UNIVERSITY OF CALIFORNIA,
IRVINE

Toward Secure and Reliable Networked Systems of Connected & Autonomous Vehicles

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Networked Systems

by

Anas Alsoliman

Dissertation Committee:
Associate Professor Marco Levorato, Chair
Associate Professor Mohammad A Al Faruque
Assistant Professor Qi Alfred Chen

2023

DEDICATION

§ وَمَا أُوتِيتُمْ مِنَ الْعِلْمِ إِلَّا قَلِيلًا §
الإسراء: ٨٥

§ You have not been given of knowledge except a little §
Al-Isra 17 : 85

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Portions of Chapter 1 of this dissertation is a reprint of the material as it appears in *ACM Transactions on Cyber-Physical Systems* (2023) https://doi.org/10.1145/3579999, used with permission from ACM. The co-authors listed in this publication are Giulio Rigoni, Davide Callegaro, Marco Levorato, Cristina M. Pinotti, and Mauro Conti.

Portions of Chapter 2 of this dissertation is a reprint of the material as it appears in *2023 IEEE Topical Conference on Wireless Sensors and Sensor Networks* (2023) https://doi.org/10.1109/WiSNeT56959.2023.10046219, used with permission from IEEE. The co-authors listed in this publication are Forough Shirin Abkenar and Marco Levorato.

Portions of Chapter 1 of this dissertation is a reprint of the material as it appears in *Proceedings of the 22nd International Conference on Distributed Computing and Networking* (2021) https://doi.org/10.1145/3427796.3428480, used with permission from ACM. The co-authors listed in this publication are Giulio Rigoni, Marco Levorato, Cristina M. Pinotti, Nils Ole Tippenhauer, and Mauro Conti.

Portions of Chapter 1 of this dissertation is a reprint of the material as it appears in *Third International Workshop on Automotive and Autonomous Vehicle Security (AutoSec) 2021 (part of NDSS)* (2021) https://dx.doi.org/10.14722/autosec.2021.23021, used with permission from NDSS. The co-authors listed in this publication are Marco Levorato and Qi Alfred Chen.

Portions of Chapter 1 of this dissertation is a reprint of the material as it appears in *4th Cyber Security in Networking Conference (CSNet)* (2020) https://doi.org/10.1109/CSNet50428.2020.9265534, used with permission from IEEE. The co-authors listed in this publication are Abdulrahman Bin Rabiah and Marco Levorato.

The work in this dissertation was partially supported by the following grants:

- NSF Grant CNS-1850533.
- NSF Grant CNS-1929771.
- NSF Grant IIS-1724331.
- NSF Grant MLWiNS-2003237.
- NSF Grant CNS-2134973.
- USDOT UTC Grant 69A3552047138.
- Project NALP-SAPR2: Navigazione Autonoma, Logistica, e agricoltura di Precisione per Sistemi Aeromobili a Pilotaggio Remoto e Robot, granted by Fondo Ricerca di Base, 2019, University of Perugia.

VITA

Anas Alsoliman

EDUCATION

Doctor of Philosophy in Networked Systems **2023**
University of California, Irvine *Irvine, CA*

Master of Science in Networked Systems **2018**
University of California, Irvine *Irvine, CA*

Master of Science in Network Engineering & Security **2014**
DePaul University *Chicago, IL*

Bachelor of Science in Information Systems **2012**
King Saud University *Riyadh, Saudi Arabia*

RESEARCH EXPERIENCE

Graduate Research Assistant **2018–2023**
University of California, Irvine *Irvine, California*

Researcher at the "Colosseum" lab **2020–2023**
Northeastern University *Boston, Massachusetts*

TEACHING EXPERIENCE

Teaching Assistant **2021–2023**
University of California, Irvine *Irvine, California*

Research Mentor **2020–2023**
University of California, Irvine *Irvine, California*

JOURNAL PUBLICATIONS

Intrusion Detection Framework for Invasive FPV Drones Using Video Streaming Characteristics **2023**
ACM Transactions on Cyber-Physical Systems

CONFERENCE PUBLICATIONS

State-Recovery Protocol for URLLC Applications in 5G Systems **2023**
IEEE Topical Conference on Wireless Sensors and Sensor Networks

COTS drone detection using video streaming characteristics **2021**
Proceedings of the 22nd International Conference on Distributed Computing and Networking

Privacy-Preserving Authentication Framework for UAS Traffic Management Systems **2020**
4th Cyber Security in Networking Conference (CSNet)

WORKSHOP PUBLICATIONS

Vision-Based Two-Factor Authentication & Localization Scheme for Autonomous Vehicles **2021**
Third International Workshop on Automotive and Autonomous Vehicle Security (AutoSec) 2021 (part of NDSS)

SOFTWARE

Traffic Profiling `https://github.com/AnasAlsoliman/Traffic-Profiling`
A Python tool for creating any network traffic profile.

Network Slicing `https://github.com/AnasAlsoliman/Network-Slice-Controller`
A slice controller for LTE networks using the Colosseum testbed.

Computational Slicing `https://github.com/AnasAlsoliman/Computational-Slicing`
A computational slicing abstraction layer.

Feedback Loop `https://github.com/AnasAlsoliman/State-Recovery-Protocol`
A state-recovery protocol for URLLC applications.

ABSTRACT OF THE DISSERTATION

Toward Secure and Reliable Networked Systems of Connected & Autonomous Vehicles

By

Anas Alsoliman

Doctor of Philosophy in Networked Systems

University of California, Irvine, 2023

Associate Professor Marco Levorato, Chair

The design choices of the networked systems for autonomous & connected vehicles (CAV) play a critical role in the security and reliability of their operations and performance. Mainly, the networked systems for CAV should be resistant to disruptions, whether they are intentional or unintentional. For instance, the security of a given restricted airspace might be governed by an authentication system for authorizing drone activity over such airspace. However, having an open authentication channel for drones would leave them vulnerable for stalking due to the broadcast nature of authentication beacons. Furthermore, such authentication system would be fruitless without the means of detecting unauthorized and disruptive drones in the first place. On the other hand, the latency-sensitive nature of CAV applications requires reliable packet delivery schemes for resisting any foreseen disruptions.

In this Dissertation, we discuss the security and reliability perspectives of CAV. In the first Chapter of the Dissertation, we start by first proposing a drone detection framework that can detect the presence of flying drones by exploiting their emitted wireless traffic. Our work shows that the proposed framework achieves a detection accuracy up to 99% for both profiled and unprofiled drones, even in the presence of network interference and benign drone-like wireless traffic. Next we propose a privacy-preserving authentication scheme for drones which is influenced by the Security Credential Management System (SCMS), the official au-

thentication system for V2V and V2I adopted by the U.S. Department of Transportation (USDoT). Our authentication scheme is then re-designed to be fitted with Remote-ID, the official identification framework for drones adopted by the Federal Aviation Administration (FAA). Finally, we propose our localized-authentication scheme that can pinpoint the physical location of authenticated vehicles and therefore differentiate between authorized and unauthorized vehicles.

In the second Chapter of the Dissertation, we go through different tools and techniques for testing and implementing reliable networked systems tailored to latency-sensitive applications such as CAV applications. First we present our packet injection tool that has various features such as generating customized traffic profiles using real data instead of padded data, creating different packetization policies, and producing fine-grained datasets. Then we go through our computing slicer module that can slice the computing resources of a device among different incoming traffic types based on different criteria such as customized scheduling policies. Next, we propose a state-recovery protocol for recovering the state of a channel in the case of channel disruptions using an adaptive loopback channel. Finally, we discuss our work on physical network slicing for 5G systems which is implemented on Colosseum, world's largest RF emulator.

Dissertation Introduction

Connected & Autonomous Vehicles (CAV) are expected to transform (and has already transformed) the shape of many industries such as transportation and logistics. For instance, the development of autonomous cars have shown promising advancements in the areas of safe and reliable self-driving applications. Furthermore, many impactful drone-based applications have emerged such as drone delivery and remote inspection. However, the operation of such vehicles may carry safety and security concerns such as crashing into people or other vehicles. Depending on the vehicle type, additional failsafe layers should be considered in the design of these vehicles to ensure safe and reliable operations.

CAVs can be categorized into two types of vehicles, ground vehicles (e.g. cars) and aerial vehicles (e.g. drones, also known as Unmanned Aerial Vehicles, or UAV for short). The applications of these vehicles can be considered mission-critical applications due to their potential to cause harmful accidents. Therefore, it is crucial to design these applications with quick response times to ensure that the vehicle is able to maintain a safe operation while navigating its surrounding environment, even when operating under uncertain conditions.

A baseline approach for implementing mission-critical applications for vehicles is to establish connectivity channels with the surrounding vehicles or with the infrastructure toward the cloud. This would enable the vehicles to communicate and coordinate among each other, as well as enabling the vehicles to access additional services hosted in the cloud through some forms of vehicular access points such as Road-side Units (RSU) or cellular networks. The security and reliability of such application are directly tied to the security and reliability of its underlying network channel; any network disruptions, whether intentional or unintentional, would impact the performance of any application built on top of the network. For example, the Security Credential Management System (SCMS, the official security system adopted by USDOT for connected vehicles) requires vehicles to have a short connectivity time for down-

loading digital certificates when passing through the coverage of an RSU. These certificates would be used later on to securely communicate with other vehicles and exchange Basic Safety Messages (BSM). Another example would be a drone performing a search-and-rescue operation autonomously. Such drones would require a constant and stable connection with a Ground Control Station (GCS) to send and receive constant updates about the mission in real time. In both examples, one can observe that the success of such applications relies on the security and reliability of the underlying network it is using.

The security and reliability requirements of vehicular networks might appear similar to one another but they have implicit distinctions between the two terms. In short, the main difference between security and reliability is the fact that reliability requirements focus on mitigating unintentional disruptions that arise from the surrounding environment, while security requirements stress on proactively acting upon intentional disruptions that arise from malicious actors. For example in the design of a LiDAR sensor, the sensor is expected to operate reliably under different environmental conditions such as snow or rain. However, it is not part of the natural environment to have maliciously crafted laser pulses fired toward a LiDAR receiver. Similarly in vehicular networks, fluctuations in a wireless channel would decrease the overall available throughput of the network but receiving spoofed packets would alter the behavior of the networked applications and ultimately the behavior of the vehicle itself. These conditions and circumstances would apply to either purely connected vehicles (i.e. a vehicle utilizing connectivity but operated by a human operator) or autonomous vehicles that intelligently self-operate while utilizing connectivity as a form of support.

In this Dissertation, the security and reliability aspect of CAV networks would be discussed under two different chapters. Chapter 1 will focus the discussion on the security aspects of CAV networks. Namely, it will discuss different authentication designs for CAV applications. Furthermore, a drone detection framework is proposed for detecting intrusive and malicious drones using their emitted wireless traffic. Chapter 2 will steer the discussion toward the

reliability design of CAV applications. This chapter will focus on different techniques for rapidly testing CAV applications under different conditions and environments, as well as proposing different supporting tools for maintaining the reliability of CAV applications.

Chapter 1

Toward Secure Networked Systems of Connected & Autonomous Vehicles

The security and privacy of Connected & Autonomous Vehicles (CAV) are critical requirements of the CAV's operational design. These requirements are intuitively critical since the aforementioned vehicles are expected to carry out various operations that would directly and indirectly impact human lives. One of the critical components of any CAV design is the underlying communication network since many CAV applications heavily rely on network support for various functions such as coordination among vehicles and offloading computational tasks to edge or cloud servers. For malicious actors, CAV networks are ripe attack vectors for exploitations. For example, a maliciously spoofed Basic Safety Message (BSM) sent out to a connected car could intimidate the vehicle driver or alter the operational behavior of the vehicle which would potentially lead to a fatal accident. Additionally, the safety-awareness messages broadcasted by drones (which is known as Remote-ID) might contain sensitive information related to the drone and its pilot which could be exploited by attackers to stalk the drone and the pilot.

The first line of defense to secure CAV networks is typically the deployment of an authentication layer in the network for verifying the authenticity of the transmitter of any message. There are already two different authentication frameworks that are officially adopted by government entities for the use of connected and autonomous vehicles. The first framework is the Security Credential Management System (SCMS) which is adopted the U.S. Department of Transportation (USDOT) as the official authentication framework for connected vehicles in the United States. The second framework is Remote-ID which is designed by the Federal Aviation Administration (FAA) as the official identification framework for drones flying under the National Air Space (NAS) of the United States. Both frameworks have their own strengths, weaknesses, and design requirements. For example, Remote-ID is designed to act as a license plate for drone identification. However, Remote-ID might reveal sensitive information such as the drone activity over its lifetime as well as revealing the location of its pilot. In this chapter, we propose a privacy-preserving authentication and identification framework for drones that preserves both the drone and the pilot privacy while keeping its operational requirements. The proposed framework utilizes some concepts from the SCMS framework to anonymize the identity of the drone and its pilot while preserving the sensitive information whenever needed for future accountability.

One of the main differences between drones and cars is the fact that cars can be physically restricted from entering restricted roads given that the car does not possess the necessary access credential for said road (similar to gated areas that can only be open using RFID fobs). However, since drones are flying vehicles, it would be simply impractical to install physical contraptions in the sky to prevent drones entry into a restricted airspace. Therefore to enable a viable and secure drone applications, it is crucial to have a drone detection system for detecting malicious and intrusive drones that are illegally entering restricted airspace such as airports. Having a drone authentication system without having the means of detecting unauthenticated drones in the first place would be meaningless. Therefore, it is essential to have a detection phase that precedes the authentication and identification phase for drone

systems. In this chapter, we also propose a drone detection framework that can detect first-person view (FPV) drones by analyzing their emitted wireless traffic. This framework is designed to detect various drone types even if the detection system has no prior profile of the intrusive drone. Furthermore, the drone detection framework assumes the intrusive drone's emitted wireless traffic to be completely encrypted.

The third CAV security challenge that will be discussed in this chapter is the challenge of localized authentication. This security challenge arise when a certain CAV application needs to associate a received message with the physical location of the transmitting vehicle. For example, if a detection system detects the presence of two drones flying over a restricted airspace but the authentication system receives a single message with valid entry credentials, then it would be difficult to decide which drone is allowed access into the restricted airspace and which drone should be intercepted for accountability and/or forcefully be taken down. To address this challenge, this chapter proposes a localized authentication framework that utilizes the light sources of a vehicle (such as the car headlights or the drone's safety light probes) to encode a random nonce in the form of visual blinks which in turn can be used by the receiver to match the sender of a message with the location of the sending vehicle.

1.1 First Module: Drone Detection

The widespread availability of commercial drones (also known as Unmanned Aerial Vehicles, or UAV for short) and their use in a myriad of applications, pose serious security and privacy concerns. For instance, unauthorized drones may operate in restricted or crowded areas such as airports and stadiums, where they can collide with airplanes or land/crash on people. Moreover, most Commercial-off-the-Shelf (COTS) drones are equipped with cameras to enable a First-Person View (FPV), which allows real-time video transmission from the drone's camera back to the controller (or any separate viewing device). Intuitively, this capability can easily lead to privacy violations. In response of such issues, the Federal Aviation Administration (FAA – the civil aviation regulatory body of the U.S. government) has released the final set of policies on the 15th of January, 2021, that governs the identification requirements for drones which is known as Remote ID [42]. These policies require any drone that operates in the National Air Space (NAS) to continuously broadcasts identifying information such as a unique identifier, a timestamp, drone velocity, latitude, longitude, and altitude of both the drone and its controller. The Remote ID compliance date for drone manufacturers is September 16, 2022 and for drone pilots is September 16, 2023. However, it is difficult to enforce the compliance of such policies on the myriad of commercial drones that are pushed to the market everyday and therefore a detection solution of non-compliant drones is still needed. Many drone detection solutions exists [33, 78], but mostly remain prohibitively expensive for the majority of citizens and some of the solutions target only specific drone models [33]. These issues motivated a surge of research efforts whose objective is to provide cost-effective solutions for detecting unauthorized drones entering restricted airspace or private areas. Toward this goal, many recent contributions (*e.g.*, [61, 15]) attempt to detect drones by analyzing their network traffic patterns.

The majority of COTS drones employ WiFi chips that allow the drone to act as WiFi

Access Point (AP). The user can install an app in a smartphone/tablet to connect to the AP and establish a bidirectional data stream to receive the live-streaming FPV video and to control the drone. Most prior work on drone detection uses the FPV video stream and analyzes traffic patterns from a statistical perspective. For instance, in [61] the authors presented a framework to detect drones' FPV streams based on channel use. That approach proved challenging as video bit rates can widely vary in response to the changing scenery that is captured by the drone's camera which triggers the video compression algorithm to add/remove video frames from the stream [31, 61]. The authors in [13] use the Received Signal Strength Indicator (RSSI) as a feature to discern moving vs. stationary devices. However, this approach may fail to differentiate drones from other moving radio sources. In [15], the authors presented a framework that use features extracted from WiFi frames exchanged between the drone and its controllers. Similar to the previously mentioned paper, the approach is not tested with changing bit rates emitted over the FPV channel and may be unable to differentiate drones' FPV streams from other WiFi video streams such as IoT cameras [31]. Furthermore, the framework requires both the drone and its controller to be within the detection range. In [8], the authors trained a machine learning model to detect drones with high accuracy using a training set composed of drone and controller traffic. Despite using controller traffic in the dataset, the authors left the problem of the "recognition of new UAV types" and "modified video patterns" as open problems.

In this work, we propose a novel and cost-effective drone detection framework (using a WiFi adapter and a laptop/desktop) that solves the problem of detecting new drone types among other similar streaming devices based solely on the drone's FPV highly-dynamic video stream. The framework accomplishes the detection task by leveraging specific packets we identified in the encrypted FPV stream sent by drones over the WiFi channel. We refer to these specific packets, which appear in video streams at periodic intervals for synchronization purposes, as "pivots".

The new framework consists of two components: the Pivot Extraction Algorithm and the Classification Model (Figure 1.1). The Pivot Extraction Algorithm identifies the pivot packets for each encountered WiFi device based on FPV video traffic patterns, while the Classification Model extracts machine learning features based on the pivot packets and trains a drone classifier.

The framework produces high drone detection accuracy under challenging settings, including

1. the scene being captured by the drone's camera is highly dynamic which produces varying bit-rates,
2. only the drone is within the packet capture range while its controller is out of capture range,
3. the network traffic of the drone is completely encrypted,
4. there are other IoT devices in the environment actively emitting video streams that exhibit an FPV packet pattern similar to those generated by drones, and
5. the classifier encounters and detects drones that were not used during the training phase of the classifier.

