SNAP : A Software-Defined & Named-Data Oriented Publish-Subscribe Framework for Emerging Wireless Application Systems

---

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Computer Science

---

by
Manveen Kaur
July 2022

Accepted by:
Dr. Abolfazi Razi, Committee Chair
Dr. Long Cheng, Co-Committee Chair
Dr. Beshah Ayalew
Dr. Jim Martin
Dr. Jacob Sorber

Abstract

The evolution of Cyber-Physical Systems (CPSs) has given rise to an emergent class of CPSs defined by ad-hoc wireless connectivity, mobility, and resource constraints in computation, memory, communications, and battery power. These systems are expected to fulfill essential roles in critical infrastructure sectors. Vehicular Ad-Hoc Network (VANET) and a swarm of Unmanned Aerial Vehicles (UAV swarm) are examples of such systems. The significant utility of these systems, coupled with their economic viability, is a crucial indicator of their anticipated growth in the future. Typically, the tasks assigned to these systems have strict Quality-of-Service (QoS) requirements and require sensing, perception, and analysis of a substantial amount of data. To fulfill these QoS requirements, the system requires network connectivity, data dissemination, and data analysis methods that can operate well within a system's limitations. Traditional Internet protocols and methods for network connectivity and data dissemination are typically designed for well-engineering cyber systems and do not comprehensively support this new breed of emerging systems. The imminent growth of these CPSs presents an opportunity to develop broadly applicable methods that can meet the stated system requirements for a diverse range of systems and integrate these systems with the Internet. These methods could potentially be standardized to achieve interoperability among various systems of the future.

This work presents a solution that can fulfill the communication and data dissemination requirements of a broad class of emergent CPSs. The two main contributions of this work are the Application System (APPSYS) system abstraction, and a complementary communications framework called the **S**oftware-Defined **NA**med-data enabled **P**ublish-Subscribe (SNAP) communication framework. An APPSYS is a new breed of Internet application representing the mobile and resource-constrained CPSs supporting data-intensive and QoS-sensitive safety-critical tasks, referred to as the APPSYS's *mission*. The functioning of the APPSYS is closely aligned with the needs of the

mission. The standard APPSYS architecture is distributed and partitions the system into multiple clusters where each cluster is a hierarchical sub-network. The SNAP communication framework within the APPSYS utilized principles of Information-Centric Networking (ICN) through the publish-subscribe communication paradigm. It further extends the role of brokers within the publish-subscribe paradigm to create a distributed software-defined control plane. The SNAP framework leverages the APPSYS design characteristics to provide flexible and robust communication and dynamic and distributed control-plane decision-making that successfully allows the APPSYS to meet the communication requirements of data-oriented and QoS-sensitive missions. In this work, we present the design, implementation, and performance evaluation of an APPSYS through an exemplar UAV swarm APPSYS. We evaluate the benefits offered by the APPSYS design and the SNAP communication framework in meeting the dynamically changed requirements of a data-intensive and QoS-sensitive Coordinated Search and Tracking (CSAT) mission operating in a UAV swarm APPSYS on the battlefield. Results from the performance evaluation demonstrate that the UAV swarm APPSYS successfully monitors and mitigates network impairment impacting a mission's QoS to support the mission's QoS requirements.

Dedication

To my partner, with whom life's milestones have more meaning; my parents and grandparents, whose many sacrifices made sure that I have opportunities to excel in life; and my sister, nephew, and niece, my inexhaustible source of happiness.

Acknowledgments

My sincere gratitude goes to my adviser, Dr. Jim Martin, for his invaluable mentorship that has helped me grow as a researcher and a person. Dr. Martin's vision and guidance have helped me produce meaningful work that I am proud to present. I want to thank my committee chair and co-chair, Dr. Abolfazi Razi, and Dr. Long Cheng, for their immense help and support during the final phase of my dissertation work. I also want to thank my committee members, Dr. Jacob Sorber and Dr. Beshah Ayalew, for being on my committee and guiding my work. I am incredibly grateful for Dr. Sorber's time and advice, which has helped me grow professionally. My special thanks to Dr. Rahul Amin and Dr. James Westall for continued guidance, support, and critical insight that has always helped me improve my work. Finally, I want to extend my gratitude to the past and present members of my research group, each of whom has been an essential part of my journey.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet is going through radical changes. Examples include the requirement for low latency networking, the integration of advanced wireless technology, and the continued integration of evolving forms of Cyber-Physical Systems (CPS). The needs of emergent CPSs, such as connecting resource-constrained devices operating under diverse environmental conditions and incorporating data-oriented network communication with low latency using available Internet protocols, are primary motivators for these changes. Changes in the Internet are more particularly motivated by a class of intelligent, reliable, and communications-oriented CPSs that support society's critical infrastructure sectors. These systems are characterized by a distributed group of mobile nodes, resource-constrained in computation, memory, communications, and battery power, and likely to support shared tasks cooperatively. Tasks assigned to these CPSs vary in complexity and may require the exchange of substantial amounts of data and distributed and cooperative operations among nodes in the CPS. Further, these tasks are likely to be safety-critical with strict Quality-of-Service (QoS) requirements.

Vehicular Ad-Hoc Networks (VANETs) and groups of Unmanned Aerial Vehicles (UAVs) called UAV swarms are examples of such systems. In recent literature, VANETs and UAV swarms have been utilized within numerous critical infrastructural sectors, including transportation, national security, agriculture, and telecommunication. VANETs augment transportation infrastructure by incorporating a range of driver assistance and road safety missions such as Cooperative Adaptive Cruise Control (CACC) and Lane Change Assistance [54]. UAV swarms comprising nodes with multi-access edge computing (MEC) facilities are used to extend 5G cellular telecommunication coverage, pro-

vide emergency back-haul communication links, and enable low-latency edge-computing [40, 47]. UAV swarms equipped with suitable video-sensing capabilities are used for surveying agricultural land and conducting high-precision crop monitoring [51]. On the battlefield, VANETs comprising semi-autonomous vehicles are used for efficient routing of military logistical vehicles, and UAV swarms are used for tactical missions such as reconnaissance, surveillance, and target searching and tracking [37, 84].

The introduction of low-cost Internet-of-Things kits [1], Unmanned Aerial Vehicles (UAVs) or drones [4], and embedded platforms [15] further creates nurturing economic conditions that contribute to the increasing demand and corresponding growth of these systems. A recent Gartner study estimates that the spending of governments worldwide on Internet-of-Things endpoints spending was $17.5 billion in 2021. This number is forecasted to increase by 22% in 2022 [14]. There are also parallel developments in domains like Machine Learning and Edge and Cloud computing that aim to improve the ability of resource-constrained systems to leverage on-board and network-wide computational resources [18, 17]. Together, these factors can enhance the CPSs' capability to support complex tasks and increase their overall utility. On the other hand, the current lack of suitable Internet protocols for supporting the network connectivity and communication necessary for these CPSs limits the usability and growth of these systems. While the current Internet has evolved to be quite efficient for traditional flow-oriented applications, the standards-based Internet protocols and methods are not designed to comprehensively support CPSs with resource constraints, intermittent connectivity arising from mobility, and limited access to reliable back-end infrastructure depending on the CPS's environmental and operating conditions. For example, WiFi does not sufficiently accommodate the extreme requirements of fleets of short-range UAVs.

The present work is motivated by the factors that limit the growth of current and emergent communication-oriented CPSs. The central research objective of this work is to provide a solution that can meet the communication and data dissemination requirements of communication-oriented CPSs supporting QoS-dependent safety-critical tasks in critical infrastructure domains. The primary contributions of this work are the introduction of a system abstraction called the Application System (APPSYS) and a complementary communications framework called the **S**oftware-Defined **NA**med-data enabled **P**ublish-subscribe (SNAP) framework that supports communication in the APPSYS. An APPSYS is a new breed of Internet applications that represents an emergent generation of mobile and safety-critical CPSs that are being utilized by critical infrastructure sectors. It is not a

standard application of the Internet but a system comprising hardware and software tightly coupled to a specified objective, referred to as the APPSYS's *mission*. It differs from current Internet applications primarily due to its ad-hoc nature and 'out in the wild' operations. An APPSYS forms in an ad-hoc manner and dynamically establishes its own communications and computation environment to provide a distributed CPS that carries out a specific mission. Nodes in an APPSYS may have a varied range of integrated sensing, computation, control, networking, and mobility capabilities. However, most nodes are expected to be constrained with respect to these capabilities. The general APPSYS architecture is distributed and partitions available nodes into multiple non-overlapping clusters capable of inter-cluster communication through selected nodes. Nodes in each cluster further form hierarchical sub-networks with the aforementioned selected nodes at the top-level.

An APPSYS is expected to support autonomous or semi-autonomous missions involving distributed and coordinated operations, node and system mobility, and varying degrees of access to backend infrastructure. For example, a UAV swarm APPSYS on a remote battlefield may not have access to reliable backend infrastructure and require fully autonomous operations. This type of mission maps to an growing class of data-oriented and compute-intensive applications that emergent CPSs are required to support [87]. A mission conducted by an APPSYS usually entails data sensing, perception, and analysis conducted in a distributed manner to accommodate resource constraints. Further, the mission is expected to be safety-critical with strict Quality-of-Service (QoS) requirements such as strict reliability or latency bounds on mission-related data exchanges between nodes. APPSYS characteristics like node mobility which may cause intermittent connectivity, link outages, and network disruptions present challenges to meeting the mission QoS. Therefore, the success on the mission relies on two critical factors: robust and adaptable communication methods that can operate well within the confines of an APPSYS's limitations; and, underlying system-level support for effective dissemination of substantial amount of mission data subject to the system state, mission state, and the mission's QoS requirements. In this work, we use the term *data services* to refer to the system-level data handling measures such as management of the mission's message transmission rate and aggregating mission messages to reduce system traffic.

The APPSYS utilizes the proposed **S**oftware-Defined **NA**med-data enabled **P**ublish-subscribe (SNAP) communication framework to provide flexible communication that can serve a mission's QoS requirements in an APPSYS. The primary enabling technologies that the SNAP framework uti-

lizes are Information-Centric Networking (ICN) and software-defined management of the APPSYS's nodes' control planes. ICN can offer named-data based resilient communication in ad-hoc networks facing intermittent network connectivity issues [21]. The ICN implementation in the SNAP framework uses a named-data based publish-subscribe communication paradigm. Communication is carried out through software brokers provisioned at each node. However, nodes at the top-level of each cluster's hierarchy carry specialized brokers with higher capabilities. The SNAP framework organizes top-level brokers from all clusters into a communication overlay that inter-connects the various APPSYS clusters. Thus, each cluster has a hierarchical distributed control plane. The communication overlay utilized rich multi-path mesh connectivity for robust and efficient forwarding of mission-related data.

Further, the SNAP framework extends the traditional ICN broker capabilities by separating the broker's control and forwarding plane and utilizing software-defined management of the broker's control plane. Brokers include abstractions allowing them to interact with the mission software and the underlying system. Therefore, the brokers are aware of the mission's critical QoS requirements and whether these requirements are being met at any given point during the mission. By coupling mission-related awareness with an understanding of the underlying system conditions, brokers dynamically determine control-plane decisions for mission-related data handling and forwarding. These decisions include connectivity decisions such as choosing the most suitable communication backhaul for a specific mission's requirements at any given point, data dissemination decisions such as critical path selection, and data-handling decisions such as merging or aggregating mission data at the top-level brokers within a cluster before forwarding. The SNAP framework also extends brokers to implement buffers for local data caching as a disruption-tolerant measure in the event of link outages. Overall, the SNAP framework allows the APPSYS to flexibly and strategically manage the mission's communication and data dissemination requirements in response to dynamically changing mission requirements and underlying system conditions.

Reinterpreting emergent CPSs like UAV swarms and VANETs as APPSYSs enables these resource-constrained systems to be able to successfully meet the dynamic demands of a wide range of data-intensive and QoS-sensitive missions. Additionally, the adherence of diverse CPSs to a common underlying architecture and communication framework provides numerous benefits. First, a common design can be extended to support interoperability among multiple APPSYSs that may need to cooperate in a joint mission—for example, utilizing UAV swarms to guide a rescue-service VANET

along available routes in a disaster mismanagement scenario [66]. Support for interoperability is a critical factor in the future evolution and management of CPSs. Second, a common APPSYS design can encourage the development of standardized solutions that can be beneficially applied to different systems. In recent works, there are numerous solutions to common CPS concerns like improving system connectivity or meeting the QoS requirements relevant to a specific task. These solutions leverage elements like adopting advanced wireless access methods, utilizing upcoming technologies like Software-Defined Networking (SDN) or Information-Centric Networking (ICN) or extending existing protocols to meet a newer set of requirements. While these elements apply to a wide range of CPSs, these solutions are typically tied to one specific CPS and can not be directly applied to other CPSs.

In this work, we illustrate the APPSYS concept through the exemplar CPS of a UAV swarm. A UAV swarm serves important roles in many critical domains, especially on the battlefield, where it can conduct reconnaissance, surveillance, and target search and tracking missions. The missions supported by a UAV swarm are safety-critical, data-intensive, and QoS-sensitive. Therefore, the UAV swarm serves as an excellent candidate to explore the benefits offered by the APPSYS design and the SNAP communication framework. We specifically utilize a use-case of a UAV swarm APPSYS conducting a Coordinated Search and Tracking (CSAT) mission on the battlefield to illustrate the concepts presented in this work. A CSAT mission utilizes the integrated sensing and compute capabilities of participating UAVs within a UAV swarm to locate evasive targets of interest in the desired area [81].

We also present a performance-evaluation study that evaluates the ability of a UAV swarm APPSYS utilizing the SNAP framework to support a data-intensive and QoS-sensitive CSAT mission, especially during network and system conditions that cause severe performance degradation and traditional communication methods. Examples of traditional communication methods include UDP-based unicast, broadcast, and common implementations of the publish-subscribe group-communication method. Our prior performance-evaluation in UAV swarms and VANETs using these communication methods demonstrates that the performance of data-intensive operations degrades significantly as aggregate traffic loads lead to high utilization of system resources. The lack of suitable network impairment detection and mitigation measures further exacerbates the performance impact. In comparison, the distributed control plane in the UAV swarm APPSYS is expected to monitor and mitigate network impairment conditions to support a mission's QoS requirements. The

study is conducted using a hardware testbed representing a heterogeneous and resource-constrained UAV swarm APPSYS. We implement relevant components of the SNAP communication framework, i.e., named-data forwarding and extensions that support control plane decision-making to support communication in the testbed. Further, we develop a form of the CSAT mission through which this study is conducted. Results from our evaluate demonstrate that control plane decision-making and data services effectively mitigate conditions causing network impairment to succesfully support a mission's QoS requirements. Overall, the research contributions of the present work include:

1. Study and characterization of existing CPSs to develop the initial APPSYS architecture and system model that can facilitate the development of future APPSYSs.

2. Design and implementation of the SNAP framework and a critical set of mission-related data services that can be utilized to meet the QoS requirements of a data-intensive and QoS-sensitive mission being conducted in a resource-constrained and mobile UAV swarm APPSYS.

3. Design and implementation of a distributed CSAT mission that can be integrated with the APPSYS architecture. The CSAT mission application software is utilized for the performance evaluation study.

4. A performance evaluation study which illustrates the benefits of the APPSYS architecture and the SNAP framework in meeting the QoS requirements of the CSAT mission in a UAV swarm APPSYS.

1.1 Organization

- Chapter 2 summarizes the prevalent system design models and emergent network and communication methods being utilized by two well-known CPSs, UAV swarms and VANETs. The APPSYS architecture and the SNAP communication framework derive functional elements from some of the studied works. We also provide overviews of the enabling technologies that are directly or indirectly applied within the SNAP communication framework.

- Chapter 3 presents the APPSYS architecture through the example of a UAV swarm APPSYS. In this chapter, we also present results from our previous performance evaluation studies within VANETs and UAV swarms that demonstrate the performance issues faced by these systems

with standard communication methods. The lessons from prior studies guide elements of the APPSYS design and the SNAP communication framework.

- Chapter 4 presents the SNAP communication framework and discusses how name-based forwarding and the distributed control plane are utilized within this framework. We also discuss the current implementation of the SNAP framework that is used for the performance evaluation in Chapter 5.

- Chapter 5 presents the performance evaluation study conducted as part of this work. The design and implementation of the CSAT mission application software used in the present work is discussed in this chapter.

- Chapter 6 summarizes this work's contributions and discusses the scope of future work.

Chapter 2

Background & Related Work

To characterize an APPSYS and develop an initial APPSYS architecture, we closely reviewed the current state of development of two well-known CPSs, UAV swarms and VANETs, that are currently the focus of numerous ongoing research efforts. This chapter summarizes the prevalent system design models and emergent network and communication methods being utilized in these systems. The APPSYS architecture and the SNAP communication framework derive functional elements from some of the studied works. While the conclusions drawn in this chapter are primarily derived from these two systems, they apply to a comprehensive set of mobile and resource-constrained APPSYSs. For example, these conclusions can broadly be applied to mobile healthcare systems comprising wearable devices, smartphones, and health monitors offering preventive or assistive care, and CPSs that leverage users' cellular devices for crowdsensing [42, 45]

2.1 System Characteristics

The primary components of a mobile CPS are one or more groups of mobile wireless nodes, called clusters. The system typically connects to backend infrastructure, which may be located in the cloud, edge, or a physical data center. This backend could be dedicated to either a single CPS, e.g., dedicated Road-Side Units (RSU) utilized in VANETs, or serve multiple CPSs through servers hosted in the cloud or on-premise in a data center. The backend can be considered the computational and storage extension of the system. The system may offload intensive computations or store historical data relevant to the application at the backend. The system's interaction with the

backend may vary with respect to the autonomy requirements of the applications being supported.

Challenges in supporting the future requirements of mobile CPSs arise in part due to their limiting network and system characteristics. Further, systems can be implemented in numerous ways tied to the applications that they support and system-specific characteristics. Below we present the broad set of characteristics that different implementations may leverage and an illustrative summary is shown in Fig. 2.1.

1. System Composition :

   A mobile CPS can have either homogeneous or heterogeneous composition. Some works consider heterogeneous systems to be robust and show improved performance as compared to homogeneous systems [74, 48]. The heterogeneity of a CPS can be attributed to the range of hardware specifications and capabilities supported by the participating nodes [74] or the use of multiple wireless access methods for connectivity [48, 88]. The nodes comprising a heterogeneous CPS might have varying computation, memory, and battery power; and support various sensors, e.g., Light Detection and Ranging (LiDAR) sensors, camera sensors, thermal sensors, etc. They may support multiple wireless access methods through a blend of hardware and software that can include any number of wireless interfaces. An example of heterogeneous access methods is a VANET supporting both IEEE 802.11p and cellular communication based on the 5G standard [48].

2. System Configuration & Wireless Access Methods :

   The network architecture of a mobile CPS can range from fully infrastructure dependent to completely Ad-Hoc in nature [28, 60]. The backend is highly involved in a fully infrastructure-dependent design with all nodes independently connecting to and communicating via the backend [28]. The backend assigns relevant application tasks to each node independently through a direct Line-of-Sight (LOS) wireless, satellite, or cellular communication link. Direct wireless links limit the operating range of the CPS. In comparison, satellite connectivity is more suitable for long-range communication. However, connectivity via a satellite results in high communication latency and packet loss [27]. Current and emergent cellular network technologies offer long-range communication and performance improvement but may be limited by existing availability in the CPS's region of operation.

   In an ad-hoc design, nodes are self-organizing and can carry out application tasks through

communicate among themselves without involving ground-based infrastructure. However, at least one node must be connected to the backend for application initiation information and any required pre- and post- processing [28]. The nodes can cooperatively achieve the mission assigned by the backend. Typically, ad-hoc communication between nodes can utilize an IEEE 802.11 wireless standard, e.g., 802.11a for ad-hoc UAV swarms or a Flying Ad-Hoc Network (FANET) [61], and 802.11p for a Vehicular Ad-hoc Networks (VANET) [54]. The ad-hoc network connectivity is established through appropriate routing protocols, and numerous works survey the available protocols for FANETs and VANETs [67, 75]. An ad-hoc architecture allows communication in real-time, provides redundancy, and promotes distributed operations. However, the communication range over which nodes in an ad-hoc CPS can reliably communicate with each other is a limiting factor. Further, the ad-hoc network is limited by bandwidth constraints and the compute capabilities of the system nodes. Some CPSs use a hybrid approach combining elements from infrastructure-dependent and ad-hoc based designs [28]. Operations in a hybrid approach are divided between the infrastructure and the CPS. Numerous examples of the hybrid approach propose the use of compute-capable edge or cloud-based backend to offload computation and disseminate data among nodes in a UAV swarm and VANET [29, 72].

3. System Mobility :

   The mobility of an CPS defines how the system nodes' position, acceleration, and velocity change over time. A CPS can have varying mobility, ranging from stationary to highly mobile. For example, a UAV swarm hovering in place is considered a stationary system even though the system is intrinsically mobile. Mobility in a CPS can be controlled centrally through pre-planned paths, cooperatively through the nodes in the CPS, or independently for each node. Further, mobility can be random or follow predictable patterns. A VANET typically follows a predictable two-dimensional mobility pattern controlled by the road layout, speed limits, and traffic rules. In contrast, a UAV swarm follows mission-specific three-dimensional mobility patterns with more variability [44]. The relatively changing node positions are likely to cause changes to the wireless network resulting in dynamic topology changes and network reconfiguration. In a CPS with low node density, the dynamic topology may result in network partitioning as well [44].

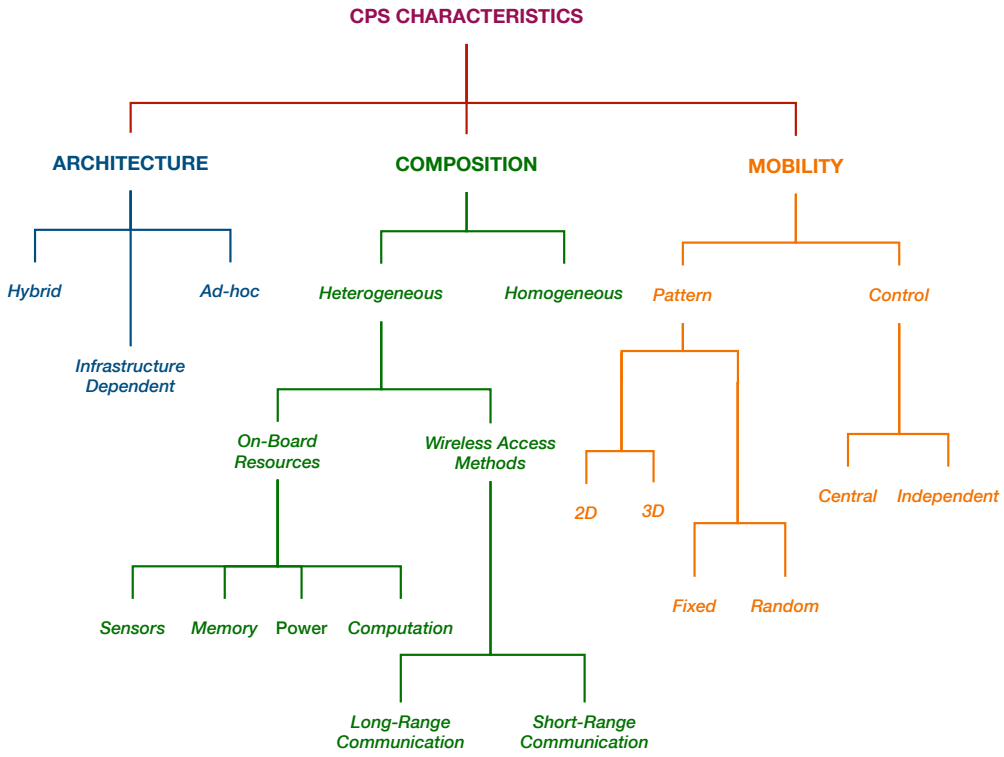


Figure 2.1: Summary of system characteristics in emergent mobile CPSs

2.2 CPS Communication Technologies and Practices

An APPSYS is expected to support missions with strict QoS requirements. Meanwhile, it is expected to be typically constrained in the availability of computational, memory, power, and network communication resources. Further, mobility and its impact on wireless connectivity are likely to result in intermittent network disruptions that may impact the APPSYS's ability to complete its mission efficiently. This section summarizes recent research within UAV swarms and VANETs that promise low latency, high reliability, or disruption tolerance and can be applied for APPSYS communication. Numerous works utilize other forms of edge computing, and fog and cloud computing for computational off-loading in efforts to reduce overall application latency.. More notably, various recent works leverage Multi-Access Edge Computing (MEC) with fifth-generation (5G) mo-

bile systems to provide reliability and low latency. 5G mobile systems utilize MEC architecture, where MEC-enabled nodes facilitating traditional cloud computational offloading are placed closer to users at the Radio Access Network's (RAN) edge [78]. While the use of MEC is not exclusive to 5G network technology, it is popularly utilized in 5G mobile systems to offer network performance enhancements, including computation offloading, distributed computing, and big data analytics for IoT systems. MEC-enabled nodes have been utilized in VANETs and UAV swarms for offloading time-consuming computation and reducing the latency of computation offloading to remote back-ends. A group of MEC-enabled UAVs flying in constant proximity to a UAV swarm can extend the 5G network slice to provide computational offloading for the UAV swarm supporting a data-intensive video sensing application - thus, providing lower latency for completion of application tasks [40]. Vehicles in a VANET can utilize MEC-enabled nodes at the network edge to collect and analyze collected vehicular data in near-real time [71]. MEC-enabled nodes in conjunction with a specific connectivity option, i.e., 5G network connectivity, assumes CPS operations in areas where such connectivity exists, which may not be valid for all CPS operational conditions. For example, a UAV swarm in a remote battlefield may not have access to 5G network communication. However, some of the principles that guide MEC standard architecture, i.e., proximity to the end-users, ability to absorb computational load from resource-constrained machines, and reusable application-programming interfaces (APIs) that a variety of tasks can use are essential requirements for a modular and standardized APPSYS design as well. Therefore, the APPSYS architecture incorporates the features mentioned above.

Some works approach the issue of lowering latency as a design problem and offer solutions in the form of hierarchical or layered architectures. A UAV swarm can use a layered architecture comprising an optimal number of UAVs in an upper layer to reduce network latency, distribute traffic flows more evenly, and improve reliability [91]. Other works utilize enabling technologies like Software-Defined Networks (SDN) and existing and emergent forms of communication protocols that facilitate group communication well-suited for CPS operations. A group communication method optimizes the transmission of messages to multiple recipients. Two group-communication methods, publish-subscribe message dissemination and multicast, have been applied to VANETs and UAV swarms. Multicast methods require only a single send operation to send a message to multiple receivers, thus, circumventing message duplication at the sender. The receivers and the sender form a multicast group, and underlying protocols create multicast trees for group message forwarding.

The publish-subscribe message paradigm and the more advanced Information-Centric Networking (ICN) concept that derives from it are further discussed in Section 2.4. Methods utilizing SDN are discussed in detail in Section 2.3.

Multicast methods can be broadly categorized into IP Multicast and Application Level Multicast (ALM) [70]. IP Multicast operates at the network layer and requires multicast-capable routers for creating multicast trees and forwarding. IP Multicast also requires separate interfaces for the inbound and outbound traffic in all participating nodes to avoid routing loops. Therefore, standard IP Multicast methods are not directly applicable to an APPSYS where wireless nodes may have single interfaces. However, modifications of IP Multicast protocols provide workarounds to extend multicast functions to an APPSYS. In PIM-MANET [22], virtual interfaces and duplicate message detection mechanisms are used at each node to avoid routing loops. Application Level Multicast (ALM) operates at the application layer. In ALM, end-systems (senders) create multicast trees using standard nodes within the network to reach a group of receivers. The multicast trees are created using limited state information at the end-systems, resulting in sub-optimal routes between the sender and receivers. Newer implementations of ALM utilize well-crafted overlay topologies to overcome issues caused by limited state information [70]. In UAV swarms, an implementation of ALM has been applied to efficiently forward video streams from a sender to multiple receivers [63]. Some works also focus on the mechanics of multicast tree formation in the context of APPSYS's mobility. Stateless and Predictive Geographic Multicast Scheme in Flying Ad-Hoc Networks (SP-GMRF) is a multicast method for UAV swarms where the multicast tree is reactively formed based on the network topology at the time of transmission [46].

A portion of recent research also focuses on routing protocols and methods suitable for wireless mobile networks where frequent disruptions and prolonged network partitioning are likely. These can be applied to incorporate disruption tolerance in a CPS. A CPS may use forwarding methods such as Store-Carry and Forward (SCF) and Greedy Forwarding (GF) [65]. In Chapter 4, we discuss how a Store-Carry and Forward (SCF) method for disruption-tolerance can be applied within the SNAP communication framework. Depending on the CPS's mobility pattern, deterministic or stochastic routing protocols may also be used to accommodate frequent changes to routing paths. Deterministic protocols such as UAV Search and Mission Protocol (USMP) [57] use predictions of network connectivity to create routing paths in an APPSYS with expected non-random mobility. On the other hand, stochastic protocols such as Shortest Expected Path Routing (SEPR) [79] use

broadcast-based message dissemination to transmit packets in a CPS with random mobility.

2.3 Software-Defined Networking

The Software-Defined Networking (SDN) paradigm led by the Open Networking Foundation (ONF) [12] allows network programmability resulting in adaptive and autonomous control over a system [23]. The main principle of SDN is the separation of the network devices used for forwarding data, and the software that makes forwarding decisions into two separate planes called the data and the control plane, respectively. Through this, SDN aims to simplify network management, provide flexible network control, and optimize network performance [23]. The complete SDN architecture comprises three vertical layers: the data plane, the control plane, and the application plane. The planes are connected via Northbound and Southbound interfaces as illustrated in Figure 2.2. Further, east and westbound interfaces are used to communicate with SDN controllers in other networks. The application plane comprises SDN applications that implement network control strategies, e.g., firewalls, load balancing, and traffic engineering. The data plane constitutes the SDN-enabled forwarding devices that operate as network switches forwarding the control plane's forwarding decisions. The control plane is the central entity of the SDN architecture. It consists of a software controller entity that accepts application requirements from the application plane and translates them into forwarding decisions for the data plane. The forwarding decisions are disseminated among the forwarding devices in the form of forwarding tables [53]. Hence, the network's control logic is centralized within the control plane. OpenFlow is known as the first widely acceptable communication protocol standardized by ONF [23].

