet and green lines represent the throughput contributed by the upper (through *n2*) and lower (through *n3*) paths, respectively. The use of MP-UDP does not allow the total throughput to drop to zero, even when there is a 50% packets loss rate in some time windows. If we increase the number of intermediate nodes, the throughput will improve further. Therefore, the QoS performance is significantly improved when the MP-UDP is implemented.

In UDP, the source ($n1$) (at its initial location (200, 500)) communicates with the destination ($n4$) through either the intermediate node, $n2$ or $n3$. When the source node ($n1$) moves vertically, its link to one of the intermediate node breaks. On the other hand, in MP-UDP protocol, the source node ($n1$) in its initial position (at 200, 500) simultaneously communicates with the destination ($n4$) through two paths (corresponding to the two intermediate nodes, $n2$ and $n3$). Therefore, one of the two links is always available when the source node moves up or down.

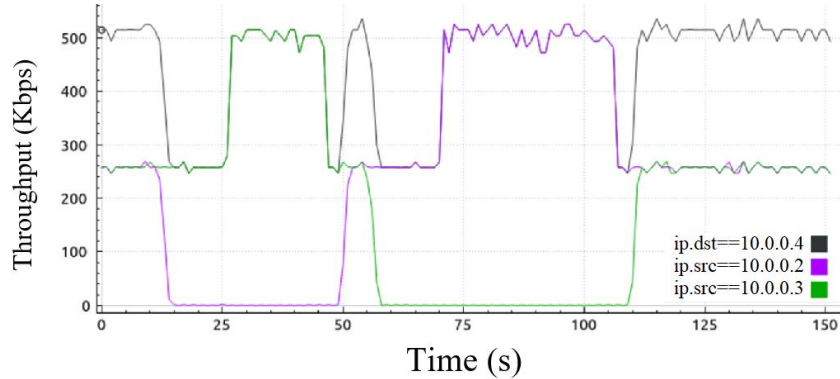
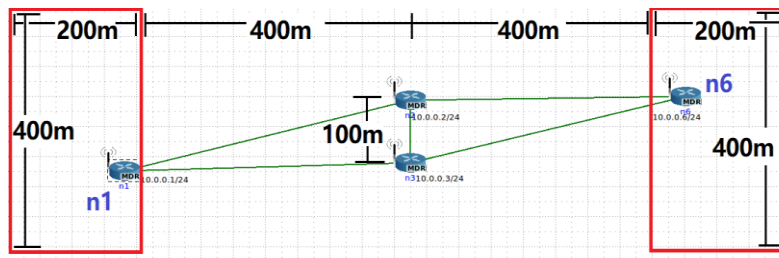


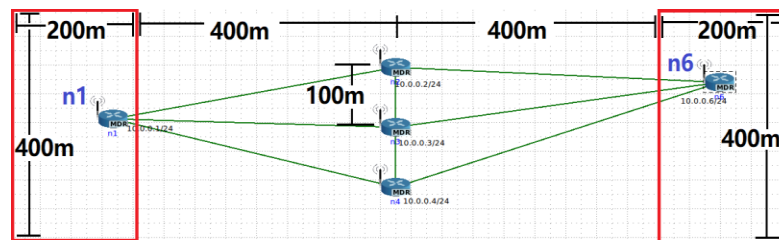
Figure 87. Throughput When Using MP-UDP

A. Throughput

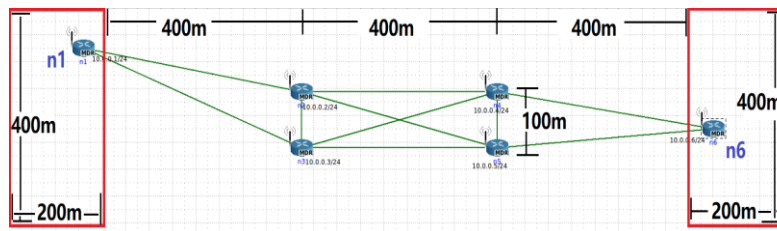
More general mobility cases are also tested for scenarios shown in Figures 88(1), (2), (3), and (4). The tests scenarios are similar to the static cases in Figure 76(2), (3), 79(2), (3). Node $n1$ on left side is the sender and node $n6$ on right is the receiver. In these new tests, Gauss Markov mobility model is applied for both sender and receiver nodes. The average speed is set at 25 m/s. The red rectangle boxes on the figure's left and right sides (with size 200m by 400m) indicate the moving ranges for the sender ($n1$) and receiver ($n2$).



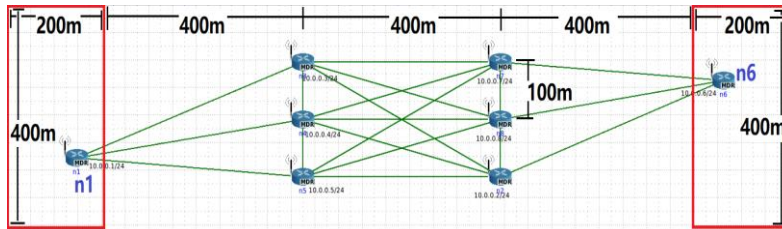
(1). 2 Hops, 2 Bridges.



(2). 2 Hops, 3 Bridges.



(3). 3 Hops, 2 Bridges.



(4). 3 Hops, 3 Bridges.

Figure 88. Simulation Scenarios for Overhead Analysis. Moving Range of Nodes are Highlighted by Red Rectangle Boxes

Figure 89 shows our test results as compared to the UDP with OLSR scheme. The OLSR has a default HELLO message interval of 2 seconds, and a default TC message distribution interval of 5 seconds. To compare with OLSR and general UDP, our MP-UDP scheme uses a 5 second PS packet distribution interval. The user data generation rate is varied from 500 kbps to 2.5 Mbps. As we can see, the MP-UDP scheme outperforms the OLSR with UDP scheme. This is because UDP sends packets by seeking a single path from the network layer, by using the OLSR protocol. However, whenever the network topology changes, the OLSR protocol needs time to converge. Also, there is no backup paths saved in buffer for transport layer protocol (UDP). These effects cause significant packet loss when an old path fails, and a new path is still being searched.