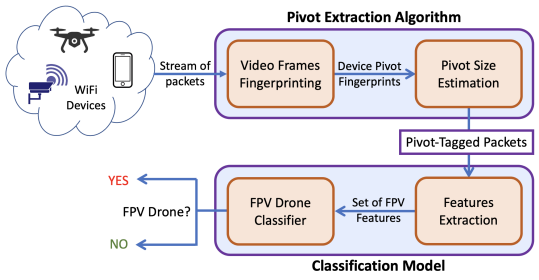


Figure 1.1: Components of the Drone Detection Framework

1.1.1 Contributions

Under the aforementioned challenges, we summarize our contributions as follows:

- We introduce the concept of *Pivot*; a special packet found in video streams (which is used as an anchor for designing drone features) that is not affected by video parameters such as the bitrate. Furthermore, we introduce the notion of *Pivot Fingerprint*; a series of packets that most likely contains a pivot packet, and an approach for locating such fingerprints within a stream of packets.
- We design a novel feature selection strategy that is executed in two phases. The first phase is *model-independent* and based on the Jaccard Similarity Index (also known as Intersection over Union, or IoU for short) which selects the drone features that have high discriminative power to detect new unprofiled drones regardless of the machine learning model used or any unique drone behaviors that exist in a training set. The second phase is *model-dependent* and based on Recursive Feature Elimination (RFE) algorithm that evaluates the sensitivity of the remaining features toward detecting unprofiled drones and selects the set of features the work best for the underlying machine learning model used in the detection framework.
- We provide an implementation and evaluation of the detection framework based on a Random Forest algorithm using features created from pivot packets that can detect drones with accuracy up to 99% even when the drones coexist with other video streaming devices such as IoT cameras and VoIP applications. We also compare our pivot features with the state-of-the-art machine learning-based drone detection features proposed by [8].

The rest of the section is structured as follows: in subsection 1.1.2 we highlight the related work on the field of drone detection. In subsection 1.1.4, we discuss the attacker and de-

fender model, then we provide a brief background on the characteristics of the uplink and downlink channels between the drone and its controller. In subsection 1.1.5 we introduce our drone detection framework and its two components, then we discuss in details the characteristics of the pivot packets and the design of the first component; the Pivot Extraction Algorithm. We also assess the algorithm's behavior when encountered with benign video traffic such as IoT cameras. subsection 1.1.6 shifts the discussion towards the framework's second components; the Classification Model, where we describe our novel pivot features creation and selection strategies, as well as our FPV drone classifier. In subsection 1.1.7, we outline the implementation design of the framework in terms of hardware and software, then we describe our data collection, processing, and sampling procedures. The evaluation and results analysis of the proposed framework is reported in subsection 1.1.8, while subsection 1.1.9 concludes the module.

1.1.2 Related Work

Drones detection is a research topic which is capturing considerable attention from the research community. Mainly, drone detection techniques fall under four different categories:

1. Radar-Based Detection [75]: based on active radars that send electromagnetic pulses toward the area to be monitored to detect the electromagnetic energy reflected by any flying objects.

2. Vision-Based Detection: based on computer vision devices (such as regular or Thermal cameras) to detect flying objects [67, 44].

3. Acoustic Detection: that detects the high-pitched sound frequency generated by the rotating propellers of the drone [23, 20]. Lastly

4. RF/WiFi Detection techniques that monitor the wireless communication links between

the drone and its controller (the Ground Control Station, or GCS) [63] [61].

Since our work proposes a new WiFi detection technique, we focus on related works on RF/WiFi detection of drones. We also summarize some previous results on detection of WiFi cameras due to the similarities between the streams transmitted by the drones and those transmitted by fixed WiFi cameras.

Detection of Drone Traffic The authors of [61] propose to detect drones by their FPV streams. The bit rate of a FPV stream emitted by a drone is compared with the bit rate of a well known and previously recorded FPV data streams. However, the more the scene changes, the higher the bit rate is, and a drone recording a highly dynamic scene might not always match a specific set of FPV bit rate. The authors also argue that since the drones move, the RSSI of their FPV channels is different from those of other WiFi video streaming services. However, they did not take into account video streaming devices that might be moving such as VoIP applications on smartphones.

In [15], drones are detected by monitoring their FPV stream sent over WiFi and applying Machine Learning models. They extracted features from WiFi frames exchanged between the drone and its controllers. As in [61], the authors do not consider either the problem of detecting drones FPV streams with changing bit rates or that of differentiating between drones FPV streams and other WiFi video streams. The same authors, in [14], took the previous work a step further by improving the robustness of the algorithm. Precisely, they aim to detect a drone flying in stealth mode (*i.e.,* a drone that is not transmitting video).

In [56], a framework is proposed to fingerprint drone WiFi communications for the identification of specific drone models. Firstly, drones are discerned by other devices via their speed (exploiting the signal strength information used to calculate the acceleration). Once a WiFi session is associated with a drone, it is further classified using different features (*i.e.,* pattern of probe messages and information in a Frame Header). However, this detection is

based on the assumption that other devices cannot move at the same speed as the drone. In some scenarios, a device could be inside a vehicle (*i.e.,* a phone inside a car) that moves at comparable or higher speeds than of a drone.

In [69, 70], standard classification algorithms are used on eavesdropped traffic exchanged between a drone and its remote controller to analyze extracted features such as packets' inter-arrival times and sizes. The main aim is to detect a drone and its status, *i.e.,* flying vs. resting.

In [8], Alipour et al. use classical features as those used in [69, 70] for differentiating FPV drones from other devices, and hence for detecting drones. The authors show that their framework detects with high accuracy drones using features extracted from packet samples of at least 50 packets exchanged between the drone and controller. The authors left the problems of "recognition of new UAV types" and "modified video patterns" as open problems. Our work aims to improve the results in [8] and to close the problem of recognizing new UAV types.

Detection of WiFi Cameras In [31], the authors detect hidden wireless cameras using smartphones. Their system, DeWiCam, automatically analyzes wireless traffic to recognize camera transmissions, specifically relying on physical and MAC layer features. Importantly, camera traffic streams have relatively stable volume and packet size pattern. Besides, wireless cameras can work continuously without interruptions in the stream.

In [57], the main objective is to identify hidden WiFi cameras by altering the ambient light captured by the cameras to induce variations in the camera's packet flow (caused by the video compression algorithm) which can be identified by statistical techniques. Both papers exploit the compression mechanism that cameras use for video transmissions, discussed below in Section 1.1.4. In our work, we are also interested in investigating characteristics of video transmissions, with the different objective of detecting drones in challenging scenarios.

1.1.3 Background on Drone Defense Systems

A general drone defense system can be realized by deploying three independent components: drone detection, drone authentication, and drone prevention. Each component is an independent system that is designed to address a specific problem. In essence, the function of each component complements each other, but at the same time, the system design of each of them is mostly independent. For example, drone detection complements drone authentication in the sense that it would be meaningless to have a system for verifying the authenticity of flying drones without having the means of detecting the presence of unauthenticated drones in the first place, but whether the detection system is radar-based or network-based, it will not influence the accuracy of the authentication system in verifying the identity of the detected drones. Similarly, the capability of a prevention system to capture and/or destroy an unauthorized drone is not influenced by how the authorization decision was actually made but such prevention system cannot decide which drone to neutralize on its own and therefore an authorization system complements a prevention system. More specific details of each component are discussed next.

1.1.3.1 Drone Detection

The purpose of the drone detection system is the detection of the physical presence of a drone within a predefined area, typically referred to as "restricted airspace". There are different approaches for addressing the detection problem. These approaches (as previously discussed) can be categorized as radar-based, vision-based, acoustic-based, and network-based detection. The selected approach (or group of approaches) is based on the threat model. For instance, the safety concerns surrounding a popular sporting event might include the risk of having spectators using FPV drones to watch and record the match, which would result in the risk of drones crashing onto the event attendees. Based on the threat model,

a network-based drone detection system might be deployed to exploit the wireless traffic generated by the FPV drone for the detection process. On the other hand, the threat model for a military installation might include risks posed by a more capable malicious adversary such as autonomous drones that do not generate any wireless traffic. Such a highly sophisticated thread model might require the deployment of multiple highly reliable drone detection systems such as radar-based and vision-based drone detection systems. Once a drone is detected, the authentication, authorization, and neutralization / countermeasurement systems are designed and implemented based on their own threat models as well.

While the objective of drone detection component is to detect the presence of drones, the task of tracking and pinpointing the exact physical location of the detected drone or its pilot for intervention falls under the Drone Prevention component. It is crucial to differentiate between detection and tracking since some threat models do not require a physical intervention with the invasive drones and therefore a tracking system might not be not needed for the prevention component of the drone defense system. For example, AeroScope [33] is a drone detection system developed by DJI - a leading drone manufacturer - which can detect the presence of DJI drones. Every DJI drone is manufactured to continuously broadcast information about the drone's flight mission and the pilot's information. Once AeroScope detects a DJI drone via its broadcast, the AeroScope operator simply attempts to contact the drone pilot based on the information obtained from the DJI registration database. Since AeroScope is intended to be used in a civilian setting, this simple prevention strategy (contacting the pilot) might be adequate for DJI's threat model.

1.1.3.2 Drone Authentication

The authentication of drones in-flight can take different forms. For instance, it can be in the form of a broadcast message transmitted by the drone, which includes digitally signed identifying information [10]. Other forms of authentication systems propose the use of the

safety flashing probes mounted on the drone frame to send visual identifying cues in the form of light pulses [9]. One of the main objectives of an authentication system is to identify the drone's identity for accountability. Another objective is to allow authorized drones to fly over a restricted airspace. For example, the FAA strictly prohibits any drone activity over national airports. However, the airspace surrounding the airport is also classified as a restricted airspace, but drone pilots are allowed to operate their drones under the exception that drone pilots obtain a flight authorization from the air traffic controller (airport control tower).

In an attempt to unify the authorization framework, the FAA and NASA have jointly developed a national identification system for drones called Remote ID [42] which enforced all drone manufacturers to equip their drones with a Remote ID module by September 16, 2022. This module continuously broadcasts identifying information regarding the drone identity and its flight status such as speed and direction. However as one can notice from such identification design is that only the complying drones can be detected by their transmitted broadcast while non-compliant drones (such as custom-made drones) are not. Therefore, a drone authentication system should be complemented by a drone detection system. Furthermore, once an unauthorized drone is detected, the authorization policy should be enforced by a drone prevention component.

1.1.3.3 Drone Prevention

Similar to the previous components, the design of a particular drone prevention system is based on the threat model. For example, Tokyo Municipal Police Office [83] proposed a guardian drone system that consists of a specialized drone equipped with a large net for capturing rogue or suspicious drones flying into restricted airspace such as near government buildings. In a more critical setting, the Israeli Ministry of Defense claimed to have designed a drone prevention system – code-named Iron Beam – which consists of a high-powered laser

system that can burn attack drones in-flight with a concentrated beam of light [50]. In a more relaxed threat models such as in a civilian setting, home owners might have concerns related to spying FPV drone. The prevention strategy in this scenario might be less invasive where a simple FPV detection system would alert the home owner to close the window blinds for example to preserve the privacy of the household members.

A notable remark about the prevention component is that if a physical intervention with drones is desired, a drone prevention systems might require the drone's current physical location to acquire a lock on the target drone. Recall that (and as discussed in Section 1.1.3.1) the objective of the drone detection component is to only detect the presence of drones within the restricted airspace while tracking the detected drones for intervention is performed by the drone prevention component. This separation of duties between detection and tracking is usually desired because a drone detection system is typically designed to be always-on while a drone prevention system is only activated once a drone is detected. Therefore, it would be more cost-effective to invoke costly targeting operations only when needed. For example, consider a guardian drone equipped with a radar imaging and tracking system that can scan the sky at a narrow angle (similar to [83]). It would be impractical to continuously operate the guardian drone over the restricted airspace while monitoring all airspace angles simultaneously, given the average operating time of quad and hexacopters on a fully charged battery to be around 30 minutes. An alternative and more practical solution is to have a drone detection system in place that detects the presence of invasive drones. Then only a general direction of the drone location (for example by using multiple FPV detection stations) is provided. Therefore only then the guardian drone can be dispatched to track and capture the invasive drone in the detected area. In a more malicious environment such as in a military setting, the threat model might require immediate and accurate location of the detected drone for a quick intervention. Therefore, the prevention system might depend on a more accurate localization information by using radar-based or vision-based detection systems for example. Then based on the drone's estimated coordinates, the drone could be

intercepted by a heat-seeker tracking prevention system for example.

1.1.4 Drone's Threat & Network Model

In this Section, we first discuss the *attacker and defender* model and its underlying assumptions. Then we provide a brief background on the characteristics of the communication channels between the drone and its controller. Specifically, Section 1.1.4.2 describes the control channel (uplink) and its components, while Section 1.1.4.3 discusses the FPV channel (downlink) and how a video compression algorithm affects the behavior of a network traffic pattern.

1.1.4.1 Attacker and Defender Model

As discussed in Section 1.1.3, a complete drone defense system is generally comprised of three different components; (i) drone detection (detects the presence of drones), (ii) drone authentication (verifies the authenticity of drones' identities), and (iii) drone prevention (stops and/or neutralizes drones' access to the restricted airspace). Since the scope of this module is focused on drone detection, the defender model described below is formulated from a drone detection perspective only. We would like to remark that our drone detection framework is designed to be independent; can either be used as a stand-alone drone detection system or be easily integrated into other drone defense systems (e.g. authentication or prevention) without having any presumptions or specific requirements on how to operate these systems.

In our defender model (Figure 1.2), we consider a restricted airspace protected by a monitoring station. The station's objective is to detect unauthorized drone's access. The unauthorized drone is defined as an unknown FPV drone (in terms of type, model, or brand) that

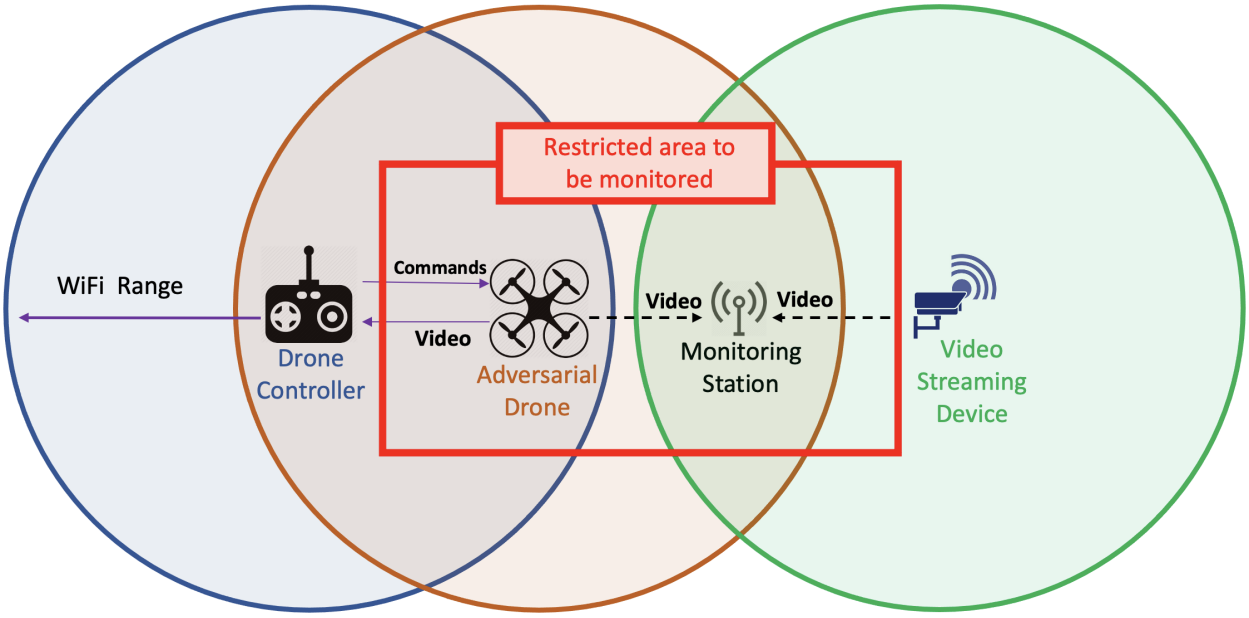


Figure 1.2: Attacker and Defender Model

violates the access policy of the restricted airspace. This unknown drone will be denoted as "unprofiled drone" throughout this module. The monitoring station is comprised of COTS hardware; a general purpose computer such as a laptop or a desktop, and a WiFi adapter that captures all WiFi packets that are sent within its vicinity. For each stream of packets, the station analyzes the stream and labels it as either 'drone' or 'non-drone' based on drone FPV signatures. We also assume the existence of other video streaming devices that can possibly generate false-alarms within the station's receiving range such as IoT cameras.

In the attacker model, we consider an unauthorized FPV drone attempting to access the restricted airspace. When the drone operates, a bi-directional connection is established between the drone and its controller. The downlink channel is a video stream sent by the drone to the controller, while the uplink channel carries control commands sent from the controller to the drone. We assume that both channels are encrypted and the drone's controller is out of the monitoring station's detection range.

1.1.4.2 Control Channel:

The control channel carries flight commands from the controller to the drone. Overall, the control channel comprises of two different network flows, periodic control packets and heartbeats packets. In case of controllers that receive a video stream over TCP, a flow of acknowledgments (ACKs) will be sent out back to the drone as well. Note that even if ACKs are encrypted, they can be easily identified by the packet size (20 bytes TCP header without options + 20 bytes IP header without payload). Some control applications embed heartbeat messages within the stream of periodic control packets. Both streams (heartbeats and control packets) have rather unique sizes and inter-arrival times. Therefore, a simple detection scheme can easily differentiate between the controller's traffic and other background traffic. In our framework, we reflect a more realistic –as well as more challenging– scenario where we assume that controllers are not always present within the range of the detection system and their traffic cannot be captured and classified (Figure 1.2).

1.1.4.3 FPV Channel & Video Encoding:

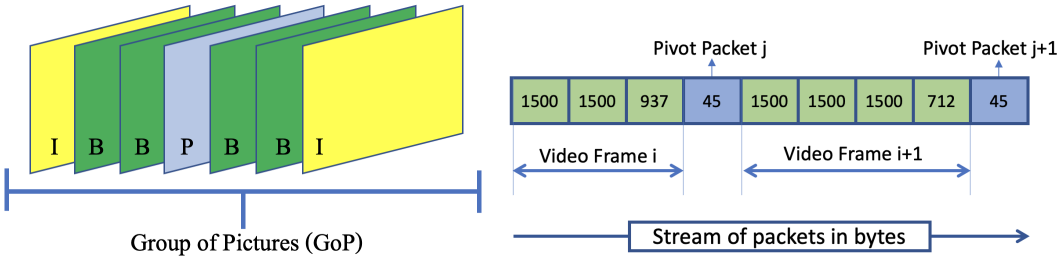
Since the wireless channel is assumed to be encrypted, the detection framework cannot identify drone FPV channels by simply inspecting the content of the intercepted packets. Therefore, the traffic profile of an FPV transmission must be identified based on understanding how video encoding and compression algorithms affect network the traffic pattern of a video streaming application.

A drone's FPV channel is considered a channel of a video streaming application that transports compressed video frames from the drone to the controller. A video is a stream or a sequence of still pictures. In order to decrease the portion of channel capacity used to transport the video over wireless channels, almost all video encoders include a video compression algorithm.