SDN can be implemented as a centralized or distributed SDN. Centralized SDN uses a physically centralized control plane with a single software controller. Various studies document the scalability, reliability, and vulnerability issues associated with centralized controllers [23]. As the network grows, the volume of application requests the software controller receives is likely to overwhelm the controller, resulting in scalability and reliability issues. A single controller is further vulnerable as a single point of failure (SPoF) and susceptible to security attacks [34]. The problems of centralized SDN can be mitigated using distributed SDN comprising multiple physically distributed controllers arranged using a flat or hierarchical architecture. The flat architecture horizontally partitions the network into numerous areas where one controller is responsible for each area. Whereas

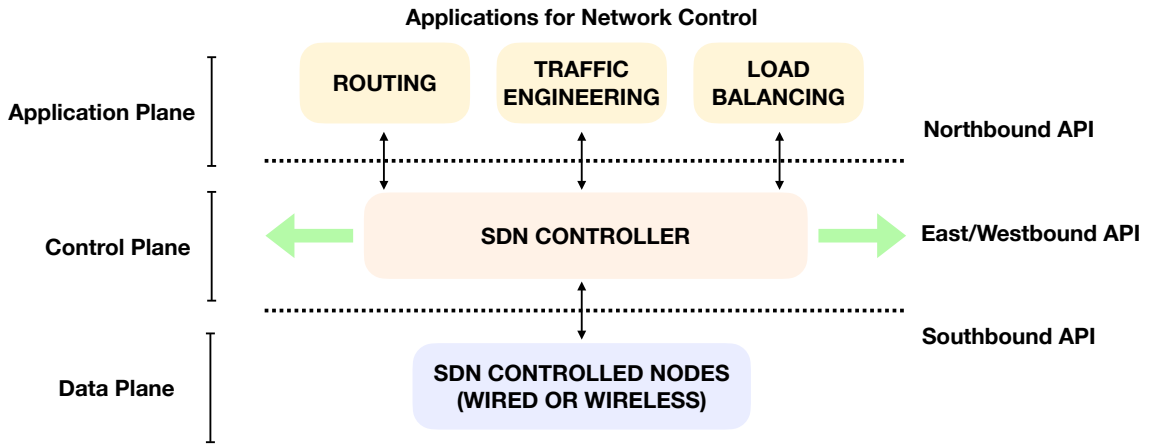


Figure 2.2: Components of Software-Defined Network Architecture

the hierarchical architecture vertically partitions the network into multiple layers using a hierarchy of controllers. The flat style promotes resiliency and fault-tolerance at the cost of scalability, while the hierarchical style promotes scalability at the expense of fault-tolerance [23].

Maintaining up-to-date global network state information across multiple controllers, known as consistency, is critical for distributed SDN. Consistency impacts the time and accuracy of applying network-wide policy updates. Networks in a single administrative domain can achieve intra-domain consistency through logically centralized controllers that replicate the global network state across all controllers. However, the measures to ensure consistency are typically time-consuming and require a trade-off between fast policy enforcement and efficient network performance [23]. While logically centralized controllers are sufficient for intra-domain distributed SDN control, they are infeasible for inter-domain distributed SDN in networks belonging to multiple administrative domains. An open research problem within the SDN community is interoperability among distributed intra-domain controllers. Limitations in interoperability arise from a lack of open standards for the Northbound interface to encourage application portability among inter-domain controllers and standardization of Eastbound and Westbound interfaces in the SDN controller architecture for communication among SDN controllers [23]. At present, minimal work in SDN focuses on interoperability. Further, the existing work only applies to Wide Area Networks (WAN) [68, 43].

The area of study that explores the utility of SDN in mobile CPSs is still under development, and all aspects of applying SDN have not been investigated yet. An SDN-enabled VANET is

proposed in [52] to load-balance traffic across multi-path topologies and optimize network utilization through centralized, decentralized, or hybrid SDN. The chosen SDN method is dependent on the state of network connectivity. Centralized SDN is likely to suffer from availability issues due to node mobility. Decentralized SDN is implemented via local SDN controllers on each wireless node. The controllers take independent control decisions resulting in local optima and not global optimum in network performance. Hybrid or distributed SDN is implemented via controllers at the backend edge infrastructure. However, the issue of maintaining consistency and handover of the VANET's operations among controllers in hybrid SDN has not been addressed. An SDN architecture for the battlefield, Software-Defined Battlefield Network (SDBN), is proposed in [64] to integrate multiple wireless access methods. SDBN is premised on the availability of numerous long and short-range communication methods through different interfaces on nodes in the UAV swarm. SDBN uses SDN to meet application QoS requirements through the selection of the most appropriate communication method. However, this work does not focus on the interaction between the application and control planes that allow mission application requirements to be translated into control logic. Currently, applications that can utilize SDN are required to provide low-level policy rules to the controller, therefore, limiting the range and flexibility of applications in an CPS that can interact with SDN.

SDN-MQTT, proposed in [84], integrates SDN and the messaging protocol, MQ Telemetry Protocol (MQTT) [10], to form a lightweight, distributed, and flexible middleware for optimized routing and forwarding in a UAV swarm. MQTT-SN, a broker-less implementation of MQTT suitable for resource-constrained networks, is utilized for message dissemination within the controller and the network. Furthermore, SDN is implemented in a distributed manner with a controller at each node. Similar to [52], the distributed controllers in SDN-MQTT also do not achieve the global optimum. This work is an example of clever integration between complementary technologies to create methods suitable for a CPS.

The separation of the control plane is conceptually beneficial to the APPSYS architecture as it allows better management of system resources and tighter system-level control over routing network traffic. However, the current state of SDN presents potential limitations for APPSYS architecture. First, SDN currently suffers from lack of interoperability and consistency issues among controllers under different administrative control. This limits the use of SDN in situations where APPSYSs under different administrative control may require to accomplish a common mission cooperatively. For example, the control plane could not be easily extended to support control-plane driven

cooperative functions between a UAV swarm belong to the U.S. Department of Defense (DOD) and a fleet of emergency-response vehicles belonging to the U.S. Department of Transportation. Second, the communication overhead introduced by the constant communication between the SDN controller and the data plane is likely to strain the APPSYS resources. Variability due to mobility in APPSYSs , resulting in frequent topology changes and dynamically changing neighbors, would require a even higher degree of communication. Additionally, the SDN control plane would need to be augmented to respond swiftly to topology changes and provide adequate neighbor discovery mechanisms [32]. Some SDN controllers such as OpenRoads [86] and OpenFlow [32] accommodate the mobility of wireless networks in SDN. However, the proposed solutions only utilize centralized SDN. Additionally, the control plane must be able to support flexible integration of a wider range of anticipated applications that would need to interact with it. The APPSYS architecture draws inspiration from SDN in the form of a distributed control plane that can flexibly interact with a wide range of applications. Chapter 4 presents details of the control plane implementation in the APPSYS architecture.

2.4 Information-Centric Networking

Information-Centric Networking (ICN) is a new Internet architecture that is said to better suit the application communication model of the present-day applications, including applications operating in CPSs, than the traditional and prevalent TCP/IP stack based Internet architecture [11]. Traditional communication that uses the TCP/IP stack is host-centric and requires source and destination communication endpoints. Meanwhile, Internet's applications are increasingly serving content-distribution requirements, often to mobile end-users. The ICN architecture generally aims to replace traditional host-centric IP communication in the Internet with an alternative architecture that relies on arbitrary application-specific data identifiers. Therefore, allowing application requests to be identified transparently by the content they are requesting as opposed to IP addresses. Name-based networking allows forwarding nodes within the system to cache named-content for future requests, therefore, creating a distributed model for content distribution. This serves as both a load distribution and disruption tolerant measure.

In comparison to the traditional Internet architecture, ICN architecture combines the TCP/IP stack's application and transport layer into an ICN application layer that creates application mes-

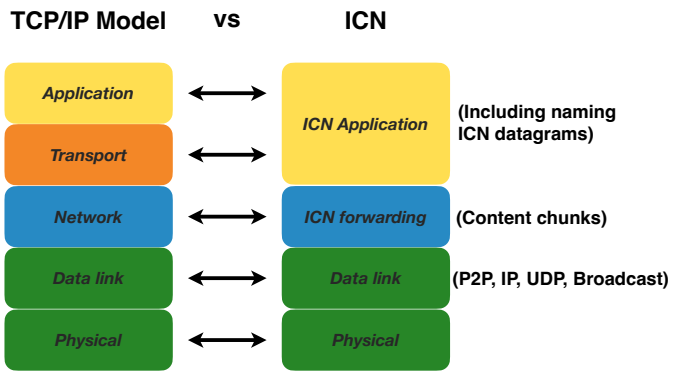
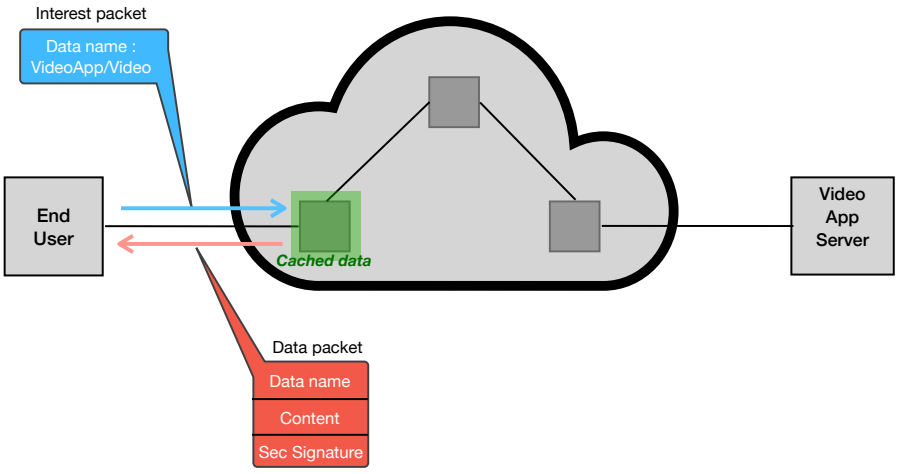


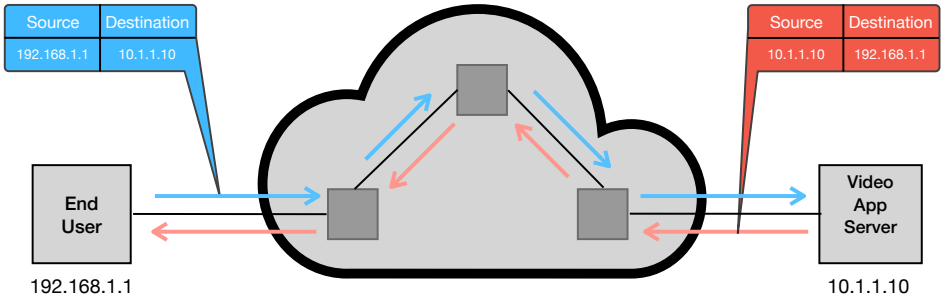
Figure 2.3: TCP/IP and ICN Architectures

sages and assigns data identifiers to them. The TCP/IP stack's network layer is replaced by an ICN forwarding layer that transports chunks of data identified with data identifiers. The ICN Link Layer can utilize any available connectivity option to forward the ICN datagrams, e.g., available IP connectivity, MAC layer broadcasts, and Point-to-Point (P2P) links. Fig. 2.3 compares the TCP/IP and ICN Internet architectures. The application clients in the ICN architecture utilize packets, called *Interest Packets*, which contain only an application-specific data name identifying the content being requested by them. As opposed to IP-based applications where only the application server or designated local caches can respond to client queries, any node with the data corresponding to the clients' Interest Packets can respond with the requested content, called *Data Packets*. As a transparent security measure, each data packet is signed by the primary Data Producer and is verifiable by the clients. Applications may use flat or hierarchical naming schemes. Flat naming schemes use unique and unconnected names for all named data, while hierarchical naming schemes use related names resembling a natural aggregation of information [77]. Fig. 2.4 illustrates ICN's request and response model as compared to the IP's request and response model.

The implementations of ICN can be broadly categorized as content-centric and publish-subscribe based. Named Data Networking [90] is an example of content-centric architecture. In NDN, a Named-Data Object (NDO) is requested by a receiver and the request is propagated through NDN routers in the network. The NDN routers utilize a broadcast-based stateful forwarding strategy; the state information stored in forwarding nodes is used to create the reverse path for the response from



(a) ICN Communication



(b) TCP/IP Communication

Figure 2.4: Illustrative comparison of TCP/IP vs. ICN communication.

the sender [90]. The use of NDN has been proposed for disaster management use-case involving a VANET and a ground-based Wireless Sensor Network (WSN) [33] where the ground-based sensors inform the VANETs of relevant information using NDN. A modified version of NDN with enhanced in-network caching methods is proposed in [92] to accommodate network disruptions in mobile VANETs. HoP-and-Pull (HoPP) [41] extends the request-response NDN architecture to support a mechanism that allows data to be pushed out without a prior request. HoPP can be used to reduce traffic in resource-constrained sensors.

Publish-Subscribe Internet Routing Paradigm (PSIRP) [80], and ICN-IoT [76] are examples of publish-subscribe architecture. The publish-subscribe architecture comprises publishers (senders), subscribers (receivers), and brokers. Publishers transmit data packets as NDOs to subscribers via one or more brokers. In PSIRP, NDOs are forwarded through brokers known as Rendezvous Points (RP). Each RP is associated with a Rendezvous Network (RN) and handles all NDOs within the RN's scope. The naming of NDOs in PSIRP includes details of the scope and the RP identifier. A subscriber must know this information apriori, thus, limiting the degree of decoupling permitted by PSIRP. ICN-IoT is an ICN architecture developed for the Internet-of-Things (IoT). It is applied to a UAV swarm to form an Internet of Drones (IoD) [24]. The objective of IoD in [24] is to provide navigational services and coordination among UAVs. ICN enables the transmission of environmental information from ground-based WSNs to the UAV swarm to guide the flight plan. Overall, while the use of ICN in AppSys-like environments is illustrated through the various presented works, study within this domain is still in its nascency. A majority of available work focuses on theoretical applications instead of performance-based evaluation.

Each of these implementations relies on a specially designed Forwarding Plane used to cache and forward named data. The major open problems that the ICN community is working towards addressing relate to the management of an unbounded namespaces created by arbitrarily named data. This problem also manifests within the Forwarding Plane where appropriate data structures must be utilized to cache and forward data. Currently, forwarding planes use data structures such as tries or hash tables which coupled with an unbounded namespace may prove unsuitable for resource-constrained systems [38]. Further, increasing adoption of ICN leads to a period of co-existence between IP and ICN based communication where networks might comprise of small ICN sub-networks within larger IP based networks or vice-versa. The former will require ICN-over-IP communication solutions while the later will require IP-over-ICN communication solutions [30].

The APPSYS architecture currently uses an ICN-over-IP model where communication is forwarded based on Interest packets, however, forwarding is conducted over underlying IP connectivity and encapsulated in traditional IP datagrams.

2.5 Summary

The system characteristics of the emergent class of mobile CPSs are differentiated through choices in system composition, configuration, and mobility. Different design choices provide different benefits and challenges. For example, while ad-hoc architecture in a CPS offers more redundancy compared to infrastructure-dependent architecture, it may also be more constrained in available bandwidth. A significant portion of recent research in CPSs is focused on proposing solutions that offer low latency, reliability, and disruption tolerance for CPS communications. Solutions include architecture-based solutions such as utilizing hierarchical or distributed systems, communication enhancements bought on through better wireless access technologies such as 5G communication, provisioning computational resources close to the CPS, and protocols designed to provide efficient group communication and disruption tolerance. However, these solutions are closely tied to distinct characteristics of specific CPSs and application scenarios. A portion of recent work also focuses on applying SDN to mobile CPSs. However, the study of SDN with respect to mobile CPSs is still under study. Various aspects of applying SDN to these systems, such as maintaining consistency between controllers in different domains, the viability of local vs. global optima using centralized vs. decentralized decision-making in CPSs, and additional node-controller communication overhead stemming from system mobility need further investigation. Another enabling technology being explored for these systems is name-based communication using ICN. ICN has the potential to provide well-suited, robust, and flexible communication support for mobile CPSs. However, ICN implementations within these systems are limited, and few performance evaluation studies exist. Similar to SDN, the impact of ICN characteristics such as forwarding plane and topic management on resource-constrained devices are yet to be sufficiently investigated within the systems of interest to this work.

Chapter 3

APPSYS Architecture

The APPSYS utilizes a decentralized architecture where participating nodes are divided into smaller sub-networks called clusters. Each cluster follows a hierarchical network organization with a designated top-level node responsible for managing intra- and inter- cluster communication. The different clusters form a mesh network and are interconnected via an overlay network with rich multi-path connectivity. Therefore, the APPSYS is a decentralized system comprising multiple hierarchical clusters connected via mesh connectivity. A decentralized APPSYS design provides advantages like increased scalability, reduced vulnerability by omitting a single point of failure (SPoF), the ability to load-balance application tasks among different clusters, and efficient traffic management in the APPSYS. Further, hierarchical design in the clusters allows the top-level cluster node to take on the role of a top-level control plane controller and apply fine-tuned control over ingress and egress traffic. Chapter 4 discusses the hierarchical and distributed control plane in further detail.

The primary objective of the APPSYS architecture is to facilitate system communication and data services that are required to meet the APPSYS's mission's QoS requirements. The APPSYS architecture proposed in this work has been informed by lessons learned through prior performance evaluation studies conducted in single-cluster VANETs and UAV swarms. In our earlier work, we conducted measurement studies showing the impact of increasing application traffic load on single-cluster VANETs or UAV swarms using standard communication protocols like UDP-based unicast, UDP-based broadcast, and publish-subscribe-based group communication through a centralized broker. The general observations from our measurement studies are as follows. As the overall application traffic load increases, the application's end-to-end latency and throughput performance

within a single-cluster system degrade. In the absence of specifically applied network impairment detection and mitigation measures, as the network observe 100% utilization on shared paths, the network performance severely degrades till it is rendered unusable for any further application communication. Further, an increase in application load has a higher impact on an application using unicast communication than broker-based publish-subscribe communication. This final observation motivates the application of broker-based ICN in the SNAP framework.

In this chapter, we first present the relevant experimental details and lessons learned from these studies. Then, we describe various components of the APPSYS architecture in further detail. In the remainder of this work, we use an exemplar UAV swarm APPSYS conducting a CSAT mission to illustrate the APPSYS architecture and the SNAP communication framework, respectively. The UAV swarm APPSYS and the CSAT mission have feature-sets representative of a typical APPSYS and mission of interest to the present work.

3.1 Earlier Performance Evaluation Studies

In a study conducted by us and detailed in [50], a measurement-based performance evaluation was conducted within a single-cluster UAV swarm. One objective of this study was to compare the end-to-end application latency or one-way delay (OWD) observed by an exemplar application in the UAV swarm using two commonly utilized communication methods, UDP-based unicast and broker-based publish-subscribe communication. The OWD represented the time taken by an application message sent by the application sender(s) within the system to the application receiver within the system. The UAV swarm communicated as an ad-hoc wireless network using standards-based IEEE 802.11a. Each node directly communicated with its one-hop neighbors within 802.11a's communication range with a R12BPSK Modulation and Coding Scheme (MCS), representing a maximum theoretical throughput of 6 Mbps. The inter-node range was 500 meters, and multi-hop communication was supported through pre-established routes using underlying routing protocols (Open Shortest Path First). Fig. 3.1 illustrates the UAV swarm used in this study.

The exemplar application being supported by this system was a generic video-transmission application involving the transmission of video data from a number of sensing capable UAVs to a single UAV designated for data aggregation. This application serves as an early form of the CSAT mission application software developed for the present work. For the purpose of this study, the video

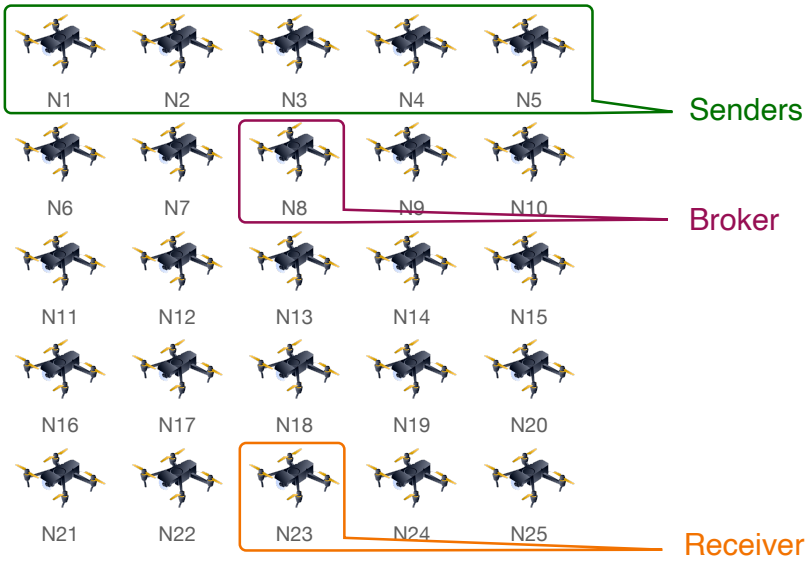


Figure 3.1: Single-cluster UAV swarm topology used in [50].

data was represented as metadata using different message sizes and application transmission rates. It was assumed that local sensing capabilities within the transmitting UAVs were able to convert video data into usable metadata that could meet the application's needs. The application was representative of a data-intensive distributed application. For the evaluation, the application load in the system was increased by increasing the number of senders transmitting 500 Bytes application messages at an application transmission rate of 100 kbps. For publish-subscribe transmission, a centralized broker was used and the broker placement was as shown in Fig. 3.1. The broker for publish-subscribe communication was specially developed and used underlying UDP transmission. For both communication methods, the senders shared a common path to the application receiver. The testing was conducted using the CORE [3] and EMANE [5] network emulation software. Each node in the emulated testbed had an isolated network stack and independent processes.

As the number of simultaneously-transmitting senders increased from 1 to 3 to 5 in Fig. 3.1, an increase in per-packet application OWD was observed at the application receiver. Fig. 3.2 shows this result. The increase in OWD was more observable for an application using UDP-based unicast transmission as compared to one using broker-based publish-subscribe transmission. The broker used for publish-subscribe transmission aggregated the message flows from all application senders and reduces the system's total number of message flows. The reduced number of flows corresponded

to reduced processing and queuing delays experienced at each intermediate node between the senders and receiver and resulted in lower end-to-end OWD at the receiver.

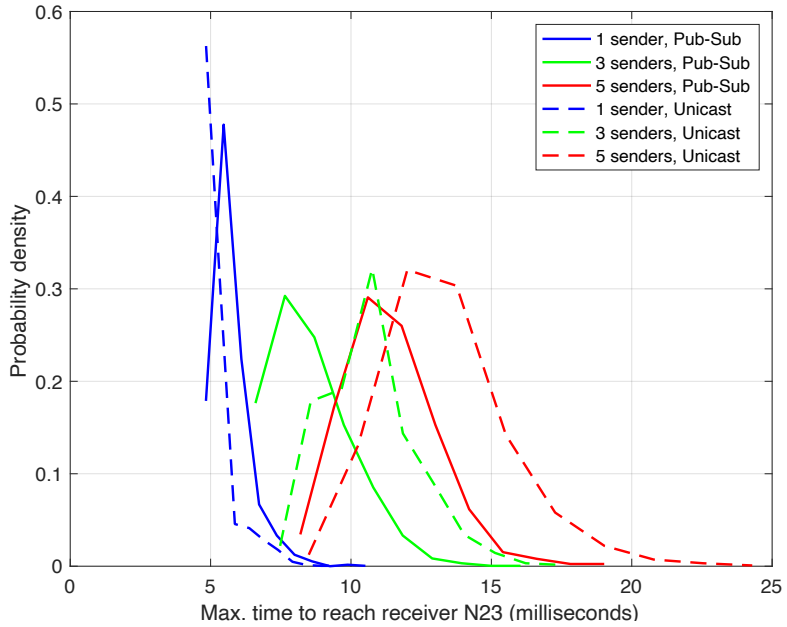


Figure 3.2: Comparison of application OWD with increasing number of senders in a UAV Swarm.

Another hypothesis can be drawn from the result in Fig. 3.2. As the application traffic load within the system increases while the application nodes share common resources, i.e., common routing paths, the application can expect to see performance degradation equivalent to the load increase. The result in Fig. 3.4 illustrates this statement. In Fig. 3.4, we observe that as we increase the transmission rate from 100 kbps to 300 kbps for 5 application senders using publish-subscribe communication, the broker OWD increases linearly as the experiment progresses. As the aggregated transmission rate from all senders is less than the maximum throughput of R12BPSK, this impact can be attributed to the increasing processing time at the broker. As the broker performance increasingly worsens, the broker queue becomes full leading to network congestion. In comparison, replacing a single centralized broker with two brokers conducting basic load-balancing greatly lessens the impact of high application load on the brokers. The single-cluster UAV swarm topology with two brokers is illustrated in Fig. 3.3. Fig. 3.4 illustrates the comparative difference in the impact of high application load on a single broker system as compared to a system with two brokers.

A similar observation can be made from an earlier performance-evaluation study conducted

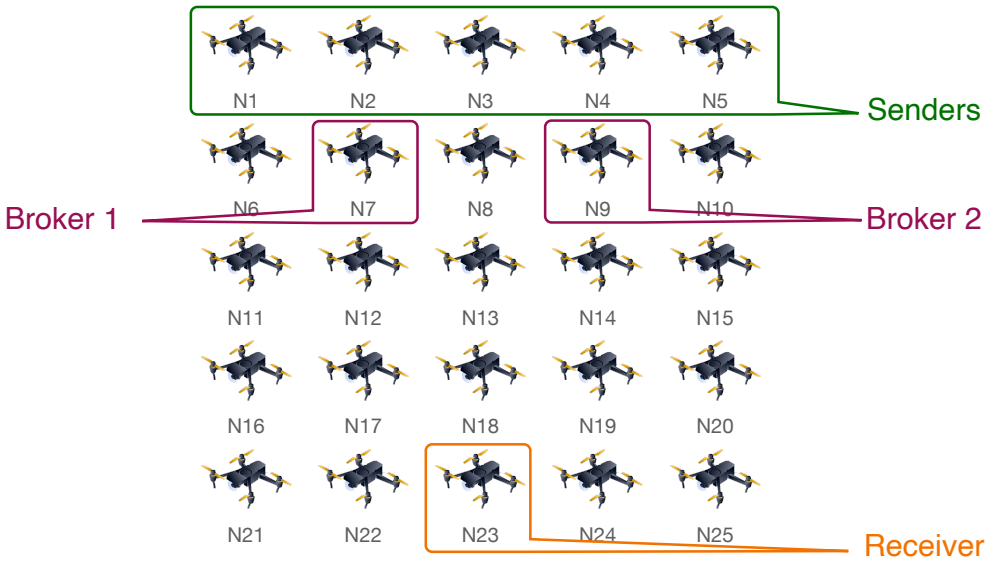


Figure 3.3: Single-cluster UAV swarm topology with decentralized brokers.

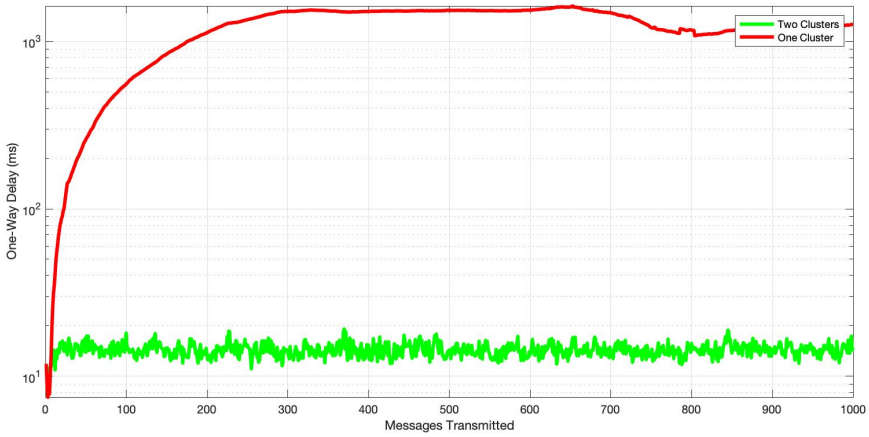


Figure 3.4: Impact of high application load on a centralized vs. decentralized brokers in a UAV Swarm.

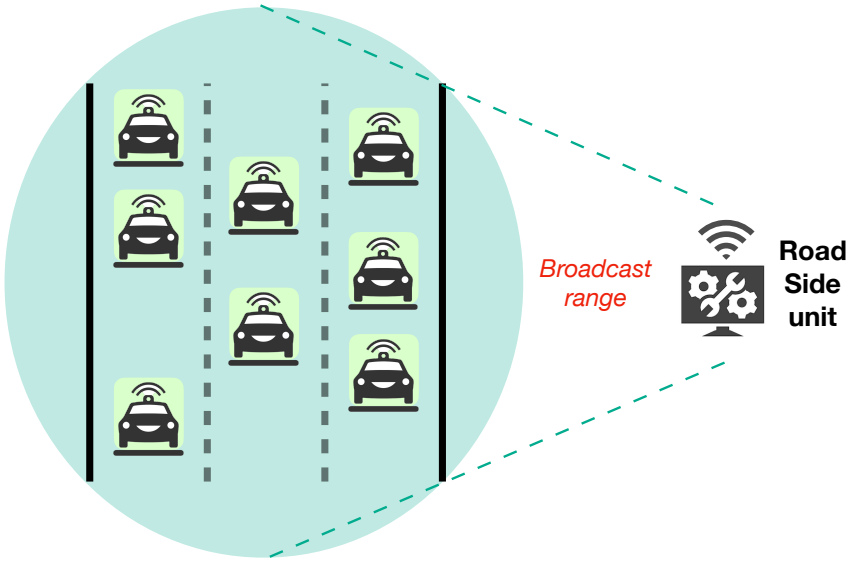


Figure 3.5: Experimental setup for the performance evaluation of a VANET in [49].

by us in VANETs [49]. In this study, vehicles within broadcast range of each other utilized the Dedicated Short Range Communications (DSRC) communication using IEEE 802.11p Wireless Access for Vehicular Environments (WAVE) at the physical (PHY) and Media Access Control (MAC) layers. The vehicles used an MCS rate of R12QPSK with a maximum achievable throughput of 3 Mbps. Vehicles in a VANET typically use broadcast-based communication. As the number of vehicles broadcasting at 3 Mbps was increased, the channel utilization approaches 100% utilization and the ensuing contention process for transmission by each sender results in prohibitively high latency. This latency value has an upper bound contingent on the maximum queue capacity and can not increase beyond that. Figs. 3.5 and 3.6 illustrate the experimental setup in NS3 for this study and the results.