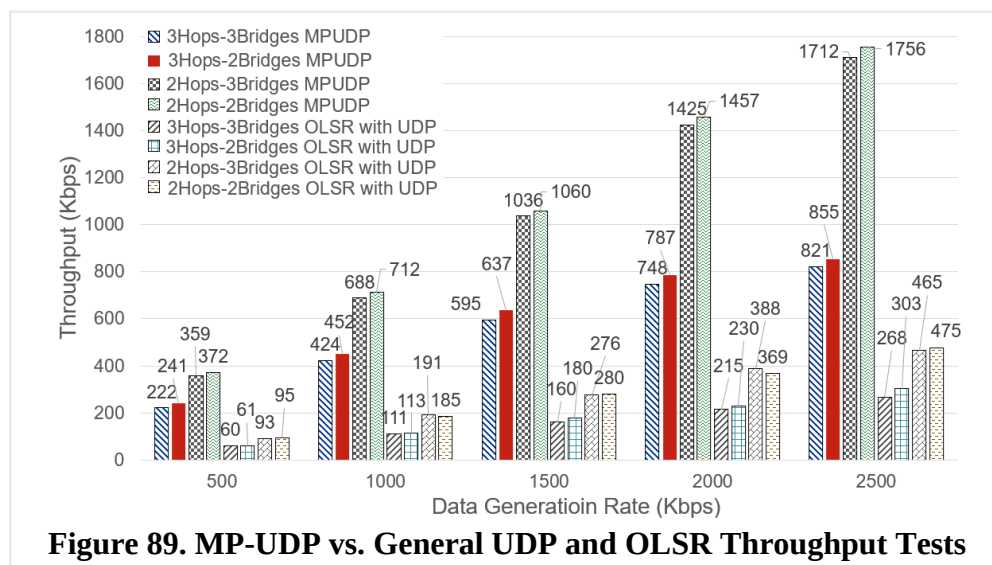


Figure 89. MP-UDP vs. General UDP and OLSR Throughput Tests

DISTRIBUTION STATEMENT A: Approved for Public Release; distribution unlimited

In contrast, our MP-UDP design provides multiple paths for traffic flows. The failure of a single path may not fully cut off the communication between the sender and receiver. Also notice that the 2-bridge test results are slightly better than 3-bridge cases. This is caused by overhead messages: 3-bridge test scenarios generate more overhead packets. Also, the interference between bridges cannot be ignored as the bridge nodes share the same wireless channel. For realistic applications, a longer distance between bridges can be applied to decrease the interference. Furthermore, complex real environment may cause a bridge teardown. Maintaining enough bridges is therefore necessary.

B. Overhead

The protocol overhead is related to network sizes. Here MP-UDP overhead is a sum of two parts: (i) PS and PS-ACK packets, and (ii) MP-UDP packet header. For part (i), PS and PS-ACK packets are responses to path search. They are similar to the general routing protocol packets, which flood the network. For part (ii), the MP-UDP packet header is the data between UDP packet header and user data (message) shown in Table 13. Overhead tests were done for topologies shown in Figure 88.

Figure 90 shows the test results for communication overhead. MP-UDP overhead ratio is higher than general UDP with OLSR. This is because our protocol has a reactive design, which is different from general proactive OLSR scheme. When there is no traffic flow between two swarms, our scheme does not generate any overhead.

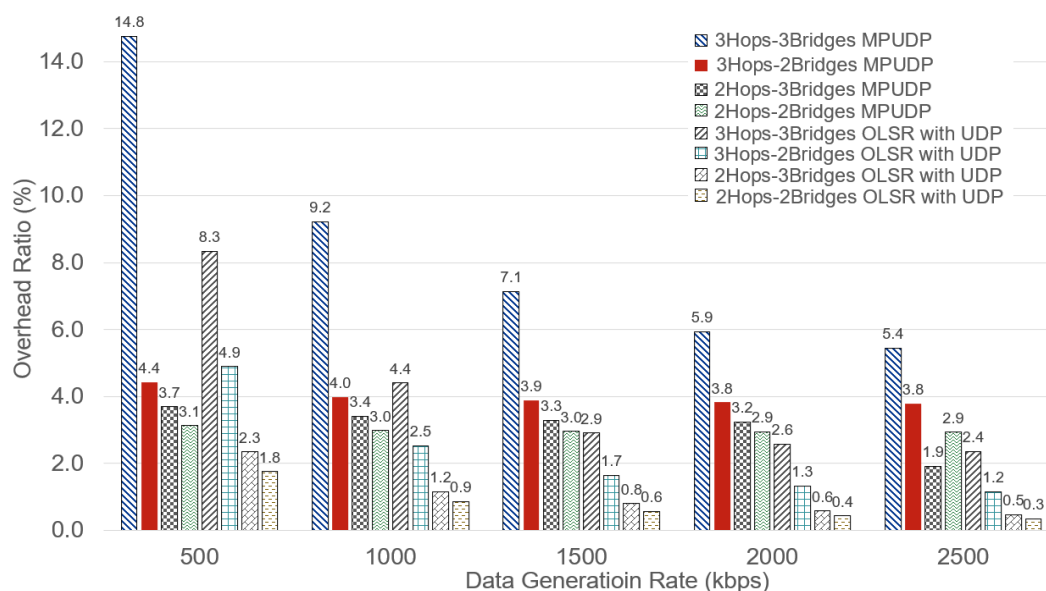


Figure 90. MP-UDP vs. General UDP and OLSR Overhead Ratio

8. CONCLUSIONS

We considered a decentralized, dynamic, multihop, ad-hoc airborne network consisting of low SWaP fixed-wing UAVs. To facilitate robust and reliable QoS-aware communication in such a network, we have designed distributed routing and transport layer protocols.

We have designed an adaptive, QoS-aware, cross-layer OLSR routing protocol, which prevents packet transmission over a short-lived, broken, or congested route, and reduces the computational overhead which can improve the network lifetime.