Typically, compression algorithms exploit the – possibly high – correlation along the spatial and temporal dimensions within Group of Pictures (GoP). In brief, while JPEG compression is used in the spatial domain, the core idea to harness temporal correlation is to encode differences compared to reference frames within each GoP. We, then, have three frame types: Intra-Coded Frames (I-Frames), Predicted Frames (P-Frames), and Bi-directional Frames (B-Frames). I-Frames are Intra-coded frames that exploit spatial redundancy (correlation among the pixels in the frame) to achieve compression at individual frame-level. P-Frames and B-Frames are Inter-coded frames that exploit temporal redundancy prediction. The difference between P- and B-Frames is that P-Frames only encode pixels that are changed compared to the last reference frame, while B-Frame considers both previous and future reference frames. These frames are grouped into a single GoP, which starts and ends with an I-Frame (Figure 1.3a). Typically, video streaming applications rely on well-known video codecs such as H.264 to compress raw images captured by some camera and turn them into compressed frames for faster transmission. In general, B-Frames (which includes futures changes in the frame) are used in prerecorded videos (such as YouTube) but are not used in real-time streaming applications (such as FPV video streaming) to minimize the video delay, on the expense of lowering the video quality or increasing the occupied bandwidth.

Intuitively, scene characteristics and the motion of the drone heavily influence the compression rate achieved by the scheme described above, as well as the temporal structure of the data stream. In other words, if the captured scene is dynamic, then, the encoding scheme will either (*i*) generate I-Frames more frequently, which in turn results in shorter GoPs or larger P-/B-Frames, or (*ii*) generate larger differential frames. In both cases, the resulting compression gain is reduced.

The keynote of a video stream transmitted over a WiFi network is that the size of these frames is larger than a WiFi MTU (Maximum Transmission Unit) which is 2304 bytes. Network interfaces translate all packets sent and received from the operating system into



(a) GoP starts and ends with an I-Frame

(b) A 45-bytes pivot packet buffered between two video frames packets

Figure 1.3: A single video frame is sent over multiple packets while pivots are contained in a single packet

Ethernet, which has an MTU of 1500 bytes. Therefore, each individual video frame that is larger than the MTU is fragmented into a series of packets of a size equal to the maximum Ethernet MTU size, except for the last packet, which contains the residual bytes of the frame.

FPV video streams also include other messages such as synchronization information for maintaining the state of the channel. Typically, these messages are small and fit a single WiFi packet and are sent periodically at predicted intervals. Packets from the compressed video stream are emitted more frequently compared to sync packets (the packets that contain the synchronization information), which then, have a *higher probability* to be buffered right after the last less-than-MTU sized frame packet (Figure 1.3b). A key observation is that the size of these sync packets is fixed for the entire duration of the video stream.

The resulting pattern contains rather unique packet sequences. The fixed-size of the sync packets, make them suitable to act as **pivots** which can be used to build FPV drone features for a detection model. The characteristics of pivot packets will be discussed in Section 1.1.5.1.

1.1.4.4 Discussion:

Literature generally treats the traffic profiling of a device connected to an encrypted network, by leveraging the packet sizes and their inter-arrivals [79, 32] (i.e. without the need to look at the content of the encrypted packets). In the specific case of traffic profiles of video streaming applications, the uplink traffic generated by the control channel is quite unique: short periodic packets (Section 1.1.4.2). Although this approach can lead to robust detectors [69, 15, 14, 36], the source of those packets is the controller, which might be out of the range of the monitoring station (Figure 1.2). On the other hand, the downlink traffic generated by the FPV channel can be profiled by the unique pattern of packet sizes generated by the underlying video compression algorithm (specifically, by looking at the timing and size of I- and P-Frames [61, 31]) as shown in Figure 1.3b. However, the FPV channel is susceptible to the frame variations in the captured scene which would change the shape of the traffic profile for a given device depending on the scene dynamics. This change in the FPV traffic would make it difficult to build video profile for a video streaming device, and even more difficult for FPV drones that are similar in their traffic profiles to other benign video streaming devices such as IoT cameras. In an attempt to differentiate between drone video and all other non-drone video profiles, we will discuss next the potential of using the sync packets (which we denote as pivot packets) to build machine learning features for a drone detection classifier that correctly classify drones even when other similar traffic to drone FPV exist in the detection range.

1.1.5 Drone Detection Framework

In this Section we describe our *Drone Detection Framework* (Figure 1.1), which is based on the concept of a *pivot*; a special packet found in the drone's FPV traffic which will be utilized as an anchor for building drone features. The Drone Detection Framework is the composition

of two main components: a *Pivot Extraction Algorithm* (Section 1.1.5.3) and a *Classification Model* (Section 1.1.6). In short, the Pivot Extraction Algorithm takes a sequence of encrypted packets as an input, tracks the packets that belong to video frames based on the video compression analysis described in Section 1.1.4.3, extracts pivot fingerprints, and estimates the size of the pivot packet for each encountered WiFi device. The engine of the Classification Model uses the pivot size to construct a set of FPV features and feeds it to a binary classifier which outputs "YES" if the packets belong to an FPV drone and "NO" otherwise.

This Section mainly focuses on the first component of the framework, Pivot Extraction Algorithm, while the second component will be discussed later in Section 1.1.6. In this Section, subsection 1.1.5.1 will discuss what does a pivot packet look like via network traffic analysis. Subsection 1.1.5.2 describes how pivot packets appear with respect to other normal packets. In subsection 1.1.5.3 we show how to locate and extract pivot packets from a network trace using our Pivot Extraction Algorithm via tracking video frames that form pivot fingerprints, while subsection 1.1.5.4 illustrates how the algorithm differentiates between FPV video streams and other benign streams such as IoT cameras.

1.1.5.1 Pivot Definition via FPV Traffic Analysis

To define pivot packets, we analyze data collected from three FPV drones from three different brands: EACHINE E58 WiFi FPV Quadcopter, Spacekey DC 014 FPV WiFi Drone, and Ryze Tello Quadcopter Drone. In the following, the three drones are referred to (according to their painted color) as Black (BLK), Red (RED), and White (WHT) drone, respectively (Figure 1.4a).

Each drone has a built-in camera with a First-Person-View (FPV) capability. Upon powering up, the drone creates an Access Point (AP) and the user connects his/her smartphone/tablet to that AP. Each drone has its own app that can be downloaded from Google Play or Apple's

(a) FPV Drones (b) IoT Cameras

Figure 1.4: The FPV drones and IoT cameras used in the pivot analysis

App Store. When the user connects to the drone's AP, a video stream can be initiated and viewed on the smartphone through the drone's designated app.

For every one of the three drones, we capture decrypted network traffic (for easier initial analysis) of the drone's FPV using an external WiFi adapter set into Monitor Mode from five meters away for 30 seconds. We consider two different video states: the first state corresponds to the drone's camera capturing a stable scene (STB), while the second state corresponds to a highly unstable (shaking) drone flight (SHK) where the camera rapidly points at different directions. The latter state induces fast changes in the captured video stream which would potentially affects the frame-rate of the video compression algorithm and ultimately the video bit-rate as explained earlier in Section 1.1.4.3.

Analyzing the BLK drone's traffic, we observe that before transmitting a series of full-sized packets (which we assume is a video frame), the drone transmits a 46-bytes packet. This packet includes an ASCII-readable command called *lewei_cmd*, which appears to be responsible for transferring media files from the drone to its associated app on the controlling smartphone [82]. The app associated with the BLK drone also sends some *lewei_cmd* packets to the drone, but only a few of them (once every second). The RED drone, instead, sends 4-bytes packets (with identical payloads) to its RED-app right before sending a video frame. The RED-app sends the same 4-bytes packet to the drone, but only four times in the entire

30 seconds network trace. Finally, the WHT drone sends an identical 35-bytes packet (except for the last 2 bytes in the payload) to its WHT-app between some video frames. These 35-bytes packets are sent at uniform intervals (every 0.1 of a second), which is independent of the varying FPV frames transmission times. For all of the three drones, traffic patterns were observed for both SHK and STB video states.

From these observations we conclude that each drone periodically sends special packets that are repetitive and identical in length, and are not part of a video frame payload. In our drone detection framework, we will leverage these packets and we name them the *pivot* packets that will be used to create features enabling drone detection.

1.1.5.2 Patterns & Behavior of Pivot Packets

We now study the occurrence and patterns of pivot packets in relation to the two different video states: SHK and STB. In both states of each drone, we compute the average bit-rate and the pivot appearance rate (pivot per second). The results are reported in Table 1.1. It can be observed that when the video state shifts from STB to SHK, the FPV bit-rate of the BLK drone on average increases by 444 kbps (25%), while the number of pivots observed per second increases by 2 (25%). In the RED drone, instead, the FPV bit-rate increases only by 15 kbps (0.8%), while the number of pivots observed per second increases by 2 (12%). In the WHT drone, although the FPV bit-rate increases by 334 kbps (13%), the number of observed pivots remains almost constant.

Although the collected data from the BLK drone seems to indicate a dependence of the number of pivots on the FPV bit-rate (both increased by almost the same percentage), the patterns observed in the RED and WHT drones streams instead supports the hypothesis that the pivots are not necessarily tied to the bit-rate (especially for the WHT drone since the inter-arrivals of the pivot packets are constant) and the number of pivot packets are

Table 1.1: Pivot frequency & bit-rate change

Drone	FPV Bit-Rate	Pivot/Second	Δ FPV Bit-Rate	Δ Pivot/Second
BLK-SHK	2.244 Mbps	9.2	≈ 444 kbps	≈ 2
BLK-STB	1.800 Mbps	7.6		
RED-SHK	1.817 Mbps	18.6	≈ 15 kbps	≈ 2
RED-STB	1.802 Mbps	16.6		
WHT-SHK	3.054 Mbps	8.0	≈ 334 kbps	≈ 0
WHT-STB	2.720 Mbps	8.0		

almost the same for both the video states, that is, pivots are independent of the bit-rate increase. The deviation in the number of pivots in the other two drones might be due to **1)** the nonuniform inter-arrivals of pivots and **2)** the inherent packet loss in the wireless environment for packets captured via WiFi Monitor Mode, which can affect the small number of transmitted pivots.

To this end, we can safely assume that the pivots are not necessarily tied to the bit-rate (*i.e.,* number of video frames) of the FPV stream and, as such, pivots can be used to create robust features for drone detection regardless of the video's motion state.

1.1.5.3 Pivot Extraction Algorithm

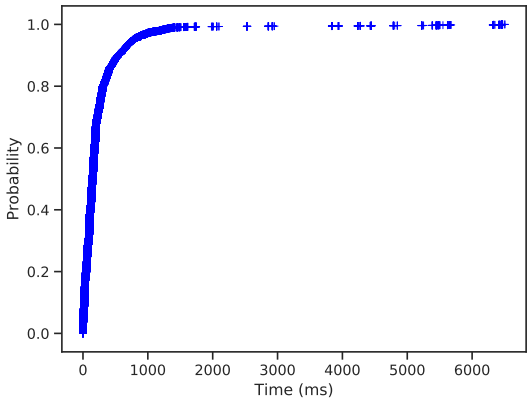
The Pivot Extraction Algorithm is the first component of the Drone Detection Framework, and its objective is to extract the size of the pivot packet for a given device. Assuming encrypted FPV streams, a fundamental problem is the identification the pivot packets. Herein, we take an approach based on their size. As different drones may have different formats for the pivot packets, the challenge is then to estimate the size of pivots from a network trace for each device we encounter during monitoring.

As discussed in Section 1.1.4.3, video frames are packetized by the Network Interface Card (NIC) and each packet's payload is capped at the Maximum Transmission Unit (MTU) of the transmitting NIC. Typically, the WiFi protocol has an MTU of 2304 bytes whereas Ethernet

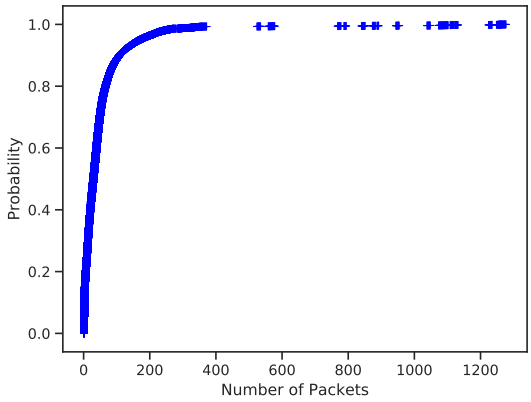
has an MTU of 1500 bytes. Since operating systems translate all sent and received packets to and from the WiFi card into Ethernet, all packets have to adhere to the Ethernet's MTU which is 1500 bytes. Therefore, video frames that are larger than 1500 bytes are packetized into a series of 1500 bytes packets where the last packet contains the residual of the video frame and its size is typically less than 1500 bytes.

Besides packetized video frames, the NIC also receives pivot packets from the drone application which, we recall, are synchronization packets sent periodically with large inter-arrivals between them (30 ms to 100 ms). On the other hand, each processed video frame is packetized and buffered at the NIC's transmission queue where each packet is sent out as soon as the wireless channel is cleared (inter-arrivals of packets belong to the same video frame ≈ 0.01 ms). Therefore, there is a high probability that pivot packets are buffered between two packetized video frames. In other words, since each packetized video frame ends with a less-than-MTU packet and begins with an MTU packet, a pivot is most likely to appear after a less-than-MTU packet and before an MTU packet (recall Figure 1.3b). From this observation, we established that a group of less-than-MTU packets between two MTU packets might include a pivot packet where the last packet is most likely the pivot. It is also observed that a pivot packet might appear after two less-than-MTU packets or slip individually between two MTUs.

During the pivot analysis, we also observed that a pivot packet has another attribute; it is sent more frequently compared to all other packet types. Based on this observation, we computed the occurrence frequency of all packet sizes within drones' traces. We observed that the highest occurring packet size is the MTU size for all of the drones we considered. Then, we observed that the second-highest occurring packet size for all three drones (BLK, RED, and WHT) is the size of the pivot packet in both motion states (STB and SHK). Knowing these characteristics, we set a new objective to determine how much time and how many packets are required to collect and have enough occurrence frequency in order to find the



(a) Cumulative Distribution Function (CDF) of packet inter-arrivals



(b) Cumulative Distribution Function (CDF) of packet sizes

Figure 1.5: Cumulative Distribution Function of time and packets needed to acquire the pivot packet

correct pivot packet size. To approach this objective, we calculate the distribution of the time and number of packets needed to correctly identify the size of the second highest occurring packet for every drone trace file. First, we extract the pivot size manually beforehand as the ground truth for each drone by distributing the occurrence frequency of packet sizes of the entire network trace of each drone. Then, we designate each packet in the trace as the starting point for accumulating all subsequent packets until the pivot size matches the ground truth of the trace file. Now for each starting point, we have the time and number of packets needed to correctly identify the pivot size. Finally, we calculate the Cumulative Distribution Function (CDF) over the detection time (Figure 1.5a) and number of packets (Figure 1.5b) for all drone traces. From the CDF, we can observe that we need at least 820 milliseconds and at least 170 packets to acquire the correct pivot size with probability of 0.95. Therefore, we set the sample size to be at least 170 packets with a time window of at least 820 millisecond.

Based on these observations, we develop a **Pivot Extraction Algorithm** that is executed in two phases, *Video Frames Fingerprinting* and *Pivot Size Estimation* (see Figure 1.1). During the Video Frames Fingerprinting phase, the algorithm moves a sliding window across

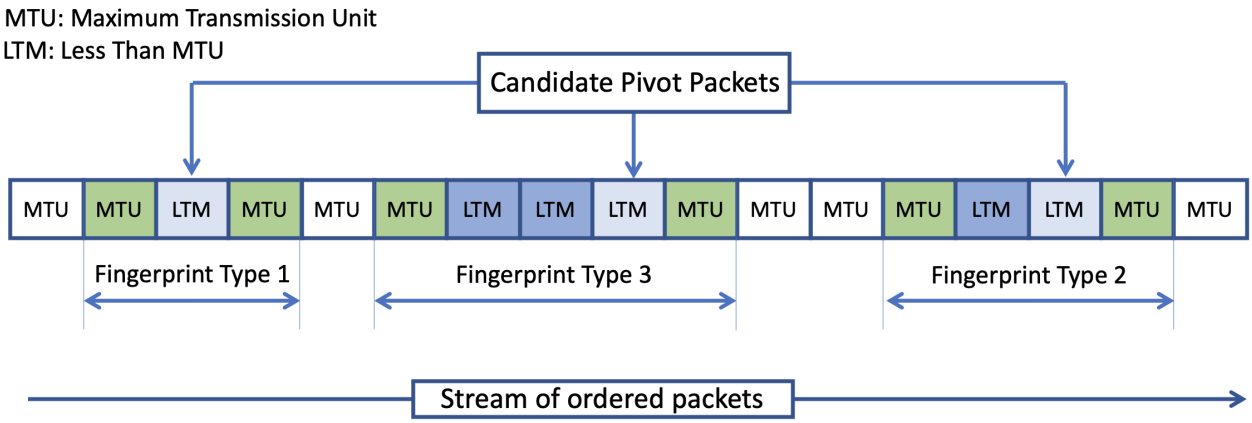


Figure 1.6: Candidate pivot packets.

a network traffic stream that can fit at least 170 packets sent within least a 820 millisecond window and records any packets sequence that (i) starts and ends with a packet of size exactly MTU, (ii) has less than six consecutive packets, (iii) and the size of all packets (except the first and last packet) is less-than-MTU. Each recorded sequence of packets are denoted as a *Pivot Fingerprint* which might include a pivot packet (Figure 1.6). In the *Pivot Size Estimation* phase, the recorded Pivot Fingerprints are evaluated against three types of fingerprints: type 1 contains three packet, type 2 contains four packets, and type 3 contains five packets (Figure 1.6). For each fingerprint, the before-the-last packet in the fingerprint is marked as a candidate pivot. Then on a separate process, the occurrence frequency of all packet sizes in the sliding window is computed. Finally, the candidate pivot size that matches the second-highest occurring packet size is declared as the size of the pivot packet.

It remains now to explain how to efficiently estimate the MTU packet size for a given sample of packets. The algorithm declares the size of the first packet in the sample as the MTU size then proceeds to find the next MTU packet. Whenever a packet size that is higher than MTU is detected, the Pivot Extraction Algorithm declares the new packet as the MTU and resumes the pivot estimation process.

Our Pivot Extraction Algorithm has correctly identified up to 99% of pivot packets within a packet trace in $O(n)$ time where n is the number of packets in the sample. Thus, the Pivot

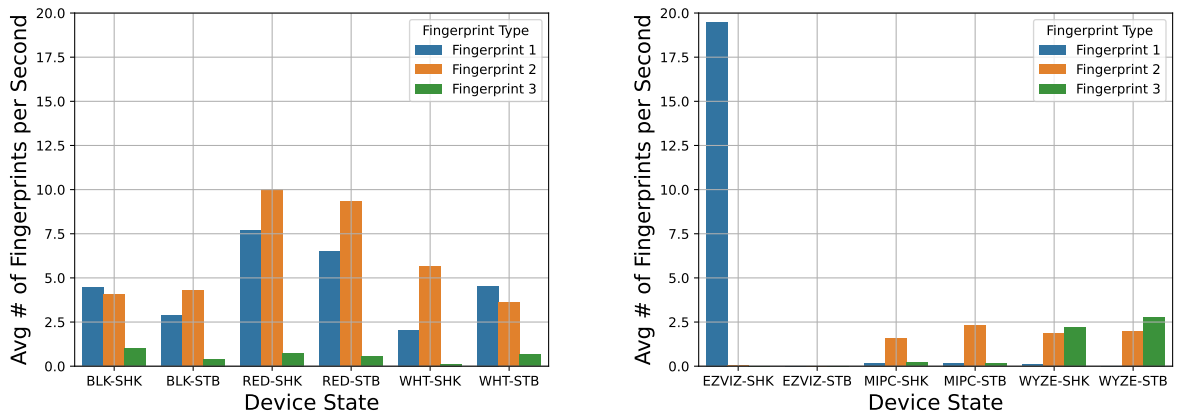
Extraction Algorithm can – with high probability – identify the pivot size.