Results in Figs. 3.4 and 3.6 lead to the conclusion that for applications where the aggregated load may be greater than the available shared resources, intelligent load-balancing of application load on system resources is necessary. Furthermore, methods for monitoring and mitigating network impairment would increase the efficacy of these systems. The APPSYS architecture achieves intelligent load-balancing by dividing the network into multiple clusters where each cluster is assigned application load proportionate to its resource availability. Further, the control brokers in the

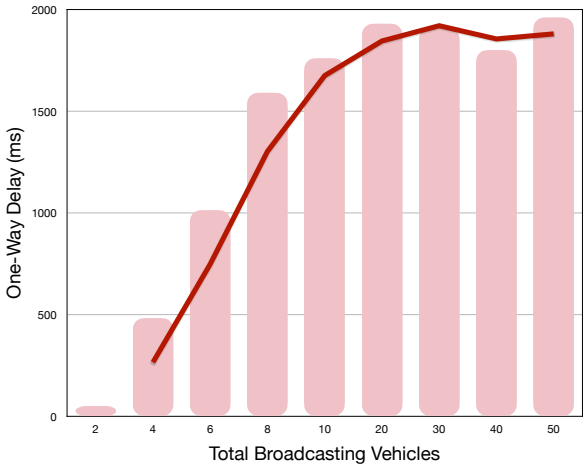


Figure 3.6: Impact on application OWD as number of broadcasting vehicles in a VANET increases.

SNAP communication framework, discussed in Chapter 4, are decentralized and offer methods for monitoring and mitigating network impairment.

Another objective of the study in [50] was to examine the role of strategic broker placement within the UAV swarm. Two experiments were conducted with a centralized broker placed at varying hop distances from the set of application senders. In the first experiment, the broker was placed closer to the group of senders. In Fig. 3.7, this broker location is called *Broker Location 1.* In the second experiment, the broker was placed farther from the group of senders at *Broker Location 2.* Application load in this set of experiments was increased by sending varying message sizes, 500 Bytes and 1250 Bytes, at a rate of 100 kbps from five application senders. Fig. 3.8 shows the result from these experiments. The result showed a significant increase in the average OWD at higher application loads due to placement of the broker farther from the senders. The placement of the broker closer to the senders reduces the total number of message flows sooner, resulting in lower OWD. This result further illustrates the requirement for strategic broker placement where brokers must be placed close to the senders. The APPSYS architecture uses the SNAP framework to achieve strategic broker placement through *local brokers* at each cluster node, and one *cluster broker* per node which is always one-hop away from all nodes within the system. Chapter 4 discusses the SNAP framework in further detail.

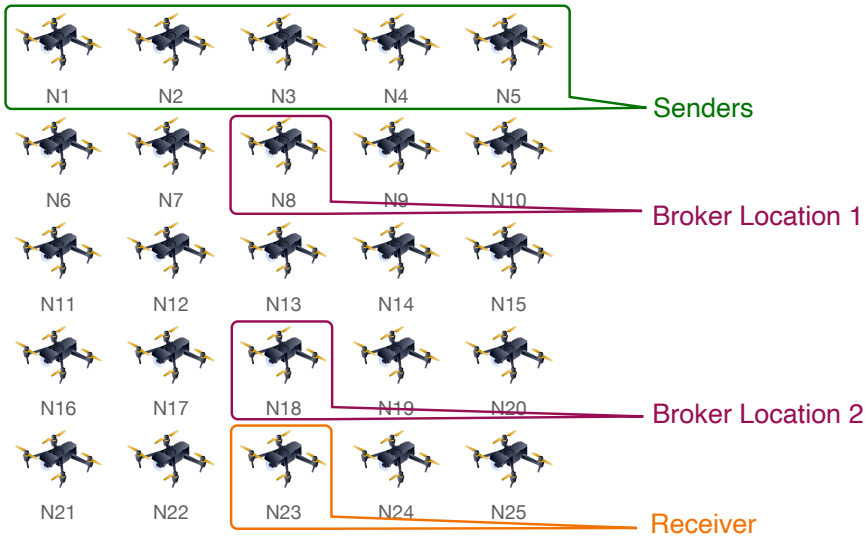


Figure 3.7: Experimental broker locations in a single-cluster UAV swarm.

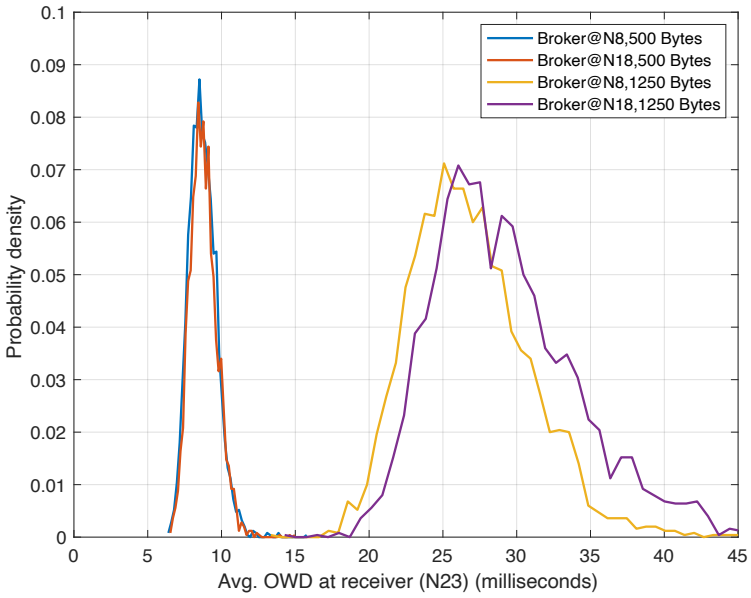


Figure 3.8: Impact of the broker position on application OWD in a UAV Swarm.

3.2 UAV Swarm APPSYS

A UAV swarm APPSYS comprises one or more clusters of wireless mobile UAV nodes. Within the APPSYS, a cluster is identified as a group of UAVs within broadcast range of each other. The mathematical problem of optimally partitioning a group of nodes into clusters has been addressed in numerous works such as [39]; however, it is not the focus of the present work and we assume that existing methods can be applied to achieve this step. Each cluster is heterogeneous and comprises UAVs with a range of power, compute, sensing, and physics-based mobility capabilities. Most UAVs in the cluster are micro UAVs with small payloads, limited speed/altitude, and constrained compute and memory capabilities. Wasp III [19] and DJI Phantom 4 [4] drones are two examples of micro UAVs. A cluster constitutes mainly of micro UAVs with a few highly functional UAVs. Due to their small size, micro UAVs can support only a limited number of sensors [26]. Therefore, different micro UAVs in the swarm may be fitted with different sensors to give well-rounded data-sensing capabilities. A cluster also supports multiple wireless access methods using a variety of interfaces, thus, allowing a flexible choice of communication methods. The UAV swarm APPSYS is closely tied to a specific application that it serves. We refer to this application as the APPSYS's *mission*. The UAV swarm operates primarily in ad-hoc mode. However, the backend is responsible for assigning missions, communicating mission-relevant details through control messages, and acquiring related outcomes periodically from the UAV swarm. Within a mission, the backend may also be secondarily responsible for high-level mobility management of the swarm. We refer to the backend in a UAV swarm APPSYS as the Control Station (CS).

3.2.1 Swarm Connectivity and Communication

One node in a cluster serves as the cluster leader node and connects to the CS over a backhaul link. The role of a leader in the cluster is discussed in Section 3.2.2. For non-line-of-sight (NLOS) communication between a CS and the clusters in an APPSYS, backhaul connectivity is through a radio, satellite, or cellular data link. Typically, significant Round Trip Time (RTT) is associated with communication using a long-haul radio or satellite link. Therefore, making infrastructure-based connectivity unsuitable for low-latency communication. The cluster leader function can only be assigned to cluster nodes equipped with the adequate interface to communicate over the backhaul link. Clusters have a failover procedure where at least one other node in the cluster is eligible to

function as the cluster leader. Within a cluster, messages are transmitted as cluster-wide broadcasts, while messages between the cluster and CS are typically transmitted as unicast messages. Nodes within a cluster are instructed to either accept or reject a message and not forward it to avoid broadcast storms. System connectivity in the APPSYS is established through an appropriate link-state routing protocol. To ensure coexistence of IP-based APPSYSs with ICN-based APPSYSs, the APPSYS design utilized IP-based end-point identification even though communication is name-based. The IP address assignment in the APPSYS ensures complete end-to-end connectivity across all clusters, ensuring multi-path connectivity among them.

3.2.2 APPSYS Node Functions

UAVs within a swarm can perform different interchangeable functions depending on their hardware capabilities. These functions could be sensing, computation, or management. We summarize the roles associated with these functions as follows:

- Cluster leader node: A cluster leader node primarily serves as a gateway between the CS and the cluster in the UAV swarm. It receives and processes control messages from the CS and forwards them to the correct destination within its cluster. The cluster leader also facilitates communication between any application end-point, i.e., an application receiver, in the cluster that may be required to communicate with the CS. For now, we assume that the cluster leader node may be responsible for resource management and assigning mission tasks within the cluster if required. However, future iterations of the APPSYS design may utilize a specialized resource manager function for the resource management tasks. As any cluster can contain an application end-point, the APPSYS architecture design guidelines dictate that a leader node be available in each cluster. However, a cluster may also utilize the nearest available leader node in adjoining clusters for communication with the CS.

- Sensor node: A sensor node is typically a micro UAV containing one or more sensors and able to transmit data over the APPSYS network. Depending on the scenario, the node might be capable of lightweight computation. For example, a node with an optical camera sensor might also possess the additional software capabilities to minimally analyze the collected data and detect moving objects. Conversely, the node might be constrained and only transmit the video stream to a different node within the APPSYS. Depending on the availability of

adequate sensors, sensor nodes are tasked with data collection relevant to the APPSYS's mission and serve as application senders within the APPSYS. The size of the node dictates the additional sensor payload it can support and limits the number of sensors on an individual node. Examples of sensors that the UAV swarm's sensor nodes can employ include optical camera sensors, infra-red sensors, radar sensors, and sensors to track electronic emissions.

- Compute node: A compute node is typically a highly functional UAV in the UAV swarm capable of aggregating and analyzing sensing data collected from various sensing nodes. Compute nodes serve as application receivers within the APPSYS and utilize the nearest cluster leader to communicate outcomes of their analysis to the CS. The level of computational capabilities that a compute node should be provisioned for varies by the set of use-cases relevant to the UAV swarm APPSYS.

Additionally, UAVs within a cluster may support one of the two specialized software broker roles discussed in Chapter 4. The software roles are assigned based on the hardware capabilities and placement of UAVs with respect to other nodes within the cluster. Communication among clusters is possible through a wireless mesh network, details of which are provided in Chapter 4 as well. Figure 3.9 illustrates the UAV swarm APPSYS and connectivity of clusters with the CS.

3.2.3 APPSYS Mobility

The UAV swarm can support two and three-dimensional mobility patterns. Mobility is mission-dependent and may range from stationary to highly mobile. Mobility may result in link outages and topology reconfiguration as nodes move out of communication range with each other [44]. We assume that each cluster can fly autonomously with a predetermined mobility pattern, and all UAVs fly at a constant velocity. At a high level, the flight plan is either provided by the CS to each UAV or to the cluster leader node that uses a leader-follower-based control strategy to execute the flight plan. The CS or the UAV cluster can modify the flight pattern through control messages to the UAVs during a mission as required. For example, the flight plan in a CSAT mission is based on waypoints based on the search target's present location.

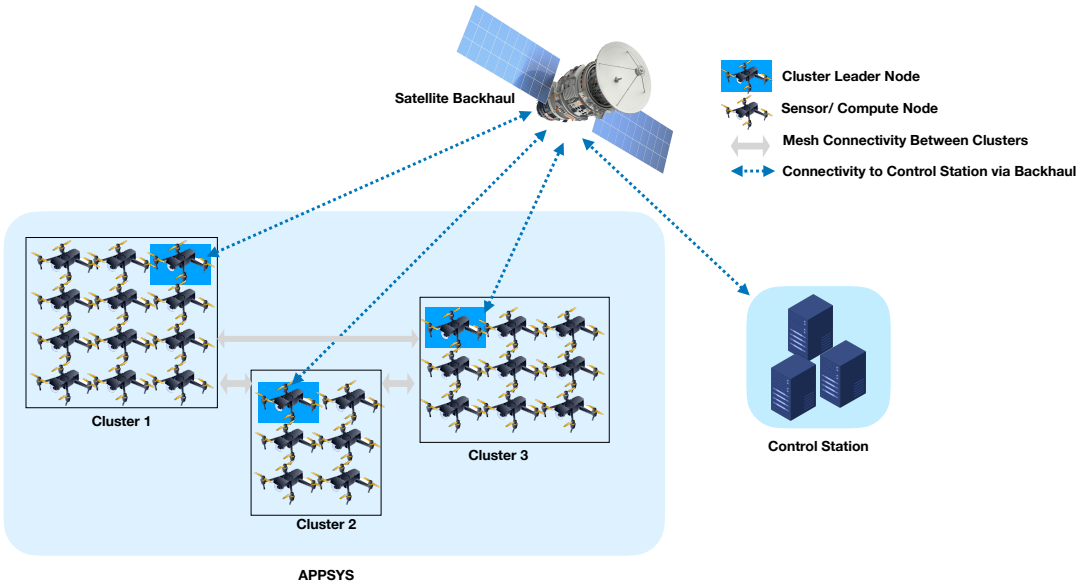


Figure 3.9: System Design of the UAV swarm APPSYS

3.3 APPSYS Missions

A UAV swarm provides support for critical infrastructure domains through missions assigned to it by the CS. The CS can either assign a mission to UAVs within the swarm APPSYS directly or delegate mission task assignments to the cluster leader nodes. In both cases, task assignments require apriori knowledge of resources available at each node. For example, video-sensing tasks can only be assigned to nodes with optical cameras and computationally-intensive tasks can only be assigned to nodes with sufficient compute capabilities. Missions may be completed autonomously or semi-autonomously by the UAV swarm APPSYS. The focus of this work is on coordinated missions that can be cooperatively supported by a UAV swarm APPSYS. The APPSYS supports missions through application software running on the UAVs. The application software translates high-level mission objectives into low-level tasks that the UAVs can carry out. For missions requiring coordination between nodes of an APPSYS, the software is distributed with multiple components running on different UAVs. The CS may also support an application component to participate in the application,i.e., provide updated directives or receive periodic mission updates.

Completing a data-intensive mission autonomously or semi-autonomously within the swarm can be generalized into three stages derived from the decision-making paradigm for autonomous applications in a CPS: data, control, and process stages [26, 69]. The data stage involves relevant data collection using sensor nodes within the UAV swarm. The control stage is sub-divided into perception and planning stages. In the perception stage, the data acquired in the previous stage is processed using algorithms for data-mining or data processing algorithms. The processed data is analyzed and converted into mission-specific actionable items for the swarm in the planning stage. Finally, the actionable items are executed in the process stage. The primary mission outcome of the process stage is a periodic mission update or a final mission result that can be transmitted to the CS. The process outcome may also form a feedback loop with the data stage to guide the sensor nodes' actions based on the outcome of the planning stage.

Figure 3.10 illustrates the stages of an autonomous mission in the context of the generic video search mission being conducted by a UAV swarm APPSYS. The participating UAVs collect and analyze video-sensing data with the objective of locating a target of interest. The feedback loop between the process and data stages in this illustration is used to aid the search function by directing each nodes' flight plan based on conducted data analysis. This mission can be thought of as a high-level form of the CSAT application discussed in Section 3.3.1. The data and control stages may involve multiple sensor and compute nodes respectively. Therefore, the data stage may include sub-steps relating to coordinated data collection to avoid duplication of efforts and conduct effective data collection. Similarly, the control stage may require sub-steps for aggregating processed data and consolidating the findings from the data. The system's network and compute resources utilized for a mission depend on the amount of data being collected, transmitted, and analyzed in the system. The sub-steps mentioned earlier may require additional system resources. Further, missions may be safety-critical with strict Quality-of-Service (QoS) requirements. A mission's QoS requirements may be defined in terms of a packet-switched network's QoS metrics such as maximum throughput, packet loss and errors, latency and variation in latency (jitter), and ordered delivery of packets. A mission may also have mission-specific QoS requirements. For example, a video search application may use time to locate target as a primary QoS requirement.

3.3.1 Illustrative Mission: Coordinated Search and Tracking

In the present work, we focus on an illustrative use case of a UAV swarm APPSYS on the battlefield assigned a Coordinated Search and Tracking (CSAT) mission. A CSAT mission involves using the integrated sensing capability of sensor nodes equipped with suitable sensors to locate evasive targets on the battlefield [81]. For example, a CSAT mission objective might be to locate a vehicle of interest fitting a particular description in the streets of an urban area. Suitable sensors might include optical camera, infra-red sensors, Radar or LiDAR sensors, and depth perception sensors. CSAT implemented through an airborne UAV swarm offers two distinct benefits for a target search. First, the airborne UAV swarm provides a better vantage point for an effective search, and second, unmanned missions conducted through a UAV swarm reduce the risk of casualties on the battlefield [31].

There are numerous implementations of the CSAT application in recent works, and different works differ in their interpretation of coordination among nodes. Sensor nodes may coordinate by simply partitioning the target region and scanning their respective areas for the target search. In this case, each node's collected data is transmitted to the CS for further analysis [73]. A more evolved coordination measure may involve flight configuration of UAVs to increase probability of target detection while minimizing UAVs, and topology reconfiguration if a participating UAV malfunctions [81]. The authors in [81] utilize parallel path search where UAVs in a line formation perform a *sweep* of a target region while exchanging periodic system update messages to handle topology reconfiguration.

Some implementations focus on advancing data perception in addition to coordination among UAVs. Authors in [62] utilize computer vision enabled data perception for data collected through different UAVs carrying different sensors. In this work, three UAVs carrying an infra-red sensor and a video camera, a fire sensor, and a depth measurement sensor respectively transmit data to a centralized decision making component where computer vision is applied for data perception. Similarly, authors in [85] utilize opportunistic learning among the participating UAVs to implement coordinated search. In this work, each participating UAV collects its own sensor data within the target region and forms a cognitive Cartesian grid map representing its situational knowledge and the probability of target detection in its current surroundings. Each UAV shares this cognitive map with its neighbors; an individual UAV participating in the CSAT map utilizes its own sensor data

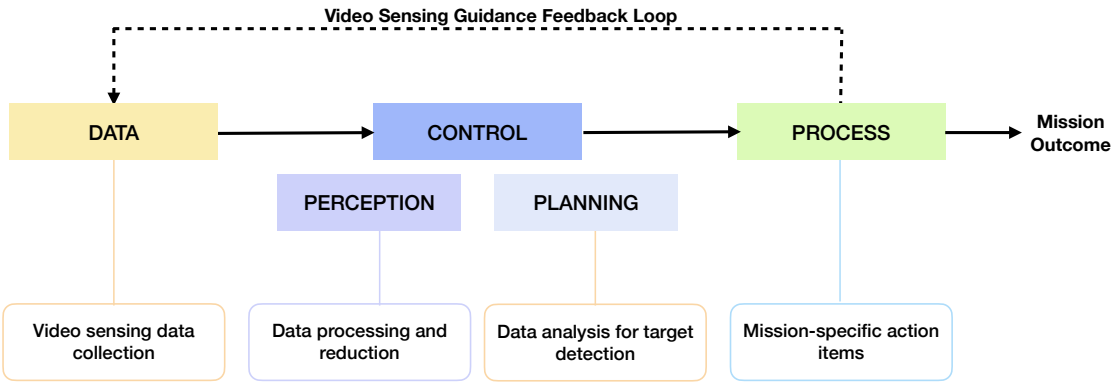


Figure 3.10: Stages of an Autonomous CSAT Mission

and an aggregated cognitive map received from its neighbors to plan its own path. Therefore, this work employs passive coordination. While [73, 81, 62, 85] focus primarily on *search*, works like [36] focus on both initial search and continuous tracking. Authors in [36] utilize a Recursive Baysean filtering framework to search and continuously track targets in a marine search and rescue scenario.

The study of recent CSAT implementations indicates that although numerous conceptually effective search and tracking mechanisms exist, there is no standard guidelines for how coordination, searching, and tracking should be implemented. Furthermore, there is a distinct lack of measurement-based studies evaluating CSAT performance in a hardware testbed representation of a UAV swarm. One contribution of the present work is an implementation of the CSAT application that can integrate with the APPSYS design and SNAP communication framework. Our CSAT implementation focuses on continuous target tracking and assumes that a method like [85] can be applied to conduct an initial search of the target. We implement a distributed CSAT application where numerous sensor nodes can collect sensing data translatable to metadata including the initial target location in Cartesian coordinates. The sensing data is collected and analyzed at a designated CSAT compute node within the UAV swarm APPSYS. The compute node utilizes a Kalman Filter to predict future target positions; the compute node uses a feedback loop between the sensor nodes and itself to share predicted target positions and inform the sensor nodes' flight plan. Within the APPSYS architecture, the feedback loop also carries QoS-relevant information that the SNAP framework's control brokers can utilize to modify the underlying system to meet CSAT's QoS re-

quirements. We discuss our implementation of the CSAT application and its performance evaluation within a UAV swarm APPSYS in Chapter 5.

3.4 Summary

The current design of the APPSYS architecture is motivated by our prior measurement-based performance evaluations in flat-architecture-based UAV swarms and VANETs using standard communication methods like UDP-based unicast, broadcast, and publish-subscribe through a centralized broker. These studies indicate that the overall performance of applications or missions operating in these systems degrades significantly as mission traffic load leads to high utilization of system resources. Further, the system is rendered unusable without suitable network impairment detection or mitigation mechanisms unless the mission is terminated. Systems using publish-subscribe communication through brokers demonstrate slower performance degradation than systems using UDP-based unicast. Further, strategies like efficient broker placement and load-balancing system traffic across multiple brokers improve the system's performance at higher application traffic loads. Guided by this prior work, the present APPSYS architecture is designed as a decentralized system comprising multiple hierarchical clusters connected via a mesh overlay. Clusters are heterogeneously composed with the ability to support sensing, computation, and communication with the backend CS. The APPSYS supports autonomous and semi-autonomous missions assigned to nodes across different clusters through publish-subscribe communication and coordination among clusters. This architecture offers scalability, robustness, and better management of system traffic load and resources in resource-constrained systems. The hierarchical architecture of clusters allows the top-level cluster node to apply critical software-defined control functions to monitor and mitigate network impairment. In this work, we illustrate the APPSYS architecture using a UAV swarm APPSYS conducting the CSAT mission.

Chapter 4

SNAP Framework & Implementation

The SNAP framework supplements the APPSYS architecture with robust, lightweight, and mission-oriented system communication and control plane decision-making. Control plane decision-making results in application of mission-appropriate data services that can be used to dynamically meet a mission's QoS requirements. The two fundamental attributes of the SNAP framework are name-based communication, offered through the ICN communication paradigm, and a distributed software-defined control plane instrumental in providing required data services. The SNAP framework conducts name-based communication through the publish-subscribe method using brokers. Senders (*publishers*) transmit mission messages tagged with a specific data names (*topics)* to the receivers (*subscribers*) through *brokers* that facilitate the message exchange. The brokers maintain data structures of interested subscribers and related topics and use this reference structure to correctly forward messages received from the publishers.

The SNAP framework identifies two broker types, control brokers and local brokers, with different scopes within the APPSYS. The control brokers have a global scope and facilitate inter-cluster forwarding; meanwhile, local brokers have a local scope and only operate locally within a particular node to enable communication with control brokers. Nodes can support only one broker type at a time. While a publish-subscribe paradigm doesn't typically depend on local brokers, the SNAP framework provisions brokers at each node to implement extensions that create a distributed

software-defined control plane. The distributed control plane enables each node to take decisions that positively impact the mission state. The SNAP framework utilizes the distributed control plane to offer data services relevant to the mission. Further, the control and local brokers provide different ranges of data services dependent on their scope within the APPSYS. In this chapter, we discuss the details of the SNAP framework and its current implementation.

4.1 Name-based Connectivity using Brokers

The SNAP framework supports two broker types, control and local brokers. The broker functionality is implemented as a common software function on nodes within the APPSYS. Both broker types utilize a common software with different features activated for each broker type. Therefore, nodes within the APPSYS can interchangeably support either broker type. For the APPSYS architecture shown in Fig. 3.9, one node in each cluster is provisioned with a control broker, while all other nodes within each cluster are provisioned with local brokers. Further, the control broker functionality may co-exist with the sensor, compute, or leader node role being supported by a node. Each cluster forms a hierarchical communication sub-network where the nodes equipped with control brokers are at the top of their respective cluster's hierarchy. The control brokers use mesh connectivity to inter-connect all clusters in the APPSYS. Certain assumptions listed later in this section govern the provisioning of all brokers within the SNAP framework design.

4.1.1 Broker Types

Control Broker

There is one primary control broker per cluster with broadcast connectivity to the entire cluster. Additionally, at least one backup node is selected to serve as the secondary control broker and provide failover support. With respect to a mission, the control broker receives mission messages from nodes located within or outside its cluster and forwards them to its list of interested subscribers. Control brokers enable robust connectivity by allowing nodes within the system to exchange messages without specific knowledge of sender or receiver end-points. They also support local data-caching of mission-critical messages for a pre-determined period of time, thus, providing disruption-tolerance and availability of critical data close to potential subscribers. The scope of the control broker is *global* and it supports both inter- and intra- cluster message dissemination. The specifics of end-to-

end message delivery through a system of control brokers between nodes participating in a mission are discussed in Section 4.1.3.

The control broker is strategically located at the top of a cluster's hierarchy and thus, has comprehensive information about the ingress and egress network traffic, including mission messages, concerning its cluster. Therefore, it is well-placed to apply impactful control plane decisions to help meet the mission QoS requirements. One of the control plane decisions at the control broker may be selecting an appropriate wireless access method suitable for inter-cluster communication concerning the mission's transmission rate and latency requirements. Therefore, in a heterogeneous system with multiple wireless access methods for inter- and intra- cluster communication, the control broker is provisioned on a cluster node with adequate real or software-defined wireless interfaces that can support multiple wireless access methods. Further, the control broker is provisioned on a node with sufficient computational and memory resources to support a mission's and system's requirements through control plane decision-making.

Local Broker

A local broker is provisioned on all nodes not serving as control brokers within the APPSYS. The local broker enables an individual node's communication with its cluster's control broker and therefore, has a *local* scope. Mission messages from the mission application software within a node are sent to its local broker using inter-process communication. Local brokers offer disruption-tolerance by caching mission messages from the mission application software for a pre-determined short period of time. This feature is leveraged to conduct Store-and-Carry-Forward (SCF) operations if the node loses network connectivity. The SNAP framework leverages the presence of local brokers within the distributed control plane. The local brokers support a small set of node-specific data services determined by the control plane.

4.1.2 Broker Mesh Overlay

The set of control brokers in an APPSYS form a mesh overlay network with rich multi-path connectivity to each other. We refer to this as the *broker mesh overlay*. The broker mesh overlay connects the clusters in the APPSYS via their control brokers. In a heterogeneous APPSYS where control brokers may have multiple choices of wireless access methods, the appropriate methods may be dynamically based on the APPSYS's data dissemination requirements. For example, an APPSYS

may use a mm-Wave mesh network to support higher data rates and ultra-low latency communication compared to an IEEE 802.11n-based wireless mesh network [93]. A potential decision by the software-defined control plane at the control brokers is to select an appropriate wireless access method based on the mission's data requirements. Therefore, the selection of control brokers may be contingent on the nodes' ability to support multiple wireless access methods.

The logical design of a UAV swarm APPSYS with three clusters (clusters 1, 2, and 3) centered around control brokers B1, B2, and B3 respectively is shown in Figure 4.1. Brokers B1, B2, and B3 form the broker overlay. It should be noted that the Control Station (CS) also incorporates a broker that may serve as either a control or a local broker depending on a mission's requirements.