Next, a novel UAV routing and morphing scheme based on the skeleton structure, called *SSR*, was proposed for swarm network. *SSR* includes an addressing system that provides nodes with geometric coordinates. Based on that, a pipe with interconnected paths is constructed between a source-destination pair, which provides a foundation for the adaptive online routing algorithm. This routing scheme has low complexity, can avoid congested areas, and achieves traffic balancing in the pipe. *SSR* uses geometric forwarding and avoids the flooding of route search messages throughout the network. It only needs a local search in a specific region, whenever the destination moves away. The simulation results verified that the *SSR* scheme can successfully establish a load-balanced high-throughput leaf-like routing pipe. *SSR* outperformed the single path throughput by up to two times and noticeably reduced the latency. The proposed skeleton structure can also simplify and facilitate the UAV swarm morphing process. The future research will investigate the application of DRL for seeking the optimal routing pipe parameters such as pipe width and trend in a centralized large-scale UAV swarm network.

We have conducted pioneering research on skeleton-based 3D UAV network routing, which is especially suitable for smooth UAV swarm formation changes. The skeleton of 3D-TR consists of a trunk, multiple stems, and many stemlets. A node just needs to reach the closest TSS node for data transmissions. The 3D-TR scheme does not need the GPS location information for skeleton identification and has a low cost of route maintenance since the skeleton nodes are relatively stable. This scheme helps to avoid frequent rerouting operations due to its semi-proactive nature. It is also a distributed implementation (thus avoids the central controller and single-point failure). The biggest advantage is that it fits the swarm behaviors very well by dividing a big route change into a series of small paths for adjustments. Thus, the 3D-TR scheme has both proactive path maintenance (i.e., the establishment of TSS) and reactive processing (i.e., small adjustment of partial skeleton).

Congestion control is a challenging issue in UAV networks since the packet loss may be caused by many different factors. Using the ABC scheme, the congestion loss and random loss can be handled efficiently with optimal coding parameters. Since the link conditions are dynamic, it is difficult to determine the coding parameters for each specific communication session. The DRL model was designed to compute the optimal batch coding parameters. The network state parameters are collected for DRL inputs, including the link capacity and PLR. Using these parameters, the DRL network can be trained efficiently. For a new communication session, the DRL suggests the optimal coding rates for the ABC algorithm based on the network conditions. The coding scheme keeps updating its parameters with the change in the network conditions. Our simulation results have shown that the DRL-based congestion control increases the throughput and decreases the end-to-end delay for SP and MPSB scenarios.

The idea of a progressive bridge transport and a cross-layer MP-UDP protocol was proposed for inter-swarm communications in UAV networks. By progressively establishing bridge communications, we can adapt to the traffic data rate changes. The implementation of MP-UDP in a bundle of EMANE and CORE network emulators, established multiple paths based on the *enhanced EKA*, which provided improved inter-swarm communication when compared with traditional UDP. Since the wireless channel has its own capacity limits, establishing more bridges may not help much when the channels reach the maximum capacity. Our enhanced MP-UDP scheme increases the network redundancy when the UAV swarms have unstable topologies.

Our protocols consider the impact of QoA parameters. In particular, the swarm formation changing rate determines the routing table refreshing frequency. Our enhanced OLSR scheme can adapt to high swarming dynamics. Our 2D skeleton routing protocol captures the swarm evolution trend, which is also a critical QoA metric. The trend-based path seeking significantly reduces the need for re-routing. Our 3D skeleton routing scheme can identify the most stable swarm regions and select the most stable nodes as skeleton nodes. The QoA-adaptive routing maintains a high QoS level under high swarm dynamics.

9. REFERENCES

1. Bujari, A., et al., "Flying ad-hoc network application scenarios and mobility models," *International Journal of Distributed Sensor Networks*, Vol. 13, No. 10, pp. 1-17, 2017.
2. Xie, J., et al., "A survey and analysis of mobility models for airborne networks," *IEEE Communication Surveys & Tutorials*, Vol. 16, No. 3, pp. 1221-1238, 2014.
3. Wan, Y., Namuduri, K., Zhou, Y., and Fu, S., "A smooth-turn mobility model for airborne networks," *IEEE Transactions on Vehicular Technology*, Vol. 62, No. 7, pp. 3359-3370, 2013.
4. Perez, A. G., and Cano, M. D., "Flying ad hoc networks: A new domain for network communications," *Sensors*, Vol. 18, No. 10, pp. 3571-3594, 2018.
5. Gupta, L., Jain, R., and Vaszkun, G., "Survey of important issues in UAV communication networks," *IEEE Communication Surveys & Tutorials*, Vol. 18, No. 2, pp. 1123-52, 2016.
6. Maxa, J. A., Mahmoud, M. S. B., and Larrieu, N., "Survey on UAANET routing protocols and network security challenges," *Ad hoc & Sensor Wireless Network*, Vol. 37, 2017.
7. Fan, X., Cai, W., and Lin, J., "A survey of routing protocols for highly dynamic mobile ad hoc networks," *IEEE International Conference of Communication Technology*, pp. 1412-1417, 2017.
8. Nivetha, S. K., and Asokan, R., "Hybrid ACO-PSO based multi objective optimization for quality-of-service routing in mobile ad hoc networks," *International Journal of Applied Engineering Research*, Vol. 9, No. 24, pp.24651-24668, 2014.
9. Riedl, A., and Schupke, D. A., "Routing optimization in IP networks utilizing additive and concave link metrics," *IEEE/ACM Transactions on Networking*, Vol. 15, No. 5, pp. 1136-48, 2007.
10. Yu, Y., et al., "Bionic optimization-based stability and congestion aware routing algorithm for airborne highly dynamic network," *International Journal of Distributed Sensor Networks*, pp. 20-37, 2016.
11. Ateya, A. A., et al., "Latency and energy-efficient multi-hop routing protocol for unmanned aerial vehicle networks," *International Journal of Distributed Sensor Networks*, Vol. 15, No. 8, 2019. DOI: 10.1177/1550147719866392.
12. Malhotra, A., and Kaur, S., "A comprehensive review on recent advancements in routing protocols for flying ad hoc networks," *Transactions on Emerging Telecommunication Technologies*, 2019. DOI: 10.1002/ett.3688.
13. Hayat, S., Yanmaz, E., and Muzaffar, R., "Survey on unmanned aerial vehicle networks for civil applications: A communications viewpoint," *IEEE Communication Surveys & Tutorials*, Vol. 18, No. 4, pp. 2624–2661, 4th Quart., 2016.
14. Arafat, M. Y., and Moh, S., "Routing protocols for unmanned aerial vehicle networks: A survey," *IEEE Access*, Vol. 7, pp. 99694-99720, 2019.
15. Jabbar, A., and Sterbenz, J. P., "AeroRP: A geolocation assisted aeronautical routing protocol for highly dynamic telemetry environments," *International Telemetering Conference*, pp. 1-11, 2009.
16. Li, X., and Huang, J., "BPP: An adaptive beacon scheme for geographic routing in FANET," *International Conference of Parallel & Distributed Computing & Appl. Technology (PDCAT)*, pp. 293-299, 2017.
17. Perkins, C., et al., "Ad hoc on-demand distance vector (AODV) routing," *RFC 3561*, July 2003.