1.1.5.4 Pivot Patterns in Other Real-time Video Streaming Devices

One can readily observe that the Video Frames Fingerprinting process is influenced by the video stream emitted by a device and therefore the ability of the Pivot Fingerprints in differentiating between FPV drones and other video streaming devices is questionable. We, then, analyze video streams emitted by IoT cameras in order to compare them with those emitted by FPV drones in terms of pivot patterns.

Real-time video streaming devices, such as IoT cameras, generate streams of packets that have a strong similarity with FPV drones' video streams. Analogously to FPV drones, video streams emitted by IoT cameras are unidirectional. That is, an IoT camera forwards a video stream to its designated app while the app maintains the video connectivity with the camera via heartbeats and keepalive messages. Conversely the video streams generated by VoIP applications (e.g. Skype) are bidirectional. Therefore, we incline toward using IoT cameras in the analysis of our pivot extraction approach since they are the most similar applications to FPV drones. However, we still included VoIP traces for the final evaluation in Section 1.1.7.1.2. In order to assess whether or not our pivot approach can distinguish IoT cameras from FPV drones, we collect video traces produced by three different IoT camera brands: Wyze Cam, EZVIZ C1C, and Lefun MIPC (Figure 1.4b). Just like the drones, we collect traces for each camera in a stable (STB) and dynamic motion (SHK) state, resulting in a total of six camera traces. We then calculate how many pivots are recorded per second under each pivot type, *i.e.,* in which type of fingerprint a pivot is found.

In Figure 1.7, for each device (drone and camera), we compute the average number of pivot fingerprint type found per second. From the results, we can see that fingerprint type 3 rarely appears in drones video. On the other hand, each drone has at least four type 2 fingerprints



(a) Drones Fingerprints.

(b) Cameras Fingerprints.

Figure 1.7: Average number of Fingerprint types per second for Drones (1.7a) and Cameras (1.7b).

per second. In Wyze Cam and Lefun MIPC cameras, we could detect some fingerprints of type 2 and 3 ($\approx$ 2.5 pivots), but almost no fingerprints of type 1 are detected in their packet traces. Instead, the EZVIZ camera in SHK state has on average about 20 fingerprints of type 1, but almost no fingerprints of type 2 and 3 in its packet trace. From the results, we conclude that other network devices that have traffic patterns similar to drones would have different pivot patterns. Thus, a pivot-based approach can discriminate the two classes of applications.

1.1.5.5 Discussion

Pivot packets show a potential in differentiating between FPV drone traffic and other video traffic. Since the network traffic is assumed to be encrypted, pivot packets cannot be identified among the stream of the intercepted packets by simply looking at the packets payload. However, the size of the pivot packet is fixed for each video stream (e.g. the size of the pivot packet for the RED drone is always 4 bytes). As discussed in Section 1.1.4.3, the video compression algorithm influences the size of non-pivot packets (packets that carry the captured video frames) and therefore it continuously alters the traffic profile. Based on the video

traffic profiles, three packet patterns (denoted as pivot fingerprints which are visualized in Figure 1.6) are identified. These fingerprints are expected to contain the pivot packet, and the size of this packet is estimated using the Pivot Extraction Algorithm. The accuracy of the algorithm is 95% when at least 170 are captured from a given device within at least 820ms.

1.1.6 Classification Model & Features Design

In this Section, we shift our discussion to the second component of the framework, the *Classification Model* (recall Figure 1.1). The classification model we adopt for the FPV Drone Classifier of our framework is the Random Forest (RF) algorithm. This is motivated by its low complexity and good performance in many settings. Moreover, RF is widely used in the literature to address similar problems.

We combine the RF algorithm with Grid Search; a technique where different hyperparameters are tested for a fine-tuning process. The hyperparameters that we used are: the maximum depth of the tree, the minimum number of samples required to split an internal node, the minimum number of samples required to be at a leaf node, and the number of trees in the forest. The basic idea is that the RF is repeated multiple times, with different hyperparameters, and the combinations of those hyperparameters that gives the best results in the validation set inside the Grid Search, are used for the final test using the test set.

For the rest of this Section, we discuss the process of utilizing pivot packets as an anchor for creating machine learning features for detecting unprofiled drones. Recall that one of the main contributions of this work is detecting drones that the FPV Drone Classifier has never trained on. In Section 1.1.6.1, we provide the list of features used by the classifier. Then in Section 1.1.6.2, we introduce two novel feature selection strategy that pick the best features for unprofiled drone detection in two phases. The first phase is based on Jaccard

Similarity Index and is independent of the underlying classification model that is used in the framework, while the second phase is based on the Recursive Feature Elimination (RFE) technique and it involves the classification model during the selection process, meaning that its performance depends on the type of the machine learning algorithm used.

1.1.6.1 Feature Creation

As discussed in Section 1.1.5, pivot packets have the potential to differentiate between FPV drones and other network devices, including video streaming devices that exhibit similar traffic patterns to FPV drones such as IoT cameras. This differentiating potential creates an opportunity for constructing machine learning features that have the discriminative power to classify network devices into either drones or non-drones based only on the traffic observed from a WiFi network. In particular, the features are expected to correctly classify (with high probability) drone devices among all other non-drone devices even if the machine learning model does not have a prior profile of the drone being classified. The set of features are organized into the following three groups:

1.1.6.1.1 Pivot Features

Pivot features are features composed from the output of the Pivot Extraction Algorithm. Recall that the algorithm outputs a sequence of packets that contains at least 170 packets which are sent within at least 820ms time window. Within this sequence, the algorithm also tags the pivot fingerprints; the location of pivot packets with respect to the MTU and less-than-MTU video frame packets that envelop the estimated pivots (see Figure 1.6).

The set of pivot features and their definitions are reported in Table 1.2 and is comprised of 13 features. A brief description of two pivot features is provided next:

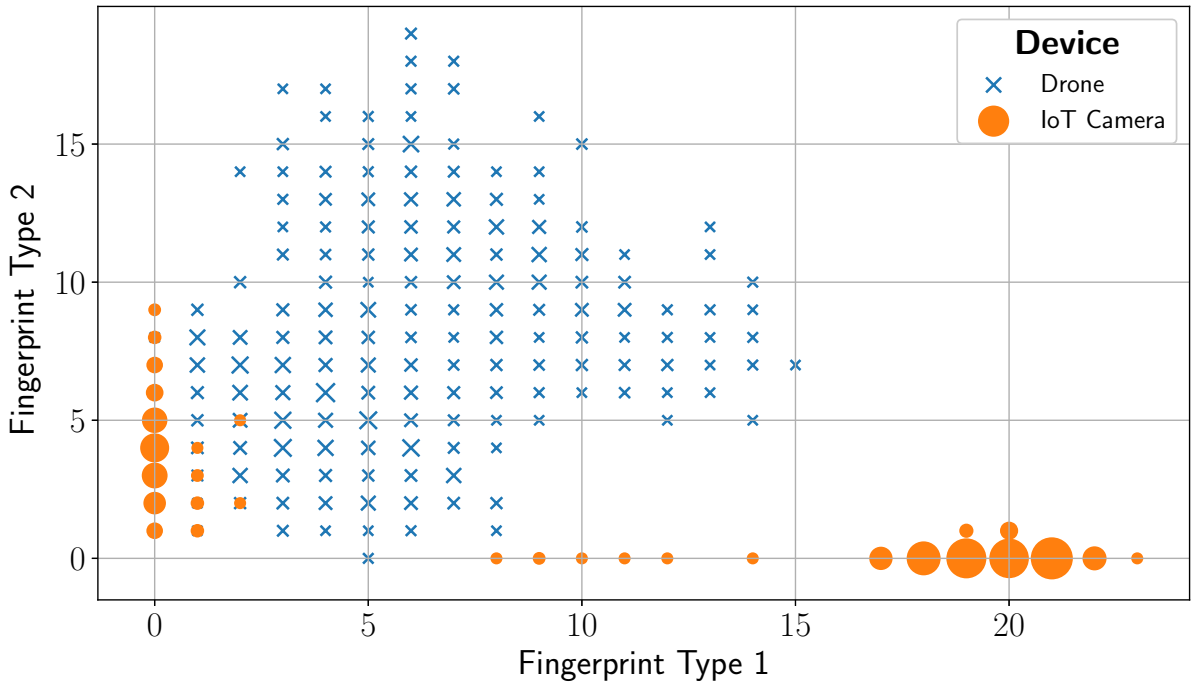


Figure 1.8: Scatter plot of fingerprint type 1 & 2 for drones and cameras

MTU Size Feature: During the drone's pivot fingerprint analysis in Section 1.1.5.1, we observed that FPV drones tend to fill their MTU packets to the maximum (1500 bytes in RED and BLK) or near maximum (1454 bytes in WHT) payload allowable by Ethernet protocol. From this observation, we created *is_MTU_Large* feature which is set to 1 if the Pivot Extraction Algorithm declares a device's MTU is more than some threshold – heuristically set to 1400 bytes – and 0 otherwise.

Pivot Fingerprint Features: We plotted the number of pivot fingerprint type 1 and 2 in a scatter plot for FPV drones and IoT cameras in Figure 1.8. From the plot, the difference in fingerprint appearance behavior between drones and cameras can be observed and therefore it has the potential of having the number of fingerprints as a feature for each device. Note that the scatter plot is derived from Figure 1.7. Because of the insignificant number of appearance of fingerprint type 3, it has not been considered as a feature.

Table 1.2: Pivot Features

Pivot Features	Description
is_large_MTU	1 if MTU is greater than 1400 bytes and 0 otherwise
fingerprint_1	Number of fingerprint type 1 in the sample
fingerprint_2	Number of fingerprint type 2 in the sample
window_time	Total packets inter-arrivals of the sample
total_packets	Total number of packets in the sample
total_size	Sum of packet sizes in the sample
pivot_size/MTU_size	The ratio of the pivot size to the MTU size
MTU_size/total_length	The ratio of MTU size to the sample size
pivot_size/total_length	The ratio of pivot size to the sample size
MTU_count	Total number of MTU packets in the sample
pivot_count	Total number of pivot packets in the sample
MTU_count/total_packets	Ratio of MTU count to packet count in the sample
pivot_count/total_packets	Ratio of pivot count to packet count in the sample

1.1.6.1.2 Statistical Features

The State-of-the-Art in machine learning-based drone detection framework that exploits wireless network traffic for drone detection [8] uses 12 statistical measurements for feature generation. These features will be denoted as *statistical features* and their statistical measurements are reported in Table 1.3 with the addition of the 'Variance' feature which was not used by [8]. Just like our framework, the drone detection framework proposed by Alipur et. al. [8] uses only packet sizes and their inter-arrival measurements for feature generation. Since both sets of features (pivot and statistical) are based on the same packet measurements, we extend our pivot-based feature set and include these statistical features to our own set as well. Including such features to our feature set is a crucial step toward performing a comparative analysis between our pivot features and the statistical features. The analysis will be conducted in Section 1.1.8.1 and it will provide an insight on the performance of our pivot features compared to the latest drone detection features proposed in the literature.

Recall that our framework assumes an encrypted network traffic and the Pivot Extraction Algorithm only extracts the size of WiFi packets and their inter-arrivals. Therefore each statistical measurement is computed twice, once over the *packet sizes* and another over their *inter-arrivals* which results in a total of 26 statistical features (13 statistical features $\times$ 2 different packet measurements).

Table 1.3: Statistical measurements for computing Statistical Features

Measurements	Description
Standard Deviation	$\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - mean(x))^2}$
Variance	$\sigma = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - mean(x))^2$
Root Mean Square	$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i)^2}$
Mean Square	$\frac{1}{N}\sum_{i=1}^{N}(x_i)^2$
Pearson Skewness	$3(mean(x) - median(x))/\sigma$
Kurtosys	$\frac{1}{N}\sum_{i=1}^{N}((x_i - mean(x))/\sigma)^4$
Skewness	$\frac{1}{N}\sum_{i=1}^{N}((x_i - mean(x))/\sigma)^3$
Minimum	$(Min(x_i)\|_{i=1,\ldots,N})$
Maximum	$(Max(x_i)\|_{i=1,\ldots,N})$
Mean	$\frac{1}{N}\sum_{i=1}^{N}(x_i)$
Median	$\left\lceil\frac{N+1}{2}\right\rceil\|(Sort_{i=1,\ldots,N})$
Mean Absolute Deviation	$\frac{1}{N}\sum_{i=1}^{N}(\|x_i - mean(x)\|)$
Median Absolute Deviation	$median(\|x_i - median(x)\|)$

1.1.6.1.3 Statistical Pivot Features

The statistical pivot features combine the statistical measurements with our pivot measurements to create *Statistical Pivot Features*. From every sequence of packets received from the Pivot Extraction Algorithm, we extract three additional measurements, the inter-arrival time between pivot packets (pivot time-distance), the number of normal packets between pivot packets (pivot packet-distance), and the total number of bytes between pivot packets (pivot length-distance). For each one of the three values, we compute the statistical measurements on each group which resulted in 39 statistical pivot features (3 different pivot measurements $\times$ 13 statistical measurements). Combined with the rest of the two previously discussed feature subsets, the total feature set considered for our framework is 78 features.

1.1.6.2 Feature Selection

In a classification problem, conventional supervised feature selection techniques such as Recursive Feature Elimination (RFE) and Permutation Feature Importance aim at selecting features that better distinguish a class label for every class that exist in a dataset.

In the context of unprofiled drone detection, the feature selection process would select the

features that work best in detecting profiled drones; the drones that were used in the training phase of the machine learning algorithm. In the testing phase however, the algorithm would correctly detect profiled drones while failing to detect unprofiled drones that were never seen by the algorithm during the training phase. This problem is known as overfitting, and it occurs –in the context of unprofiled drone detection– because the samples that are extracted from the same drone are correlated. To illustrate, consider two samples that are extracted from the same drone, sample SP_A and SP_B. Then consider some feature x. If we compare the value of feature x between samples SP_A and SP_B, we find that the value of x_A from sample SP_A will be statistically similar –with high probability– to the value of x_B from sample SP_B (i.e. $SP_{A_x} \approx SP_{B_x}$). In other words, because the feature values of different samples extracted from the same device are statistically similar, these samples are correlated. Consequently, if correlated samples (samples that are extracted from the same device) appear in the training set and in the testing set, it would create an undesired strong correlation between certain features and a specific device, which in turn would cause overfitting. It is a natural behavior for data points to be statistically similar to each other if extracted from the same drone because the network traffic generated by the drone is fairly unique for that drone. For example, the pivot size of WHT drone is 35-bytes. Therefore, 95% of samples extracted from WHT drone would have the "Pivot Size" feature equals to 35 (Recall that the Pivot Extraction Algorithm can find the correct pivot size with a probability of 0.95). This would cause the classifier to declare any WiFi device that has a pivot size of 46 bytes (BLK), 4 bytes (RED), or 35 bytes (WHT) as *drone* and any other pivot sizes as *non-drone* devices. This is an unwanted behavior because the model associates the unique pivot dimension with a class label, and ignoring the other features. This problem will persist regardless of the dataset size; the number of packets collected from drones since all drone-labeled samples are representing (in our case) three distinct drones and their three pivot sizes are uniquely consistent with every drone sample.

Of course this would not be a problem if we implement our model under the assumption that every drone type in the market must be profiled prior to the detection process. However, this is not a realistic assumption for creating a practical drone detection framework. To the best of our knowledge, no prior efforts that adopt a machine learning-based approach for drone detection has addressed this issue.

To this end, we next propose our novel feature selection technique that overcomes the overfitting problem via a two-phase process. The first phase computes a Feature Similarity Score based on Jaccard Similarity Index (model-independent feature selection), while the second phase applies a modified RFE for unprofiled drones (model-dependent feature selection). The objective of our feature selection technique is to select features which describe a generalized drone behavior rather than features that describe a behavior of a specific networked device. In other words, the selected features are expected to detect unprofiled drones; drones that the framework has never seen before and the classifier has never trained on.

The main motivation of applying Jaccard Similarity Index (JSI) testing before using an RFE-based feature selection, is that the computation of JSI scores is much faster than running an RFE-based algorithm which helps us weed out the weak features before passing them to the RFE process. For a comparison, it took our computer (a laptop with Intel i7) around 20 seconds to calculate the JSI scores for 78 features while it took the same computer around four hours to compute the RFE scores of the remaining features selected by JSI (which is 37 as we will see next). For the second phase, we opt for using a feature elimination-based technique (such as RFE) rather than feature extraction-based technique because feature extraction techniques such as Principal Component Analysis (PCA) would compress and retain the overfitting features into lower dimensions rather than discarding them.

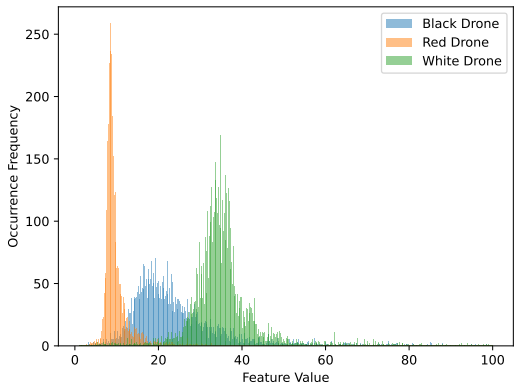
1.1.6.2.1 Phase I – Feature Similarity Score

As we have seen in Section 1.1.6.2, some features might describe a drone as a unique device rather than an FPV drone. For this reason, we need to identify the features that capture a generalized drone behavior and not a specific drone type. Our first step toward accomplishing this task is computing a Feature Similarity Score; a score that describes how a certain feature can capture the behavior of all the three drones we have in our experiment. Note that this feature selection approach is independent from the classification model used in the framework. In other words, Feature Similarity Score pre-processes the features without involving the adopted machine learning algorithm and therefore Feature Similarity Score results are always consistent regardless of the used classification algorithm.

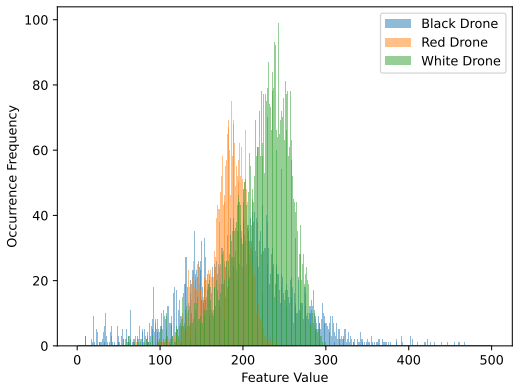
We first start by plotting the distribution of each feature as a histogram across all drones and measure the similarity of their distributions. For example, Figure 1.9a represents the distributions of *pivot_packet_distance_mean_root_square* feature for all three drones while Figure 1.9b represents the distribution for *pivot_packet_distance_total_size* feature for the same drones. From the Figure we can see that the first feature represents each drone individually while the second feature shows some shared characteristics among the drones. The main intuition is that if a feature shares similar distributions among different drones then this feature will most likely generalize a drone behavior while other features with distinctive distributions can help identify the specific device it represents. An extremely distinctive distribution such as the pivot size could overfit the data.