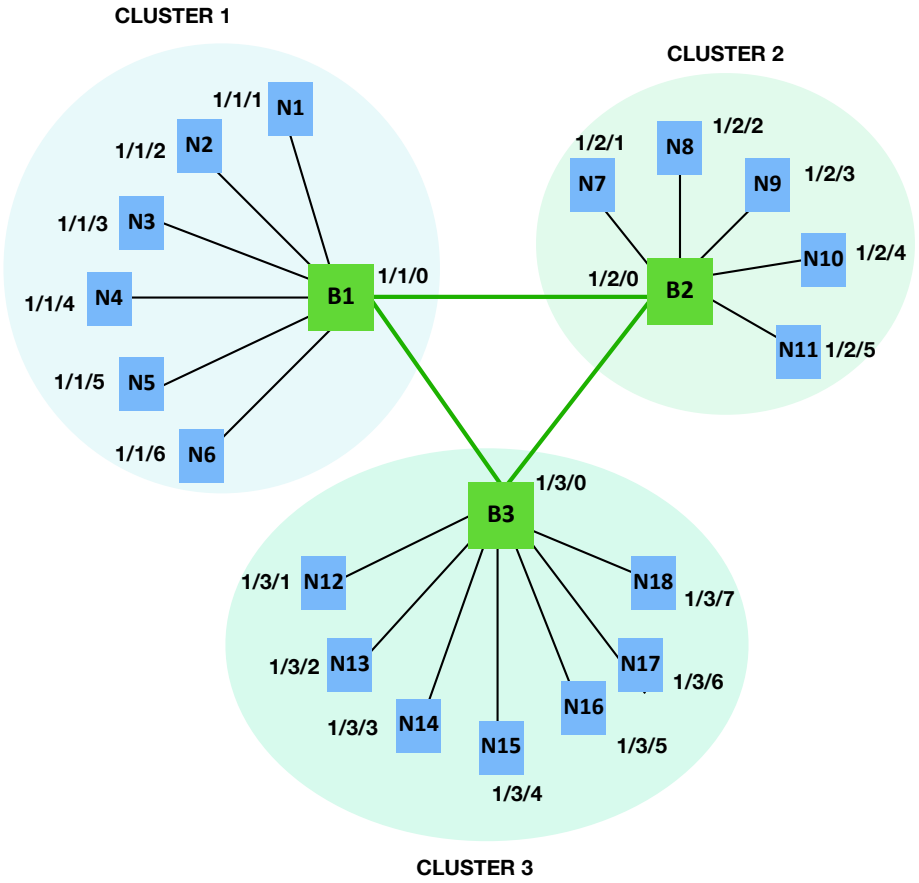


Figure 4.1: Logical Design of the UAV swarm APPSYS implementing the SNAP framework

4.1.3 Communication Setup

Naming Data

The primary requirement for ICN communication setup is assigning the APPSYS nodes and data objects human-readable and contextual names. We refer to the data object names as *topics* in this work. Messages from a publisher or subscription requests from a subscriber to the broker contain the topic name identifying the mission data that is being transmitted or requested. A topic can be a simple unique string value that can identify the type of data being requested or transmitted, e.g., the identifier `targetVehiclePosition` suffices for CSAT mission messages transmitting the target vehicle's positions. The current implementation of the SNAP framework represents topics using unique strings. However, this section also describes a hierarchical naming scheme which is more scalable and recommended for real-world APPSYS deployments. A hierarchical naming template can be used to assign both node names and data topics. Node names may be recursive, starting with a top-level APPSYS identifier assigned by the APPSYS's administrative domain. Therefore, they may be assigned hierarchical values comprising the APPSYS, cluster, and node identifiers following the convention `APPSYS_ID/ CLUSTER_ID/ NODE_ID`. For example, in Figure 4.1 , node N3 in Cluster 1 of APPSYS 1 would be named as `1/1/3`. Nodes names may be required for the named-data based exchange of node specific system state information.

Similarly, data topics can be hierarchically assigned using the template `MISSION_STATE / DATA_IDENTIFIER`. The Mission State and Data Identifier refer to the execution stage within the mission and the an identifier for the data included in the named data object. We assume that a mission includes predefined lists of mission states and data identifiers to assist with naming. Like wildcards in MQTT, topics can use a wildcard character such as an asterisk (*) to cover all available options at a level of hierarchy. For example, in the CSAT mission, the topic `DATA/VIDEO_STREAM` can be used to subscribe to all video streams being collected and transmitted during a video-sensing mission's data collection stage; meanwhile, the topic `DATA/*` can be used to subscribe to all data being collected and transmitted during the same mission. A set of publish and subscribe messages based on the topics of interest is used to set up mission data message forwarding between sensor nodes, compute nodes, and the CS.

Name-based Forwarding

In Section 2.4, we discuss that ICN forwarding uses state information stored at each forwarding node to create a reverse path from the data source to the node that has requested the data. The SNAP framework utilizes similar stateful forwarding. A node within a cluster subscribes to a topic of interest through a subscription request to the cluster's control broker. The subscription request may utilize a time-based metric to indicate how long the request should remain active. The current implementation assumes that mission requests related to the dissemination of sensor data stay active for the length of the mission. The control broker stores each valid subscription request in its topic look-up data structure along with the name of the interface at which the request was received. Valid subscription requests are identified through checks at the control broker ensuring that the APPSYS's topic-naming conventions are followed. Similarly, a subscription request at a control broker is forwarded as a subscribe message from this control broker to other APPSYS control brokers and stored within their topic data structures. When the topic of an incoming published message matches an existing topic in a control broker's topic look-up data structure, the message is forwarded through the interface at which the request was received. For subscription requests received from nodes within the cluster, the message is sent as a broadcast message to all cluster nodes. Nodes other than a message's intended recipients are instructed to ignore incoming broadcast messages. For subscription requests received from other control brokers, the message is received by the set of control brokers forming the broker mesh overlay. The control broker may utilize available unicast, broadcast, or multicast options for forwarding within the broker mesh overlay.

State Maintenance in the Broker Overlay

The broker overlay nodes subscribe to one another using topics with wildcard characters to maintain relevant system state information among control brokers. The system state is required for control plane decision-making and is discussed in Section 4.2. Initial state exchange is proactive and may include relevant node names, topic names, and system state information required to initiate subscriptions. For the remainder of the mission, state may be reactively exchanged based on subscription requests made by control brokers.

4.1.4 Assumptions about Communication Initialization

We make the following assumptions about the initialization of a UAV swarm APPSYS communication using the SNAP framework:

1. The UAV swarm has been divided into clusters using existing mathematical solutions and required system steps. For example, a combination of a k-Connected m-Dominating sets mathematical formulation [39] and a constrained k-means clustering algorithm [25] can be used to provision control brokers and divide a swarm into clusters with a fair distribution of nodes.

2. Failover methods for the election of control brokers are in place and candidates for backup control brokers have been identified.

3. Inter- and intra- cluster connectivity is established through appropriate underlying routing protocols. Further, required communication channels between a control broker provisioned on a non-leader node and the CS has been established through the cluster leader node.

4.2 Distributed Control Plane

The SNAP framework extends the control and local brokers to create a distributed and hierarchical control plane that offers mission QoS-aware data services and forwarding. The majority of services that the distributed control plane provides are implemented at the control brokers due to their strategic location and global scope. Meanwhile, local brokers provide a small set of control plane decisions relevant to their local node. The broker software for control and local brokers is the same, however, different functionalities are activated both each broker type. Figure 4.2 illustrates the software modules that form a broker's control logic. A broker's control plane is formed of four primary modules that provide mission control abstraction, network state management, mission management, and data communication functions.

Mission Control Abstraction

The mission control abstraction enables the interaction between an APPSYS mission and the underlying system through the broker. A sensor node collecting mission data can utilize this

abstraction to communicate mission requirements to the cluster's control broker as a part of the published message's header. The APPSYS requires all mission application software to include dedicated *control fields* in the mission message header. The control fields in the header indicate information such as the mission's QoS requirements and current QoS state, relevant directives about the preferred transport protocol, and embedded scripts for mission data services. The control broker uses the mission QoS requirements and current QoS state to make forwarding decisions such as selection of suitable wireless access methods (in heterogeneous systems) to transmit mission data from the mission sensor nodes to the compute nodes and offer data services relevant to the mission. A mission may inform the control broker of a preferred transport layer protocol, e.g., preference of QUIC over UDP to transmit CSAT video steams. However, the broker does not guarantee that the preferred protocol will be available and used. The published message also includes embedded scripts to guide the control plane decisions and the data services applied at the broker. The control header fields can be adapted and extended to meet the perceived needs of a specific APPSYS or class of missions. However, the general objective remains to provide critical information about the mission to the APPSYS control plane.

Network State Management

Control plane decisions made as a result of mission information provided through relevant control fields in the message header also require up-to-data information of the APPSYS's system state. APPSYS state information is collected through the network state management module. Relevant state information includes metrics like link utilization, expected latency, and packet loss for links extending from the broker to directly reachable nodes. This module may obtain relevant information as a part of mission messages or through a dedicated set of underlying health scripts collecting system state information. The link state sub-module is responsible for analyzing the system state information and providing input to the mission manager module. In addition, the control broker periodically collects host state information at the host state sub-module. The host state refers to the internal measurements of the node hosting the control broker, including battery levels and CPU and memory utilization. The broker utilizes host state information for internal management tasks, e.g., initiating failover to the secondary control broker or offloading stored mission data from its cache to the CS.

Mission Manager & Data Communication

The mission manager combines the mission state information from the mission control abstraction and the network state information form the network state management module to implement control plane decisions at a broker. Each broker makes its control decisions independently. The data services and forwarding decisions taken by the control plane are implemented and passed on to the data plane by the mission manager. All control function modules receive data and execute control plane decisions through publish and subscribe messages transmitted using underlying UDP sockets.

Further, while control brokers utilize the full range of functionality discussed in this section, local brokers only offer limited functions relevant to a local node. For example, the local broker may provide data services like caching for disruption tolerance or apply the relevant transport protocol headers to the mission messages.

4.2.1 Consistency among Control Brokers

The broker overlay does not suffer from the consistency issue that limits distributed SDN. In distributed SDN, control plane decisions are typically applied as selection of an appropriate routing path between the sender and receiver, and informing the sender's data plan of the recommended routing path. Thus, maintaining consistency among distributed controllers is critical and remains an open issue within distributed SDN research. The use of named-data networking in the SNAP framework circumvents this consistency issue. Forwarding in the SNAP framework only depends on point-to-point or point-to-multi-point forwarding where each control broker can determine its own control broker decisions at its forwarding interface.

4.2.2 Communication Extensions for Interoperability

Missions involving a UAV swarm APPSYS and a secondary APPSYS require interactions among the two APPSYSs. These APPSYSs may be within the same or different administrative domains. We refer to all APPSYSs within the same administrative domain as an APPSYS Wireless System (AWS). The AWS is conceptually similar to Autonomous Systems (AS) utilized in traditional Internet architecture. Like AS, the AWS concept provides abstractions for interoperability among APPSYSs with different ownerships. For connectivity among APPSYSs within the same AWS,

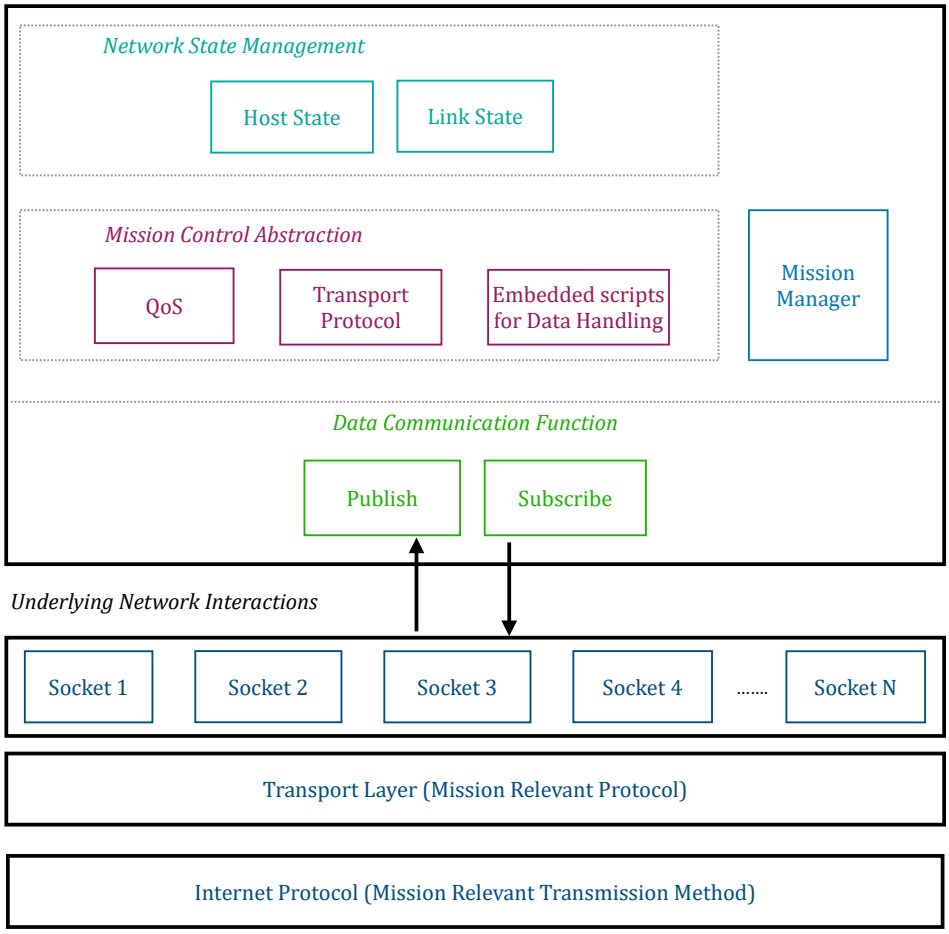


Figure 4.2: Broker Control Function Software Modules

the control plane of one of the involved APPSYSs can provision new interfaces connecting the IP subnets of the APPSYSs. For connectivity among APPSYSes in different administrative domains, the control plane establishes connectivity through the CSs of the involved APPSYSes. The CSs are expected to be connected through a traditional Internet backbone, and the underlying connectivity can be leveraged to connect the two APPSYSes.

4.3 Mission Message Set

An APPSYS mission utilizes a set of messages representing different stages of the mission. A representative message set for an autonomous or semi-autonomous mission can be drawn from

the stages of such a mission, discussed in Section 3.3. Different messages within the set may carry different control header fields to provide mission state and forwarding assistance to the APPSYS control plane. Each message carries an associated topic and is received at the subscribers using a pull-based mechanism in response to pre-existing subscriptions. At a high level, an autonomous or semi-autonomous mission in the APPSYS entails the exchange of the following messages:

- *Assignment Message (AM)*: The AM is used by the CS to assign the mission to the specific sensor and compute nodes within the APPSYS. This message is transmitted to relevant control brokers through their cluster leader nodes and then forwarded from the control brokers to the relevant sensor and compute nodes. The AM includes mission-relevant information required by the participating nodes to initiate the mission.

- *Data Message (DM)*: The DM contains the sensing data from sensor nodes participating in the mission. It is published by the sensor nodes involved in a mission. The DM header utilizes certain control header fields to indicate mission-relevant information such as mission QoS requirements, transport protocol requests, or embedded scripts to assist the control plane with providing suitable data services.

- *Process Message (PM)*: The PM contains intermediate mission results from the mission's data perception and planning stages. It is published by the compute nodes participating in the mission. Depending on the mission's requirements, a PM may be utilized in the message feedback loop to inform sensor nodes of the outcome of data analysis. When multiple compute nodes are involved in distributed perception and planning, they may coordinate to publish a single PM to the sensor nodes. For semi-autonomous missions where the CS is in the loop, PMs may be sent to inform the CS of the mission status periodically; however, the frequency of PMs to the CS is dependent on a mission's requirements. The PM header utilizes control header fields to indicate the mission's QoS state at the present time during mission operations. PMs may leverage mission feedback messages to include system state information required by the control plane.

- *Outcome Message (OM)*: The OM is used by the compute nodes to publish the final mission outcome at completion to the CS either individually or cooperatively.

With respect to the CSAT mission discussed in Section 3.3.1, the mission is assigned to

sensor nodes equipped with suitable on-board sensors and located in the proximity of the area of interest through AMs. Further, AMs are also published to several compute nodes based on the anticipated processing load and the nodes' compute capabilities. The sensor nodes collect data relevant to the CSAT mission and publish it as DMs to their respective clusters' control brokers. The brokers' control decisions depends on the information within the DMs, which includes helpful directives for data forwarding and handling. A mission DM indicates the mission's QoS requirements, e.g., throughput, latency, reliability, or other application-specific requirements to guide the broker's forwarding decision-making. A DM may indicate the mission's choice of transport protocol; if the broker's preferred transport layer protocol is available, it is used for the mission data exchange. Further, embedded scripts may be included in the DMs to guide the application of data services suitable for the mission. A PM contains the current state of a mission's QoS. For a CSAT mission, the PM utilizes the CSAT feedback messages to the sensor nodes to include system state information. Information contained in the PMs is used to prompt the control plane to offer appropriate data services in response to a mission QoS requirements not being met. Thus, control plane decisions depend on information provided by the DMs and feedback provided by the PMs. We conjecture that the impact of AMs and OMs on the system state and mission performance is trivial as compared to DMs. For example, the message sizes and transmission rates for DMs impact the system state and mission QoS. Further, PMs are critical to the SNAP framework's system state exchange mechanism. Therefore, we focus specifically on DMs and PMs within the implementation and performance evaluation of the SNAP framework.

4.4 Current Implementation

The named-data communication in the SNAP framework offers robust and disruption-tolerant connectivity to support any mission within the APPSYS. In the current implementation, we focus on the basic name-based communication setup in a UAV swarm APPSYS with two clusters as shown in Fig. 4.3. The sensor nodes, S1, S2, and S3, participating in the CSAT mission are located in cluster 1, while the compute node participating in the CSAT mission, C1, is located in cluster 2. B1 and B2 are the control brokers for clusters 1 and 2, respectively. Due to the control plane benefits offered by the control brokers, we focus our current implementation on topic-based forwarding and control plane extensions for the control brokers only; local broker functionality is

not implemented. The broker software code is implemented using the C programming language and additional code details are provided in Appendix **??**.

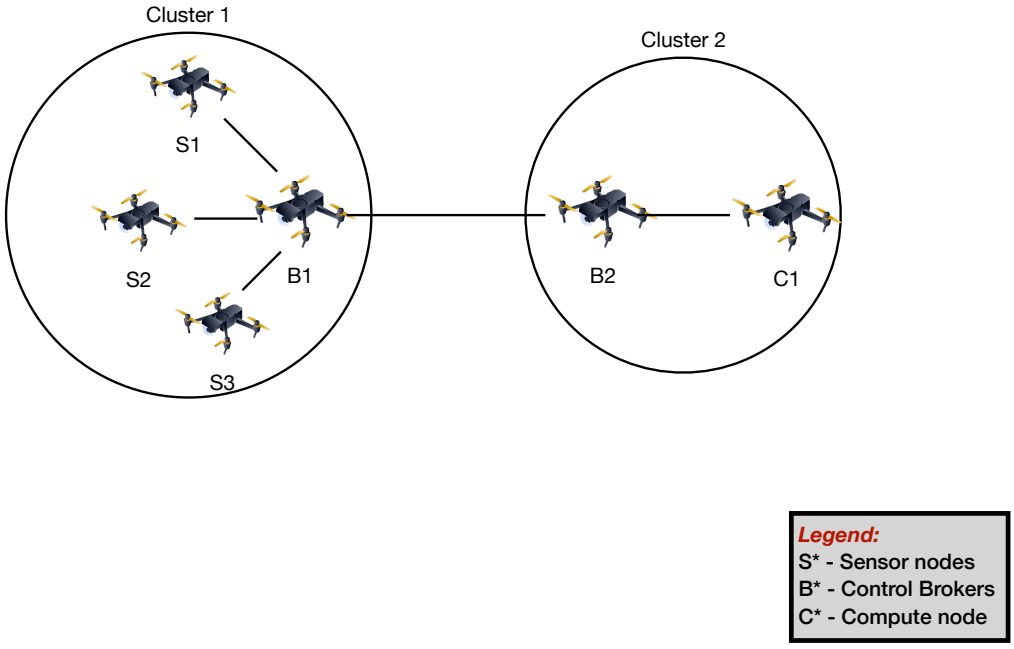


Figure 4.3: Illustrative UAV swarm APPSYS used for current implementation of the SNAP framework

The control plane abilities can be customized and adapted by modifying the set of control header fields to collect information of interest to an APPSYS. For example, a UAV swarm APPSYS with node power constraints may utilize a control header field that includes node battery level for its decision-making; meanwhile, a VANET that can derive adequate power for the vehicle battery may consider this field unnecessary. Through this flexibility, the control plane in the SNAP framework can be extended to support any class of missions or the comprehensive set of requirements most applicable to a specific APPSYS. The current implementation of the SNAP framework focuses on a data-intensive and QoS-sensitive class of missions operating in a UAV swarm APPSYS. The mission requires the collection and dissemination of substantial amounts of sensor data. Further, the mission QoS fulfillment is dependent on the quality of collected data and data dissemination characteristics such as the chosen forwarding paths or overall network traffic that impact the timely reception of messages at compute nodes. The CSAT mission being considered for this work is a representative

example within this class. The quality of target search and tracking within the CSAT mission depends on the timely reception of mission messages and quality of sensing data collected by the sensor nodes participating in the CSAT mission. Therefore, the set of data services utilized within the current implementation comprise methods to improve mission data dissemination characteristics and quality of collected data.

4.4.1 Message Header & Control Fields

The mission message header contains fields relevant to both name-based forwarding and control plane decision-making. The message header used in the current implementation of the control broker software is shown in Fig. 4.4. The fields 'nodeID' and 'sequenceNum' are 16 and 32 bit values that identify the message source and the message sequence number respectively. The fields 'ts_sec' and 'ts_nsec' are 32 bit values that hold the message timestamp in seconds and nanoseconds respectively. The CSAT mission uses these fields to keep track of incoming mission messages and compute related QoS metrics at the compute node. As a simplification, the topic is currently implemented as a unique string value with statically allocated memory based on the length of the mission's longest topic string. The 'topic' value is used by the control broker for name-based forwarding.

The fields 'isPub', 'isFwd', and 'qoSMet' are the control fields in the CSAT mission message header. These fields represent flags that can be set to 0 or 1 to provide input to the control plane at the control brokers. For 'isPub', a value of 0 represents a subscription message while a value of 1 represents a publish message. The field 'isFwd' represents the direction of mission message flow and is used for name-based forwarding to the correct interface. DMs from the CSAT sensor nodes to compute nodes have an 'isFwd' value of 1 while PMs from CSAT compute nodes to sensor nodes have an 'isFwd' value of 0. The CSAT feedback messages also use the 'qoSMet' flag in every feedback message to indicate whether the mission QoS was met. A 'qosMet' value of 0 indicates that the QoS was not met. To ensure that a subscription request does not keep getting forwarded among control brokers indefinitely, a SUB message includes a time-to-live (TTL) metric equivalent to the maximum number of hops required to connect any pair of control brokers in the APPSYS. The TTL metric is decremented at each forwarding control broker and not forwarded after it reaches a value of 0. We implement mission message transmission rates at each sensor node using an array of available transmission rates where each rate is accessible using the correct index value. In DMs,

the 'txRateIndex' represents the current transmission rate being used by the sensor nodes. In PMs, this field represents any changes to the transmission rate that the broker control plane applies. Sections 4.4.2 and 4.4.3 provide further details of how these fields are utilized.

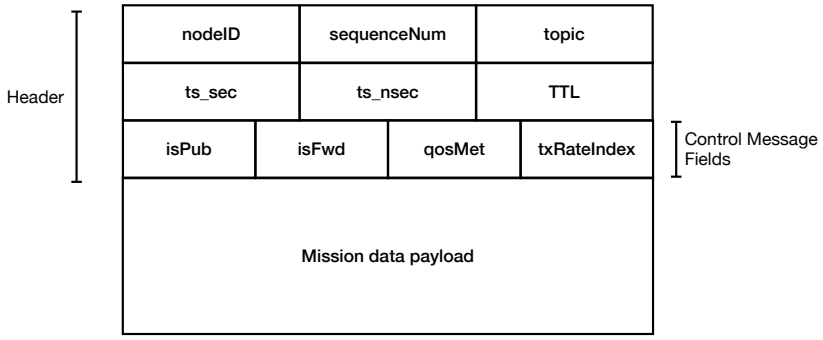


Figure 4.4: Message Header with Control Fields

4.4.2 Name-based Forwarding

The SNAP framework transmits mission message using a pull-based mechanisms where interested nodes must first send subscription requests for the topics of interest. Subscription requests relevant to a mission are assumed to stay active for the duration of mission operations. When a control broker receives a subscription request with a valid topic name, it adds the subscription topic to its topic look-up data structure. It also forwards the subscription request to other control brokers within the mesh broker overlay. In the example UAV swarm APPSYS in Fig. 4.3, the TTL value for subscription requests is set to 1.

ICN implementations widely utilize trie-based, hash table-based, or bloom filter-based schemes to implement topic look-up data structures [56]. In the current SNAP framework, we implement the topic look-up data structure as a standard trie where each topic is entered as a string key. The trie data structure is useful for topic look-ups due to its efficient lookup time and its property of storing common name prefixes only once [38]. While a standard trie suffices for our current implementation due to the use of simple string based topics and a very limited number of topics, future iterations

of the SNAP framework development expect to support a larger range of hierarchical topics and will potentially utilize data structures like path compressed character level tries [38]. For a mission message published to a control broker, the control broker looks up the published message's topic in its topic look-up data structure; if a topic exists, the message is forwarded from the control broker.

Broadly, named-data forwarding is conducted by ensuring that messages published to a forwarding node are forwarded out of the same interfaces at which subscription requests were received. This information is used to create reverse paths for forwarding published messages to subscribers. The current implementation of named-data forwarding follows this principle and applies it to a system with 2 clusters where the control brokers have two interfaces. Each control broker in Fig. 4.3 has two interfaces - an *inbound* interface connected to the cluster and an *outbound* interface connected to the broker overlay connection. Further, each message contains a direction flag represented using the 'isFwd' field to indicate the intended direction of the message. When 'isFwd' and 'isPub' are equal to 1, the message is understood to be a DM the sensor nodes and intended for the compute nodes. Therefore, the control broker B1 in Fig. 4.3 forwards it out of its outbound interface towards the broker overlay. At B2, messages with 'isFwd' and 'isPub' are understood to be intended for the compute node within cluster 2. Similarly, 'isFwd' equal to 0 represents a PM originating from the compute node and intended for the sensor nodes; B1 forwards this message out of its inbound interface while B2 forwards it out of its outbound interface. While sufficient for the performance evaluation of the SNAP framework in a two-cluster APPSYS, this method isn't readily scalable for a multi-cluster system. For a multi-cluster system, the topic look-up data structure can be appended with an identifier representing the interface at which the request was received. This feature is within the future scope of the SNAP framework implementation.

4.4.3 Control Plane Extensions

The current implementation of the SNAP framework focuses on the control plane extensions and decision-making possibilities at the control brokers within the APPSYS. The control broker offers data services relevant to a mission whose success depends on the characteristics of mission data transmission and the quality of mission data. At a high-level, the SNAP control plane at the control broker utilizes system and mission state information to take control plane decisions that dynamically enable the system to support mission QoS requirements. The system state metric in the current experimental system is the link utilization observed at the control brokers. We define link

utilization as the total network traffic being carried over a given broker link as a factor of the total broker link capacity. The link utilization metric is expressed as a percentage of the total available link capacity.

The mission QoS state is provided as a part of the mission feedback message from the compute node to the sensor nodes. The 'qosMet' field in the control message header shown in Fig. 4.4 is set to 1 if the mission QoS is met at the current time during operations, otherwise the value is set to 0. A value of 0 prompts control plane decision-making resulting in the implementation of a suitable data service. As a part of mission state information, the control broker is also provided with a metric representing the noise associated with a sensor nodes' sensors. For the current implementation, we assume that the sensor noise associated with a single on-board sensor or a combination of on-board sensors is provided through the sensor manufacturer or empirical measurement studies in controlled experimental environments. When a mission's QoS requirements are not met at any point during mission operations, the control broker utilizes the state information available to it to apply a relevant data service. Fig. 4.5 shows the set of inputs provided to the control plane and the choices of data services available within the current implementation. The control plane utilizes available system state metrics, mission QoS requirements, and mission state information like on-board sensor noise levels to provide an array of data services including transmission rate control, intelligent data fusion, and sensor noise reduction. The choice of the applied data service is flexible and more than one data service may be suitable for a specific situation. For example, intelligent data fusion and transmission rate control are both valid choices to curb the impact of high link utilization at a control broker on the mission QoS; and, sensor noise reduction and intelligent data fusion can both be applied to reduce the impact of sensor noise on the mission QoS. In such cases, the mission may provide additional state information through embedded scripts to guide the application of the appropriate data service.

4.4.3.1 Transmission Rate Control

Transmission rate control and intelligent data fusion can be applied in response to the impact of high link utilization on the mission QoS. For example, a control broker utilizing 100% of its allocated link capacity will observe increased packet queuing delays or packet loss for incoming mission messages from sensor nodes; therefore, any mission QoS requirements dependent on timely or reliable reception of mission messages will be impacted by the increased delay. Our past performance

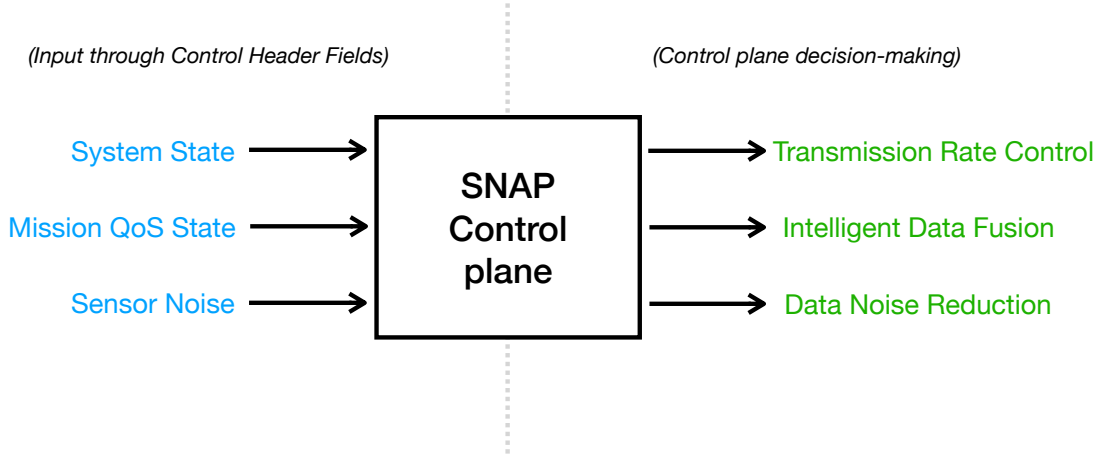


Figure 4.5: Data services offered by the SNAP framework control plane

evaluations in VANETs and single-cluster UAV swarms, discussed in Section 3.1, illustrate that severe performance degradation results from systems reaching 100% utilization due to high network traffic loads. However, the transmission rate control data service provides a way to dynamically reduce system traffic in response to high utilization, improve system performance, and support a mission's QoS requirements.