18. Johnson, D., et al., "The dynamic source routing protocol (DSR) for mobile ad hoc networks for IPv4," *RFC 4728*, Feb 2007.
19. Clausen, T., et al., "Optimized link state routing protocol (OLSR)," *RFC 3626*, 2003.
20. Perkins, C. E., and Bhagwat, P., "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers," *ACM SIGCOMM Computer Communications Review*, Vol. 24, No. 4, pp. 234-44, 1994.
21. Zheng, Y., et al., "A mobility and load aware OLSR routing protocol for UAV mobile ad-hoc networks," *International Conference of Information Communications & Technologies (ICT)*, pp. 1-7, 2014.
22. Abid, M. A., and Belghith, A., "Asynchronous locally self-adjusted routing protocol for mobile multi hop ad hoc networks," *International Conference of Computer Systems Applications (AICCSA)*, pp. 1-8, 2010.
23. Belhassen, M., et. al, "Performance evaluation of a cartography enhanced OLSR for mobile multi-hop ad hoc networks," *Wireless Advanced*, pp. 149-155, 2011.
24. Pu C., "Link-quality and traffic-load aware routing for UAV ad hoc networks," *International Conference of Collaboration Internet Computing (CIC)*, pp. 71-79, 2018.
25. Rondinone, M., et al., "Designing a reliable and stable link quality metric for wireless sensor networks," *Workshop on Real-World Wireless Sensor Networks*, pp. 6-10, 2008.
26. Bao, K., et al., "Intelligent software-defined mesh networks with link-failure adaptive traffic balancing," *IEEE Transactions on Cognitive Communication & Networking*, Vol. 4, np. 2, pp. 266-276, 2018.
27. Gerharz, M., et al., "Link stability in mobile wireless ad hoc networks," *IEEE Conference of Local Computer Networks*, pp. 30-39, 2002.
28. Oddi, G., et al., "A proactive link-failure resilient routing protocol for MANETs based on Reinforcement learning," *20th Mediterranean Conference of Control Automation*, pp. 1259-1264, 2012.
29. Guo, Z., et al., "Multi-objective OLSR for proactive routing in MANET with delay, energy, and link-lifetime predictions," *Applied Mathematical Modeling*, Vol. 35, No. 3, pp. 1413-1426, 2011.
30. Su, W., Lee, S. J., and Geria, M., "Mobility prediction and routing in ad hoc wireless networks," *International Journal of Network Management*, No. 1, pp. 3-30, 2001.
31. Li, Z., and Haas, Z. J., "On residual path lifetime in mobile networks," *IEEE Communication Letters*, Vol. 20, No. 3, 2016.
32. Sun, Y., et al., "Delay constraint multipath routing for wireless multimedia ad hoc networks," *International Journal of Communication Systems*, Vol. 29, No. 1, pp. 210-225, 2014.
33. Song, M., et al., "A mobility prediction and delay prediction routing protocol for UAV networks," *International Conference of Wireless Communication & Signal Processing (WCSP)*, pp. 1-6, 2018.
34. Kuiper, E., and Tehrani, S. N., "Mobility models for UAV group reconnaissance applications," *International Conference of Wireless Mobile Communication*, pp. 33-40, 2006.
35. Lakew, D. S., et al., "Routing in flying ad hoc networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, Vol. 22, No. 2, pp. 1071-1120, 2020.
36. Anjum, M. N., and Wang, H., "Mobility modeling and stochastic property analysis of airborne network," *IEEE Transactions on Networks Science & Engineering*, 2019. DOI: [10.1109/TNSE.2019.2921482](https://doi.org/10.1109/TNSE.2019.2921482).

37. Bouachir, O., Abrassart, A., Garcia, F., and Larrieu, N., "A mobility model for UAV ad hoc network," *International Conference of Unmanned Aircraft Systems*, pp. 383-388, 2014.
38. Wang, W., et al., "A novel mobility model based on semi-random circular movement in mobile ad hoc networks," *Information Sciences*, Vol. 180, No. 3, pp. 399-413, 2010.
39. Liang, B., and Haas, Z. J., "Predictive distance-based mobility management for PCS networks," *INFOCOM*, Vol. 3, pp. 1377-1384, 1999.
40. Biomo, J. D. M. M., et al., "An enhanced Gauss-Markov mobility model for simulations of unmanned aerial ad hoc networks," *Wireless Mobile Networking Conference*, pp. 1-8, 2014.
41. Oubbati, O. S., et al., "Routing in flying ad hoc networks: Survey, constraints, and future challenge perspectives," *IEEE Access*, Vol. 7, pp. 81057-81105, 2019.
42. Maza, I., Ollero, A., Casado, E., and Scarlatti, D., "Classification of multi-UAV architectures," in *Handbook of Unmanned Aerial Vehicles*, Springer, pp. 953-975, 2014.
43. Yun, B., Chen, B. M., Lum, K. Y., and Lee, T. H., "Design and implementation of a leader-follower cooperative control system for unmanned helicopters," *Journal of Control Theory Applications*, Vol. 8, No. 1, pp. 61-68, Feb. 2010.
44. LeCun, Y., Bengio, Y., and Hinton, G., "Deep learning," *Nature*, Vol. 521, pp. 436-444, May 2015.
45. Mnih, V., et al., "Playing ARARI with deep reinforcement learning," 2013, *arXiv:1312.5602*. [Online]. Available: <http://arxiv.org/abs/1312.5602>
46. Bruck, J., Gao, J., and Jiang, A., "MAP: Medial axis based geometric routing in sensor networks," *Wireless Networks*, Vol. 13, No. 6, pp. 835-853, Dec. 2007.
47. Fang, Q., et al., "GLIDER gradient landmark-based distributed routing for sensor networks," in *Proceedings of IEEE 24th Annual Joint Conference of IEEE Computer Communication Societies*, Vol. 1, pp. 339-350, Dec. 2005.
48. Cai, K., et al., "Onion- Map: A scalable geometric addressing and routing scheme for 3D sensor networks," *IEEE Transactions on Wireless Communication*, Vol. 14, No. 1, pp. 57-68, Jan. 2015.
49. Jiang, J., and Han, G., "Routing protocols for unmanned aerial vehicles," *IEEE Communication Magazine*, Vol. 56, No. 1, pp. 58-63, Jan. 2018.
50. Yuan, Z., Huang, X., Sun, L., and Jin, J., "Software defined mobile sensor network for micro UAV swarm," in *Proceedings of IEEE International Conference of Control Robot. Eng. (ICCRE)*, pp. 1-4, Apr. 2016.
51. Xiong, F., Li, A., Wang, H., and Tang, L., "An SDN-MQTT based communication system for battlefield UAV swarms," *IEEE Communication Magazine*, Vol. 57, No. 8, pp. 41-47, Aug. 2019.
52. Qi, W., Song, Q., Kong, X., and Guo, L., "A traffic-differentiated routing algorithm in flying ad hoc sensor networks with SDN cluster controllers," *Journal of Franklin Institute*, Vol. 356, No. 2, pp. 766-790, Jan. 2019.
53. Silva, T. D. E., et al., "STFANET: SDN-based topology management for flying ad hoc network," *IEEE Access*, Vol. 7, pp. 173499-173514, 2019.
54. Dai, F., Chen, M., Wei, X., and Wang, H., "Swarm intelligence-inspired autonomous flocking control in UAV networks," *IEEE Access*, Vol. 7, pp. 61786-61796, 2019.
55. Zeng, T., et al., "Wireless communications and control for swarms of cellular-connected UAVs," in *Proceedings of 52nd Asilomar Conference of Signals, System & Computers*, pp. 719-723, Oct. 2018.
56. Azari, M. M., Geraci, G., Garcia-Rodriguez, A., and Pollin, S., "Cellular UAV-to-UAV