There are different techniques in the literature for testing the similarity between data distributions such as Pearson's chi-squared test and Kolmogorov–Smirnov test. We chose the Jaccard Similarity Index method because it is easier to interpret and provides greater flexibility in setting the appropriate similarity threshold that fits our desired application.

Pre-processing: Before testing the similarity of a feature between drones, we need to prepare



(a) Distribution of *pivot-packet-distance-mean-root-square* feature.



(b) Distribution of *pivot-packet-distance-total-size* feature.

Figure 1.9: Feature (a) uniquely describes each drone individually while feature (b) generalizes an FPV drone behavior.

the data points of that feature to be plotted as a histogram. First we remove the anomalies of each feature; the data points that are scattered (very few) and are either extremely large or extremely small compared to the majority of the data points. Since we don't have a prohibitively large number of features, we removed the anomalies for each feature manually by setting the histogram range for each feature to be within the range that capture most of the data points while leaving out the very few data points that are extremely out of range. Then we perform 'data binning'; a process of dividing the range into same-length intervals, each interval (a histogram bar) is referred to as a 'bin'. A data point that falls within the range of a certain bin will increase the value of that bin by one. If anomalies are not removed, it would collapse the bins together and severely change the shape of the histogram's distribution since the histogram range will be proportionally large for the majority of the data points. After removing the anomalies, we set the number of bins to 100 for all features as a unified bin interval.

Jaccard Similarity Index Measurement: After creating the histograms, we measure the size of the intersection area between the three drones' distributions. Then, we measure the occupation ratio of the intersection area to the total distribution area using a Jaccard index.

For each bin position B_i, we record the minimum bin value among the three drones bins. The intersection area is then computed as $\sum_{i=1}^{n} min(B_{i_{1...m}})$ where n is the number of bins and m is the number of histograms. Note that since we have three histograms (one for each drone), then $m = 3$. Similarly, the total area of the three histogram is $\sum_{i=1}^{n} max(B_{i_{1...m}})$. Finally, the Jaccard Similarity Index JSI is computed as:

$$JSI = \frac{\sum_{i=1}^{n} min(B_{i_{1...m}}))}{\sum_{i=1}^{n} max(B_{i_{1...m}})} \tag{1.1}$$

The resulted JSI gives us the ratio of how much of the intersected region occupies from the total sum of histograms areas.

Feature Selection Based On Jaccard Similarity Index: We computed the Jaccard Similarity Index for all of the 78 features based on equation 1.1. The results are sorted in ascended order and reported in Figure 1.10. Note that the Jaccard indices are ratios (≤ 1) and in the Figure they are multiplied by 100 for clarity. From the Figure, we notice that there are high variations between the top eight features where the index difference between features ft_i and ft_{i+1} ($\Delta JSI = ft_i - ft_{i+1}$) can reach up to 13 points between two consecutive features for the top eight indices. Then the ΔJSI started to stabilize to be ≈ 1 until it reaches to the 30^{th} feature which has $JSI = 12.71$ and $\Delta JSI = 2.64$. In other words, the number of useful features started to drop by $\Delta JSI = 2.64$ points after $JSI < 12$ and therefore we select the acceptance threshold to be 12, that is, the intersection region must occupy at least 12% of the total histogram areas. With this threshold, we reduce our features set from 78 features to 30 features. Additionally, we add another threshold that accepts a feature if it has $JSI > 70$ of intersection area between any two drones. The intuition of this additional threshold is

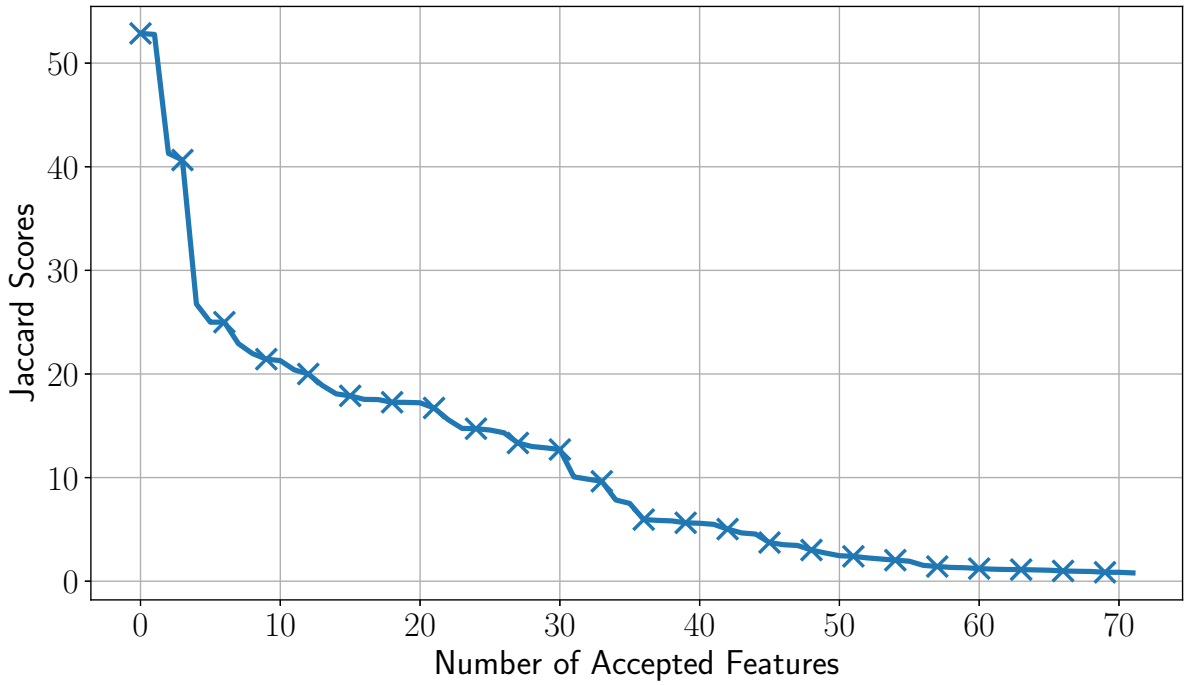


Figure 1.10: Jaccard Scores × 100. Each score defines the threshold of accepted features.

that, a certain feature might be a strong candidate for detecting FPV drones but one of the three drones we used during testing might have a unique and outlying distribution towards that particular feature and therefore would inject an anomaly in the JSI testing which would potentially lower the JSI score of rather a strong feature. Therefore, we re-computed JSI score for each feature thrice, once between each two drones. In other words, for each feature we computed JSI for drones combinations BLK-WHT, WHT-RED, and BLK-RED. Then, we set a heuristically high threshold in which we set to $JSI > 70$ and selected the features that has $JSI > 70$ in at least one of the three drones combinations. The new threshold gave us additional 7 more features which resulted in a total of 37 features.

1.1.6.2.2 Phase II – Recursive Feature Elimination for Unprofiled Drones

The features selected in Section 1.1.6.2.1 were selected independently from the underlying classification model used in the detection framework. Now we need to test the sensitivity

of the remaining features and further select the features that are most sensitive toward our Random Forest classifier in detecting new unseen drones. For this task, we implement a modified version of the Recursive Feature Elimination (RFE) technique that involves our FPV Drone Classifier which influences the RFE process to select features that have the discriminative power to detect unprofiled drones (Algorithm 1).

Algorithm 1: Modified Recursive Feature Elimination

```
1  iteration_results.initialize();
2  while feature_set.count ¿ 1 do
3      features_scores.initialize();
4      for ft in feature_set do
5          feat_set_no_ft = feature_set - {ft};
6          drone_subsets = k_fold_cross_validation(dataset);
7          ft_scores.initialize();
8          for drone_set in drone_subsets do
9              validation_set = drone_set;
10             train_set = drone_subsets - drone_set;
11             model = RandomForest(train_set, feat_set_no_ft);
12             accuracy = model(validation_set);
13             ft_scores.add(accuracy);
14         features_scores.add(ft, ft_scores.mean());
15     feature_set.remove_weakest_feature(features_scores);
16     iteration_results.record(feature_set);
17 final_feature_set = iteration_results.best_feature_set();
```

Since RFE involves the classification model in the selection process, the model would eventually influence the RFE to select the features that would cause overfitting for the reasons discussed in the beginning of Section 1.1.6.2. To overcome this issue, we modify the basic RFE technique and inject a Cross-Validation test in the step that invokes the classifier (line 9 to 13 in Algorithm 1). Since we have three drones, we apply a 3-folds cross-validation strategy, where we divide the dataset into three distinct sets or folds (line 6).

However, the samples are not distributed equally among the folds as in a conventional cross-validation. Instead, each fold contains samples from a single drone plus non-drone samples selected randomly. Then the RFE train on two folds and validate on the third fold. This process is repeated once for each drone/fold as the validation set. In other word, the RFE train on two drones and validate on new unseen drone then shuffle between the drones and train and validate again.

The accuracy for each fold is accumulated out of 300% (# of drones × 100 at line 13) then mean of the accuracy is computed (line 14) after the cross-validation is performed on all drones. At the end of each iteration (line 15), the feature that yields the maximum accuracy when removed (*i.e.,* weakest feature) is eliminated. Next, new and smaller set of features and their respective scores are then recorded (line 16) and the final feature set is selected at the end of the algorithm (line 17) based on the highest detection accuracy yielded by the best feature set. Our modified RFE process has further decreased the number of features from 37 to 12. It is worth noting that our features that are built from pivot packets are among the remaining features, and out of 28 statistical features, only three of them are kept including our 'Variance' feature in the final set. In summary, the selected features are: 3 from the Pivot Features group (is_MTU_Large, fingerprint_1, and fingerprint_2), 6 from the Pivot Statistical Features group (2 Pivot Time Distance features, 1 Pivot Packet Distance feature, and 3 Pivot Length Distance features), and 3 from the Statistical Features group (2 from Alipur et. al. [8] and our 'Variance' feature). We refer to the final set of selected features as **Pivot-Based Features**.

1.1.7 Implementation

In this Section, we provide an overview of our experimental setup, the devices and applications that are used to generate network traffic, and the data collection strategy used to generate our datasets.

1.1.7.1 Data Collection

This subsection provides a quick overview on the hardware and software setup used to collect the network traces to generate the framework datasets, as well as the devices and applications that generated such traces.

1.1.7.1.1 Hardware & Software Setup

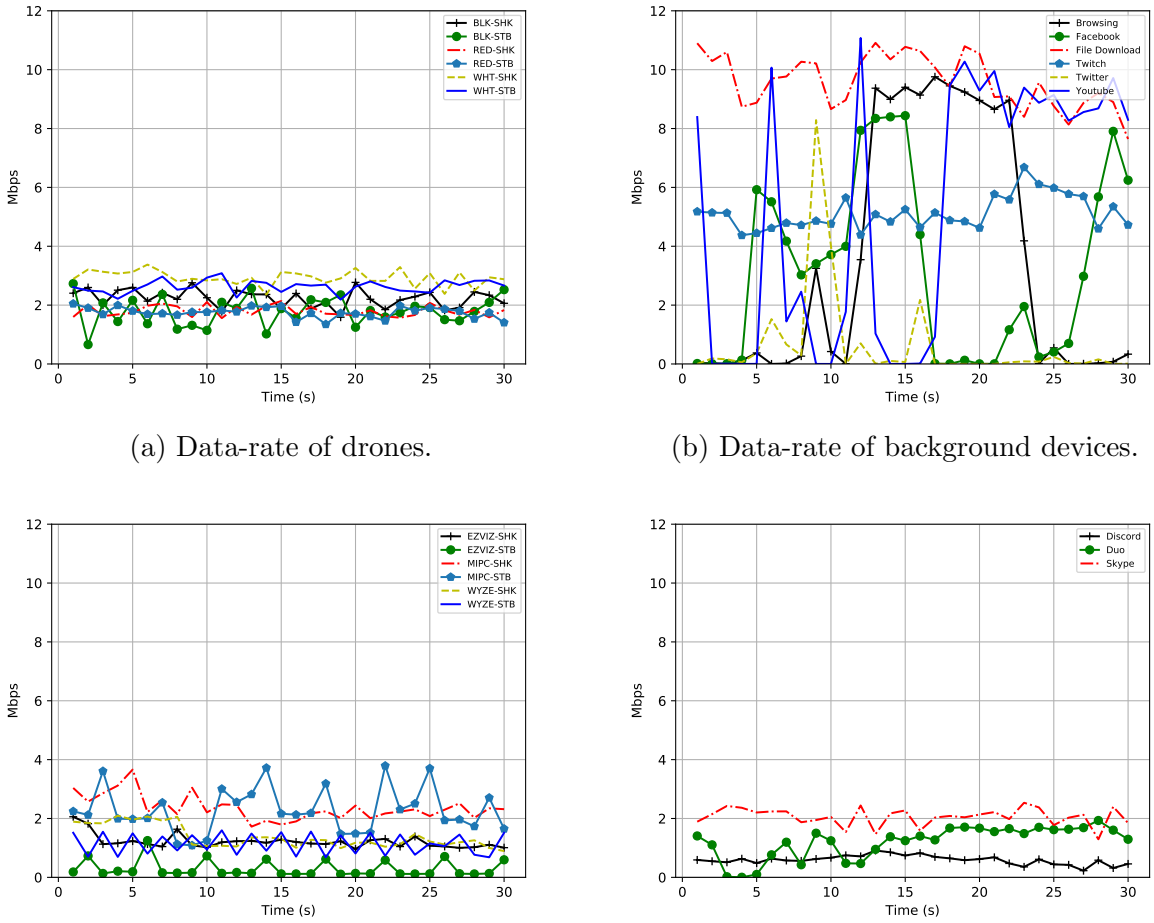
To collect the traffic emitted by WiFi devices, we used an Acer Aspire F5-573G-759N laptop with Intel Dual-Core i7 CPU and Alfa AWUS036ACH WiFi adapter. We used Kali Linux 64-Bit version 2020.1, which was installed as a virtual machine using Oracle VirtualBox version 6.0.14. The WiFi adapter was set to Monitor Mode and tuned to the operating frequency of the drone's AP being monitored. We used the Android app "WiFi Analyzer" [3] to find the correct operating frequency of the AP. The network traces were collected using Wireshark [4] and parsed using Scapy [2].

1.1.7.1.2 Network Traffic Generation

The network traffic traces are generated from various devices and applications in different conditions and scenarios. All traces are collected using the WiFi Monitor Mode (rather than on-device traffic capture) to include packet loss and other distortion effects that characterize real-world environments. The WiFi packets were captured from an encrypted WiFi network configured with a WPA2 security standard. WPA2 establishes a secure channel for each device using the CCMP protocol that uses the AES encryption algorithm to encrypt the packets payload. To maintain the state of the secure channel, CCMP attaches an 8-bytes header and an 8-bytes Message Integrity Code (MIC) to each packet. From each captured packet, we remove the CCMP header and its MIC which leaves us with only the WiFi header and the encrypted payload. The encrypted network traces are collected from the following devices and network services:

VoIP Apps: To test the performance of our framework, we ensure that our datasets include samples extracted from network devices that have traffic patterns similar to FPV drones'. In addition to the IoT camera traces collected for the pivot analysis in Section 1.1.5.4, we also

collect traces of video traffic originated from Skype, Google Duo, and Discord VoIP apps while making video calls for five minutes. In Figure 1.11, we can clearly see that bit-rates of FPV drones, IoT cameras, and VoIP applications have similar traffic patterns.



(a) Data-rate of drones.

(b) Data-rate of background devices.

(c) Data-rate of cameras.

(d) Data-rate of VoIP.

Figure 1.11: Data-Rate for Drones, Background Devices, Cameras and VoIP applications over 30 seconds.

Non-Drone Background Devices: To ensure the completeness of our dataset, we collect six different traces of network traffic originated from network services for three minutes each. These traces were collected during 1) file downloading, 2) non real-time video streaming (YouTube), 3) live internet video streaming (Twitch), 4) Internet browsing, and social media browsing: 5) Twitter and 6) Facebook. The bit-rate of each application is plotted in Figure

1.11b. Furthermore, we included traces from publicly available WiFi packet traces [72]. Similar to all of the network traffic traces collected by us, these traces were collected in WiFi Monitor Mode which make them subject to the uncertainties of wireless networks.

Drones & IoT Cameras: The traces from drones and IoT cameras that are collected for the pivot analysis in Section 1.1.5.2 and 1.1.5.4 respectively, are collected under two different extreme cases; a completely stable video scene (STB) and a highly dynamic video scene (SHK). To include network traffic generated from a more realistic video pattern, we place all of the three drones and all of the three cameras in front of a screen playing a five minutes documentary video that contains slow, moderate, and fast-moving scenes. From each device, we capture the emitted video traffic which resulted in an additional six network traffic traces.

Flying Drones: To further ensure that our dataset includes realistic samples, we recorded the FPV video of our drones during flight. First, we performed an experiment to determine the maximum distance from the monitoring station at which a drone can be detected. We set up a laptop as a monitoring station in the center of a university football field. Then we set up seven waypoints in a straight line starting 10 meters from the monitoring station. The distance between each two consecutive waypoints is 10 meters. We then flew each drone starting from the monitoring station in a straight line to the first waypoint while collecting the packets sent by the drone. Then, we flew the drone from the first waypoint to the second, up to the seventh while collecting the transmitted packets at each step. The drones we used are considered "micro-drones", and move at a relatively low speed with a rather small tilting of the drone body during motion.

We notice that the RSSI severely drops after 40 meters at waypoint 4 and the received power level at the monitoring station's antenna is below -80 dBm. This is expected, as these drones are low-power devices with limited battery capacity. The packet loss between waypoint 3

and waypoint 4 was excessively high (around 70%). Therefore, we conclude that in the considered hardware configuration, the effective detection distance for the micro-drones is 30 meters.

From our monitoring station in the center of the football field, we flew each drone for three minutes inside a detection area of radius equal to 30 meters while the drone occasionally exits and re-enters the area to simulate a more realistic scenario of invasive drones. For each drone, we collected the emitted video traffic which resulted in three new network traffic traces.

1.1.7.2 Data Preprocessing

For each captured WiFi packet P_i transmitted by device D_j, we extract from the WiFi packet header the transmitter's MAC address, fragment number, sequence number, header flags, payload size $S_{i,j}$, and packet's arrival time $I_{i,j}$. We also extract the RSSI from the radiotap header that is added by the WiFi card upon receiving the packet. In case of fragmented packets, we use the sequence number, fragment number, and the "more fragments" flag in the header to put fragmented WiFi packets back into a complete packet. This approach is needed as WiFi adapters set into Monitor Mode do not reconstruct fragmented packets, and pass them individually to the operating system. We use the sequence number to identify retransmitted packets that were already captured. Note that we could not rely on the "retry" flag to capture duplicates because is not guaranteed that the original packet transmission has already been captured. Also a simple check of consecutive duplicate sequence numbers would not work either, since some retransmissions might arrive out of order, and thus the last captured packet is not always followed by its retransmitted duplicate. As an alternative solution for identifying duplicates, we simply keep track of the last eight captured packets for each D_j and whenever a new packet is received, we check if the packet is already in this window.