To apply transmission rate control, the control broker directs sensor nodes transmitting mission data to reduce their message transmission rates. At the sensor nodes, reducing the transmission rates also corresponds to reducing the rate at which the on-board sensors collect new data samples. To maintain uniformity of operations across all sensor nodes participating in a mission, we utilize a range of permissible transmission rates as an array where specific transmission rates can be accessed using the correct index. We assume that the control broker and the sensor nodes have *apriori* knowledge of the array of permissible transmission rates. The current transmission rate is known to the control broker through the incoming DMs from sensor nodes. Further, the control plane decision to reduce sensor nodes' transmission rates is applied by setting the index value in 'txRateIndex' to the location of the desired transmission rate in the PMs being sent as feedback from the compute nodes to the sensor nodes. Reducing the transmission rate is well-suited measure when link utilization is prohibitively high.

4.4.3.2 Intelligent Data Fusion

While reducing the transmission rate can be broadly applied to any mission collecting sensing data, it may not be well-suited for scenarios where the data demonstrates a high degree of variance, e.g., a target vehicle in CSAT moving at a fast pace with non-linear mobility. In such cases, reducing the data rate may result in loss of critical information and impact the mission objective. n such cases, intelligent data fusion serves as a suitable alternative. Using intelligent data fusion, information from collected sensor data can be preserved while reducing the number of mission messages being transmitted through the control broker. Here, we use the sensor noise information available to the control broker to prioritize the transmission of data from more reliable sensor nodes. This data service can also be utilized to limit the network resources used by a mission in an APPSYS with shared resources, and improve the quality of data being transmitted to mission end-points with respect to the sensor noise present in data from different sensor nodes.

Consider $M_1, M_2, M_3, ..., M_t$ samples or sensor data readings from sensor nodes $S_1, S_2, S_3, ..., S_t$ with the mean μ and different sensor noise errors, where, $M_t \sim N(\mu, \sigma_t^2)$ and σ_t^2 is the variance in the measurement M_t. We first use the method of Maximum Likelihood Estimation (MLE) to formulate an expression through which values of $M_1, M_2, M_3, ..., M_t$ that provide a good estimate of μ, known as μ^{MLE} can be found. Then, we utilize this expression to create a non-convex optimization problem that minimizes the difference between μ and μ^{MLE}.The formulation of the intelligent data fusion problem is shown below. This problem is solved using an exhaustive search technique.

Let $X_1, X_2, ..., X_n$ be a random sample of n sensor readings observed from a sensor node S_t with a mean μ and variance σ_t^2. Then, the probability density function as a function of μ and σ_t^2 can be written as:

$$f(x_i; \mu, \sigma_t^2) = \frac{1}{\sqrt{\sigma_t^2}\sqrt{2\pi}} exp\left[- \frac{(x_i - \mu)^2}{2\sigma_t^2} \right] \tag{4.1}$$

Therefore, the likelihood function is

$$L(\mu, \sigma_t^2) = \prod_{i=1}^{n} f(x_i; \mu, \sigma_t^2) \tag{4.2}$$

And, the log-likelihood function is

$$l(\mu, \sigma_t^2) = -\frac{-n}{2} \log \sigma_t^2 - \frac{-n}{2} \log(2\pi) - \frac{1}{2} \sum_{i=1}^{n} \frac{(x_i - \mu)^2}{\sigma_t^2} \tag{4.3}$$

We can extend equation 4.3 to solve the problem of computing the maximum likelihood estimate, μ^{MLE} for different number of samples from each sensor node subject to the individual sensor noise variance at each node's on-board sensors.

Here, we solve this problem for 3 sensor nodes, S_1, S_2, S_3 by computing μ^{MLE} for $(m+n+q)$ samples received by S_1, S_2, and S_3, where

$$\begin{aligned} X_1, X_2, ..., X_m &\sim N(\mu, \sigma_x^2) \\ Y_1, Y_2, ..., Y_n &\sim N(\mu, \sigma_y^2) \\ Z_1, Z_2, ..., Z_q &\sim N(\mu, \sigma_z^2) \end{aligned} \tag{4.4}$$

$$\begin{aligned} l(\mu, \sigma_x^2, \sigma_y^2, \sigma_z^2) &= -\frac{m}{2}\log\sigma_x^2 - \frac{n}{2}\log\sigma_y^2 - \frac{q}{2}\log\sigma_z^2 - \frac{1}{2}\left[\sum_{i=1}^{m}\frac{(x_i-\mu)^2}{\sigma_x^2} + \sum_{i=1}^{n}\frac{(y_i-\mu)^2}{\sigma_y^2} + \sum_{i=1}^{q}\frac{(z_i-\mu)^2}{\sigma_z^2}\right] \\ &= -\frac{m}{2}\log\sigma_x^2 - \frac{n}{2}\log\sigma_y^2 - \frac{q}{2}\log\sigma_z^2 - \frac{1}{2}\left(\frac{1}{\sigma_x^2}\sum_{i=1}^{m}x_i^2 + \frac{1}{\sigma_y^2}\sum_{i=1}^{n}y_i^2 + \frac{1}{\sigma_z^2}\sum_{i=1}^{q}z_i^2\right) \\ &\quad + \left(\frac{m\bar{x}}{\sigma_x^2} + \frac{n\bar{y}}{\sigma_y^2} + \frac{q\bar{z}}{\sigma_z^2}\right)\mu - \frac{1}{2}\left(\frac{m}{\sigma_x^2} + \frac{n}{\sigma_y^2} + \frac{q}{\sigma_z^2}\right)\mu^2 \end{aligned} \tag{4.5}$$

where, $\bar{x} = \frac{1}{m}\sum_{i=1}^{m} x_i, \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i, \bar{z} = \frac{1}{q}\sum_{i=1}^{q} z_i$ are the means of samples in S_1, S_2, S_3 respectively.

Taking the derivative of the log-likelihood w.r.t. μ and setting it to 0, we get

$$\frac{\partial l}{\partial \mu}(\mu, \sigma_x^2, \sigma_y^2, \sigma_z^2) = \left(\frac{m\bar{x}}{\sigma_x^2} + \frac{n\bar{y}}{\sigma_y^2} + \frac{q\bar{z}}{\sigma_z^2}\right) - \left(\frac{m}{\sigma_x^2} + \frac{n}{\sigma_y^2} + \frac{q}{\sigma_z^2}\right)\mu = 0 \tag{4.6}$$

$$\hat{\mu}^{MLE} = \frac{m\bar{x}\sigma_y^2\sigma_z^2 + n\bar{y}\sigma_x^2\sigma_z^2 + q\bar{z}\sigma_x^2\sigma_y^2}{m\sigma_y^2\sigma_z^2 + n\sigma_x^2\sigma_z^2 + q\sigma_x^2\sigma_y^2} \tag{4.7}$$

The non-convex optimization is formulated as:

$$
\begin{aligned}
\min \quad & E[(\mu - \hat{\mu}^{MLE})^2] \\
\text{s.t.} \quad & 0 \leq m \leq N \\
& 0 \leq n \leq N \\
& 0 \leq q \leq N \\
& m + n + q = T
\end{aligned}
\tag{4.8}
$$

where, N is the number of data samples or mission messages being transmitted from each sensor node per unit time according to the current transmission rate; and, T is the total number of messages per unit time that the control broker would like to forward based on the reduced transmission rate.

The control broker applies data fusion by knowing the optimal number of messages to transmit per unit time for a given sensor node, and performing a local averaging function to reduce the current number of messages from that sensor node to the optimal number of messages. For example, the control broker would average the sensing data from every $\frac{N}{M}$ messages from S_1 before forwarded.

4.4.3.3 Sensor Noise Reduction

Sensor noise in an on-board sensor is generally defined as unwanted variations in the sensor output that are independent of variations in the actual data being captured. Sensors like optical cameras and LiDAR sensors utilized on UAVs typically come with expected noise or accuracy level in the specifications provided by the manufacturers. Further, noise-levels from multiple on-board sensors operating together contribute to the overall sensor noise. Filtering techniques like Kalman filtering or wavelet transforms are applied to reduce the noise in noisy sensor readings [55]. The objective of sensor noise reduction as a data service is to reduce the sensor noise associated with the data being forwarded by the control broker. There are numerous benefits to applying this data service. First, it provides better quality of data to missions in the APPSYS reliant on the quality of collected data. Specifically, missions where noise reduction is not applied at the mission end-points can benefit from this data service. Second, it abstracts the computation required for applying a critical compute-intensive technique like filtering away from resource-constrained nodes and on to more computationally capable control brokers. Third, the APPSYS's network and communication

resources are better utilized when the data quality of mission messages being transmitted is high. We assume that this service is provided in response to a special flag set by a mission end-point to indicate high inaccuracies in sensor data. In response, the control broker applies Kalman filtering to reduce the sensor noise in all mission messages related to the mission in question. The Kalman filtering method is discussed in detail in Section 5.1.5. For this data service, the portion of Kalman filtering relevant only to improving the estimation of the received sensor measurement is applied.

4.5 Summary

The SNAP framework supplements the APPSYS architecture to meet the communication requirements of data-intensive and QoS-sensitive missions in an APPSYS. The two key technologies used in this framework are name-based message dissemination, conducted using the publish-subscribe communication paradigm, and the extension of brokers used for name-based forwarding to create a distributed and hierarchical software-defined control plane. Further, the SNAP framework provisions two broker types: *control brokers* and *local brokers*. Each cluster comprises one node provisioned as the control broker while all other nodes function as local brokers. Control brokers facilitate intra- and inter- cluster communication; inter-cluster communication is conducted over a *broker mesh overlay* with rich multipath connectivity. Control brokers form the top level of the hierarchical control plane within a cluster and are strategically well-placed to make impactful control plane decisions that can support a mission's QoS requirements. Control plane decisions are based on system and mission state information available to the brokers through special control fields in the mission message header and underlying health scripts.

The current implementation of the SNAP framework is focused on enabling named-data forwarding within the APPSYS and control plane extensions that enable the control brokers within a UAV swarm APPSYS to support the QoS requirements of a data-intensive and QoS-sensitive CSAT mission. The mission involves transmitting sensing data as mission messages from a group of sensor nodes in one cluster to a compute node in another cluster. Forwarding is conducted through the cluster's control brokers. The control brokers accept system state, represented using link utilization at various broker links, and mission state, represented using the mission QoS state and provided as a part of the mission's feedback messages from the compute node to the sensor nodes, to make control plane decisions and implement data services that can meet a mission QoS. Further, the

control brokers are also provided with the state of sensor noise in various sensor nodes. When a mission QoS requirement is not met, the control broker may utilize one of its available data services to fulfill the mission's QoS requirements. The current implementation allows the control brokers to implement transmission rate reduction, intelligent data fusion, and sensor noise reduction as data services.

Chapter 5

Performance Evaluation of a UAV Swarm APPSYS

UAVs provide critical airborne functions in many critical domains, especially on the battlefield, where they can conduct reconnaissance, surveillance, and target search and tracking missions. UAV-supported missions on the battlefield are expected to be safety-critical, QoS-sensitive, and require the collection, transmission, and analysis of a substantial amount of sensor data. Further, UAVs utilized on the battlefield may have a varied range of size, distance and power capability, physics-based abilities, and available computational and memory resources. For example, the MQ-1 Predator drone weighing 512 kilograms with a wingspan of 16.8 meters is an example of a large UAV deployed on the battlefield [9]. In contrast, a Wasp III, weighing 6 kilograms with a wingspan of 72.3 centimeters, is an example of a small UAV deployed on the battlefield[19]. While large UAVs are more resource-capable than small UAVs, small UAVs bring numerous benefits in operations where scalable and flexible deployment is required. For example, small UAVs can better serve a mission in an urban area due to their higher maneuverability. Further, smaller UAVs are cost-effective compared to large UAVs.

Standard UAV operations on the battlefield are infrastructure-based, where each UAV communicates with the control station (CS) using a wireless radio, satellite, or data link. The UAVs transmit mission-relevant sensing data in the form of mission messages to the CS, which may enlist a computational system and a human operator for data analysis and decision-making. However,

the frequent closed-loop or human-in-the-loop decision-making over a long-range wireless link incurs high communication latency and impacts the performance of time-sensitive missions. The efficiency of UAVs on the battlefield can be increased by measures that allow a subset of the data analysis tasks to be conducted locally and limit the dependency on the CS. Thus, requiring the provisioning of additional computational capabilities on the UAVs. Additional computational capabilities can be easily provisioned for larger-sized UAVs with adequate power and physics-based capabilities. For example, a large battlefield UAV, MQ-9 Reaper, can be retrofitted with a high-performance embedded computer (HPEC) to support on-board data analysis [6]. However, small-sized UAVs cannot sustain such a solution's additional payload and power requirements. For these UAVs, computational power can be provisioned by organizing multiple small UAVs into a UAV swarm and leveraging their integrated computational capabilities in a distributed manner [84, 89].

The organization of resource-constrained small-sized UAVs as a swarm gives rise to several intellectual challenges in network and communication complexities, task allocation and resource management, and dynamic and coordinated path planning and formation control [28, 85]. The APPSYS design with the SNAP communication framework addresses network and communication complexities challenges. UAVs within a swarm are low-powered; therefore, they have limited transmission ranges and require replacement during time-consuming missions. Further, mobility patterns of airborne UAVs may result in a UAV drifting away from the swarm's transmission range. These factors result in frequent topology changes and require dynamic swarm reconfiguration. In sparse UAV swarms, topology changes may also result in network partitioning [44]. Therefore, a UAV swarm requires a robust communication framework to meet mobility and intermittent connectivity challenges. Additional network challenges are presented due to constraints on the UAV swarm's network and computational resources that result in increased processing, propagation, and transmission times. Meanwhile, a UAV swarm is expected to support data-intensive and QoS-sensitive missions conducted in a distributed manner. Network and computational resources limitation impacts the mission QoS. Thus, the communication framework utilized in the swarm must also provide support to handle data disruption and the impact of various network and communication challenges on the mission QoS requirements. The communication framework should also be flexible in response to dynamically changing network and system conditions and mission QoS requirements. Further, a prudent feature in the communication framework would be applicability to a broad range of CPSs supporting a general category of data-intensive and QoS-sensitive missions.

The aim of the APPSYS design and the SNAP communication framework is to provide underlying communication and data services that any mission in an APPSYS can leverage to fulfil its QoS requirements. Transforming a UAV swarm into a UAV swarm APPSYS can help overcome the swarm's network and communication challenges and meet the goals of a data-intensive and QoS-sensitive mission. First, ICN-based communication through brokers provides disruption tolerance and faster connection reinstatement. Within each cluster, the use of name-based broadcast communication ensures that a reconnected node can rapidly resume transmission of relevant mission data. If disconnected, the broker cache at the disconnected node can store a reasonable amount of recently collected sensor data for future transmission. Second, the SNAP framework's distributed and hierarchical control plane provides data services required for a data-intensive and QoS-sensitive mission. The control plane determines the appropriate data service dynamically based on the current state of the mission QoS, the mission's QoS requirements, and the current system state. The data services currently implemented in the SNAP framework are stated in Section 4.4.3. The available data services include *transmission rate control*, *intelligent data fusion*, and *sensor noise reduction*. Transmission rate control is generally applicable to all missions, while intelligent data fusion and sensor noise reduction are applicable to missions reliant on collection, transmission, and analysis of numeric sensor data from the sensor nodes' on-board sensors.

The present performance evaluation's objective is to assess the UAV swarm APPSYS to support the QoS requirements of data-intensive and QoS-sensitive missions, especially in adverse communication conditions where traditional methods demonstrate performance limitations. Our prior performance evaluation study using a single cluster UAV swarm and a VANET, discussed in Section 3.1, indicate that traditional communication methods like UDP-based unicast, broadcast, and publish-subscribe communication using a single broker demonstrate degradation in performance due to prohibitively high latency as overall system traffic load is increased and network resource utilization approaches 100% [49, 50]. Further, traditional methods do not offer "graceful" standardized measures to respond to network impairment. The control plane features within the SNAP framework are expected to apply data services in response to various network impairment conditions that impact mission QoS, thus, providing the ability to support mission QoS in adverse conditions. For the purpose of this evaluation, we implement a UAV swarm APPSYS with two clusters, shown in Fig. 5.1, using a physical hardware testbed. Communication in the testbed utilizes the SNAP framework implementation discussed in Section 4.4. The CSAT mission is the representative data-intensive

and QoS-sensitive mission considered in this work. To aid our study, we develop CSAT mission application software that can integrate with the APPSYS design and operate within the hardware testbed. The objective of the CSAT application is to accurately detect a moving object of interest. Further, we specifically focus the performance evaluation on the impact of control plane decisions offered at the control brokers on the system's ability to support the mission QoS requirements.

5.1 CSAT Implementation

In this section, we present details of the CSAT mission application software developed for this work. The CSAT mission utilizes the integrated sensing and compute capabilities of the UAV swarm APPSYS to search for and continuously track a target of interest. This CSAT implementation can be integrated with the APPSYS architecture and utilizes numerous UAVs distributed over multiple APPSYS clusters to accomplish mission-related functions. The primary mission-related functions for the CSAT mission are data sensing and perception, conducted by one or more sensor nodes equipped with adequate sensors, and data analysis of collected data, conducted by one or more compute nodes provisioned with sufficient compute capabilities.

To aid our study, we assume that all sensor nodes participating in the CSAT mission are located in a single cluster in the experimental system, and a single compute node located in a different cluster is used for data analysis. The system of interest for this measurement-study is illustrated in Fig. 5.1. The sensor nodes are S1, S2, and S3 in cluster 1, and the compute node is C1 in cluster 2. Nodes B1 and B2 are the control brokers for clusters 1 and 2 respectively. Communication between the sensor nodes and the compute node is through the respective clusters' control brokers. These assumptions allow us to further focus our study on the control broker B1 as the strategic point to apply control plane decision-making. The present system can be extended to incorporate multiple compute nodes by creating mappings between sensor nodes and their corresponding compute nodes, and selection of a compute node for final data aggregation and analysis. Messages from the sensor nodes to the compute node travel in the 'Forward' direction, while feedback messages from the compute node to the sensor nodes travel in the 'Feedback' direction shown in Fig. 5.1.

For the CSAT mission, participating sensor nodes collect sensor data which is forwarded to the compute nodes as *Data Messages (DMs)*. The sensor data includes the target's current position in the XY Cartesian plane. The compute node aggregates input related to the target's current position

from all DMs. It then uses Kalman filtering to predict the target location at the next time step based on the aggregated sensor input. The predicted position is transmitted in the form of *Process Messages (PMs)* to the Control Station(CS) associated with the UAV swarm APPSYS and fed back to the sensor nodes to inform the nodes' flight plans and assist with continuous target tracking. Therefore, the objective of the implemented CSAT mission is to continuously and accurately predict a target's future positions at each time step following a successful initial search confirming the target location. We assume that the sensor nodes participating in the CSAT application identify the initial target location using an existing method such as an expensive one-time sweep of the target region discussed in [81]. When at least one participating sensor node identifies the target, it uses an intra-cluster CSAT message to share the target location with other sensor nodes. This step orients all sensor nodes to the correct location and establishes trust through the confirmation of the target location by multiple sensor nodes. The CSAT application software comprises two components, *csatTx* and *csatRx*, implemented on the sensor and compute nodes respectively. The software code is implemented using the C programming language.

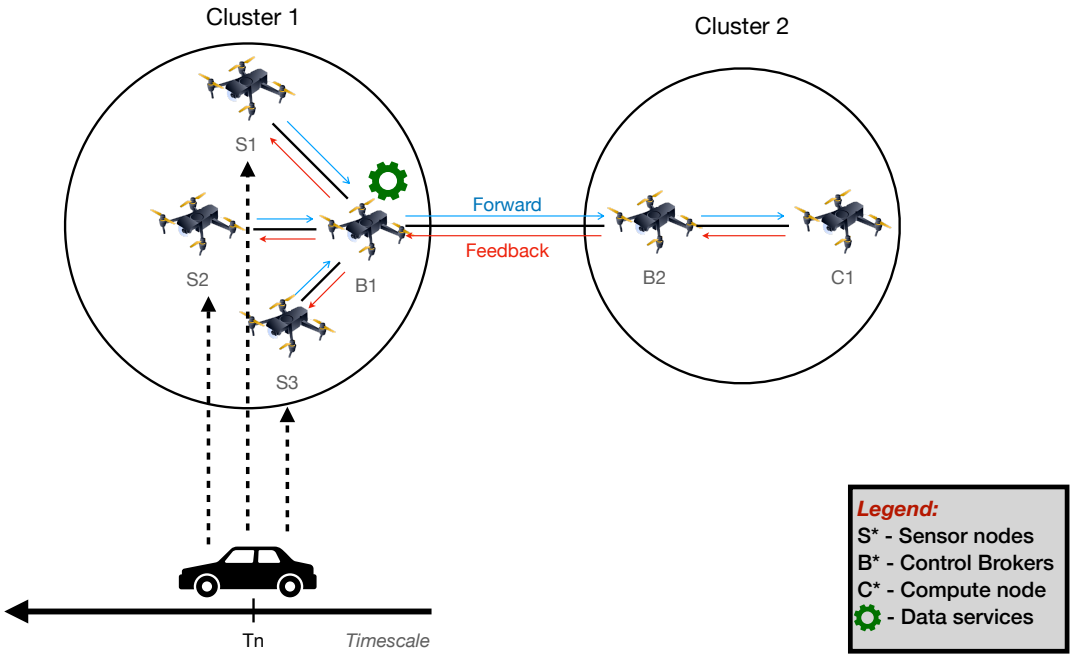


Figure 5.1: CSAT mission operating in a two cluster UAV swarm APPSYS

5.1.1 Participating UAVs

The CSAT mission using k nodes within the UAV swarm APPSYS has $(k-1)$ sensor nodes represented as S_i where $i = 1, 2, 3, ..., k-1$ and 1 compute node represented as C_j where $j = 1$. The illustrative target is an adversarial combatant vehicle, V_T, that the CSAT mission is tasked with tracking. We assume that the Control Station (CS) associated with the APPSYS assigns appropriate sensor and compute nodes to the mission *apriori*, and notifies them of the set of named-data topics relevant to CSAT mission operations. This CSAT implementation uses two simple string based topics, *csatfwd* and *csatfdb*. DMs from sensor nodes to the compute node are published with *csatfwd*, while PMs from the compute node to the sensor nodes are published with *csatfdb*. Therefore, sensor and compute nodes send subscription requests to their clusters' control brokers for the topics *csatfdb* and *csatfwd*, respectively, to set up mission communication. Their respective cluster's control brokers forward the subscription request to all control brokers in the mesh overlay to facilitate topic-based forwarding. For example, S1, S2, and S3 in Fig. 5.1 send subscription requests for the topic *csatfdb* to B1 which forwards the request to B2; and, C1 sends a subscription request for the topic *csatfwd* to B2 which forwards the request to B1. We assume that the subscription requests remain active for the duration of mission operations. Section 4.1.3 provides relevant details of name-based forwarding through clusters' control brokers in the SNAP framework. Further, we assume that the flight endurance of all nodes participating in the CSAT application is greater than the length of the mission. Regarding each UAV's flight plan, we assume that UAVs use Global Positioning Systems (GPS) or Inertial Navigation Systems (INS) for position determination [83]; and, there exists an underlying swarm control algorithm that ensures successful translation of the predicted target position into collision-free trajectories followed by the UAVs [81].

CSAT Sensor Nodes

Each sensor node S_i collects sensing data relevant to the task of locating and tracking V_T. A sensor node may have more than one on-board sensors, for example, any combination of an optical camera, infrared sensor, radar sensor, and a depth perception sensor. We assume that each sensor conducts local data perception and transmits the target vehicle's position represented as a (x, y) point in the two-dimensional Cartesian coordinate plane. For sensor nodes with multiple on-board sensors, we assume that sensor data fusion is used to provide the final measurement. However, S_i's

observed position of V_T at time t deviates from the true position of V_T as a factor of the cumulative noise error in the on-board sensors and the sensor node's distance from V_T at time t. Therefore, S_i reports a variation of the true position of V_T.

The message transmission rate is a crucial parameter at the CSAT sensor nodes. The transmission rate in our CSAT implementation is directly proportional to the sampling rate at which S_i collects sensing data. While a higher sampling rate may improve the accuracy of predicting a target's position, it may also result in higher link utilization and additional latency within the system. For cohesive operations, all S_i utilize the same transmission rate. DMs from sensor nodes carry the index of the transmission rate applied at S_i in the 'txRateIndex' header field. Control brokers apply a reduction in transmission rate by changing the index value in the same field; S_i monitors incoming messages for a change in the 'txRateIndex' header field. DMs from sensor nodes may also contain raw sensor data or a data digest within each message for additional data analysis required at the compute node or the CS. The message size of the DM reflects the additional sensor data included in each message.

CSAT Compute Nodes

Compute nodes C_j are selected based on their computational ability to support the required computation and proximity to participating sensor nodes. At time t, the compute node C_1 receives mission messages from each S_i containing its observation about V_T's position, $(x_i(t), y_i(t))$. It accumulates the incoming messages from each S_i for a time period ΔT, referred to as C_TIME in *csatTx*. The average value of the accumulated messages within time ΔT, calculated as the arithmetic mean, is used to represent a singular position for V_T at the end of each ΔT time period.

A singular V_T position at time n in the compute node, where $n = T_{n-1} + \Delta T$, is expressed as:

$$(x\prime_n, y\prime_n) = \frac{1}{M} \sum_{t=1}^{\tau} \sum_{i=1}^{k} (x_i(t), y_i(t)) \tag{5.1}$$

where, M is the total number of mission messages received from all S_i in C_TIME = ΔT; k is the total number of CSAT sensor nodes; and, each S_i transmits messages from time $t = 1, 2, 3, ...\tau$. It should be noted that each S_i may not send an equal number of messages within ΔT. Further, ΔT is carefully selected such that the target vehicle position does not change significantly during this

time and the position $(x\prime_n, y\prime_n)$ remains a fair approximation of V_T's position at time n with respect to the perceived sensor nodes' V_T positions.

Each position $(x\prime_n, y\prime_n)$ at time n is input into a Kalman Filter to form a good estimate of V_T's current position at time n and predict the next position at time $n+1$ where $n+1$ is the generally written to signify $n + \Delta T$. Fig. 5.2 illustrates the process that the compute node follows to predict V_T's position. In the figure, $(\hat{x}_{n,n}, \hat{y}_{n,n})$ represents the estimated position of V_T at time n; and, $(\hat{x}_{n+1,n}, \hat{y}_{n+1,n})$ represents the predicted position of V_T at time n for time $n+1$. The compute node sends the predicted position $(\hat{x}_{n+1,n}, \hat{y}_{n+1,n})$ to the sensor nodes as a PM. The aggregation of mission messages from the sensor nodes before using the Kalman filtering technique is motivated by conservation of computational resources required to repeatedly run Kalman filtering on resource-constrained UAVs. Section 5.1.5 discusses the target position prediction using Kalman filter in further detail.

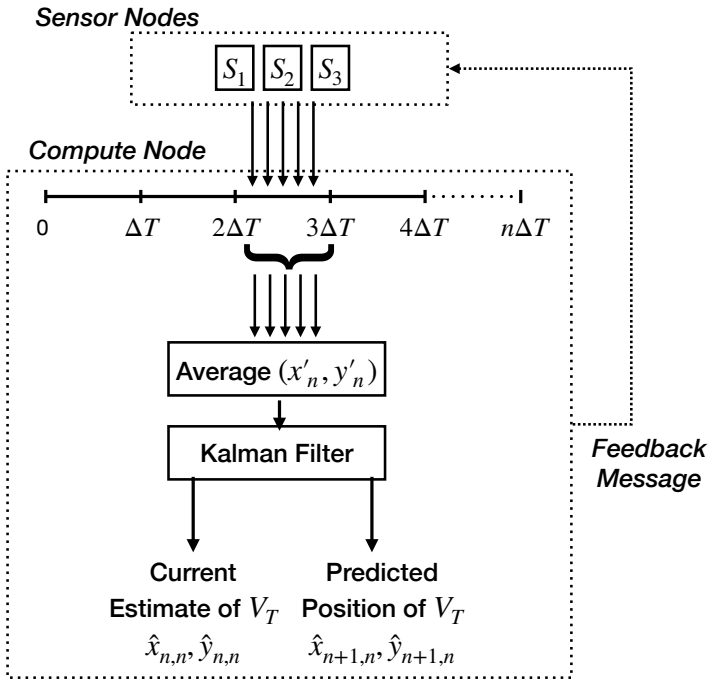


Figure 5.2: Data analysis at the CSAT compute node

5.1.2 Target of Interest

The target vehicle of interest is mobile and capable of changing its velocity. However, we assume that the average speed for the duration of the mission is ≈ 15 meters/second approximately. We assume that the vehicle follows constant acceleration dynamics. We do not possess any additional information about the target's actual speed, direction, or chosen course during the mission. We further assume that after the initial search, the sensor nodes do not lose the vehicle even though their tracking might become inaccurate.

5.1.3 QoS Requirements & Metric

This CSAT mission's success is defined by *prediction accuracy*, i.e., the accuracy of predicting the target vehicle V_T's future position at the next time step at the compute node. The compute node uses DMs received from all sensor nodes within a time-period ΔT to compute an average value that is input into the Kalman filter for predicting V_T's future position. Generally, the quality and quantity of sensing data received during this time-period leads to better accuracy in the average value input into the Kalman filter and improves the overall prediction accuracy. Factors like increased latency and packet loss experienced by DMs result in stale or lost sensing data and impact the quality and quantity of messages received at the compute node. In the *csatRx* application software, DMs with an expected arrival time $<$ or $>$ the start and end time representing the ΔT time period are considered stale and rejected. We measure expected arrival time, T_A as the $T_M + \delta$, where T_M is the DM's transmission timestamp and δ is the expected latency for the arrival of a DM within the system.