- communications,” in *Proceedings of IEEE 30th Annual International Symposium Personal, Indoor Mobile Radio Communication (PIMRC)* pp. 1–7, Sep. 2019.
57. Ho, D.-T., Grotli, E. I., Shimamoto, S., and Johansen, T. A., “Optimal relay path selection and cooperative communication protocol for a swarm of UAVs,” in *Proceedings of IEEE Globecom Workshops*, pp. 1585–1590, Dec. 2012.
 58. Mahmud, I., and Cho, Y.-Z., “Adaptive hello interval in FANET routing protocols for green UAVs,” *IEEE Access*, Vol. 7, pp. 63004–63015, 2019.
 59. Gankhuyag, G., Shrestha, A. P., and Yoo, S.-J., “Robust and reliable predictive routing strategy for flying ad-hoc networks,” *IEEE Access*, Vol. 5, pp. 643–654, 2017.
 60. Wang, F., et al., “Greedy forwarding and limited flooding based routing protocol for UAV flying ad-hoc networks,” in *Proceedings of IEEE International Conference of Electron. Inf. Emergency Communication (ICEIEC)*, pp. 1–4, Jul. 2019.
 61. Jacquet, P., et al., “Optimized link state routing protocol for ad hoc networks,” in *Proceedings of IEEE International Multi Topic Conference of Technol. 21st Century*, pp. 62–68, Dec. 2001.
 62. Rosati, S., Kruzelecki, K., Traynard, L., and Rimoldi, B., “Speed-aware routing for UAV ad-hoc networks,” in *Proceedings of IEEE Globecom Workshops*, pp. 1367–1373, Dec. 2013.
 63. Arafat, M. Y., and Moh, S., “Location-aided delay tolerant routing protocol in UAV networks for post-disaster operation,” *IEEE Access*, Vol. 6, pp. 59891–59906, 2018.
 64. Yin, C., Xiao, Z., Cao, X., Xi, X., Yang, P., and Wu, D., “Enhanced routing protocol for fast flying UAV network,” in *Proceedings of IEEE International Conference of Communication System (ICCS)*, pp. 1–6, Dec. 2016.
 65. Pu, C., “Stochastic packet forwarding algorithm in flying ad hoc networks,” in *Proceedings of IEEE Military Communications Conference (MILCOM)*, pp. 490–495, Nov. 2019.
 66. Toorchi, N., et al., “Volcano routing: A multi-pipe high-throughput routing protocol with hole avoidance for multi-beam directional mesh networks,” *IEEE Transactions on Mobile Computation*, Vol. 19, No. 12, pp. 2981–2996, Dec. 2020.
 67. Smith, P., Hunjet, R., Aleti, A., and Barca, J. C., “Adaptive data transfer methods via policy evolution for UAV swarms,” in *Proceedings of 27th International Telecommunication Networks Applications Conference (ITNAC)*, pp. 1–8, Nov. 2017.
 68. Toorchi, N., Chakareski, J., and Mastronarde, N., “Fast and low-complexity reinforcement learning for delay-sensitive energy harvesting wireless visual sensing systems,” in *Proceedings of IEEE International Conference of Image Processing*, pp. 1804–1808, Sep. 2016.
 69. Koushik, A. M., Hu, F., and Kumar, S., “Deep Q-learning-based node positioning for throughput-optimal communications in dynamic UAV swarm network,” *IEEE Transactions on Cognitive Communication Networks*, Vol. 5, No. 3, pp. 554–566, Sep. 2019.
 70. Shapira, M., and Rappoport, A., “Shape blending using the star-skeleton representation,” *IEEE Computer Graphics Applications*, Vol. 15, No. 2, pp. 44–50, Mar. 1995.
 71. Porteus, E. L., *Foundations of Stochastic Inventory Theory*, Stanford University Press, Palo Alto, CA, 2002.
 72. Puterman, M. L., *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, Wiley, Hoboken, NJ, 2014.
 73. Sutton, R. S., and Barto, A. G., *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 2018.
 74. Goodfellow, I., Bengio, Y., and Courville, A., *Deep Learning*, MIT Press, Cambridge, MA, 2016.