For each captured stream of n WiFi packets, we reconstruct the fragmented, drop the duplicates, then group them as a list of packets based on the transmitter's MAC address such as $D_j = \{P_{1,j}, ..., P_{n,j}\}$. For each packet i, we only save 1) the packet size S_i and 2) its arrival time I_i. Therefore, each device's packet list can be interpreted as $D_j = \{(S_{1,j}, I_{1,j}), ..., (S_{n,j}, I_{n,j})\}$ where $P_{i,j} = (S_{i,j}, I_{i,j})$. Then, we add each device list D_j to our Packet Dataset $DS_p = \{D_1, ..., D_m\}$.

1.1.7.3 Data Sampling

After preparing DS_p, the packets are grouped into batches called samples (SP). Each SP consists of at least 170 packets that are sent within at least 820ms interval (The reason for sampling packets in batches with a minimum of 170 packets and a minimum of 820ms in inter-arrival window is discussed in Section 1.1.5.3). This process transforms DS_p into a Samples Dataset DS_{sp}. To illustrate, to generate the first sample $SP_{1,j}$ for a given device $D_j \in DS_p$, a sliding-window strategy is used; the sliding-window is filled with 170 $P_{i,j}$ packets for device D_j. Then if $I_{170,j} - I_{1,j} < 820ms$, the sliding-window is filled with additional n packets until $I_{170+n,j} - I_{1,j} \geq 820ms$. Finally, the current packets in the sliding-window is recorded as $SP_{1,j}$ and added under device D_j. For the next sample, the sliding-window shifts by 30 packets to create the second sample $SP_{2,j} = \{P_{31,j}, ..., P_{201,j}\}$ and add additional n packets until $I_{201+n,j} - I_{31,j} \geq 820ms$ is satisfied. This sampling process continues until all packets for D_j are batched into m samples such that $D_j = \{SP_{1,j}, ..., SP_{m,j}\}$ where typically $n > m$ since each two consecutive samples (with at least 170 packets) have 30 interleaving packets offset. The reason for this offset choice is because during the pivot analysis, we observed that two pivot packets are separated by a maximum of 30 normal non-pivot packets. In other words, 30-packets is the maximum offset that includes all possible pivots combinations among consecutive samples.

The final dataset contains the following number of samples: Drones: {WHT Drone: 6041

samples, RED drone: 3194 samples, BLK drone: 4626 samples}, Non-Drone Devices: {IoT cameras: 11449 samples, Internet activity: 4523 samples, VoIP apps: 1463 samples, public traces [72]: 474 samples}. 70% of the samples of each class will be used in our RFE feature selection process (Section 1.1.6.2.2), while the remaining 30% of the samples will be retained to evaluate the performance of the model (Section 1.1.8).

1.1.7.4 Features Extraction

The last step in creating our datasets is extracting features from the samples. Since the samples are already processed, features are easily computed from each sample $SP_{i,j}$ producing a machine learning record $R_{i,j}$ that contains a set of k features Ft where $R_{i,j} = \{Ft_{i,j_1}, ..., Ft_{i,j_k}\}$. The feature design is already discussed in detail in Section 1.1.6.1 and reported in tables 1.2 and 1.3 while the feature selection strategy is discussed in Section 1.1.6.2. Since the framework follows a supervised machine learning paradigm, each record is labeled 1 if it belongs to a drone, and 0 otherwise.

1.1.7.5 Scalability

As discussed in Section 1.1.5.3, the proposed Pivot Extraction Algorithm runs linearly in $O(n)$ of time complexity. When tested on our hardware and software (Acer laptop with 2-cores i7 CPU running Python 3.8), the algorithm traversed 100,000 WiFi packets within 758ms in a single Python process. When running the algorithm through 200,000 WiFi packets using two Python processes running in parallel (each process traverses 100,000 packets), the execution time for each process was around 896ms. Processing 400,000 WiFi packets using four Python processes (each process traverses 100,000 packets), the execution time for each process was around 960ms. On average, the packet generation time of a drone is 200 packets per second. Therefore without any code or hardware optimizations, our algorithm

can process wireless traffic of 2000 devices simultaneously using only four parallel Python processes running on a 2-cores machine (assuming every device generates 200 packets per second).

1.1.8 Evaluation

In this Section, we evaluate the detection performance of our proposed framework using the features created and selected in Section 1.1.6 and the dataset generated in Section 1.1.7. During the feature selection process, we used only 70% of the dataset and kept the remaining 30% for the final evaluation. With the the remaining 30%, we employ a 3-folds cross-validation technique. In a conventional cross-validation, the dataset is split into training and testing sets by specific ratios (such as 80/20). However, to test the framework's ability in detecting unprofiled drones, we split the remaining 30% into training and testing sets by drone type. In particular, the training set is designated to contain samples of two different drones plus randomly selected non-drone samples drawn from the designated 30% pool, while the testing set contains samples from the third drone plus randomly selected non-drone samples drawn from the designated 30% pool as well. To balance the datasets, the non-drone samples are added to each set (training and testing) in a 50/50 drone to non-drone ratio. To create the second train/test set pair, we move the test set drone samples to the training set while moving one of the training set drones into the testing set. Then we redistribute the non-drone samples to 50/50 drone to non-drone ratio. The third train/test set pair follows the same process; designate the test set for the last drone then redistribute the non-drone samples. This 3-folds cross-validation setup covers all possible combinations of having two drones in the training set and the third drone in the testing set. Finally, to assert that the initial traffic analysis of the three drones used in Sections 1.1.5.1 and 1.1.6.2 did not influence or interfere with the results of the evaluation of our experiments, we collected additional samples from a fourth drone as a baseline; a 3DR Solo drone (Figure 1.12). The

Solo drone was never used in any shape or form in neither the analysis of the pivot behavior nor the feature selection process. The samples of the new drone were collected in the same manner as in Section 1.1.7.1 and is comprised of 398 samples. Similar to the previous three train/test set pairs, we use the Solo drone to create a fourth train/test set pair where we designate the samples of the Solo drone to be part of the test set.

The evaluation of the framework is divided into two sets of experiments, each set is executed under four different experiment settings. In the first set, we apply the features that were selected by our feature selection strategy proposed in Section 1.1.6.2 which is comprised of our pivot-based features. In the second set, we apply the statistical features only (Table 1.3) which was proposed by [8]. Recall that one of the main motivations of the proposed framework is detecting unprofiled drones that the framework has never seen before. To test the framework's performance in detecting unprofiled drones, we designate a drone to the testing set that was never used in the training set. In particular, the setup of the four experiments are the following; for each experiment, the classifier is 1) trained on RED and BLK and tested on WHT, 2) trained on WHT and RED and tested on BLK, 3) trained on BLK and WHT and tested on RED, and 4) trained on RED, BLK, and WHT and tested on Solo. As mentioned before, to balance the datasets (training and testing) for each experiment, we include non-drone samples in a 50/50 drone to non-drone ratio. Note that in each experiment, we construct a new Random Forest algorithm and fit it with its designated training set to make sure the new drone used in the testing set is unprofiled and never seen by the classifier. The setup and results for each experiment is reported in Table 1.4.

1.1.8.1 Results Analysis

The results reported in Table 1.4 demonstrate that pivot-based features can detect unprofiled drones with accuracy of at least 93%. On the other hand, statistical features – such as those used in state-of-the-art machine learning-based drone detection – struggle at detecting

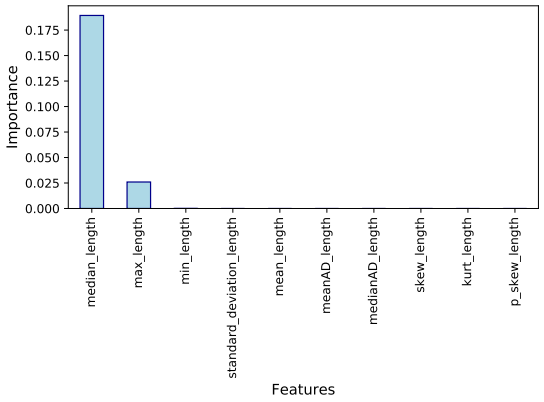
Table 1.4: Final results.

	# of samples per device in training set	# of samples per device in testing set	Pivot Features			Statistical Features		
			Accuracy	Precision	Recall	Accuracy	Precision	Recall
Experiment 1	Red Drone = 959 samples Black Drone = 1388 samples Noise Samples = 2347 samples	White Drone = 1812 samples Noise Samples= 1812 samples	97%	98%	97%	49%	0%	0%
Experiment 2	Red Drone = 959 samples White Drone = 1812 samples Noise Samples = 2771 samples	Black Drone = 1388 samples Noise Samples= 1388 samples	93%	98%	87%	69%	99%	39%
Experiment 3	White Drone = 1812 samples Black Drone = 1388 samples Noise Samples = 3200 samples	Red Drone = 959 samples Noise Samples= 959 samples	99%	98%	99%	89%	99%	79%
Experiment 4	Red Drone = 959 samples Black Drone = 1388 samples White Drone = 1812 samples Noise Samples = 4159 samples	Solo Drone = 398 samples Noise Samples= 398 samples	98%	97%	99%	50%	0%	0%

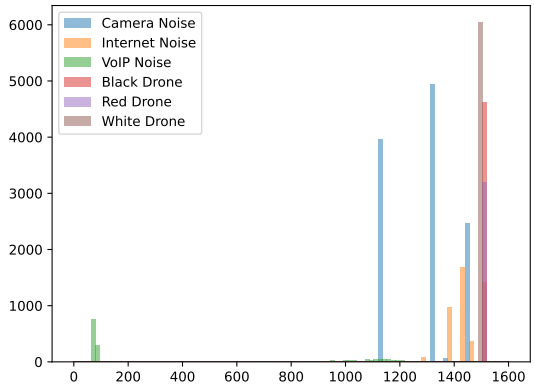
Figure 1.12: 3DR Solo Drone that was used as part of the final test set

unprofiled drones, especially when detecting WHT and Solo drone; statistical features failed to detect even a single WHT or Solo drone sample. Recall that the network traffic originated by the Solo drone is completely new and unknown to the framework and were never used neither during the analysis of FPV drone behavior (Section 1.1.5.1) nor during the RFE feature selection process (Section 1.1.6.2.2). To understand the drawbacks of the statistical features, we apply a permutation feature importance technique to score the importance index for the statistical features only (Figure 1.13a). From the Figure, we observe that the one and only important feature is the median; the middle value of a sorted list of values. As discussed in Section 1.1.4.3, a video stream is comprised mostly of MTU packets and therefore the median of a series of 170 packets is almost always the MTU size; maximum packet size in the sample. The second most important feature is the packet with the maximum payload (*i.e.,* MTU) which essentially reflects the same value of the median feature. Then we plot the distribution of the MTU for each device in Figure 1.13b. As we can observe, there is almost no variance in the distribution of the MTU. In other words, every device has a unique MTU size that is tied to its network interface card (NIC) and is represented as a single bar in Figure 1.13b. This explains why the classification model privileges this single feature in the detection decision. In fact, the model can associate MTUs of 1500 bytes (RED and BLK drones) and 1482 bytes (WHT drone) to drone devices and any other MTU sizes to a non-drone device. This also explains why the detection accuracy drops to 55% when WHT drone is removed from the training set (since RED and BLK have the same MTU size).

On the other hand, our feature selection strategy ensures the removal of features that have a single or very few distinct values across samples drawn from the same device such as MTU or pivot size which have the potential to uniquely identify a device rather than a group of devices that share the same characteristics (*i.e.,* FPV drones). Furthermore, the selected features are mostly influenced by the behavior of pivot packets that indeed capture a generalized drone behavior rather than a single unique device behavior.



(a) Permutation Feature Importance for Statistical Features.



(b) Distribution of the MTU feature. Note that "Camera Noise" is comprised of three distinct IoT cameras.

Figure 1.13: Fixed-valued features such as MTU force the classifier to heavily rely on a single feature

1.1.9 Drone Detection Conclusion

In this module, we proposed a COTS Drone Detection Framework that can detect invasive FPV drones flying into a restricted area without the need of having a prior profile of the invasive drone or requiring the drone's controller to be within the detection range. The framework rely synchronization packets that are interleaved with the video stream emitted by drones. These packets are denoted as "pivots" and unlike prior work, pivot packets are not affected by the varying bit-rate of the drone's video stream which can be used as a guideline to construct features for a machine learning-based system built from a Random Forest classifier. We also proposed a *Pivot Extraction Algorithm* algorithm that quickly searches a sample of packets for pivots in linear time. Furthermore, we proposed two-phase feature selection strategy that selects drone features which have the discriminative power to detect unprofiled drone. The first phase is a model-independent feature selection process which is based on Jaccard Similarity Index, while the second phase is model-dependent and based on the RFE selection process.

Our experiments demonstrated that pivot-based features can detect unprofiled FPV drones even when other video streaming devices such as IoT cameras are within the detection environment. The detection accuracy using our pivot-based features is at least 93% using at least 170 captured packets that are transmitted within at least 820ms, while the detection accuracy of using the state-of-the-art drone features [8] is at least 49% using the same number of packets and detection window.

1.2 Second Module: Drone Authentication

In the last 10 years, commercial unmanned aerial vehicles (UAVs) have emerged as a key component in many applications such as goods delivery, photogrammetry, environmental research and surveying, emergency first-responders, and public safety monitoring [60, 54, 76, 34]. However, an increased drone activity within the national airspace would eventually create various technical challenges and safety concerns. To address the latter issue, the FAA and NASA have jointly established a Research Transition Team (RTT) to initiate the development of a UAS Traffic Management (UTM) system that is separate from, but complementary to, the Air Traffic Management (ATM) system which manages manned aircrafts [40]. Under the current regulations, a UAV operator would typically need to apply for a flight authorization via an automated service called Low Altitude Authorization and Notification Capability (LAANC) [39] for piloted line of sight (LOS) flight missions only. The anticipated UTM system on the other hand will set and enforce regulations for efficient and safe autonomous and beyond visual line of sight (BVLOS) UAV operations. One of the main UTM components is the **Remote-ID** (RID) module, which will be designed to help identify UAVs in flight [41].

There are some efforts in the industry attempting to implement an actual Remote-ID system. AirMap [7], Wing [86], and Kittyhawk.io [53] (each is a UAS Service Suppliers (USS) for LAANC) have demonstrated a network-based Remote-ID system [6] that exchanges UAV information across different USSs via InterUSS Platform, a platform that is designed to connect multiple USSs together [49]. On the other hand, Intel has introduced "Open Drone ID project" [64], a broadcast-based Remote-ID implementation of the "ASTM F3411 Remote ID and Tracking Specification" [11]. Open Drone ID has an app implementation for iOS and Android. It uses Bluetooth 5.0 to search for drones that are equipped with Open Drone ID for up to 1 km. The drone sends two types of broadcast messages, static messages (include static information such as drone ID) and dynamic messages (include dynamic information

such as current GPS coordinates) which are broadcasted more frequently than the static ones.

The UAS Identification and Tracking (UAS ID) Aviation Rulemaking Committee (ARC) has introduced two different approaches for implementing Remote ID [37]: Direct Broadcast and Network Publishing (Figure 1.14). In the direct broadcast approach, the UAV continuously broadcasts identifying information such as its unique ID, tracking information, and UAS owner information. In the network publishing approach, these identifying information are uploaded to a database over the internet. Anyone with an access to such database would be able to verify the identity of any flying UAVs without the need of receiving direct broadcasts from the UAV itself.

On the 31st of December 2019, the FAA has proposed a new set of regulations for Remote Identification of Unmanned Aircraft Systems [38], which included rules defining what classes of UAVs are required to support the RID hardware and software. The final rule of these regulations were published on the 15th of January 2021 [41] and it removed the Network Publishing requirement while setting the effective compliance date for drone manufacturers to equip their UAVs with Remote ID capability by September 16th 2022. For drone pilots, the effective compliance date to start operating UAVs equipped with Remote ID capability is September 16th 2023. Most importantly, the final rule included the structure of the Remote ID message. The broadcasted message would include several elements such as a unique UAV identifier, UAV location, take-off location, and pilot location. Revealing such information to the public is intended to enhance the public awareness toward UAV activities. While on the other hand, making this information available to the public have raised **privacy concerns** among the drone hobbyist community and UAS operators at large [46][59][26].

Threat Model: Given the discussed Remote ID framework, we formulate the following threat model. We assume a given restricted airspace that only allows authorized UAVs to pass through the airspace by broadcasting authorization credentials. Furthermore, we

assume a malicious receiver on the ground that receives the Remote ID broadcasts and attempts to stalk and the UAV and its pilot (i.e. follows the UAV and locate the position of the pilot). Under this threat model, we formulate the following attacks:

1) UAV Authorization: a UAV might attempt to fly over a restricted airspace without possessing the required authorization credentials.

2) UAV Impersonation: a UAV might attempt to use the flight credentials of another UAV in order to access a restricted airspace.

3) UAV Stalking a broadcast receiver on the ground might attempt to target and stalk a specific UAV by following the UAV unique identifier embedded in Remote ID message.

4) Pilot Stalking a broadcast receiver on the ground might attempt to target and stalk the UAV pilot using the controller location embedded in Remote ID message.

Contributions: In this module, we propose a Privacy-Preserving Authentication framework that extends the Remote ID framework to 1) securely and anonymously authenticate a UAV during its flight without revealing the identity of its operator except to the authorities, and 2) verify the authenticity of the flight permissions held by the UAV for its current flying area and flying time without revealing the UAV's entire flight path that might lead to the UAS operator's location. With these two features, all of the formulated attacks in the threat model can be mitigated. Furthermore, our framework leverages the signature aggregation capability of the Boneh–Gentry-Lynn–Shacham (BGLS) digital signature scheme to shorten the message size and saves the bandwidth of the broadcast channel.

The rest of the module is organized as follows. In Section 1.2.1, we discuss related work in the area of UAV authentication. Section 1.2.2 goes over a brief overview of the proposed framework. In Section 1.2.3, we provide background and a formal description of aggregate signatures. In Section 1.2.4, we discuss our flight plan slicing technique and provide the

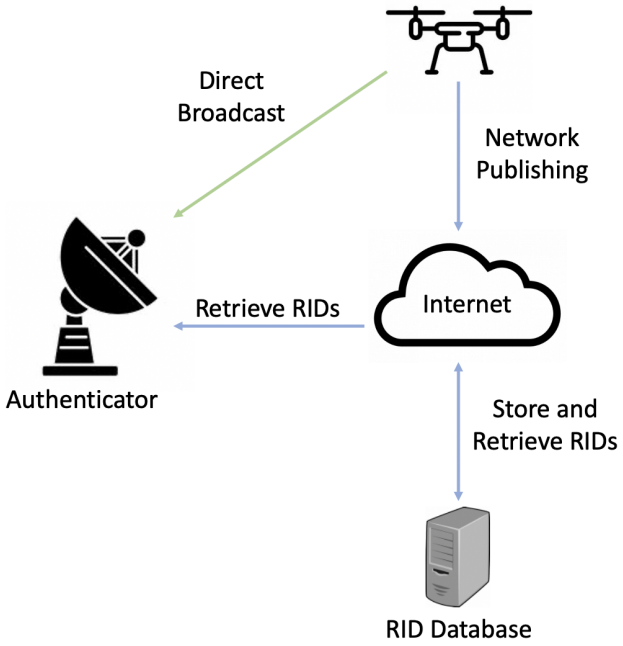


Figure 1.14: Authentication Channels.

reasoning behind the proposed technique. Section 1.2.5 discusses the design of the privacy-preserving authentication framework and describes in detail how RID messages are constructed and disseminated among different entities in the framework. Section 1.2.6 provides a security analysis of the proposed framework. In Section 1.2.7, we provide experimental results to evaluate the computational cost of BGLS as well as comparative analysis of the ECDSA and BGLS signature schemes. Section 1.2.8 concludes the module.