Therefore, we express CSAT's QoS requirement in terms of the One-Way Delay (OWD) experienced by DMs arriving at the compute node. We define OWD for a DM m from a sensor node S_i as

$$OWD_m = T_{arrival} - T_{transmission} \tag{5.2}$$

where, $T_{arrival}$ marks the time at which a DM is received at the compute node and $T_{transmission}$ marks the time at which it was transmitted from S_i.

For M messages received from all sensor nodes during a time-period ΔT, average OWD is

defined as

$$OWD_{avg} = \frac{1}{M} \sum_{i=1}^{M} d_m(M) \tag{5.3}$$

The compute node uses the OWD_{avg} at the end of every ΔT time to monitor the mission's QoS state. Specifically, we use an upper-bound value of acceptable OWD that has been determined experimentally as the QoS requirement. Additionally, we use distance error for the analysis of prediction accuracy observed for conducted experiments. Distance error is defined as the measured distance between V_T's actual position as time n and its position predicted by the compute node for time n at time $n-1$. Distance error is expressed as

$$d_{error} = \sqrt{(x_{Tn} - \hat{x}_{n+1,n})^2 + (y_{Tn} - \hat{y}_{n+1,n})^2} \tag{5.4}$$

where, (x_{Tn}, y_{Tn}) represents the V_T's true position at time n, and $(\hat{x}_{n+1,n}, \hat{y}_{n+1,n})$ represents the position predicted by the compute node. Prediction accuracy is generally inversely proportional to distance error, i.e., high distance error indicates low prediction accuracy.

5.1.4 CSAT Message Header

DMs and PMs used within the CSAT mission utilize the message header illustrated in Fig. 4.4. The 'nodeId', 'sequenceNum', 'ts_sec', and 'ts_nsec' are set individually at each sensor node. For all DMs, the 'topic' field is *csatfwd*, 'isPub' and 'isFwd' are set to 1, and the qosMet field is not considered. The 'txRateIndex' is set to reflect the same transmission rate at all sensor nodes. For all PMs, the 'topic' field is *csatfdb*, and the 'isPub' and 'isFwd' fields are set to 0. Each PM verifies whether the mission QoS requirement was met prior to creating a PM and sets the 'qosMet' field in the PM to either 0 (not met) or 1 (met). The control broker applies a suitable data service in response to the 'qosMet' value being set to 0 in an incoming PM. The TTL field is used for subscription messages and discussed in Section 4.1.3.

5.1.5 Target Position Prediction using Kalman filtering

The discrete Kalman filter is a recursive set of mathematical equations that is used to estimate the state of a discrete time-controlled process from noisy measurement inputs[82]. Kalman filters have been applied in mobile CPSs like VANETs and UAV swarms to the general problem of

estimating the position of an object of interest from noisy input [35, 58, 59]. The general problem that the Kalman filter addresses is that of estimating the state $\boldsymbol{X_{n+1}}$ of a process governed by the state extrapolation equation

$$\boldsymbol{x_{n+1}} = \boldsymbol{A}\boldsymbol{x_n} + \boldsymbol{w_n} \tag{5.5}$$

with a measurement $\boldsymbol{Z_n}$ expressed using the measurement equation

$$\boldsymbol{Z_n} = \boldsymbol{H}\boldsymbol{x_n} + \boldsymbol{v_n} \tag{5.6}$$

In equation 5.5, the state $\boldsymbol{X_{n+1}}$ is a $n \times 1$ matrix representing the minimal data required to describe the system's behavior. The $n \times n$ matrix $\boldsymbol{A}$ is the state transition matrix that relates the state at time step n to time step $n+1$. In 5.6, $\boldsymbol{Z_n}$ measurement vector input to the system at time step n where $\boldsymbol{x_n}$ represents the true system state. $\boldsymbol{H}$ is a $m \times n$ matrix that relates the state $\boldsymbol{x_n}$ to the measurement $\boldsymbol{Z_n}$. $\boldsymbol{H}$ is known as the Observation Matrix.

The random variables $\boldsymbol{w_n}$ and $\boldsymbol{v_n}$ in 5.5 and 5.6 respectively represent the process noise and the measurement uncertainty. They are Gaussian with probability distributions

$$p(\boldsymbol{w_n}) \sim N(0, \boldsymbol{Q}) \tag{5.7}$$

$$p(\boldsymbol{v_n}) \sim N(0, \boldsymbol{R}) \tag{5.8}$$

where, $\boldsymbol{Q}$ and $\boldsymbol{R}$ are the $n \times n$ and 2×2 process noise and measurement covariance matrices respectively.

The Kalman filter estimates the state of a system through two steps following a feedback loop: *update* and *measurement*. The equations used by the Kalman filter fall into one of these two categories. The update equations project the current state estimate and the estimate uncertainty at time step n forward to obtain the *a priori* estimates the next time step $n+1$. The measurement equations incorporate a new measurement $\boldsymbol{Z_n}$ into the *a priori* estimate from the last time step $n-1$ to obtain an improved *a posteriori* state estimate and compute the estimate uncertainty for the present time step n.

The specific equations for Kalman Filter's measurement and update steps are shown below.

For every new measurement $\boldsymbol{Z_n}$ at time step n, the *a posteriori* state estimate is expressed using the state update equation

$$\boldsymbol{\hat{x}_{n,n} = \hat{x}_{n,n-1} + K_n(Z_n - H\hat{x}_{n,n-1})} \tag{5.9}$$

where, $\boldsymbol{K_n}$ is the Kalman Gain at time n and $\boldsymbol{\hat{x}_{n,n-1}}$ is the *a priori* system state.

Kalman Gain $\boldsymbol{K_n}$ is expressed as

$$\boldsymbol{K_n = P_{n,n-1}H^T(HP_{n,n-1}H^T + R_n)^{-1}} \tag{5.10}$$

where, $\boldsymbol{P_{n,n-1}}$ is the *a priori* estimate uncertainty matrix.

The covariance update equation computes the *a posteriori* estimate uncertainty as a $n \times n$ matrix.

$$\boldsymbol{P_{n,n} = (I - K_nH)P_{n,n-1}(I - K_nH)_T + K_nR_nK_n^T} \tag{5.11}$$

Equations 5.9, 5.10, and 5.11 represent the measurement equations. A simplified version of the state extrapolation equation in equation 5.5 and a covariance extrapolation equation are used to obtain the *a priori* estimates for time step $n + 1$.

In the case of a target vehicle detection, no known control input is applied. Therefore, equation 5.5 is simplified to the form

$$\boldsymbol{\hat{x}_{n+1,n} = A\hat{x}_{n,n}} \tag{5.12}$$

The covariance extrapolation equation is expressed as

$$\boldsymbol{P_{n+1,n} = AP_{n,n}A^T + Q} \tag{5.13}$$

Equations 5.12 and 5.13 represent the update equations.

Position Prediction Algorithm at the Compute Node

The objective of the position prediction algorithm at C_j is to predict the location of the target vehicle, V_T, at time $n + 1$ given all S_i's reporting of the target's positions till time n. The Kalman filter predicts V_T's position $\hat{x}_{n+1,n}, \hat{y}_{n+1,n}$ for time $n + 1$ at time n on the XY Cartesian

plane. We assume constant acceleration dynamics for V_T. However, it is expected that any turning motions made by V_T will result in angular acceleration whose projection on the X and Y axis will not be constant. Therefore, the system state for the target vehicle V_T at time n is defined by by a 6×1 matrix as

$$\boldsymbol{x_n} = \begin{bmatrix} x_n \\ \dot{x}_n \\ \ddot{x}_n \\ y_n \\ \dot{y}_n \\ \ddot{y}_n \end{bmatrix} \tag{5.14}$$

where, x_n, $\dot{x}_n$, and $\ddot{x}_n$ represent V_T's position, velocity, and acceleration in the x axis; and, y_n, $\dot{y}_n$, and $\ddot{y}_n$ represent V_T's position, velocity, and acceleration in the y axis.

The following set of equations extrapolate the estimated state at time $n+1$ given the state at time n, where ΔT is the rate at which the Kalman filter receives a new measurement in *csatRx*.

$$\begin{aligned} \hat{x}_{n+1,n} &= \hat{x}_{n,n} + \hat{\dot{x}}_{n,n}\Delta T + \frac{1}{2}\hat{\ddot{x}}_{n,n}\Delta T^2 \\ \hat{\dot{x}}_{n+1,n} &= \hat{\dot{x}}_{n,n} + \hat{\ddot{x}}_{n,n}\Delta T \\ \hat{\ddot{x}}_{n+1,n} &= \hat{\ddot{x}}_{n,n} \\ \hat{y}_{n+1,n} &= \hat{y}_{n,n} + \hat{\dot{y}}_{n,n}\Delta T + \frac{1}{2}\hat{\ddot{y}}_{n,n}\Delta T^2 \\ \hat{\dot{y}}_{n+1,n} &= \hat{\dot{y}}_{n,n} + \hat{\ddot{y}}_{n,n}\Delta T \\ \hat{\ddot{y}}_{n+1,n} &= \hat{\ddot{y}}_{n,n} \end{aligned} \tag{5.15}$$

Therefore, $\boldsymbol{A}$ can be written as a 6×6 matrix

$$\boldsymbol{A} = \begin{bmatrix} 1 & \Delta T & 0.5\Delta T & 0 & 0 & 0 \\ 0 & 1 & \Delta T & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta T & 0.5\Delta T \\ 0 & 0 & 0 & 0 & 1 & \Delta T \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.16}$$

We assume a discrete-noise model and derive the process noise matrix $\boldsymbol{Q}$ for a model with constant acceleration motion as

$$\boldsymbol{Q} = \begin{bmatrix} \frac{\Delta T^4}{4} & \frac{\Delta T^3}{2} & \frac{\Delta T^2}{2} & 0 & 0 & 0 \\ \frac{\Delta T^3}{3} & \Delta T^2 & \Delta T & 0 & 0 & 0 \\ \frac{\Delta T^2}{2} & \Delta T & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\Delta T^4}{4} & \frac{\Delta T^3}{2} & \frac{\Delta T^2}{2} \\ 0 & 0 & 0 & \frac{\Delta T^3}{3} & \Delta T^2 & \Delta T \\ 0 & 0 & 0 & \frac{\Delta T^2}{2} & \Delta T & 1 \end{bmatrix} \sigma_a^2 \tag{5.17}$$

where, σ_a^2 represents random variance in acceleration for V_T.

The measurement, $x\prime_n, y\prime_n$, provided to the Kalman filter is in the form of a 2×1 matrix $\boldsymbol{z_n}$ written as $\begin{bmatrix} x\prime_n \\ y\prime_n \end{bmatrix}$. Therefore, from equation 5.6, the observation matrix $\boldsymbol{H}$ is a 2×6 matrix

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{5.18}$$

The measurement covariance matrix $\boldsymbol{R}$ is written as

$$\boldsymbol{R} = \begin{bmatrix} \sigma_{x\prime_n}^2 & 0 \\ 0 & \sigma_{y\prime_n}^2 \end{bmatrix} \tag{5.19}$$

where, $\sigma_{x\prime_n}^2$ and $\sigma_{y\prime_n}^2$ represent the measurement error in the x and y coordinates. Off-line tuning is performed to set constant values of $\boldsymbol{R}$ and $\boldsymbol{Q}$ suitable for V_T's trajectory.

Finally, the following initial values for $\hat{\boldsymbol{x}}_{\mathbf{0,0}}$ and $\boldsymbol{P}_{\mathbf{0,0}}$ are set.

$$\hat{\boldsymbol{x}}_{\mathbf{0,0}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{5.20}$$

$$\boldsymbol{P}_{\mathbf{0,0}} = \begin{bmatrix} 1000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1000 & 0 \\ 0 & & 0 & 0 & 0 & 1000 \end{bmatrix} \tag{5.21}$$

Algorithm 1 elaborates on the position prediction steps implemented in the *csatRx* application software code implemented on the CSAT compute node, C_j. Here, (x_i, y_i) represents the V_T's position received from any S_i; N is the total number of messages received at C_j from all S_i during time ΔT; $cTime_{start}$ and $cTime_{end}$ mark the beginning and end of every ΔT time; and, T represents current system time. With respect to Kalman filter operations, *csatRx* initializes the values for $\boldsymbol{A}$, $\boldsymbol{Q}$, $\boldsymbol{H}$, and $\boldsymbol{R}$ as specified in Equations 5.16, 5.17, 5.18, and 5.19. *curX*, *curP*, *nextX*, and *nextP* refer to $\hat{x}_{n,n}$, $\hat{P}_{n,n}$, $\hat{x}_{n+1,n}$, and $\hat{P}_{n+1,n}$ at time n. The initial values for *curX* and *curP* are set as shown in Equations 5.20 and 5.21. *lastX* and *lastP* have the same dimensions as *curX* and *curP*.

5.2 Experimental Setup and Methodology

5.2.1 Hardware Testbed

The hardware testbed setup for this evaluation represents a UAV swarm APPSYS with 2 clusters shown in Fig. 5.1. The testbed comprises commodity hardware devices representing a heterogeneous UAV swarm with a varied range of resource capabilities. Devices used in the testbed include Raspberry Pi 2, Intel NUC mini PC, and Beelink SEi 10 mini PC [15, 7, 2]. Raspberry Pi

Algorithm 1: Position Prediction Algorithm at the CSAT Compute Node

Data: $x_i, y_i, x_{Total}, y_{Total}, N, cTime_{start}, cTime_{end}, \delta, \Delta T, T$
Result: $x_{n+1,n}, y_{n+1,n}$
initialization $A, Q, H, R, curX, curP$;
begin;
$cTimeflag \leftarrow true$;
$cTime_{end} \leftarrow T_{firstArrival} + \Delta T$;
$lastX = curX$;
$lastP = curP$;
while *incoming messages from* S_i **do**
  if $cTimeflag$ **then**
    if $T_{msgArrival} + \delta \geq cTime_{start}$ **then**
      $x_{Total} \leftarrow x_{Total} + x_i$;
      $y_{Total} \leftarrow y_{Total} + y_i$;
      $N \leftarrow N + 1$;
      if $T \geq cTime_{end}$ **then**
        $cTimeflag \leftarrow false$
  $x'_n \leftarrow x_{Total}/N$;
  $y'_n \leftarrow y_{Total}/N$;
  $x_{Total}, y_{Total}, N \leftarrow 0$;
  $Z \leftarrow x'_n, y'_n$;
  $kalmanGain \leftarrow lastP^T \times (H \times lastP \times H^T + R)^{-1}$;
  $curX \leftarrow lastX + kalmanGain \times (Z - H)$;
  $curP \leftarrow$
  $(I - kalmanGain \times H) lastP \times (I - kalmanGain \times H)^T + kalmanGain \times R \times kalmanGain^T$;
  $nextX \leftarrow A \times curX$;
  $nextP \leftarrow A \times curP \times A^T + Q$;
  $x_{n+1,n} \leftarrow nextX[0][0]$;
  $y_{n+1,n} \leftarrow nextX[3][0]$;
  $lastX \leftarrow nextX$;
  $lastP \leftarrow nextP$;
end

2 nodes are used as sensor nodes (S1,S2,S3) while Intel NUC and Beelink SEi 10 mini PCs are used as the compute node C1 and the control broker B1 respectively. C1 is the primary device under study. All devices run Linux-based operating systems. Wired connectivity is established in order to decouple the present evaluation from the complexities of a specific wireless access method. However, the evaluation results are generally applicable to wireless radio access systems that use scheduled or random medium access control.

Connectivity within the testbed resembles APPSYS communication such that nodes within each cluster use the cluster's broadcast address for message transmission; messages between clusters 1 and 2 are forwarded through control brokers B1 and B2 over a point-to-point link representing the broker overlay. Wired links used in the testbed supported high link throughput of 100 Mbps. However, rate limiting measures are applied through the Linux Traffic Control utility to limit the egress and ingress bandwidth for all relevant nodes to 100 kbps [16]. Even for wireless access methods supporting higher bandwidth, it is possible for a specific UAV swarm mission to only have access to limited bandwidth due to network slicing. Further, results from the performance evaluation are applicable for any APPSYS where the mission data transmission requirements result in considerably high network traffic and high utilization of system resources. For reliable measurements of the QoS metric, OWD, all APPSYS nodes are time-syncronized using the chrony implementation of the Network Time Protocol (NTP) [8]. Additional specifications of the hardware testbed are provided in Appendix A.

5.2.2 Performance Evaluation Software

The software used for this evaluation includes the CSAT mission application software and the SNAP framework's control broker software written using the C programming language. The CSAT mission software has two main components, *csatTx* and *csatRx*. Sensor nodes use *csatTx* to first subscribe to the *csatfdb* topic of interest to them and then publish DMs with V_T's observed positions and the topic *csatfwd* to the control broker B1. Communication from *csatTx* to the control broker uses broadcast-based messages sent over a UDP socket. The message transmission rate, total number of messages to transmit, and the message size are experimental parameters that can be adjusted in *csatTx*. The code accepts a text data file with comma-separated XY positions to simulate sensing data representing its observations of V_T's position. Synchronization among sensor nodes is implemented using a busywait period that ensures that all sensor nodes start transmitting

sensing data at the same time. Busywait is also used to implement the correct transmission rate. The 'nodeID' for each S_i in the message header is unique. 'sequenceNum', 'ts_sec', and 'ts_nsec' are set individually per DM per S_i. The topic for the initial subscribe message and the following pub messages are set appropriately, and a TTL value of 1 to aid the forwarding of subscribe messages among control brokers. Each sensor node also sets the 'isPub', 'isFwd' and 'txRateIndex' values. *csatTx* inspects the 'txRateIndex' value in incoming PMs for a notification of a transmission rate change. Other header values are set to default and considered relevant.

Similarly, the *csatRx* code in C1 sends a subscribe message to enable receiving messages on the topic *csatfwd*. It receives and processes DMs for a C_TIME 1 second. The number of messages received within this time-period depend on the message transmission rates and message sizes used at the sensor nodes. For example, 15 total DMs are received at the compute node for three sensor nodes transmitting 500 Bytes messages at 20 kbps. The OWD for each incoming message, computed every time a message is received, and the total number of received messages are used to compute the average OWD for this time period. At the end of each time-period, *csatRx* compares the OWD_{avg} with the OWD QoS requirement to populate the 'qosMet' field in the PM's message header. Additionally, the target vehicle's positions obtained from all DMs received within this C_TIME are averaged and provided as input to the Kalman filtering function at the end of this time-period. The compute node sets the 'nodeID', 'sequenceNum, 'ts_sec', 'ts_nsec', 'isPub', 'isFwd', and 'qosMet' values within a PM's message header. Finally, *csatRx* outputs two data files at the end of each experiment - one with all raw samples obtained for all sensor nodes during the experiment, and another with the output of the Kalman filter algorithm at the end of every C_TIME.

The main component of the control broker software is the *broker*. The *broker* code is implemented on the control brokers B1 and B2. However, the range of functionality currently offered within the SNAP framework is implemented in the control broker under study, B1. Generally, the *broker* code accepts broadcast-based messages over a UDP socket from within its cluster. It handles subscription requests by first confirming that the 'isPub' value is set to 0, and then checking if the topic in the message header exists in the topic lookup trie data structure. A topic that doesn't previously exist is added to the trie. Similarly, it forwards published messages by first checking that an existing subscription for the topic exists. Then, messages are forwarded through appropriate interfaces using the value in the 'isFwd' flag as discussed in Section 4.4.2.

The broker uses *modes* corresponding to the three data services implemented in the SNAP

Broker Mode	Data Service
0	Forward Only
1	Transmission Rate Reduction
2	Intelligent Data Fusion
3	Sensor Noise Reduction

Table 5.1: Broker modes and corresponding data services

framework as shown in Table 5.1. Broker B2 only operates in Mode 0 within the current experimental system, while B1 can operate in any mode. Mode 0 conducts publish-subscribe forwarding using UDP sockets and sets the performance baseline for this study, and modes 1 - 3 implement data services offered through the control plane. At B1, the *broker* code inspects the message header for all incoming messages with 'isFwd'= 0, i.e., all PMs. If the 'qosMet' flag within an incoming message is set to 0, B1 sets an internal flag that indicates that a data service needs to be applied. It is assumed that the applied data service is selected based on the system state and embedded scripts provided by the sensor nodes. Within this study, the system state is expressed as B1's links' utilization and is know while the knowledge provided through embedded scripts is assumed.

5.2.3 Target Vehicle and UAV Swarm Mobility Models

We assume that the target is mobile with constant acceleration dynamics. However, turning points in target trajectory result in angular acceleration whose impact on the X and Y axis is not constant. We utilize two target vehicle profiles for our performance evaluation, PPRZ and CRV. Both models follow a mobility model based on random waypoints. However, PPRZ utilizes straight linear paths between defined waypoints, and CRV utilizes curved paths between defined waypoints. Both models follow a vehicle speed of ≈ 15 meters/sec. The PPRZ model is generated using an open-source implementation of the Paparazzi mobility model for the NS3 discrete-event simulator [13]. The CRV model is generated using the Waypoint Trajectory Generator in MATLAB [20]. Each target vehicle profile is a set of data-points taken at the rate of 100 samples/second from these models. The UAV swarm mobility models are obtained from the target vehicle profiles using computations described in this section and implemented using MATLAB.

Sensor Nodes Perception of Target Vehicle Positions

We assume that the UAV swarm center loosely follows a smoothed version of the vehicle trajectory. Each swarm center position is the moving mean of 30 vehicle position samples. Further, we assume that sensor nodes S1, S2, and S3 are equidistant from the swarm center (x_c, y_c). Their relative positions are defined as $(x_c + d, y_c + d)$, $(x_c - d, y_c - d)$, and $(x_c + d, y_c - d)$ and d is a constant distance measurement. A given sensor node, S_i, observed V_T's position with respect to V_T's true position, (x_T, y_T) at time t as

$$\begin{aligned} x_{S_i}(t) &= x_T(t) + n_i(t) \\ y_{S_i}(t) &= y_T(t) + n_i(t) \end{aligned} \tag{5.22}$$

where,

$$n_i(t) \sim N(0, \sigma_i^2) + \alpha d_i$$

and,

$$d_i = |(x_T(t) - x_i(t))^2 + (x_T(t) - x_i(t))^2|$$

Here, $(x_i(t), y_i(t))$ represent the position of S_i at time t; $n_i(t)$ represents Gaussian noise added to the true target position with a mean 0 and variance σ_i^2; α represents a constant factor set to 0.01; and, d_i is the distance between V_T and S_i. For each sensor node, S1, S2, and S3, the values σ_i^2 are set to 5, 10, and 15 respectively.

For each target vehicle profile, we use Equation 5.22 to generate a set of data points representing each S_i's observations. The *csatTx* code at each S_i accepts these data points through input from a data file. Further, multiple data files corresponding to different message transmission rates used for our study are created for each S_i. For example, the data file has 2 samples/second and 12 samples per second for a transmission rate of 10 kbps and 50 kbps, respectively, and message size of 500 Bytes. Figs. 5.3 and 5.4 illustrate the actual target trajectory for the CRV and PPRZ model along with each S_i's observed target positions for the duration of experiment. It should be noted that the observed target positions look more scattered in PPRZ as compared to CRV due to the difference in the scales of the two graphs. The mobility of the target vehicle in the PPRZ model is

confined to a smaller region as compared to the CRV model.

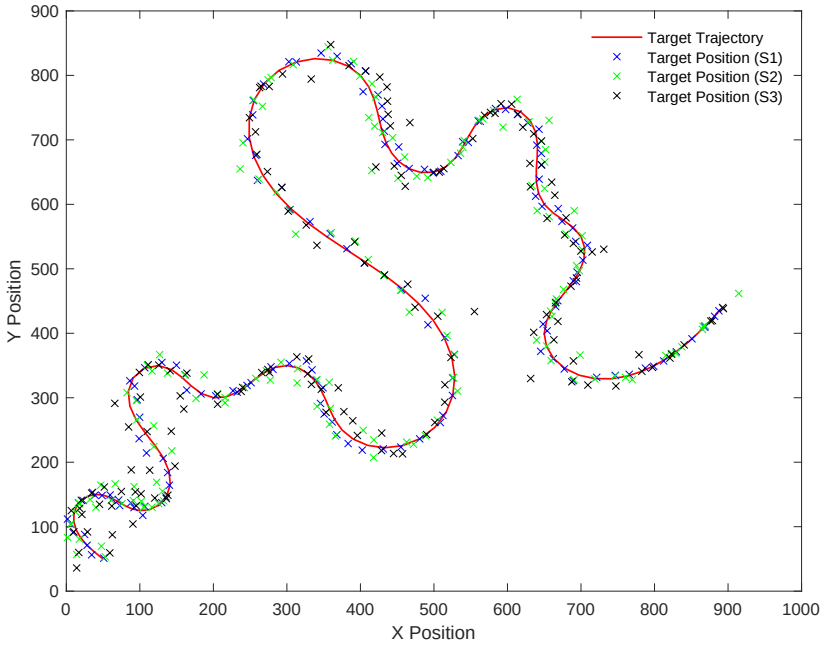


Figure 5.3: V_T's true positions vs. observations at S_i (CRV Model)

5.2.4 Experiment Sets

We first conducted a set of experiments to characterize the network and communication factors that could potentially impact the CSAT mission QoS state and quantify the impact observed on the CSAT mission accuracy. To characterize the network and communication factors, we applied increasing amounts of mission-related traffic by varying the number of sensor nodes participating in the mission and the message transmission rate utilized for the experiment. For Fig. 5.1, these experiments were designed to drive up the link utilization on B1's interfaces. As link utilization increased and network congestion ensued, messages being transmitted were expected to experience higher OWD or packet loss, thus, impacting the prediction accuracy at the compute node. Table 5.2 summarizes the experimental parameters utilized for this experiment, and Table 5.3 shows how the link utilization is expected to increase for a 100 kbps link at broker B1 as a factor of varying mission traffic load conditions. The message size remains the same throughout all conducted experiments. Through this set of experiments, we primarily aimed to identify system and mission state metrics

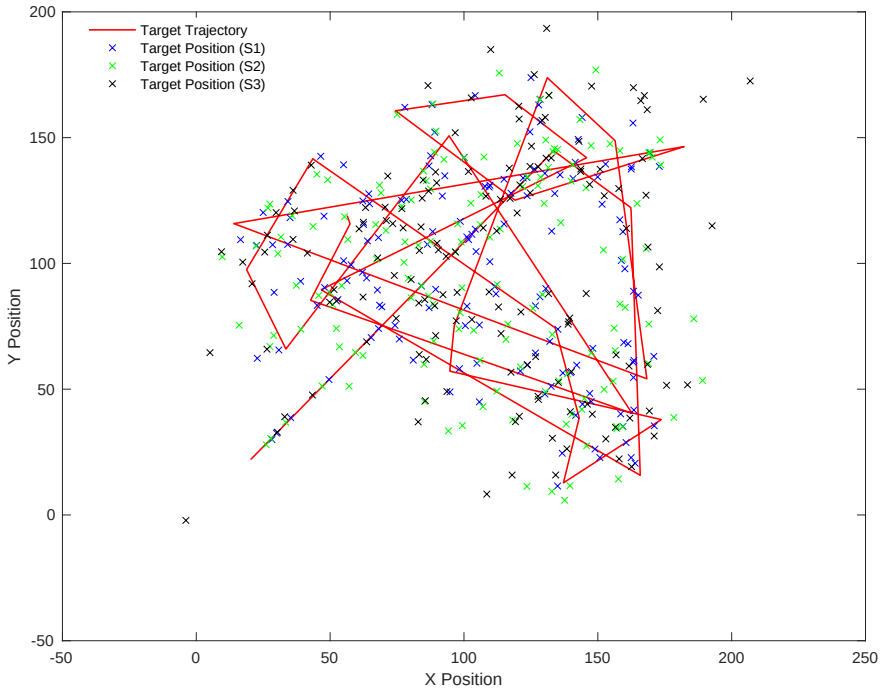


Figure 5.4: V_T's true positions vs. observations at S_i (PPRZ Model)

relevant to selecting appropriate data services at B1. Secondarily, these experiments helped identify experimental conditions through which the broker's capabilities could be assessed. For this set of experiments, the broker was set to operate in mode 0 references in Table 5.1.

We then conducted a second set of experiments to improve mission QoS through data services applied at the control broker B1. The current implementation of the SNAP communication framework at broker B1 offers three data services, referenced in Table 5.1, that can be applied by changing the mode within the broker software. These data services can be applied in response to

Experimental Parameter	Range
Message Transmission Range	10 kbps - 50 kbps
Message Size	500 Bytes + 48 Bytes (Header)
No. of Sensor Nodes	1,3
B1 Link Capacity	100 kbps - 200 kbps
B1 Link Utilization	10% - 150%
Target Vehicle Mobility Model	CRV, PPRZ

Table 5.2: Experimental parameters to characterize network conditions impacting mission QoS

No. of Senders	Message Transmission Rate	Link Utilization
1	10 kbps	10%
1	25 kbps	25%
3	10 kbps	30%
1	50 kbps	50%
3	25 kbps	75%
3	50 kbps	150%

Table 5.3: Impact of mission-related traffic conditions on link utilization at control broker B1

changing system and mission states. Further, multiple data services may apply to specific mission scenarios. Table 5.4 shows potential scenarios in which each data service can be applied. Transmission rate reduction (mode 1) is applicable when the overall link utilization at B1 exceeds 100%. When overall link utilization is $< 100\%$, however, the link capacity assigned to a specific mission is $\geq$ 100%, both transmission rate reduction (mode 1) or intelligent data fusion (mode 2) are applicable. Intelligent data fusion is also effective in reducing data noise at C1, therefore, both intelligent data fusion (mode 2) and sensor noise reduction (mode 3) are applicable for scenarios where mission QoS is impacted by high levels of sensor noise.

In this set of experiments, we implement all three data services in response to mission scenarios reflecting network impairment. Another experimental parameter used is the target vehicle mobility model. There are two objectives of this set of experiments: first, to determine that the data services implemented by B1 can mitigate network impairment and improve mission QoS; second, to identify any additional system or mission state parameters that can aid the control-plane in deciding which data service to apply. Through the results of this set of experiments, we also identify additional information within the mission state that could be helpful in determining the appropriate data service that B1 should implement.