75. Debowski, B., Spachos, P., and Areibi, S., "Q-learning enhanced gradient based routing for balancing energy consumption in WSNs," in *Proceedings of IEEE 21st International Workshop on Computer Aided Modeling Design & Communication Links Networks (CAMAD)*, pp. 18–23, Oct. 2016.
76. Wallace, K., Cisco IP Telephony Flash Cards: Weighted Random Early Detection (WRED), Cisco Press, Indianapolis, IN, 2004.
77. Dai, F., and Wu, J., "Proactive route maintenance in wireless Ad Hoc networks," in *IEEE International Conference of Communications*, Vol. 2, pp. 1236–1240, 2005.
78. Abushiba, W., and Johnson, P., "Performance comparison of reactive routing protocols for Ad Hoc network," in *4th International Conference of e-Technologies and Networks for Development (ICeND)*, pp. 1–5, 2015.
79. Garnepudi, P., Damarla, T., Gaddipati, J., and Veeraiah, D., "Proactive, reactive and hybrid multicast routing protocols for wireless mesh networks," in *IEEE International Conference of Computational Intelligence and Computing Research*, pp. 1–7, 2013.
80. Chhieng, V. M., Choi, R. H., and Wong, R. K., "Efficient geometric routing in ad-hoc wireless networks," in *4th IEEE Consumer Communications and Networking Conference*, pp. 259–263, 2007.
81. He, Y., et al., "Routing protocol for complex three-dimensional vehicular Ad Hoc networks," in *Connected Vehicles and Expo (ICCVE)*, pp. 739–744, 2014.
82. Lin, Q., Li, C., Wang, X., and Zhu, L., "A three-dimensional scenario oriented routing protocol in vehicular Ad Hoc networks," in *IEEE 77th Vehicular Technology Conference (VTC Spring)*, pp. 1–5, 2013.
83. Brust, M. R., and Strimbu, B. M., "A networked swarm model for UAV deployment in the assessment of forest environments," in *IEEE 10th International Conference of Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pp. 1–6, 2015.
84. de Souza, B. J. O., and Endler, M., "Coordinating movement within swarms of UAVs through mobile networks," in *IEEE International Conference of Pervasive Computing and Communication Workshops (PerCom Workshops)*, pp. 154–159, 2015.
85. Champion, M., Ranganathan, P., and Faruque, S., "UAV swarm communication and control architectures: a review," *Unmanned Vehicle Systems*, Vol. 7, No. 2, pp. 93–106, 2018.
86. Qin, X., Sang, X., Zhu, S., and Cheng, S., "Line-skeleton extraction of 3D meshes based on geometry segmentation," in *1st ACIS International Symposium of Cryptography, and Network Security, Data Mining and Knowledge Discovery, E-Commerce and Its Applications, and Embedded Systems*, 2010, pp. 354–357.
87. Au, O. K.-C., et al., "Skeleton extraction by mesh contraction," in *ACM Transactions on Graphics (TOG)*, Vol. 27, No. 3, p. 44, 2008.
88. Zhu, D., et al., "The extraction and evaluation of skeleton in sensor networks," in *IEEE 9th International Conference of Mobile Ad-hoc and Sensor Networks*, pp. 47–52, 2013.
89. Zhou, H., et al., "Distributed algorithms for bottleneck identification and segmentation in 3D wireless sensor networks," in *SECON*, pp. 494–502, 2011.
90. Xia, S., Wu, H., and Jin, M., "Trace-routing in 3D wireless sensor networks: a deterministic approach with constant overhead," in *Proceedings of 15th ACM International Symposium of Mobile Ad Hoc Networking and Computing*, pp. 357–366, 2014.
91. Huang, H., et al., "Three-dimensional geographic routing in wireless mobile Ad Hoc and sensor networks," *IEEE Network*, Vol. 30, No. 2, pp. 82–90, 2016.
92. Su, H., Wang, Y., and Fang, D., "An efficient geographic surface routing algorithm in 3D Ad

- Hoc networks,” in *5th IEEE International Conference of Pervasive Computing and Applications*, pp. 138–144, 2010.
93. Xia, S., Jin, M., Wu, H., and Zhou, H., “Bubble routing: A scalable algorithm with guaranteed delivery in 3D sensor networks,” in *9th Annual IEEE Conference of Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, pp. 245–253, 2012.
 94. Mir, Z. H., Jung, W.-S., and Ko, Y.-B., “Continuous neighbor discovery protocol in wireless ad hoc networks with sectored-antennas,” in *IEEE 29th International Conference of Advanced Information Networking and Applications*, pp. 54–61, 2015.
 95. Jiang, H., Zhang, S., Tan, G., and Wang, C., “CABET: Connectivity-based boundary extraction of large-scale 3D sensor networks,” in *IEEE INFOCOM*, pp. 784–792, 2011.
 96. Zhou, H., Wu, H., and Jin, M., “A robust boundary detection algorithm based on connectivity only for 3D wireless sensor networks,” in *INFOCOM*, pp. 1602–1610, 2012.
 97. Shirahama, J., and Ohtsuki, T., “RSS-based localization in environments with different path loss exponent for each link,” in *IEEE Vehicular Technology Communication (VTC Spring)*, pp. 1509–1513, 2008.
 98. Zhao, Y., et al., “Cut-and-sew: a distributed autonomous localization algorithm for 3D surface wireless sensor networks,” in *Proceedings of 14th ACM International Symposium Mobile Ad Hoc Networking and Computing*, pp. 69–78, 2013.
 99. Shang, Y., and Ruml, W., “Improved MDS-based localization,” in *IEEE INFOCOM*, Vol. 4, pp. 2640–2651, 2004.
 100. Jiang, H., et al., “Connectivity-based segmentation in large-scale 2-D/3-D sensor networks: algorithm and applications,” *IEEE/ACM Transactions on Networking*, Vol. 23, No. 1, pp. 15–27, 2013.
 101. Huang, H., et al., “ L_1 -medial skeleton of point cloud,” *ACM Transactions on Graphics*, Vol. 32, No. 4, pp. 65–1, 2013.
 102. Ma, W.-C., Wu, F.-C., and Ouhyoung, M., “Skeleton extraction of 3D objects with radial basis functions,” in *IEEE Shape Modeling International*, pp. 207–215, 2003.
 103. Wang, W.-T., and Huang, H.-C., “Regularized principal component analysis for spatial data,” *Computational and Graphical Statistics*, Vol. 26, No. 1, pp. 14–25, 2017.
 104. Murphy, T. I., “Line spacing in LaTeX documents,” URL: https://diffgeom.subwiki.org/wiki/Injectivity_radius, accessed April 4, 2010.
 105. Chakeres, I. D., and Belding-Royer, E. M., “AODV routing protocol implementation design,” in *24th International Conference of Distributed Computing Systems Workshops*, pp. 698–703, 2004.
 106. Eun, J. S., and Jung, H., “An efficient multipath generation method for hierarchical network model,” in *International Conference of Information and Communication Technology Convergence (ICTC)*, pp. 559–562, 2014.
 107. Deng, C. and Zheng, P., “A new hidden markov model with application to classification,” in *6th World Congress on Intelligent Control and Automation*, Vol. 2, pp. 5882–5886, 2006.
 108. Qi, J., et al., “3-ent (resilient, intelligent, and efficient) medium access control for full-duplex, jamming-aware, directional airborne networks,” *Elsevier Journal of Computer Networks*, Vol. 129, pp. 251–260, 2017.
 109. Wang, X. Z., “The comparison of three algorithms in Shortest Path Issue,” in *Journal of Physics Conference Series*, Vol. 1087, No. 2, 2018.
 110. Zhang, W., Chen, H., jiang, C., and Zhu, L., “Improvement and experimental evaluation Bellman-Ford algorithm,” in *International Conference of Advanced ICT and Education*