1.2.1 Related Work

In [77], the authors proposed a double-authentication watermarking scheme for a network of UAVs that are clustered as nodes in a tree network architecture where cryptographically hashed watermarks are aggregated at cluster heads using a chaotic logistic mapping. Unlike our framework, the proposed scheme incurs additional overhead on the cluster head node that continuously verifies and aggregates all received data streams. Furthermore, only the nodes who possess the secret key can verify the watermarks in this scheme.

In [85], the authors propose an authentication and key agreement scheme between drones and users who connect and acquire real-time information from them. The scheme relies on secure session establishment using pseudo-identities that are facilitated by a trusted server. However, in this scheme the server must be online during the authentication phase for each session between drones and users, whereas in our proposed framework, all pseudo-identities are signed and uploaded to the drone prior to each flight mission.

In [28], the authors proposed Traceable and Privacy-Preserving Authentication scheme for UAV systems based on elliptic curve cryptography. The scheme describes a trusted authority that generates public and private key pairs for UAV manufacturers, users, and ground control stations (GCS) as certificates. Furthermore, whenever a user buys or rents a UAV, a public and private key pair is generated and provided by the UAV manufacturer as a UAV certificate. The Traceable Privacy-Preserving property comes from the fact that only the trusted authority knows the real identity of the secret key owner. The drawback of this approach is that the authority and the manufacturer are given excessive trust since they possess the secret keys of the users and drones respectively. Furthermore, the authentication process involves the UAV manufacturer in every flight request which would add an unnecessary key management overhead to the system.

In [80], the authors proposed a privacy-preserving authentication framework for UAVs to authenticate each other via mobile edge computing (MEC) using pseudonym UAV certificates that have short expiration periods (minutes to hours) to prevent UAV tracking across different MECs. Despite utilizing pseudonym certificates, a UAV's unique master public key is used each time when it connects to a MEC, which implies that a compromised MEC has the potential to revoke the anonymity of UAVs connecting to it. Furthermore, the framework does not implement any authentication procedures with external parties outside of the network nodes (UAVs and MECs) as well as it does not authenticate whether or not a UAV has a permission to fly over a MEC at the connection time.

In [12], the authors proposed a privacy preserved authentication architecture based on ID-Based Signcryption. In this architecture, UAVs are required to equip an RFID tag that contains the UAV's real ID while distributed WiFi Access Points (AP) are required to equip an RFID reader. Whenever a UAV enters an AP coverage, the UAV scans its tag via the AP's reader to send its real ID to an identity server which in turn replies back with a pseudonym ID for the UAV to use in its current AP coverage for authenticating with other UAVs under the same coverage. The main drawback of this architecture is the impractical use of RFIDs where a UAV must fly too close (6 to 9 meters) over an AP. Furthermore, the architecture assumes an AP coverage for the authentication process to take place.

1.2.2 Framework Overview

As discussed in the threat model, the Remote ID framework is susceptible to different types of attacks. To mitigate each one of them, we propose our Privacy-Preserving Authentication framework that can be used as an extension to Remote ID. The two key components that comprised the framework are described next.

1) Flight Plan Slicing Technique: typically before each flight, a UAV is required to submit a flight request to the authorities, which includes the flight plan the UAV is intending to follow [39]. Upon request approval, the UAV receives a permission to fly in a specified region at a specified time. In case of autonomous UAV flight, a flight plan includes a flight path that is represented as a series of predetermined waypoints. Our proposed flight plan slicing technique divides the flight plan into a series of contiguous flight zones (Figure 1.15). Each zone describes a localized UAV trajectory which estimates the times at which the UAV enters and exits the zone along with the expected altitude, direction, and speed. Each flight zone is represented as timed waypoints and signed by an authoritative entity (e.g., FAA). The signed flight zone acts as a spatial and temporal *flight permission* which is used by the UAV to prove

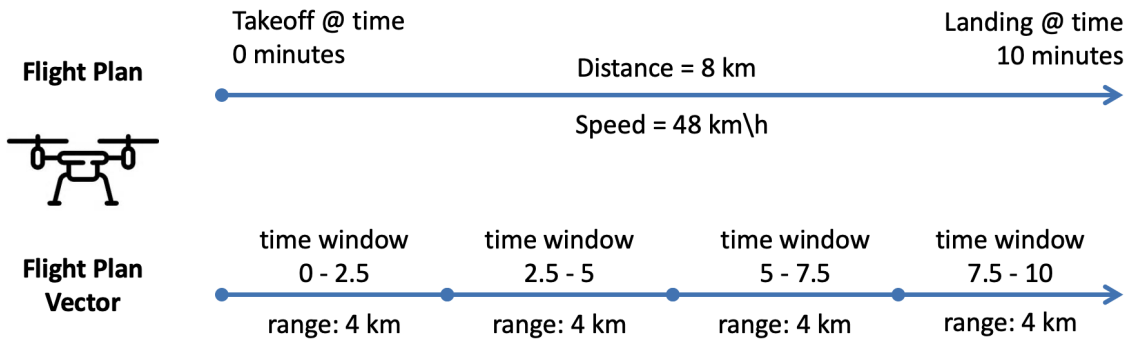


Figure 1.15: Flight Plan Slicing: dividing the flight path into contiguous Flight Zones.

to any third party that the UAV has a permission to fly in or pass through its current region at the current time. By using this flight planning technique, we can introduce **unlinkability** into the framework; distributing the identity of the UAV over multiple Pseudonym IDs that cannot be traced back to the same UAV. In other words, the UAV would replace its unique ID with a continuously changing pseudonym ID where the UAV would use a new anonymous identifier whenever it enters a new flight zone. This process makes it more difficult for a stalker to follow a UAV based on its broadcasts.

2) Remote-ID Message Structure: a Remote-ID (RID) is an identification message that is continuously broadcasted by the UAV. It includes an authorization for the current flight zone and a pseudonymous UAV certificate signed by an authoritative entity. It also includes UAV telemetry information signed by the UAV's pseudonymous certificate. To hide the pilot location without removing it from the broadcasted message, the location coordinates are encrypted by the authoritative entity's public key such that only the authorities can decrypt the retrieve the pilot location. This message structure enables authenticators to anonymously verify the authenticity of flying UAVs as well as authenticate UAVs' flight permissions without revealing neither the identity of its operators nor its entire flight path, while at the same time keeping any identifying information accessible to the authorities in case of a dispute. Furthermore to decrease the message size, the message structure leverages the fast signature aggregation capability of the Boneh–Gentry-Lynn–Shacham (BGLS)

digital signature scheme, which allows any arbitrary number of signatures to be aggregated as a single signature for fast and energy-efficient transmission of authentication credentials over the network. The use of signature aggregation is motivated by the fact that energy-constrained devices such as UAVs benefit from reduced size of transmitted messages more than the reduced size of messages processed locally in terms of energy consumption [22].

1.2.3 Cryptographic Building Blocks

In this section, we review the cryptographic primitives that we utilize throughout this work. We start by describing traditional cryptographic signature schemes. We, then, provide a formal definition for aggregate signatures. Readers who are familiar with these primitives can skip this section.

Signature schemes are fundamental objects from cryptography that allow a sender to use a secret key SK to cryptographically sign a message msg and send a signature σ to a receiver. Successful validation of σ indicates that σ has been created by the sender holding SK; the security of the signature schemes makes sure that for all adversaries $\mathcal{A}$ who observes valid pairs (msg, σ), $\mathcal{A}$ cannot forge a valid σ' for any message $\mathsf{msg}' \neq \mathsf{msg}$ without knowledge of SK. For applications that require exchange of many signatures $\Sigma := \{\sigma_i\}_i$ however, such signature schemes impose additional bandwidth overhead since each $\sigma_i \in \Sigma$ has to be individually exchanged, and thus the cumulative size of signatures grows linearly with the number of signatures.

Aggregate signatures are signature schemes with features that make them attractive in many applications: they allow a set of N signatures $\Sigma := \{\sigma_i\}_i$ created by N users under N key pairs $\{(\mathsf{PK}_i, \mathsf{SK}_i)\}_i$ on N messages $\mathcal{M} := \{msg_i\}_i$, where $i \in \{1, \ldots, N\}$, to be aggregated into a single signature $\sigma = \sigma_1 \circ \cdots \circ \sigma_N$ by an aggregating entity (not necessarily associated with the users or even a trusted entity). The advantage is that the aggregate signature σ

has the size of a single signature of the underlying signature scheme. We formally describe aggregate signatures next.

Definition 1: An aggregate signature scheme is a tuple of efficient algorithms $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Aggregate}, \mathsf{VerifyAggregate})$ which satisfies the following syntax, correctness, and security properties.

- **Syntax:** $\mathsf{KeyGen}(1^n)$ outputs a key pair $(\mathsf{PK}_i, \mathsf{SK}_i)$; $\mathsf{Sign}(\mathsf{SK}_i, \mathsf{msg}_i)$ outputs a signature σ_i; $\mathsf{Verify}(\mathsf{PK}_i, \mathsf{msg}_i, \sigma_i)$ outputs a bit; $\mathsf{Aggregate}(\{\sigma_i\}_{i\in\{1,\ldots,N\}})$ outputs a signature σ; $\mathsf{VerifyAggregate}(\{\mathsf{PK}_i\}_i, \{\mathsf{msg}_i\}_i, \sigma)$ outputs a bit where $i \in \{1, \ldots, N\}$.

- **Correctness:** For all $\{(\mathsf{PK}_i, \mathsf{SK}_i)\}_i$ in the support of KeyGen, and messages $\{\mathsf{msg}_i\}_i$: $\mathsf{Verify}(\mathsf{PK}_i, \mathsf{msg}_i, \sigma_i) = 1$ holds with probability 1 where $\sigma_i = \mathsf{Sign}(\mathsf{SK}_i, \mathsf{msg}_i)$, and $\mathsf{VerifyAggregate}(\{\mathsf{PK}_i\}_i, \{\mathsf{msg}_i\}_i, \sigma) = 1$ holds with probability 1 where $\sigma = \mathsf{Aggregate}(\{\sigma_i\}_i)$ for $i \in \{1, \ldots, N\}$.

- **Security:** The probability that any efficient adversary $\mathcal{A}$ wins the aggregate signature forgery game described in [19] is negligible.

Remark. In the aggregate signature security game described in [19], $\mathcal{A}$ is given a lot of power, namely it holds $(\mathsf{PK}_1), (\mathsf{SK}_2, \mathsf{PK}_2), \ldots, (\mathsf{SK}_N, \mathsf{PK}_N)$, chooses messages $\mathcal{M} := \{\mathsf{msg}_1, \ldots, \mathsf{msg}_N\}$ to be signed, has access to a signing oracle for SK_1 (without $\mathcal{A}$ knowledge of SK_1) to sign any $\mathsf{msg}' \neq \mathsf{msg}_1$ and is challenged by a challenger $\mathcal{C}$ to produce the aggregate signature $\sigma = \mathsf{Aggregate}(\{\sigma_i\}_i)$ where $i \in \{1, \ldots, N\}$, indicating inability of $\mathcal{A}$ to forge an aggregate signature σ; please see [19] for details on security.

In our authentication framework, we use the provably secure aggregate signature scheme BGLS proposed in [19]. Our framework design is modular, so that if future, more efficient, aggregate signature schemes are proposed, we could flexibly use them instead of BGLS.

1.2.4 UAV Flight Planning & Flight Plan Slicing

In autonomous flight missions, an Unmanned Aerial System (UAS) operator generates a flight plan for the UAV to follow. There are different techniques and algorithms for different objectives and motivations to calculate and design a UAV flight plan. Generally, when a flight planning algorithm calculates the optimal path based on the task's navigational requirements, it outputs the path as a series of waypoints; a 3D point in space that includes the latitude, longitude, and altitude of the waypoint. In this framework, our main objective is to anonymously authenticate the flight permissions of UAVs without revealing their entire flight path while asserting to any outside authenticator that the UAV possesses a permission to fly over a particular area.

In this section, we introduce a Flight Plan Slicing technique; an approach for dividing the waypoints that is being followed by a UAV into smaller blocks of contiguous Flight Zones (FZs). Each FZ is a localized UAV trajectory that estimates the time at which the UAV enters and exists the FZ given the speed of the UAV and the path it follows. Therefore, each FZ can be transformed into a signed Flight Certificate (FC) which can be used to prove to any third party that the UAV has a permission to fly over its current time and location (Figure 1.17). To introduce anonymity between different FZs, each FC includes a unique cryptographic public key (PK) that belongs to the UAV. Therefore, the UAV would appear as a new UAV whenever it enters and exits a new FZ. A more detailed discussion about the structure of Remote-ID message (Figure 1.17) will be explored in section 1.2.5. In this section, we discuss the basis of determining the size of each FZ. In our discussion, we consider Linear Flight Planning; a flight planning technique for UAVs traveling mostly in straight lines. Linear Flight Planning is a popular flight planning technique since most UAV flight missions prefer the shortest straight path between two waypoints. Furthermore, more complex flight plans can be constructed from a composition of linear flight plans.

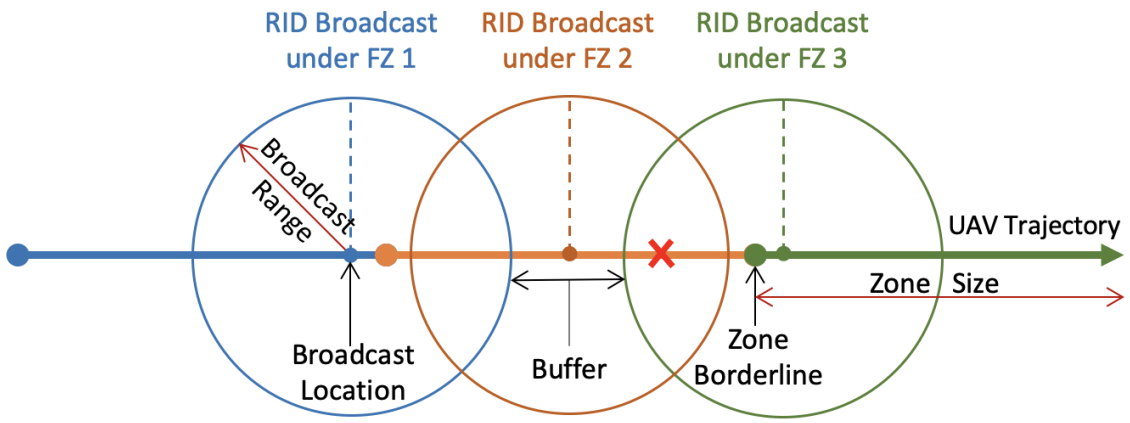


Figure 1.16: Three (color-codded) Flight Zones describe a worst-case scenario where a UAV performed a broadcast right before exiting FZ1 and right after entering FZ3. The buffer range prevented broadcasts from FZ1 and FZ3 to overlap. Authenticator at the Red X can only correlate a maximum of two zones (FZ2 and FZ3) to the same UAV.

Generally, having many FZs in a flight plan makes it harder for authenticators to successfully correlate the PKs to the same UAV by collecting all (or most of) the FZs from Remote-ID (RID) broadcasts. In contrast, if we have very few FZs such as two FZs for the entire flight path, an authenticator located near or right at the borderline separating the two FZs would be in the broadcast coverage of both FZs (Figure 1.16). Therefore, when the authenticator receives the RID broadcasts (which includes the FC) from the UAV, it can deduce that the RIDs received from FZ_i and FZ_{i+1} belong to the same UAV even if the PKs are not correlated. In another scenario, if we have three FZs for the entire flight path and two authenticators are located near the borderlines separating the FZs, each authenticator would possess part of the entire flight path by listening to RID broadcasts, and if the two authenticators collaborate with each other, they both can deduce that the collected RIDs belong to the same UAV. It is worth noting that even if an authenticator was able to reconstruct the entire flight path of a UAV, the authenticator cannot conclude for certain that the reconstructed path is the complete path of the UAV. This also implies that successfully correlating some of the FZs to the same UAV would partially compromise the UAV's anonymity and the more FZs are correlated the bigger the compromise would get.

Our main approach to tackle this issue is to first determine the UAV's broadcast coverage area. Each wireless protocol and hardware has a minimum acceptable signal strength. For example, typical WiFi adapters use omnidirectional antennas and require at least -80 dBm signal strength. Let's denote the minimum acceptable signal strength threshold as θ. Next we use the Link Budget equation $Receivedpower(dB) = transmittedpower(dB) + \sum gains(dB) - \sum losses(dB)$ to estimate θ, which can be written as: $P_{RX} = P_{TX} + G_{RX} + G_{TX} - L_{RX} - L_{TX} - L_{LM} - L_{FS}$ where P_{RX} is the authenticator's received power which is also the threshold θ, P_{TX} is transmitter output power, G_{RX} is receiver antenna gain, G_{TX} is transmitter antenna gain, L_{RX} is receiver losses, L_{TX} is the transmitter losses, and L_{FS} is the free space loss. Furthermore, L_{FS} is expressed as: $L_{FS} = 20\log_{10}(\frac{4\pi d}{\lambda})$ where λ is the wavelength and d is the distance in the same unit as the wavelength. Then, we use the Link Budget function to solve for d such that: $d = \frac{\lambda \sqrt[20]{10^{P_{TX}} + 10^{G_{RX}} + 10^{G_{TX}} - 10^{L_{RX}} - 10^{L_{TX}} - 10^{L_{LM}} - 10^{P_{RX}}}}{4\pi}$ Once we solve for d, we can use the solution to estimate the broadcast coverage radius r given the UAV's altitude alt such that: $r = \sqrt{d^2 - alt^2}$.

After finding r, we can now define the size of each FZ to be at least bigger than the coverage diameter $2r$. In real-life scenarios, it is inevitable that an authenticator might be located between two adjacent FZs where the authenticator can receive two RIDs from the same UAV that happened to pass over two adjacent FZs. Therefore, setting the size of each FZ to $2r$ ensures that no authenticator can receive more than two RIDs of adjacent FZs from the same UAV. Furthermore, to ensure that broadcasts from neighboring FZs do not overlap on a 3rd FZ and to defend against unexpected high G_{RX}, a buffer b is added to the FZ size such that $FZSize = 2r + b$ where $b \leq \frac{r}{2}$ (Figure 1.16).

After defining the size of each FZ, the entire flight path can now be divided into FZs prior to the flight mission where each two waypoints that have a distance longer than $2r + b$ is divided by FZ size and any series of waypoints that is shorter than $2r + b$ are combined together until it reaches the acceptable FZ size. For each FZ, a UAV's entry and exit time

to the FZ is estimated based on the takeoff time, the distance to the FZ, and the expected UAV's speed.