Mission QoS State	System State	Broker Mode
qosMet=0	Aggregate Link Utilization $\geq$ 100%	Mode 1
qosMet=0	Utilization of Link Capacity assigned to mission $\geq$ 100%	Mode 1
qosMet=0	Utilization of Link Capacity assigned to mission $\geq$ 100%	Mode 2
qosMet=0	High sensor noise level	Mode 2
qosMet=0	High sensor noise level	Mode 3

Table 5.4: Applicability of different broker modes with respect to mission and system state

5.3 Results and Discussion

In this section, we provide the results from experiments discussed in Section 5.2.4. Figs. 5.5 and 5.6 show the impact of increasing mission traffic load within the UAV swarm APPSYS testbed on the OWD observed at Broker B1. B1's link capacity is set to 100 kbps for both tests. In both figures, we show OWD observed at C1 for sequential messages received from the sensor node. In cases where three sensor nodes contribute to mission traffic lot, we show the OWD observed for one of the 3 sensor nodes. However, all sensor nodes demonstrated similar OWD behavior when traffic loads is equivalent to $\leq 90\%$ link utilization at B1. Fig. 5.5 shows that as the mission traffic load from three sensor nodes (S1, S2, and S3) sending mission data is increased from 10 kbps to 30 kbps, and B1's link utilization at the corresponding link increases from 30% to 90%, the overall OWD observed at C1 is ≤ 2 milliseconds. However, we observed that at 90% utilization, $\approx 7\%$ of messages received at C1 were rejected on account of arriving outside their valid ΔT time-period.

As traffic load is further increased through 3 sensors transmitting at 150 kbps, B1's link utilization is increased to 150%. Here, we observe that as the mission traffic load reaches the total link capacity, message queues start to build at the corresponding interface and the system enters a state of congestion. This increases the queuing delays for messages arriving at C1 and contributes to an increase in OWD for each message. Fig. 5.6 shows this observation. Using observations from Figs. 5.5 and 5.6, we empirically determine that the QoS requirement for the CSAT mission can be expressed as average OWD ≤ 2 milliseconds.

The first set of conducted experiments also helps us make important observations about the characteristics of the sensing data from each model and how these characteristics respond to variability in mission operating conditions, e.g., change in message transmission rate or observed OWD. Figs. 5.7 and 5.8 show the results of C1's predictions of V_T's positions made at each time $n-1$ for time n with respect to the true position of V_T at time n. The graphs used to represent these results present the real and predicted V_T values in an XY Cartesian plane. We modified the experimental parameters, number of sensor nodes and message transmission rates, for these experiments. Additionally, we measured the average distance error observed with each set of experimental parameters used for these experiments.

Fig. 5.7 shows comparison of C1's predictions against true V_T positions for the CRV model which has high trajectory variability as it follows a curved path between defined waypoints. We

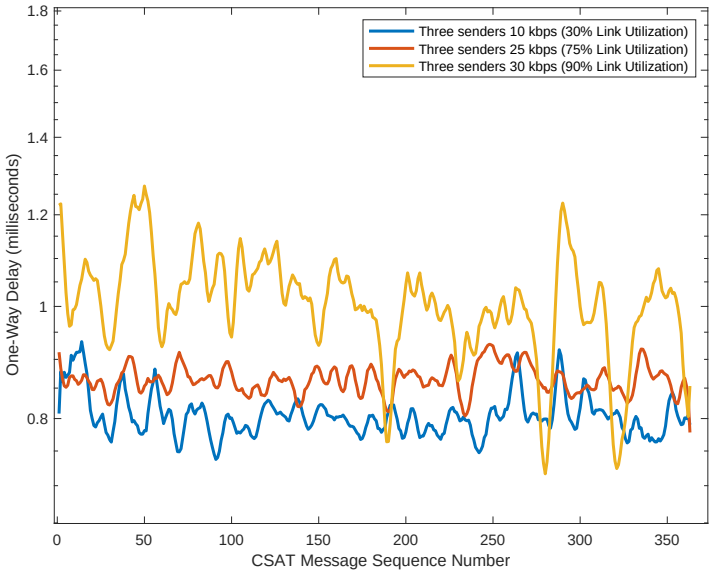


Figure 5.5: OWD observed at C1 when B1's link utilization $\leq 90\%$

present results for a single sensor node transmitting at 25 kbps and 50 kbps, and three sensor nodes transmitting at 25 kbps and 50 kbps. For this experiment, the link capacity of B1's links is set to 100 kbps. For this model, we observe that a single sensor node is transmitting at a message transmission rate of 25 kbps has an average distance error of 19.37 meters. When the message transmission rate for the single sensor node is increased from 25 kbps to 50 kbps, we observe that the increased number of sensing data samples provided to C1 improves its accuracy prediction and average distance error reduces by 3.02 meters. Further, even at a lower message transmission rate of 25 kbps, when the data used for predictions is provided through three sensors as opposed to a single sensor, the availability of additional sensing data also improves the accuracy of prediction moderately. Therefore, it can be stated that accuracy prediction improves with the amount of sensing data provided to C1. However, with 3 sensor nodes using a message transmission of 50 kbps, B1's link utilization is 150%. As shown in Fig. 5.6, this results in prohibitive increase in OWD which significantly lowers the prediction accuracy. This result is observable in Fig. 5.7 where the average distance error observed when 3 senders are sending at 50 kbps is 37.41 meters.

We present similar results for the PPRZ model in Fig. 5.8. The PPRZ model follows linear motion between defined waypoints and has lower trajectory variability as compared to the CRV

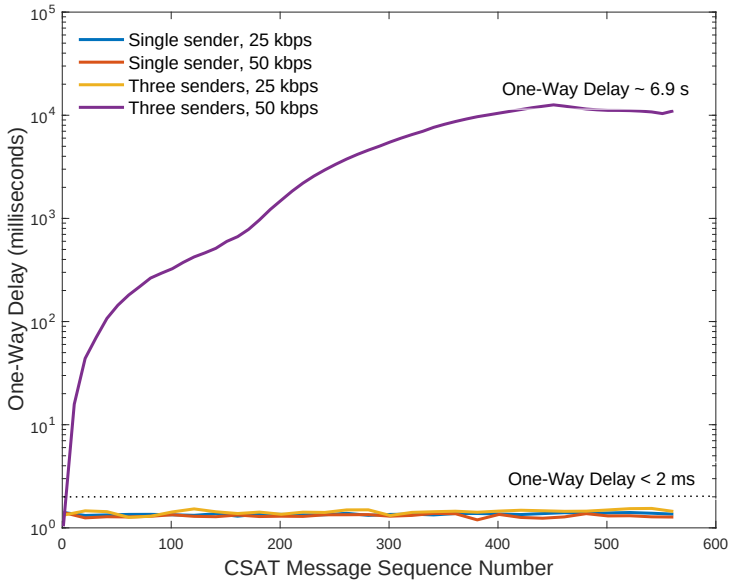


Figure 5.6: OWD observed at C1 when B1's link utilization $>$ 100%

model. Here, we observe that a single sensor node using the message transmission rate of 25 kbps has an average distance error of 17.45 meters, comparatively lower than a single sensor using the same message transmission rate in Fig. 5.7. As the message transmission rate for the single sennsor node is increased to 50 kbps, we observe that the average distance error further reduces by 1.24 meters. In contrast, changing the message transmission rate from 25 kbps to 50 kbps for a single sender in the CRV model resulted in a higher impact on prediction accuracy. Further, increasing the number of sensor nodes using the message transmission rate of 25 kbps has no impact on the prediction accuracy. With three sensor nodes using the message transmission rate of 50 kbps and B1's link utilization increasing to 150%, the PPRZ model also demonstrates a significantly high distance error of 23.91 meters. However, this distance error is lower than that observed in the CRV model for the same operating conditions.

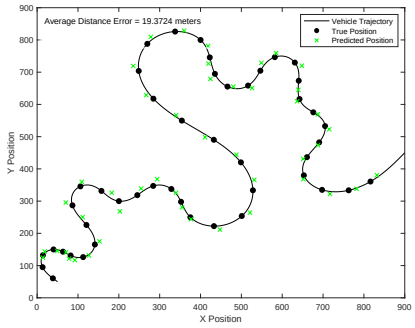
Fig. 5.9 summarizes the impact of message transmission rates, link utilization, and target's trajectory variability due to varying mobility models on the prediction accuracy at C1. We show

the average distance error observed as we increase the message transmission rate from 10 kbps to 25 kbps to 50 kbps. For these three cases, B1's link capacity is set at 200 kbps, therefore, observed link utilization for each case is 15%, 37.5%, and 74%. We observe that increasing the message transmission rate, and consequently the data sampling rate at the sensor nodes, results in improvements in accuracy across these three cases. Therefore, the CSAT mission benefits from increased message transmission rates. However, when message transmission rates cause network congestion as shown with the transmission rate 50kbps* in Fig. 5.9 where B1's link capacity is set to 100 kbps, the increased transmission rate lowers prediction accuracy.

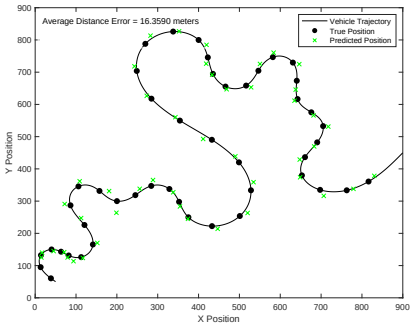
Further, we see more significant improvement in prediction accuracy as the message transmission rate from 3 sensor nodes increases for the CRV model as compared to the PPRZ model. The CRV model has a higher trajectory variability than the PPRZ model, therefore, increasing the number of data points improves the prediction accuracy more significantly. Through an additional 'trajectory variability metric' included as a part of the mission state information, the broker can discern than the appropriate data service to reduce link utilization and improve accuracy in scenarios where the target vehicle has higher variability is intelligent data fusion as opposed to transmission rate reduction.

5.3.1 Transmission Rate Reduction at B1

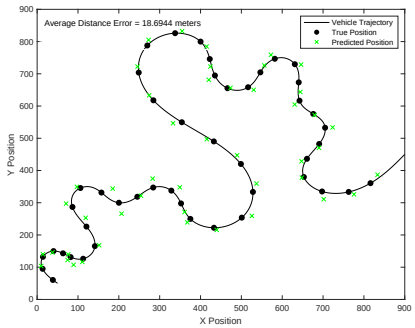
In this experiment, transmission rate reduction is applied when B1 receives a feedback message (PM) from C1 with the 'qosMet' field set to 0. In response to the updated QoS state information and system state information of $> 100\%$ link utilization, the broker directs all sensor nodes to reduce their message transmission rate. For this performance evaluation, we reduce the transmission rate by a factor of 2. Therefore, when the QoS flag is triggered due to the 3 sensor nodes, S1, S2, and S3, using a 50 kbps message transmission rate and causing a link utilization of 150% on a 100 kbps B1 link, the new transmission rate to be applied is 25 kbps. Fig. 5.10 shows the impact on OWD at C1 for each sensor node as the transmission rate reduction is applied. Before the transmission rate is applied, we observe an increasing OWD that reaches a value of ≈ 1 second for each sensor node. Fig. 5.10 marks the point in the experiment where the qoSFlag value is triggered. We note that it takes a few seconds for the applied reduction to lower broker utilization. This can be attributed to the pre-existing messages that have been queued at B1's interface for forwarding. When the queue clears, we observe a lowered link utilization at B1 and an equivalent decrease in



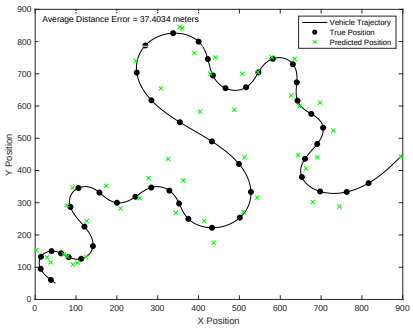
(a) Single sensor node transmitting at 25 kbps



(b) Single sensor node transmitting at 50 kbps

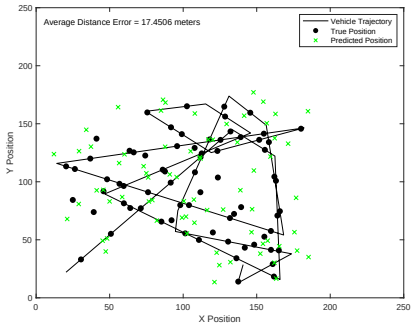


(c) Three sensor nodes transmitting at 25 kbps

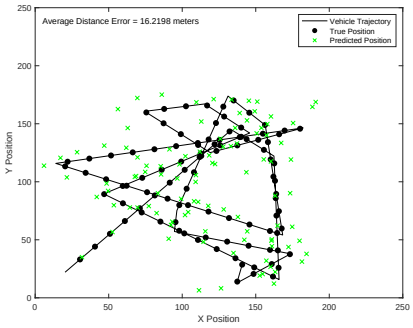


(d) Three sensor nodes transmitting at 50 kbps

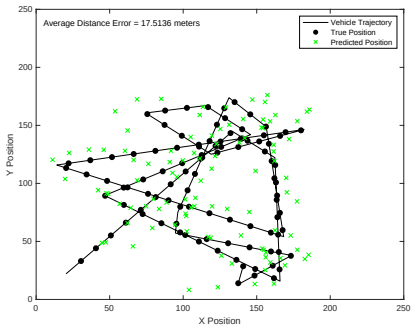
Figure 5.7: Impact of B1's link utilization on prediction accuracy (CRV model)



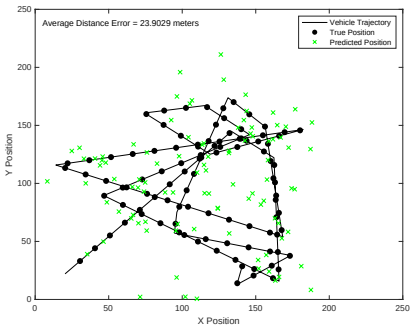
(a) Single sensor node transmitting at 25 kbps



(b) Single sensor node transmitting at 50 kbps



(c) Three sensor nodes transmitting at 25 kbps



(d) Three sensor nodes transmitting at 50 kbps

Figure 5.8: Impact of B1's link utilization on prediction accuracy (PPRZ model)

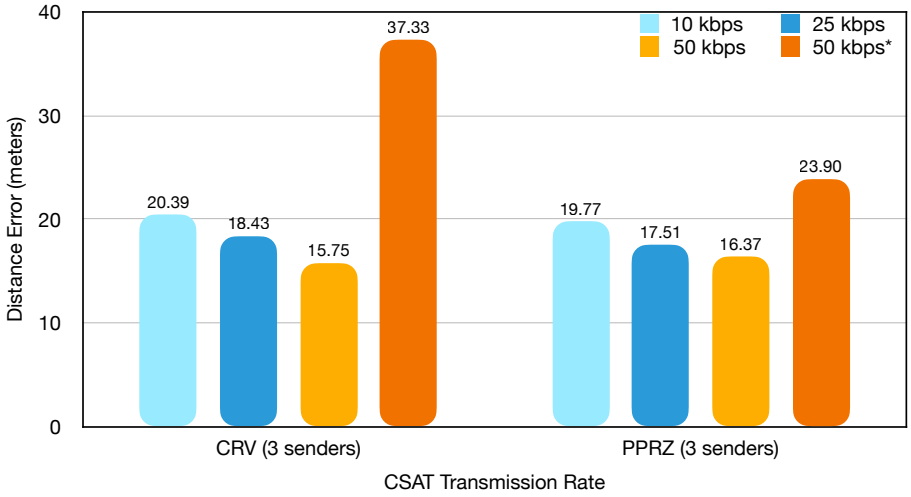


Figure 5.9: Impact of message transmission rate, link utilization, and target trajectory variability on prediction accuracy at C1

the observed OWD.

Figs. 5.11 and 5.12 show the comparative improvement in C1's prediction accuracy after the transmission rate reduction is applied. In both figures, part (a) of the figure shows corresponding results from Figs. 5.7 and 5.8, respectively, where 3 sensor nodes are using the message transmission rate of 50 kbps and B1's link utilization is 150%. Part (b) shows the comparative improvement in prediction accuracy after applying transmission rate reduction. We observe that applying transmission rate reduction significantly improves the prediction accuracy for both the CRV and PPRZ models. However, in comparison to the CRV model, the average distance error for the PPRZ model is closer to the average distance error observed in the scenario where 3 sensor nodes transmit at 50 kbps over a 200 kbps B1 link; the difference in average distance error observed for the PPRZ and the CRV models concerning the latter scenario is 2.46 meters and 4.02 meters respectively. Therefore, while transmission rate reduction brings improvement in both cases, it is more suitable for the PPRZ model with lower trajectory variability than the CRV model with higher trajectory variability.

5.3.2 Intelligent Data Fusion at B1

In this experiment, intelligent data fusion is applied in response to B1 receiving a PM with the 'qosMet' flag set to 0. Intelligent data fusion is applicable when B1 intends to reduce the link

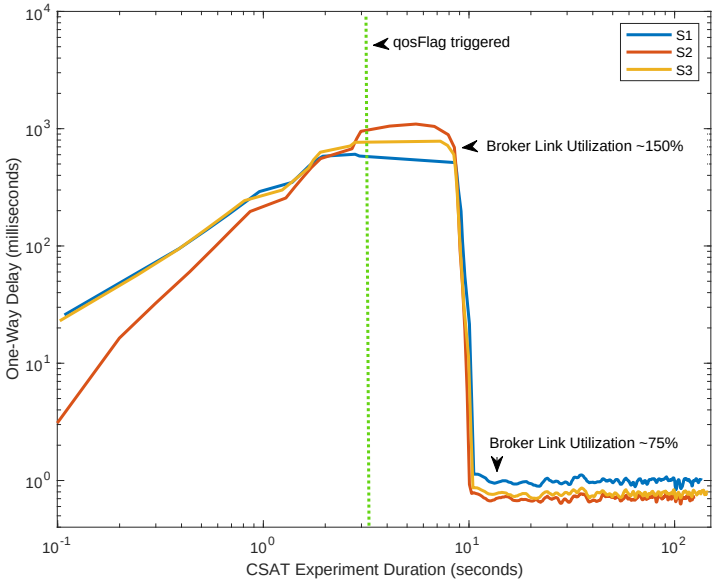
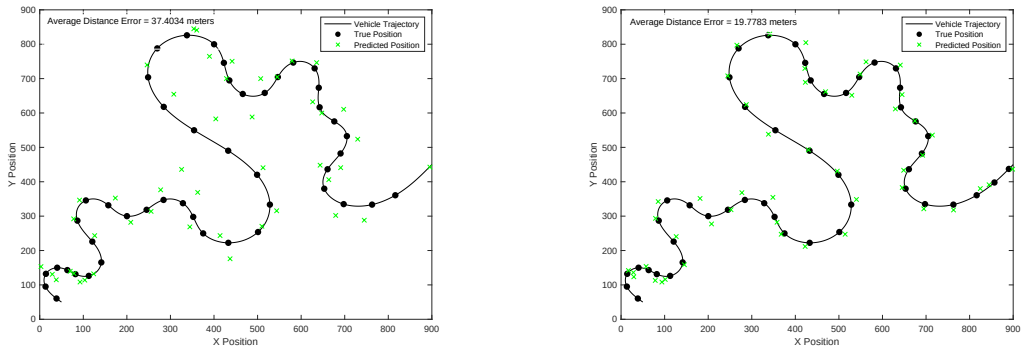


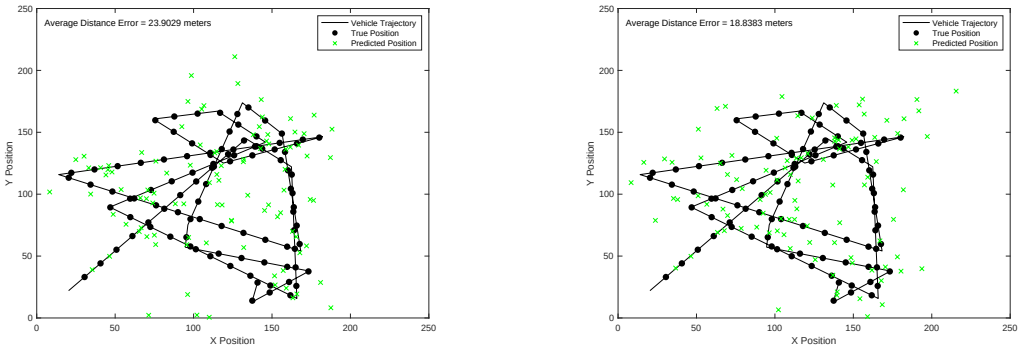
Figure 5.10: Impact of transmission rate reduction on observed OWD at C1



(a) Broker Mode 0 (No data service applied)

(b) Broker Mode 1 (Transmission rate reduction applied)

Figure 5.11: Impact of transmission rate reduction on prediction accuracy at C1 (CRV model)



(a) Broker Mode 0 (No data service applied)

(b) Broker Mode 1 (Transmission rate reduction applied)

Figure 5.12: Impact of transmission rate reduction on prediction accuracy at C1 (PPRZ model)

utilization of a specific mission, however, it is not applicable when overall link utilization at B1 is $> 100\%$. In addition to reducing the mission traffic load, intelligent data fusion also improves the quality of data being transmitted to C1. Here, we apply intelligent data fusion in response to high link utilization observed by 3 sensor nodes using a message transmission rate of 50 kbps when B1's link allocation for the CSAT mission is 100 kbps. With intelligent data fusion, B1 aims to limit the number of CSAT mission messages being transmitted out of its inter-cluster link while preserving the data quality. The reduction factor is similar to transmission rate reduction and reduces the number of messages to an equivalent of a factor of 2 reduction in transmission rate. Therefore, an aggregate of 33 messages received per second from all sensor nodes at 50 kbps is reduced to 15 messages being transmitted per second from B1's outbound interface. An aggregate of 15 messages/second from all sensor nodes is equivalent to a 25 kbps message transmission rate. B1 determines the optimal number of messages per sensor per second using an exhaustive search and knowledge of sensor noise error associated with each sensor node. For our experiment, 33 messages/second were reduced to 11 messages from S1 and 2 messages each from S2 and S3. For reference, sensor error levels for S1, S2, and S3 were set to 5, 10, and 15 meters.

Fig. 5.13 shows the impact on OWD at C1 for each sensor node as the intelligent data fusion

is applied. The observed trend for OWD is similar to Fig. 5.10 and not discussed again. Figs. 5.14 and 5.15 show the comparative improvement in C1's prediction accuracy after the intelligent data fusion is applied. In both figures, part (a) of the figure shows corresponding results from Figs. 5.7 and 5.8, respectively, where 3 sensor nodes are using the message transmission rate of 50 kbps and B1's link utilization is 150%. Part (b) shows the comparative improvement in prediction accuracy after applying intelligent data fusion. We observe that applying intelligent data reduction also significantly improves the prediction accuracy for both the CRV and PPRZ models. Compared to applying transmission rate reduction, both models show a higher improvement in prediction accuracy with respect to the scenario where 3 sensor nodes transmit at 50 kbps over a 200 kbps B1 link. Using intelligent data fusion, the difference in average distance error observed for the PPRZ and the CRV models with respect to the high-utilization scenario is 1.21 meters and 1.92 meters respectively. However, intelligent data fusion is expected to be more computationally intensive than transmission rate reduction. Therefore, the decision to chose this data service over transmission rate reduction depends on mission specifications, variability in the sensing data being collected, and computational capability of B1 during the mission.

5.3.3 Sensor Noise Reduction at B1

In this experiment, we consider a form of the implemented CSAT mission that is limited in its scope. The CSAT mission relevant to this experiment accepts incoming sensing data from sensor nodes and uses an arithmetic mean to estimate the *current* position of V_T. The incoming sensing data includes sensor noise. This form of the CSAT mission represents data-sensitive and QoS-intensive missions dependent on sensing data quality; however, without local resources to apply computations like Kalman filtering to improve data quality. We assume that a suitable metric exists within this limited form of CSAT to set the 'qosMet' flag to 0 when high levels of inaccuracy are observed in C1's data analysis.

In this scenario, B1 utilizes sensor noise reduction to reduce the sensor noise in the sensing data that is forwarded through it. This data service is implemented through Kalman filtering's estimation process at B1. Fig. 5.16 demonstrates the impact of sensor noise reduction applied at B1 on the estimation accuracy observed at C1. We define estimation accuracy as the accuracy with which C1 can estimate the current position of V_T from a simple arithmetic mean. Estimation accuracy is measured through distance error, just like prediction accuracy. In Fig. 5.16, we compare

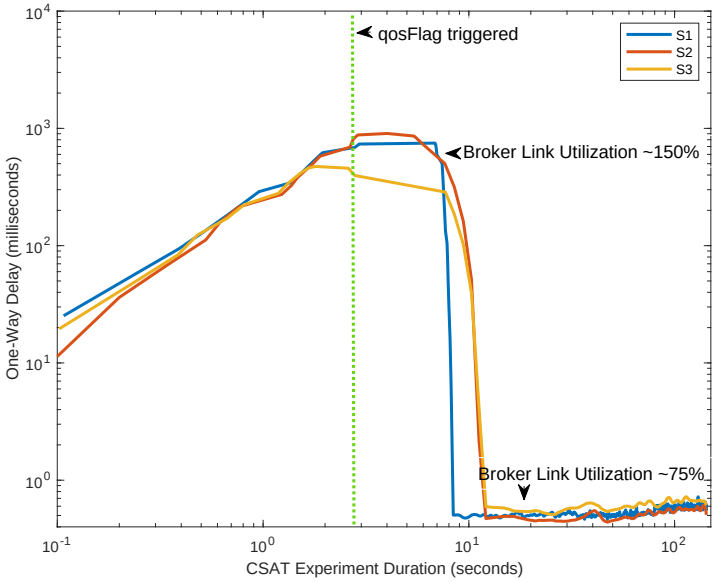
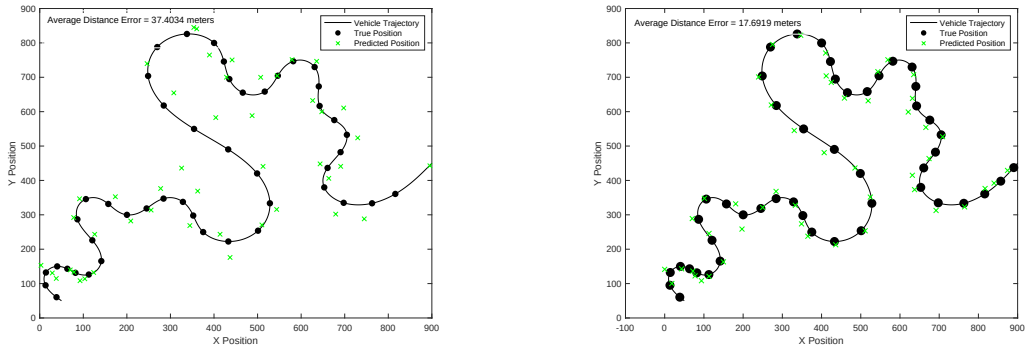


Figure 5.13: Impact of intelligent data fusion on observed OWD at C1

the estimation accuracy achieved at C1 for two cases - one when sensor noise reduction is applied prior to the arithmetic mean and another when the arithmetic mean is computed from noisy sensor data. We show that the average distance error observed when sensor noise reduction is applied is significantly lower as compared to when noisy sensing data is used.

5.4 Summary

The performance evaluation study presented in this work evaluates the ability of the APPSYS architecture and the SNAP framework to support data-intensive and QoS-sensitive missions, especially during network and system conditions that cause severe performance degradation using traditional communication methods. We implement an exemplar UAV swarm APPSYS with two clusters using a hardware testbed comprising heterogeneous commodity devices. Communication in the experimental system used the SNAP communication framework. The exemplar scenario we consider is the UAV swarm APPSYS supporting a CSAT mission. A form of the general CSAT

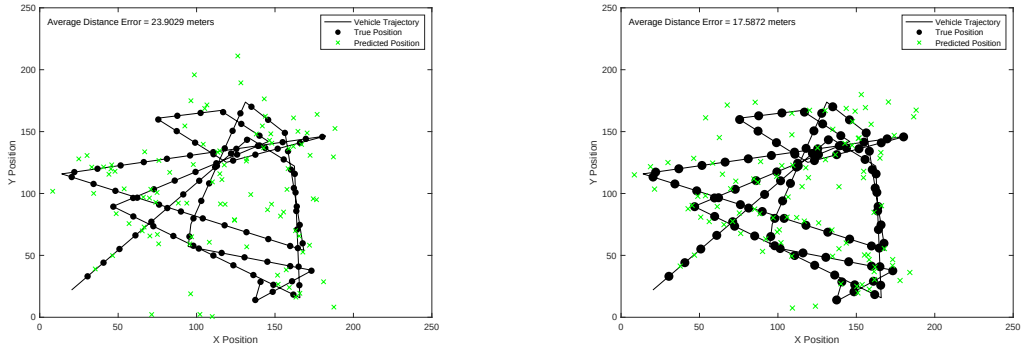


(a) Broker Mode 0 (No data service applied) (b) Broker Mode 1 (Intelligent data fusion applied)

Figure 5.14: Impact of intelligent data fusion on prediction accuracy at C1 (CRV model)

application is developed to aid this study. The developed CSAT application integrates with the APPSYS architecture and SNAP communication framework. Its objective is to continuously track a target vehicle through sensing data provided by CSAT sensor nodes and prediction of the target's future positions provided by the CSAT compute node.

The success of the CSAT mission is defined by its prediction accuracy; one set of conducted experiments establishes that OWD observed at the CSAT compute node impacts the prediction accuracy. Therefore, the CSAT mission utilizes an empirically determined upper bound of OWD $\leq$ 2 milliseconds as its QoS metric. We experimentally trigger high utilization of system resources and show that control plane decision-making and data services offered at the control brokers effectively mitigate network impairment and support the mission's QoS requirements. Experiments also illustrate how the control plane decision-making at control brokers can be strengthened through additional mission and state information. For the CSAT mission, we identify target trajectory variability are a factor using which the control broker can offer mission-appropriate data services. We also utilize a limited form of the CSAT mission to illustrate the value offered by abstracting computationally-intensive tasks such as noise reducting for sensing data as data services within the control plane.