- (ICAICTE-13). Atlantis Press, 2013.
111. Zhao, L., and Zhao, J., "Comparison study of three shortest path algorithm," in *IEEE International Conference of Computer Technology, Electronics and Communication (ICCTEC)*, pp. 748–751, 2017.
 112. Barbehenn, M., "A note on the complexity of Dijkstra's algorithm for graphs with weighted vertices," *IEEE Transactions on Computers*, Vol. 47, No. 2, p.263, 1998.
 113. Mao, Q., Hu, F., Bao, K., and Kumar, S., "Adaptive batch coding: A balanced congestion control strategy for multi-beam antenna networks," *IEEE Transactions on Vehicular Technology*, Vol. 67, No. 4, pp. 3575–3585, 2017.
 114. Silver, D., et al., "Mastering the game of go with deep neural networks and tree search," *Nature*, Vol. 529, No. 7587, p. 484, 2016.
 115. Helmstaedter, M., et al., "Connectomic reconstruction of the inner plexiform layer in the mouse retina," *Nature*, Vol. 500, No. 7461, p. 168, 2013.
 116. Zhao, N., et al., "Interference alignment and its applications: a survey of research issues and challenges," *IEEE Communication Surveys & Tutorials*, Vol. 18, No. 3, pp. 1779–1803, 2016.
 117. Han, G., Xiao, L., and Poor, H. V., "Two-dimensional anti-jamming communication based on deep reinforcement learning," in *IEEE International Conference of Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2087–2091, 2017.
 118. Peng, S., et al., "Modulation classification using convolutional neural network based deep learning model," in *IEEE 26th Wireless and Optical Communication Conference (WOCC)*, pp. 1–5, 2017.
 119. Nachmani, E., Be'ery, Y., and Burshtein, D., "Learning to decode linear codes using deep learning," in *IEEE 54th Annual Allerton Conference of Communication, Control, and Computing (Allerton)*, pp. 341–346, 2016.
 120. Challita, U., Dong, L., and Saad, W., "Deep learning for proactive resource allocation in LTE-U networks," in *23rd European Wireless Conference*, pp. 1-6, 2017.
 121. Wang, J., et al., "Spatiotemporal modeling and prediction in cellular networks: A big data enabled deep learning approach," *IEEE INFOCOM*, pp. 1–9, 2017.
 122. Liu, L., et al., "Deep learning based optimization in wireless network," *IEEE International Communications Conference (ICC)*, pp. 1–6, 2017.
 123. Hu, T., and Fei, Y., "QELAR: A machine-learning-based adaptive routing protocol for energy-efficient and lifetime-extended underwater sensor networks," *IEEE Transactions on Mobile Computing*, Vol. 9, No. 6, pp. 796–809, 2010.
 124. Kato, N., et al., "The deep learning vision for heterogeneous network traffic control: Proposal, challenges, and future perspective," *IEEE Wireless Communication*, Vol. 24, No. 3, pp. 146–153, 2016.
 125. Mao, B., et al., "Routing or computing? the paradigm shift towards intelligent computer network packet transmission based on deep learning," *IEEE Transactions on Computers*, Vol. 66, No. 11, pp. 1946–1960, 2017.
 126. Gwon, Y. L., and Kung, H., "Inferring origin flow patterns in wi-fi with deep learning," in *11th International Conference of Autonomic Computing (ICAC)*, pp. 73–83, 2014.
 127. Tang, T. A., et al., "Deep learning approach for network intrusion detection in software defined networking," in *IEEE International Conference of Wireless Networks and Mobile Communication (WINCOM)*, pp. 258–263, 2016.
 128. Kang, M.-J., and Kang, J.-W., "Intrusion detection system using deep neural network for in-vehicle network security," *PloS one*, Vol. 11, No. 6, p. e0155781, 2016.

129. Jay, N., et al., "A deep reinforcement learning perspective on internet congestion control," in *International Conference on Machine Learning*, pp. 3050–3059, 2019.
130. Li, W., Zhou, F., Chowdhury, K. R., and Meleis, W. M., "QTCP: Adaptive congestion control with reinforcement learning," *IEEE Transactions on Network Science and Engineering*, 2018.
131. Bachl, M., Zseby, T., and Fabini, J., "RAX: Deep reinforcement learning for congestion control," *IEEE International Communications Conference (ICC)*, pp. 1–6, 2019.
132. Peng, Q., Walid, A., Hwang, J., and Low, H., "Multipath TCP: Analysis, Design, and Implementation," *IEEE/ACM Networking*, Vol. 24, pp 596-609, Feb 2016.
133. Javaid, A., "Understanding Dijkstra's algorithm," Available at SSRN: <https://ssrn.com/abstract=2340905>, April 2013.
134. Hu, P., Dellar, M., and Hu, C., "Task Scheduling on Flow Networks with Temporal Uncertainty," *IEEE Symposium on Foundations of Computational Intelligence*, Honolulu, HI, pp. 128-135, 2007.
135. Ahrenholz, J., Danilov, C., Henderson, T. R., and Kim, J. H., "CORE: A real-time network emulator," *IEEE Military Communications Conference*, San Diego, CA, pp. 1-7, 2008.
136. Ahrenholz, J., Goff, T., and Adamson, B., "Integration of the CORE and EMANE Network Emulators," *IEEE Military Communications Conference*, Baltimore, MD, pp. 1870-1875, 2011.
137. Ouacha, A., et al., "OLSR protocol enhancement through mobility integration," *10th IEEE International Conference of Networking, Sensing and Control (ICNSC)*, Evry, France, pp. 17-22, 2013.
138. Romanik, J., Kraśniewski, A., and Golan, E., "RESA-OLSR: RESources-aware OLSR-based routing mechanism for mobile ad-hoc networks," *International Conference of Military Communications and Information Systems (ICMCIS)*, Brussels, Belgium, pp. 1-6, 2016.
139. Wang, Q., Xu, R., and Yuan, D., "TCP rate control mechanism based on channel utilization in mobile adhoc networks," *IEEE 13th International Conference of Communication Technology*, Jinan, China, pp. 572-576, 2011. doi: 10.1109/ICCT.2011.6157941.
140. Gajjar, S., and Gupta, H. M., "Improving performance of adhoc TCP in mobile adhoc networks," *Annual IEEE India Conference*, Kanpur, India, pp. 144-147, 2008. doi: 10.1109/INDCON.2008.4768816.
141. Tripathy, T., and Mohapatra, S., "Comparative analysis of physical propagation model in modified versions of OLSR Ad-Hoc routing protocol," *Annual IEEE India Conference (INDICON)*, Mumbai, India, pp. 1-6, 2013. doi: 10.1109/INDCON.2013.6726059.
142. Cui, Q., Liu, P., Wang, J., and Yu, J., "Brief analysis of drone swarms communication," *IEEE International Conference of Unmanned Systems (ICUS)*, Beijing, China, pp. 463-466, 2017. doi: 10.1109/ICUS.2017.8278390.
143. Yin, J., Wang, L., Han, C., and Yang, Y., "NC-OLSR: A network coding based OLSR multipath transmission scheme for FANETs," *4th International Conference of Systems and Informatics (ICSAI)*, Hangzhou, China, pp. 1007-1012, 2017. doi: 10.1109/ICSAI.2017.8248433.
144. Soderi, S., et al., "Emulation of secure Wi-Fi communication: A performance gap analysis against a virtual test-bed," *13th International Conference of ITS Telecommunications*, Tampere, pp. 226-231, 2013.
145. Wang, S., et al., "Emulation-based study of dynamic service placement in mobile micro-clouds," *IEEE Military Communications Conference*, Tampa, FL, pp. 1046-1051, 2015.

146. Lass, R. N., et al., "An evaluation of serverless group chat," *IEEE Military Communications Conference*, Baltimore, MD, pp. 1639-1644, 2011.
147. Labonte, G., "Self-Organization of Formation and Swarms," in *Defense R&D Canada Contract Report*, Jan 2010.
148. Neto, E. P., and Callou, G., "An Approach Based on Ford-Fulkerson Algorithm to Optimize Network Bandwidth Usage," *Brazilian Symposium of Computing Systems Engineering (SBESC)*, Foz do Iguacu, Brazil, pp. 76-79, 2015.
149. Aydogdu, C., and Karasan, E., "Goodput and throughput comparison of single-hop and multi-hop routing for IEEE 802.11 DCF-based wireless networks under hidden terminal existence," *Wireless Communication & Mobile Computation*, Vol. 16, No. 9, pp. 1078–1094, June 2016.
150. Sheth, A., et al., "Packet Loss Characterization in WiFi-Based Long Distance Networks," *IEEE INFOCOM*, Barcelona, pp. 312-320, 2007.

LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

3D-TR	3D Transformative Routing
ABC	Adaptive Batch Coding
ACK	Acknowledgement
AI	Artificial Intelligence
AN	Airborne Network
AODV	Ad-Hoc On-Demand Distance Vector
BO	Buffer Occupancy
CORE	Common Open Research Emulator
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
DL	Deep Learning
DRL	Deep Reinforcement Learning
EKA	Edmonds-Karp Algorithm
EMANE	Extendable Mobile Ad-hoc Network Emulator
FGQPA	Fountain-code based Greedy Queue and Position Assisted
HMM	Hidden Markov Model
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
LLT	Link Lifetime
LTA-OLSR	Link-quality and Traffic-load Aware Optimized Link State Routing
LTE	Long Term Evolution
MAC	Medium Access Control
MANET	Mobile Ad-hoc Network
MAP	Medial Axis-based Naming and Routing Protocol
MDP	Markov Decision Process
ML-OLSR	Multi-Level Optimized Link State Routing
MP-UDP	Multipath User Datagram Protocol
MPR	Multi-Point Relay
MPSB	Multipath, Single-Beam
MRMP	Multi-Region, Multipath
OLSR	Optimized Link State Routing

OSPF	Open Shortest Path First
OSPF-MDR	Open Shortest Path First-MANET Designated Router
PDR	Packet Delivery Ratio
PLR	Packet Loss Rate
P-OLSR	Position Based - Optimized Link State Routing
PSO-GLFR	Particle Swarm Optimization - Greedy Limited Forwarding Routing
QoA	Quality of Autonomy
QoS	Quality of Service
ReLU	Rectified Linear Units
RARP	Reverse Address Resolution Protocol
RF	Radio Frequency
RH	Route Handoff
RL	Reinforcement Learning
RLT	Route Lifetime
RREP	Route Reply
RREQ	Route Request
RSS	Received Signal Strength
SINR	Signal to Interference plus Noise Ratio
SPZ	Skeleton Prediction Zone
SSR	Skeleton-based Swarm Routing
STFAN ET	SDN-Based Topology Management for Flying Ad Hoc Network
TC	Topology Control
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TSDP	Two-Step Discarding Policy
TSS	Trunk-Stem-Stemlet
TTL	Time-to-Live
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
USN	UAV Swarm Network