(a) Broker Mode 0 (No data service applied) (b) Broker Mode 1 (Intelligent data fusion applied)

Figure 5.15: Impact of intelligent data fusion on prediction accuracy at C1 (PPRZ model)

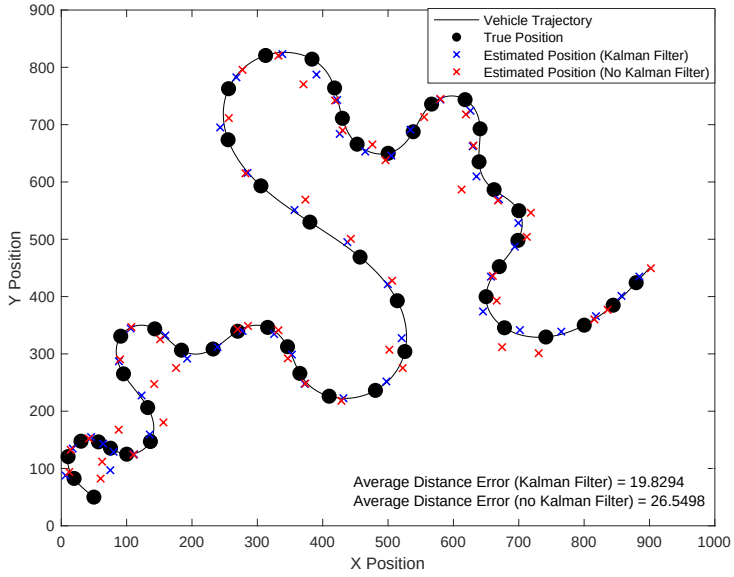


Figure 5.16: Impact of sensor noise reduction on estimation accuracy at C1 (CRV model)

Chapter 6

Conclusions & Future Work

The overarching objective of this work was to introduce, implement, and evaluate the APPSYS architecture and SNAP framework as a solution that provides robust and flexible methods for network communication and data dissemination to support the QoS requirements of a data-intensive and QoS-sensitive mission relevant to an emergent class of mobile, wireless, and resource-constrained CPSs, e.g., UAV swarms and VANETs. In the present work, we first studied these example CPSs to develop the initial architecture and system model of a general APPSYS. Our prior performance evaluation studies conducted using UAV swarms and VANETs further inform the APPSYS architecture. These studies demonstrate the performance benefits of using ICN-based communication over UDP-based unicast, requirement for strategic broker placement close to senders, and the utility of multiple brokers for resource sharing. Guided by prior work, the present APPSYS architecture is designed as a decentralized system comprising multiple hierarchical clusters connected via a mesh overlay. The high-level template for APPSYS and APPSYS mission design introduced in this work can be used to guide the development of future APPSYSs. Further, the potential adoption of these designs accross a diverse class of emergent CPSs promotes interoperability, offers opportunities to create standardized solutions, and extends the utility of CPSs to society.

The SNAP communication framework supplements the APPSYS architecture. The two primary enabling technologies in the SNAP framework are name-based message dissemination conducted using ICN's publish-subscribe communication paradigm, and the extension of brokers used for publish-subscribe communication to create a distributed software-defined control plane. The brokers include abstractions through which they interact with the mission and the underlying sys-

tem to acquire relevant mission and system state information. This knowledge enables control-plane decision-making in the form of data services that brokers can apply in response to change in the mission QoS state. While all brokers participate in the distributed control plane, the control brokers provisioned on the top-level node within a cluster's hierarchical network is well-placed to apply effective data services in response to changing mission state. The data services applied within the current implementation of the SNAP framework include transmission rate reduction, intelligent data fusion, and sensor noise reduction.

The performance evaluation focuses on the ability of the control broker's control plane extensions to support a data-intensive and QoS-sensitive mission, especially during adverse network and system conditions caused by the data transmission requirements of the mission. We conduct the study through a UAV swarm APPSYS supporting the CSAT mission. A hardware testbed and implementations of the SNAP framework and CSAT mission application software in the C programming language are used to aid the study. We implement relevant components of the SNAP communication framework, i.e., named-data forwarding and extensions that support control plane decision-making to support communication in the testbed. The study demonstrates that in operational conditions where traditional communication methods suffer severe performance impact, control plane decision-making and data services offered through the control brokers in the UAV swarm APPSYS successfully utilizes available system and mission state information to monitor and mitigate network impairment and support the CSAT mission's QoS requirements. In addition, the control plane can elevate its decision-making and selection of appropriate data services through additional mission and system state metrics. The study also demonstrates the value offered by abstracting computationally-intensive functions as data services offered by the control broker's control plane.

The APPSYS is intended as a vast, modular, and flexible system abstraction that can support a diverse range of CPSs and missions. We expect that elements of the SNAP communication framework like the control fields in message headers and offered data services will strengthen and become more comprehensive as the APPSYS concept is applied to more CPSs. Therefore, a natural progression of the present work is to apply the APPSYS concept to other CPSs. For example, a VANET APPSYS may be more dynamic and operate in a less controlled environment as compared to a UAV swarm APPSYS; therefore, the state information and data services offered through the SNAP framework would have to be extended to support the needs of a VANET APPSYS. Further,

an APPSYS design suitable for VANETs and UAV swarms can be leveraged to implement the interoperability features theoretically discussed in this work.

Offering an extended set of 'services' through the distributed control plane also offers possibilities for future work. In the present work, we abstract the computation required for sensor noise reduction to the control brokers. Resource-constrained CPSs are currently limited in using highly advantageous computationally-intensive methods that could truly augment their capabilities. Through computation abstracted to the control plane, compute-intensive functions, e.g., machine learning, could be leveraged by missions. This data service could allow the control broker to serve a function similar to a MEC node within the APPSYS. Similarly, the ability of the control brokers within a cluster to inspect and monitor all ingress and egress network traffic relevant to the cluster can be applied towards a security-focused data service. Finally, the current SNAP communication framework can also be applied to conduct a measurement-based evaluation of the efficacy of ICN in mobile and resource-constrained systems. Minimal work currently studies practical aspects of applying ICN to these systems. Through the extension of the current ICN components within the SNAP framework, an ICN-focused performance-evaluation study can be conducted to improve the ICN community's understanding of issues and challenges concerning the systems of interest in this work.

Appendix A

Hardware Testbed

This section describes additional details of the hardware testbed used for the performance evaluation in Chapter 5. The hardware testbed is shown in Fig. A.1, and the network setup is Fig. A.2. The established network connectivity resembles the UAV swarm APPSYS referenced in Figs. 4.3 and 5.1. It comprises commodity hardware that represent heterogeneous UAV swarm nodes. Raspberry Pi 2 devices are used for sensor nodes [15]. Raspberry Pi 2 devices are second-generation Raspberry Pi models with 32-bit ARM Cortex-A7 processor and a 1 GB RAM. The control broker under study, broker B1 in Fig. 5.1, is represented using a Beelink SEi 10 mini PC with an Intel Core i3 processor and 16 GB RAM [2]. Similarly, the compute node handling compute-intensive CSAT processing is represented using an Intel NUC NUC5i5RYK mini PC with an Intel Core i5 processor and 16 GB RAM [7]. Broker B2 which only conducts forwarding within the testbed is a Raspberry Pi 2 device. The Raspberry Pi nodes use the Raspberry Pi Operating System which is based on Debian. The mini-PCs use Ubuntu 18.06.6 LTS.

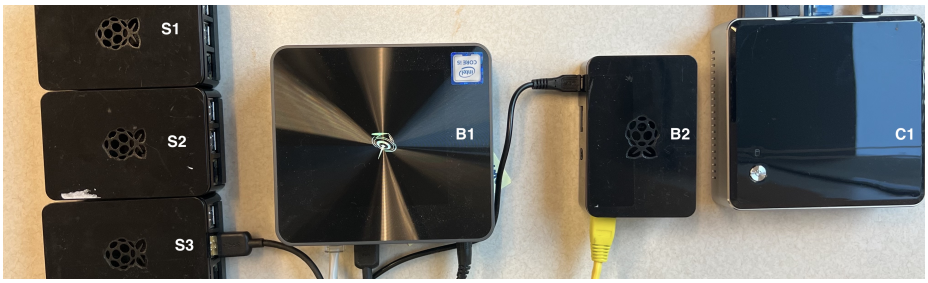


Figure A.1: Hardware testbed implementation

The different clusters are represented using different IP address subnets; the mesh overlay uses a separate subnet as well. Fig. A.2 shows the IP addressing scheme for the testbed. TP-Link network switches are used to connect different subnets. To avoid complexities arising from using a specific wireless method and for broad applicability of the performance evaluation results, nodes within the testbed are connected using wired connections. By default, the wired connections support very high throughput. However, rate limiting has been applied at all nodes using the Linux traffic control (tc) utility. We implement rate limiting using the classful Hierarchy Token Bucket (HTB) queuing discipline. System clocks of all nodes in the testbed are synchronized to a GPS receiver reference clock located two hops away from the testbed.

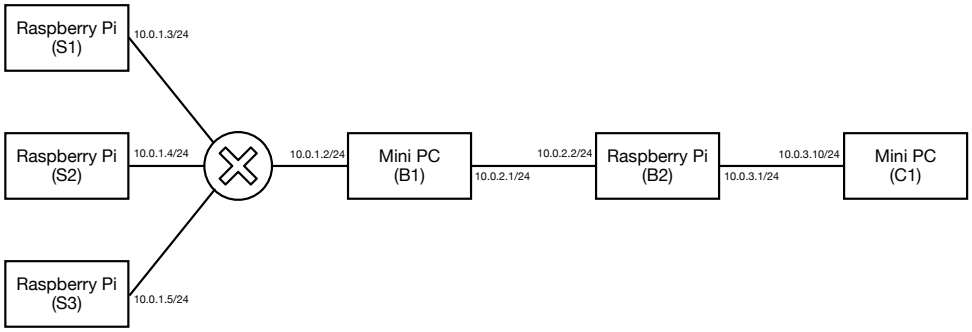


Figure A.2: Network connectivity in the hardware testbed

Bibliography

[1] Arduino products - internet of things. https://www.arduino.cc/en/Main/Products/. Accessed: 11-08-2021.

[2] Beelink sei series product specification. https://www.bee-link.com/sei8-i5-8279u-41924825. Accessed: 06-26-2022.

[3] Common open research emulator. https://github.com/coreemu/core. Accessed: 07-23-2021.

[4] Dji phantom 4. https://www.dji.com/phantom-4/info/. Accessed: 11-06-2021.

[5] Emane wiki. https://github.com/adjacentlink/emane/wiki. Accessed: 07-30-2021.

[6] Ga-asi's mq-9 rpa flies with afrl's agile condor pod. https://www.airforce-technology.com/news/ga-asis-mq-9-rpa-flies-with-afrls-agile-condor-pod/. Accessed: 07-15-2021.

[7] Intel® nuc kit nuc5i5ryk product specification. https://ark.intel.com/content/www/us/en/ark/products/83254/intel-nuc-kit-nuc5i5ryk.html. Accessed: 06-26-2022.

[8] Introduction to chrony. https://chrony.tuxfamily.org. Accessed: 06-26-2022.

[9] Mq-1b predator fact sheet. https://www.af.mil/About-Us/Fact-Sheets/Display/Article/104469/mq-1b-predator. Accessed: 07-15-2021.

[10] Mqtt: The standard for iot messaging. https://mqtt.org. Accessed: 2021-11-04.

[11] Named data networking: Motivation details. https://named-data.net/project/archoverview/. Accessed: 06-11-2022.

[12] Open networking foundation. https://opennetworking.org. Accessed: 2021-11-03.

[13] Paparazzi mobility model for ns-3. https://github.com/sbindel/PPRZM-ns3/blob/master/README.md. Accessed: 04-25-2022.

[14] Press release: Gartner says global government iot revenue for endpoint electronics and communications to total $21 billion in 2022. https://www.gartner.com/en/newsroom/press-releases/2021-06-30-gartner-global-government-iot-revenue-for-endpoint-electronics-and-communications-to-total-us-dollars-21-billion-in-2022/. Accessed: 06-05-2022.

[15] Raspberry pi personal computer kits. https://www.raspberrypi.com/products/. Accessed: 11-08-2021.

[16] tc(8) — linux manual page. https://man7.org/linux/man-pages/man8/tc.8.html/. Accessed: 06-20-2022.

[17] Tensorflow lite. https://www.tensorflow.org/lite. Accessed: 06-05-2022.

[18] Tinyml project. https://www.tinyml.org. Accessed: 06-05-2022.

[19] Wasp iii fact sheet. https://www.af.mil/About-Us/Fact-Sheets/Display/Article/104480/wasp-iii/. Accessed: 07-15-2021.

[20] Waypoint trajectory generator. https://www.mathworks.com/help/fusion/ref/waypointtrajectory-system-object.html. Accessed: 06-01-2022.

[21] Alex Afanasyev, Jeff Burke, Tamer Refaei, Lan Wang, Beichuan Zhang, and Lixia Zhang. A Brief Introduction to Named Data Networking. In *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, pages 1–6, Los Angeles, CA, October 2018. IEEE.

[22] Rahul Amin and Greg Kuperman. Pim-manet: Extension to pim for multicast routing in manets. In *MILCOM 2015-2015 IEEE Military Communications Conference*, pages 115–120. IEEE, 2015.

[23] Fetia Bannour, Sami Souihi, and Abdelhamid Mellouk. Distributed SDN Control: Survey, Taxonomy, and Challenges. *IEEE Commun. Surv. Tutorials*, 20(1):333–354, 2018.

[24] Pietro Boccadoro, Mauro Losciale, Giuseppe Piro, and Luigi Alfredo Grieco. A Standard-Compliant and Information-Centric Communication Platform for the Internet of Drones. page 6.

[25] Paul S Bradley, Kristin P Bennett, and Ayhan Demiriz. Constrained k-means clustering. *Microsoft Research, Redmond*, 20(0):0, 2000.

[26] Axel Bürkle, Florian Segor, and Matthias Kollmann. Towards Autonomous Micro UAV Swarms. *J Intell Robot Syst*, 61(1-4):339–353, January 2011.

[27] Carlo Caini, Haitham Cruickshank, Stephen Farrell, and Mario Marchese. Delay-and disruption-tolerant networking (dtn): an alternative solution for future satellite networking applications. *Proceedings of the IEEE*, 99(11):1980–1997, 2011.

[28] Mitch Campion, Prakash Ranganathan, and Saleh Faruque. UAV swarm communication and control architectures: a review. *J. Unmanned Veh. Sys.*, 7(2):93–106, June 2019.

[29] Wuhui Chen, Baichuan Liu, Huawei Huang, Song Guo, and Zibin Zheng. When uav swarm meets edge-cloud computing: The qos perspective. *IEEE Network*, 33(2):36–43, 2019.

[30] Mauro Conti, Ankit Gangwal, Muhammad Hassan, Chhagan Lal, and Eleonora Losiouk. The road ahead for networking: A survey on icn-ip coexistence solutions. *IEEE Communications Surveys & Tutorials*, 22(3):2104–2129, 2020.

[31] Kendra L. B. Cook. The silent force multiplier: The history and role of uavs in warfare. In *2007 IEEE Aerospace Conference*, pages 1–7, 2007.

[32] Peter Dely, Andreas Kassler, and Nico Bayer. OpenFlow for Wireless Mesh Networks. In *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–6, Lahaina, HI, USA, July 2011. IEEE.

[33] Rajvardhan Somraj Deshmukh and Michael Zink. An information centric networking approach for sensor to vehicular network communication in disasters. In *2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 227–234, Rome, October 2017. IEEE.

[34] Mohan Dhawan, Rishabh Poddar, Kshiteej Mahajan, and Vijay Mann. Sphinx: detecting security attacks in software-defined networks. In *Ndss*, volume 15, pages 8–11, 2015.

[35] Huifang Feng, Chunfeng Liu, Yantai Shu, and Oliver WW Yang. Location prediction of vehicles in vanets using a kalman filter. *Wireless personal communications*, 80(2):543–559, 2015.

[36] Tomonari Furukawa, Frederic Bourgault, Benjamin Lavis, and Hugh F Durrant-Whyte. Recursive bayesian search-and-tracking using coordinated uavs for lost targets. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2521–2526. IEEE, 2006.

[37] Nicole Gempton, Stefanos Skalistis, Jane Furness, Siraj Shaikh, and Dobrila Petrovic. Autonomous control in military logistics vehicles: Trust and safety analysis. In *International Conference on Engineering Psychology and Cognitive Ergonomics*, pages 253–262. Springer, 2013.

[38] Chavoosh Ghasemi, Hamed Yousefi, Kang G Shin, and Beichuan Zhang. On the granularity of trie-based data structures for name lookups and updates. *IEEE/ACM Transactions on Networking*, 27(2):777–789, 2019.

[39] Ada Gogu, Dritan Nace, Arta Dilo, and Nirvana Mertnia. Optimization problems in wireless sensor networks. In *2011 International Conference on Complex, Intelligent, and Software Intensive Systems*, pages 302–309. IEEE, 2011.

[40] Christian Grasso and Giovanni Schembra. A Fleet of MEC UAVs to Extend a 5G Network Slice for Video Monitoring with Low-Latency Constraints. *JSAN*, 8(1):3, January 2019.

[41] Cenk Gundogan, Peter Kietzmann, Thomas C. Schmidt, and Matthias Wahlisch. HoPP: Robust and Resilient Publish-Subscribe for an Information-Centric Internet of Things. In *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*, pages 331–334, Chicago, IL, USA, October 2018. IEEE.

[42] Yanxiang Guo, Xiping Hu, Bin Hu, Jun Cheng, Mengchu Zhou, and Ricky YK Kwok. Mobile cyber physical systems: Current challenges and future networking applications. *IEEE Access*, 6:12360–12368, 2017.

[43] Arpit Gupta, Laurent Vanbever, Muhammad Shahbaz, Sean P. Donovan, Brandon Schlinker, Nick Feamster, Jennifer Rexford, Scott Shenker, Russ Clark, and Ethan Katz-Bassett. SDX: a software defined internet exchange. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 551–562, Chicago Illinois USA, August 2014. ACM.

[44] Lav Gupta, Raj Jain, and Gabor Vaszkun. Survey of Important Issues in UAV Communication Networks. *IEEE Commun. Surv. Tutorials*, 18(2):1123–1152, 2016.

[45] Xiping Hu, Terry HS Chu, Henry CB Chan, and Victor CM Leung. Vita: A crowdsensing-oriented mobile cyber-physical system. *IEEE Transactions on Emerging Topics in Computing*, 1(1):148–165, 2013.

[46] Hassen Redwan Hussen, Sung-Chan Choi, Jaeho Kim, and Jong-Hong Park. Stateless and predictive geographic multicast scheme in flying ad-hoc networks. In *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 685–690. IEEE, 2017.

[47] Zeeshan Kaleem, Muhammad Yousaf, Aamir Qamar, Ayaz Ahmad, Trung Q Duong, Wan Choi, and Abbas Jamalipour. Uav-empowered disaster-resilient edge architecture for delay-sensitive communication. *IEEE Network*, 33(6):124–132, 2019.

[48] Konstantinos Katsaros, Mehrdad Dianati, and Long Le. Effective implementation of location services for VANETs in hybrid network infrastructures. In *2013 IEEE International Conference on Communications Workshops (ICC)*, pages 521–525, Budapest, Hungary, June 2013. IEEE.

[49] Manveen Kaur, GG Md Nawaz Ali, Anjan Rayamajhi, Beshah Ayalew, and Jim Martin. Network driven performance analysis in connected vehicular networks. In *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*, pages 1–6. IEEE, 2019.

[50] Manveen Kaur, Rahul Amin, and Jim Martin. The design and validation of icn-enabled hybrid unmanned aerial system.

[51] Jeongeun Kim, Seungwon Kim, Chanyoung Ju, and Hyoung Il Son. Unmanned Aerial Vehicles in Agriculture: A Review of Perspective of Platform, Control, and Applications. *IEEE Access*, 7:105100–105115, 2019.

[52] Ian Ku, You Lu, Mario Gerla, Rafael L. Gomes, Francesco Ongaro, and Eduardo Cerqueira. Towards software-defined VANET: Architecture and services. In *2014 13th Annual Mediterranean Ad Hoc Networking Workshop (MED-HOC-NET)*, pages 103–110, Slovenia, June 2014. IEEE.

[53] James F Kurose and Keith W Ross. *Computer Networking : A Top-Down Approach*. 2021.

[54] Michael Lee and Travis Atkison. VANET applications: Past, present, and future. *Vehicular Communications*, 28:100310, April 2021.

[55] Hongxu Li, Jianhua Chang, Fan Xu, Zhenxing Liu, Zhenbo Yang, Luyao Zhang, Shuyi Zhang, Renxiang Mao, Xiaolei Dou, and Binggang Liu. Efficient lidar signal denoising algorithm using variational mode decomposition combined with a whale optimization algorithm. *Remote Sensing*, 11(2):126, 2019.

[56] Zhuo Li, Yaping Xu, Beichuan Zhang, Liu Yan, and Kaihua Liu. Packet forwarding in named data networking requirements and survey of solutions. *IEEE Communications Surveys & Tutorials*, 21(2):1950–1987, 2018.

[57] Robert L Lidowski, Barry E Mullins, and Rusty O Baldwin. A novel communications protocol using geographic routing for swarming uavs performing a search mission. In *2009 IEEE International Conference on Pervasive Computing and Communications*, pages 1–7. IEEE, 2009.

[58] Chunbo Luo, Sally I McClean, Gerard Parr, Luke Teacy, and Renzo De Nardi. Uav position estimation and collision avoidance using the extended kalman filter. *IEEE Transactions on Vehicular Technology*, 62(6):2749–2762, 2013.

[59] Sandy Mahfouz, Farah Mourad-Chehade, Paul Honeine, Joumana Farah, and Hichem Snoussi. Target tracking using machine learning and kalman filter in wireless sensor networks. *IEEE Sensors Journal*, 14(10):3715–3725, 2014.

[60] Barbara M Masini, Gianluigi Ferrari, Cristiano Silva, and Ilaria Thibault. Connected vehicles: Applications and communication challenges, 2017.

[61] Luis Merino, Fernando Caballero, J Ramiro Martínez-de Dios, Joaquin Ferruz, and Aníbal Ollero. A cooperative perception system for multiple uavs: Application to automatic detection of forest fires. *Journal of Field Robotics*, 23(3-4):165–184, 2006.

[62] Luis Merino, Fernando Caballero, J.R. Martínez-de Dios, Joaquín Ferruz, and Aníbal Ollero. A cooperative perception system for multiple UAVs: Application to automatic detection of forest fires. *J. Field Robotics*, 23(3-4):165–184, March 2006.

[63] Raheeb Muzaffar, Evşen Yanmaz, Christian Raffelsberger, Christian Bettstetter, and Andrea Cavallaro. Live multicast video streaming from drones: an experimental study. *Autonomous Robots*, 44(1):75–91, 2020.

[64] Jeferson Nobre, Denis Rosario, Cristiano Both, Eduardo Cerqueira, and Mario Gerla. Toward software-defined battlefield networking. *IEEE Commun. Mag.*, 54(10):152–157, October 2016.

[65] Omar Sami Oubbati, Mohammed Atiquzzaman, Pascal Lorenz, Md Hasan Tareque, and Md Shohrab Hossain. Routing in flying ad hoc networks: survey, constraints, and future challenge perspectives. *IEEE Access*, 7:81057–81105, 2019.

[66] Omar Sami Oubbati, Abderrahmane Lakas, Pascal Lorenz, Mohammed Atiquzzaman, and Abbas Jamalipour. Leveraging communicating uavs for emergency vehicle guidance in urban areas. *IEEE Transactions on Emerging Topics in Computing*, 2019.

[67] Omar Sami Oubbati, Abderrahmane Lakas, Fen Zhou, Mesut Güneş, and Mohamed Bachir Yagoubi. A survey on position-based routing protocols for flying ad hoc networks (fanets). *Vehicular Communications*, 10:29–56, 2017.

[68] Kevin Phemius, Mathieu Bouet, and Jeremie Leguay. DISCO: Distributed multi-domain SDN controllers. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–4, Krakow, Poland, May 2014. IEEE.

[69] Siby Jose Plathottam and Prakash Ranganathan. Next generation distributed and networked autonomous vehicles. In *2018 10th International Conference on Communication Systems & Networks (COMSNETS)*, pages 577–582. IEEE, 2018.

[70] Adrian Popescu, Doru Constantinescu, David Erman, and Dragos Ilie. *A survey of reliable multicast communication*. IEEE, 2007.

[71] Guanhua Qiao, Supeng Leng, Ke Zhang, and Yejun He. Collaborative task offloading in vehicular edge multi-access networks. *IEEE Communications Magazine*, 56(8):48–54, 2018.

[72] Anjan Rayamajhi, Mizanur Rahman, Manveen Kaur, Jianwei Liu, Mashrur Chowdhury, Hongxin Hu, Jerome McClendon, Kuang-Ching Wang, Abhimanyu Gosain, and Jim Martin. Things in a fog: System illustration with connected vehicles. In *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)*, pages 1–6. IEEE, 2017.

[73] Jürgen Scherer, Saeed Yahyanejad, Samira Hayat, Evsen Yanmaz, Torsten Andre, Asif Khan, Vladimir Vukadinovic, Christian Bettstetter, Hermann Hellwagner, and Bernhard Rinner. An Autonomous Multi-UAV System for Search and Rescue. In *Proceedings of the First Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*, pages 33–38, Florence Italy, May 2015. ACM.

[74] M. Scheutz, P. Schermerhorn, and P. Bauer. The utility of heterogeneous swarms of simple uavs with limited sensory capacity in detection and tracking tasks. In *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.*, pages 257–264, Pasadena, CA, USA, 2005. IEEE.

[75] Baraa T Sharef, Raed A Alsaqour, and Mahamod Ismail. Vehicular communication ad hoc routing protocols: A survey. *Journal of network and computer applications*, 40:363–396, 2014.

[76] Sabrina Sicari, Alessandra Rizzardi, Luigi Alfredo Grieco, and Alberto Coen-Porisini. A secure icn-iot architecture. In *2017 IEEE international conference on communications workshops (ICC workshops)*, pages 259–264. IEEE, 2017.

[77] John Skovronski and Kenneth Chiu. An Ontology-Based Publish-Subscribe Framework. page 10.

[78] Tarik Taleb, Konstantinos Samdanis, Badr Mada, Hannu Flinck, Sunny Dutta, and Dario Sabella. On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials*, 19(3):1657–1681, 2017.

[79] Kun Tan, Qian Zhang, and Wenwu Zhu. Shortest path routing in partially connected ad hoc networks. In *GLOBECOM'03. IEEE Global Telecommunications Conference (IEEE Cat. No. 03CH37489)*, volume 2, pages 1038–1042. IEEE, 2003.

[80] Sasu Tarkoma, Mark Ain, and Kari Visala. The publish/subscribe internet routing paradigm (psirp): Designing the future internet architecture. In *Future Internet Assembly*, pages 102–111, 2009.

[81] Patrick Vincent and Izhak Rubin. A framework and analysis for cooperative search using UAV swarms. In *Proceedings of the 2004 ACM symposium on Applied computing - SAC '04*, page 79, Nicosia, Cyprus, 2004. ACM Press.

[82] Greg Welch, Gary Bishop, et al. An introduction to the kalman filter. 1995.

[83] Richard Wise and Rolf Rysdyk. Uav coordination for autonomous target tracking. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, page 6453, 2006.

[84] Fei Xiong, Aijing Li, Hai Wang, and Lijuan Tang. An SDN-MQTT Based Communication System for Battlefield UAV Swarms. *IEEE Commun. Mag.*, 57(8):41–47, August 2019.

[85] Yanli Yang, Marios M. Polycarpou, and Ali A. Minai. Multi-UAV Cooperative Search Using an Opportunistic Learning Method. *J. Dyn. Sys., Meas., Control*, 129(5):716, 2007.

[86] Kok-Kiong Yap, Masayoshi Kobayashi, Rob Sherwood, Te-Yuan Huang, Michael Chan, Nikhil Handigol, and Nick McKeown. OpenRoads: empowering research in mobile networks. *SIGCOMM Comput. Commun. Rev.*, 40(1):125–126, January 2010.

[87] Hao Ye, Le Liang, Geoffrey Ye Li, JoonBeom Kim, Lu Lu, and May Wu. Machine learning for vehicular networks: Recent advances and application examples. *ieee vehicular technology magazine*, 13(2):94–101, 2018.

[88] Abdennour Zekri and Weijia Jia. Heterogeneous vehicular communications: A comprehensive study. *Ad Hoc Networks*, 75-76:52–79, June 2018.

[89] Tengchan Zeng, Omid Semiari, Mohammad Mozaffari, Mingzhe Chen, Walid Saad, and Mehdi Bennis. Federated Learning in the Sky: Joint Power Allocation and Scheduling with UAV Swarms. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pages 1–6, Dublin, Ireland, June 2020. IEEE.

[90] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, KC Claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. Named data networking. *ACM SIGCOMM Computer Communication Review*, 44(3):66–73, 2014.

[91] Qixun Zhang, Menglei Jiang, Zhiyong Feng, Wei Li, Wei Zhang, and Miao Pan. IoT Enabled UAV: Network Architecture and Routing Algorithm. *IEEE Internet Things J.*, 6(2):3727–3742, April 2019.

[92] Weicheng Zhao, Yajuan Qin, Deyun Gao, Chuan Heng Foh, and Han-Chieh Chao. An Efficient Cache Strategy in Information Centric Networking Vehicle-to-Vehicle Scenario. *IEEE Access*, 5:12657–12667, 2017.

[93] Pei Zhou, Xuming Fang, Yuguang Fang, Rong He, Yan Long, and Gaoyong Huang. Beam management and self-healing for mmwave uav mesh networks. *IEEE Transactions on Vehicular Technology*, 68(2):1718–1732, 2018.