1.2.5 Authentication Framework Design

The authentication framework defines the process of constructing and exchanging the Remote-ID (RID) messages between different entities in the framework. To recap, an RID is a message that is continually broadcasted by a UAV to anonymously authenticate itself to any third party authenticators as well as to prove to the authenticator that the UAV has a permission from an authoritative entity to fly over its current flying location at the time of the broadcast.

There are five participants in the authentication framework (Figure 1.18), Unmanned Aerial Vehicle (UAV), Unmanned Aerial System (UAS), UAS Traffic Controller (UTC), UAS Service Supplier (USS), and an Authenticator. The UAV is the entity that is required to anonymously and continually authenticate itself by broadcasting/uploading its RID messages. The UAS is the operator of the UAVs (e.g. a UAV hobbyist or a commercial drone delivery company). A single UAS operator may have one or multiple UAVs simultaneously. UTC is the authoritative entity that regulates and oversees the UAV operations over an airspace (e.g. FAA). The UTC is the equivalent to Air Traffic Controller (ATC) that manages manned aircraft operations. Similar to the current regulations [39], all UAVs participating in the authentication framework are required to obtain a digitally signed flight authorization from a UTC prior to each flight mission. The USS is an entity that provides services to other entities in the framework to support various UTM operations (e.g. an online database that stores all UAVs digital certificates and RIDs). In our framework, an RID repository is required to support the anonymous authentication process via the internet (Figure 1.18). A public authenticator is any entity that receives UAV RIDs (either via direct broadcast from

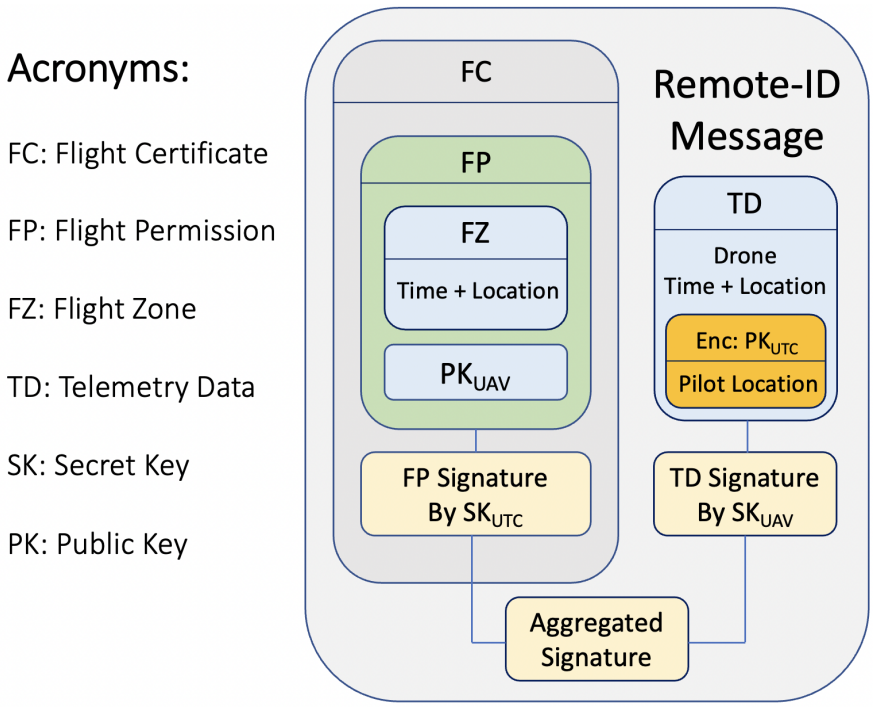


Figure 1.17: Remote-ID Message Structure.

the UAV or over the internet from an RID repository) to anonymously authenticate the UAV based on the RID credentials.

Framework Workflow

The framework process is executed in two different phases: a Setup Phase and an Authentication Phase (Figure 1.18). In the setup phase, a UAS operator submits a flight authorization request to the overseeing UTC before each flight mission and obtains an authorization approval that contains a list of Flight Certificates (FCs), where each FC is used to authenticate the UAV over a specific region within a specific time period. In the authentication phase, the UAV uses the appropriate FC to construct an RID message and continually disseminates the RIDs over two different wireless channels, a direct RID broadcast or uploading RIDs to an RID repository over the internet (Figure 1.14).

Every RID includes an FC which is a spatiotemporal flight permission, a flight permission bounded by a geographic location and a time window. Furthermore, each FC has its own

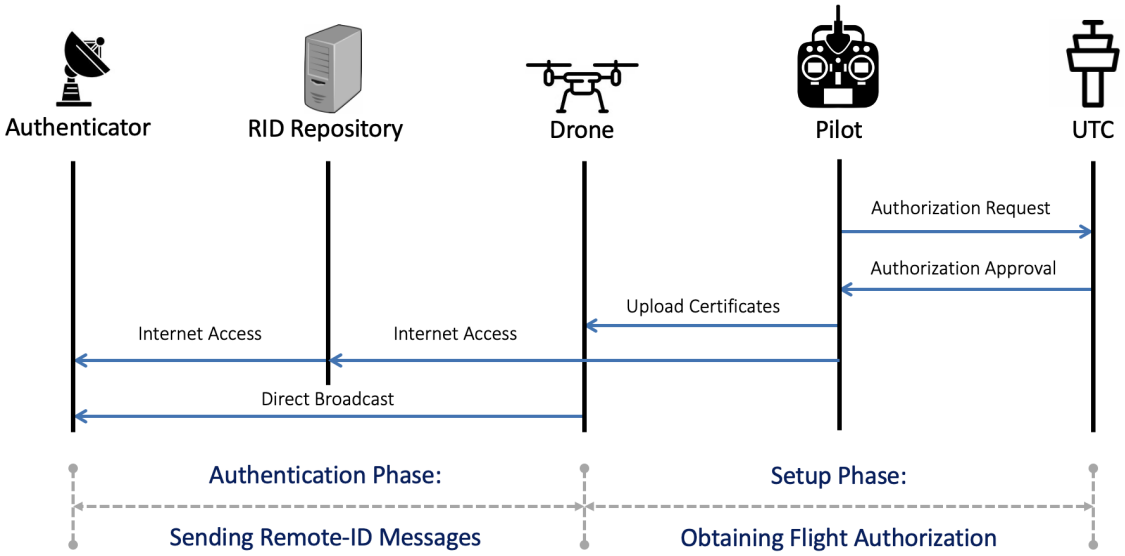


Figure 1.18: Framework Workflow.

PK which makes every FC acting as a pseudonymous digital certificate for the UAV that certifies its PK and its flight permissions for the current fly zone and time. Since every FC has a different PK, an authenticator cannot track a specific UAV to its destination or back to its takeoff site without physically following the UAV since the UAV would use a new PK for each flight zone and would appear as a new UAV whenever exiting or entering a new flight zone. Furthermore, since the list of Flight Certificates (FCs) form a series of contiguous flight zones that are bounded by a sequential arrival and departure times, a UAV cannot deviate from its designated flight path or fly at a different times other than the time that is approved by the UTC without risking a violation detection by authenticators.

A. Setup Phase:

Authorization Request: Prior to each flight, a UAS operator must obtain a flight authorization from the UTC which will be used to construct each RID message (Figure 1.17) during the UAV's flight time.

1. First, the UAS operator creates a flight plan for the UAV assigned to the mission and divides it into flight zones $FZ_i \in \{FZ_1, FZ_2, ..., FZ_n\}$ where $i \in \{1, 2, ..., n\}$ and each

FZ_i is the expected UAV trajectory over a specific region within a time period as discussed in section 1.2.4.

2. For each FZ_i, the operator generates a public/private key pair $(PK_i, SK_i) \in \{(PK_1, SK_1), (PK_2, SK_2), ..., (PK_n, SK_n)\}$ and appends each PK_i to a FZ_i as a pair to create a "flight permission" $FP_i = (PK_i, FZ_i)$.

3. The set "Flight Plan Vector" FPV is defined as $FPV = \{FP_1, FP_2, ..., FP_n\}$ where $FP_i = (PK_i, FZ_i)$.

4. Then the operator creates an authorization request Req that includes the following attributes: `UAS ID, UAV ID, FPV`, and `UAS public key` PK_{UAS}, and signs the request with the operator's private key SK_{UAS} such that $\sigma_{Req} := Sign(SK_{UAS}, Req)$.

5. Next, the UAS operator sends (Req, σ_{Req}) to the UTC over a secure channel such as TLS. The UTC then verifies the UAS request signature such that $Verify(PK_{UAS}, Req, \sigma_{Req}) = 1$.

Authorization Approval: Once the signature is verified and the flight request gets approved, the UTC creates an authorization approval as follows:

1. The UTC signs each $FP_i \in FPV$ using UTC's private key SK_{UTC} to create FP_i signature σ_{FP_i} such that $\sigma_{FP_i} := Sign(SK_{UTC}, FP_i)$. Each signed FP_i is a flight certificate $FC_i := (FP_i, \sigma_{FP_i})$ that is used by the UAV to authenticate itself over a flight zone within a specific period of time.

2. The UTC, then, populates a "Flight Certificates Vector" set FCV which is defined as $FCV = \{FC_1, FC_2, ..., FC_n\}$ where $FC_i := (FP_i, \sigma_{FP_i})$

3. The UTC sends the FCV back to the UAS as an authorization approval over a secure channel such as TLS.

4. Once the UAS receives the FCV from the UTC, the UAS groups all the secret keys SK_{UAV_i} generated at the Authorization Request step as a "Secret Key Vector" SKV where $SKV = \{SK_{UAV_1}, SK_{UAV_2}, ..., SK_{UAV_n}\}$ and uploads the FCV and SKV to the UAV assigned to the mission.

B. Authentication Phase:

During flight time, the UAV is required to authenticate itself by continually sending out RID messages. Each RID message can be communicated via two different channels: by direct broadcast and through the internet (Figure 1.14). Authenticators within the the UAV's broadcast range can receive and decode the RIDs. Authenticators with an active internet connection can retrieve the RIDs from an RID repository over the internet. When the UAV flies over a region, the authentication process takes place as follows:

Signing the RID:

1. The UAV selects the FC_i from FCV that has an FZ_i matching its current flight area at its current flight time.

2. Then the UAV records its current telemetry data TD that includes the current time and GPS location readings.

3. After that, the UAV creates and signs an RID message using the secret key SK_{UAV_i} matching the PK_i in the currently used FC_i such that $\sigma_{RID} := Sign(SK_{UAV_i}, RID)$ where $RID = (TD, FC_i)$.

4. To reduce the message size, the UAV uses a signature aggregation function to aggregate the UTC signature σ_{FP_i} with the UAV signature σ_{RID} to get a new σ_{RID} such that $\sigma_{RID} := \mathsf{Aggregate}(\sigma_{FP_i}, \sigma_{RID})$.

5. Finally, the UAV sends out the tuple (RID, σ_{RID}) via a direct broadcast and to the internet.

Verifying the RID:

When an authenticator receives an RID either via a direct broadcast or from the internet, it performs the following:

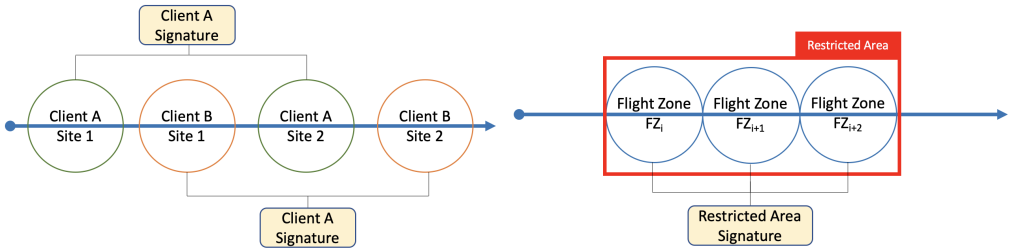
1. The authenticator takes the aggregate signature σ_{RID}, PK_{UAV_i}, and PK_{UTC} as inputs for an aggregate signature verification function such as $\mathsf{VerifyAggregate}(\{PK_{UAVi}, PK_{UTC}\}, \{FP_i, TD\}, \sigma_{RID})$.

2. If the aggregate signature verifies successfully, the authenticator cross-references the time and location of FZ_i in FC_i with its own current time and location. If the authenticator is within range of FZ_i and the time of receiving the RID is within FZ_i time window, the RID is accepted. Otherwise, appropriate actions are taken for a suspected spoofing attempt. Note that the authenticator did not cross-reference with TD because the UAV cannot be fully trusted since it can forge its own telemetry readings. The main purpose of TD is to provide non-repudiation and holds the UAV accountable when a forged TD is detected. Another purpose of the TD is when the actual time and location of the UAV is required such as UAV-to-UAV communications. A more thorough security analysis is discussed in the next section, section 1.2.6.

3. Finally, the authenticator retains a copy of the encrypted pilot location for future accountability.

Other Applications:

The main application that motivated the proposed framework is the base use case for the Remote ID framework operating under the previously described threat model. This base use



(a) First application. (b) Second Application.

Figure 1.19: Other use cases of the authentication framework.

case (which the rest of the module is be built upon) is for UAVs to authenticate themselves to the open public while keeping their identity secured by using one flight certificate with each RID message. However, this framework can be extended to other use cases as described below.

A. The second use case (Figure 1.19a) considers a scenario where a UAV performs services for multiple clients and each client has multiple sites that the UAV needs to visit. Having multiple certificates allows the UAV to choose the permitted flight zones including all sites of a single client and revealing it to that client only, while keeping the rest of the flight path hidden.

B. In the third use case (Figure 1.19b), a UAV might encounter a heavily restricted area which has a UAV detection system at its perimeter. In this case, the UAV needs to reveal the certificates of the entire route that passes through the area to the perimeter keeper while keeping the rest of the flight path hidden.

1.2.6 Security Analysis

In this section, we analyze the security of the proposed framework and the RID structure with respect to different attack scenarios. Note that all of these attacks stem from the original threats that comprise the threat model.

Impersonation Attack: Suppose a rogue UAV attempts to enter a restricted area but does not have permissions from the UAS Traffic Controller (UTC) to do so. The rogue UAV might attempt to forge its own flight permissions and use it to construct its own RID messages, but since the UAV does not have Flight Certificates (FCs) legitimately signed by the UTC, the RIDs will not be validated by the authenticators.

Replay Attack: A rogue UAV might attempt to intercept a freshly broadcasted RID that are constructed by a legitimate UAV and re-broadcast (replay) it at a different time and/or at a different location to gain access to a restricted area. Each RID includes a FC that is bounded by a specific time and a specific location. Therefore when an authenticator receives an RID with a FC that does not match its current time and location, the RID will be rejected.

Misbehaving UAVs: Although only registered and certified Unmanned Aerial System (UAS) operators are allowed to submit flight requests by using their own certificates to obtain flight approvals for their UAVs, there is still the possibility of having misbehaving UAVs in the network that are hijacked, malfunctioned, or maliciously altered by their UAS operators to deviate from its designated paths. We remark that in the proposed scheme, each UAV appends its Telemetry Data (TD) to every RID message which includes current time and location of the RID broadcast. Therefore, if a misbehaving UAV attempts to change the course of its designated flight path, its telemetry data will not match the spatiotemporal bounds of the accompanying FC. Furthermore, if the UAV attempts to forge its own TD to falsely match the bounds of FC, it will not match any authenticator's current time and location. It is worth noting that the main purpose for including TD is to follow the FAA's guidelines for safe navigation when interacting with other participants in the UAS Traffic Management (UTM) system.

Leaked Flight Certificate Vector (FCV): FCV is a list of Flight Certificates (FCs) that includes the signed path for the UAV to follow. If the FCV (or part of it) lost its confidentiality before the start time of the leaked FCs, any forged RIDs by rogue UAVs using the leaked

Table 1.5: BGLS and ECDSA Comparison.

Signature Scheme	Key Generation (milliseconds)	Message Signing (milliseconds)	Public Key Size (bits)	Signature Size (bits)
BGLS	0.25	4.45	768	384
ECDSA	0.714	0.76	512	512

FCs will not be accepted by authenticators since for each RID to be accepted, it must be signed by the owner of the Secret Key SK_{UAV_i} which belongs to the public key PK_{UAV_i} that is embedded inside each FC_i for each FC period i. The list of all secret keys $\{SK_1, ..., SK_n\}$ are generated by the controlling UAS and are never sent over the network.

Anonymity and Untraceability: Each FC_i in FCV is individually signed by the UTC and includes a unique PK_{UAV_i}. Each PK_{UAV_i} is randomly generated and is not tied to any identifying information which implies that authenticators cannot deduce the owner of the UAV from its RIDs. The mapping to each PK_{UAV_i} with the actual identity of the UAV is traced via the public key of the UAV operator PK_{UAS} which is only accessible to the UTC since it is the entity that signed all PK_{UAV_i} at the setup phase in section 1.2.5. Furthermore, authenticators cannot pinpoint from where the UAV is coming from or where the UAV is going to since the FC_i in the RID is only exposing a limited range of where the UAV is allowed to operate in.

1.2.7 Performance Evaluation

In this section, we provide a performance evaluation of the signature aggregation scheme using Mariano Sorgente's implementation of Boneh–Gentry–Lynn–Shacham (BGLS) signature scheme [74] as well as a comparative analysis with Brian Warner's implementation of Elliptic Curve Digital Signature Algorithm (ECDSA) signature scheme [84]. Since our framework focuses on the performance on the UAV side, we evaluate the message sizes of both BGLS and ECDSA as well as the processing time of key generation and the message signing operation

of both signature schemes. We also evaluate the processing time of signature aggregation using BGLS.

Message Sizes:

Table 1.5 shows the size of the signature and the public key of both schemes. At first, it might appear that the total size of ECDSA (Signature + Key = 1024 bit) is less than BGLS (Signature + Key = 1152 bit), but each RID message requires at least two signatures and up to the entire FC_i signatures. Therefore, the minimum size of an ECDSA is 1536 bits with additional 512 bits for each additional signature, while BGLS will always have a fixed cryptographic size which is 1152 bit regardless of the number of signatures. In energy-constrained devices such as UAVs, it is critical to reduce the size of messages to be communicated since the energy consumption of sending a single byte over the network is higher than the energy consumption of processing a single byte locally [22].

Computational Costs of Key Generation & Message Signing: We have generated 10,000 key pairs and signatures for each scheme. It took BGLS 2.5 seconds to generate 10,000 public and private keys which means 0.25 ms for each key generation operation (Table 1.5). On the other hand, it took ECDSA 7.14 seconds to generate the same number of keys, that is 0.714 ms per key pair. This shows that BGLS provides an efficient key generation algorithm and having a key pair for each Flight Certificate is not a computational bottleneck.

Regarding message signing, BGLS took 44.5 seconds to sign 10,000 distinct messages (4.45 ms per signature) while it took ECDSA 7.6 seconds to sign the same set of messages (0.76 ms per signature). Even though ECDSA's signing operation is more efficient than BGLS's, our framework requires a UAV to perform a single signing operation with every broadcast, and 4.45 ms per broadcast is not a prohibitive performance since the broadcasting frequency in realistic scenarios is expected to be reasonably low (e.g. five broadcasts per second).

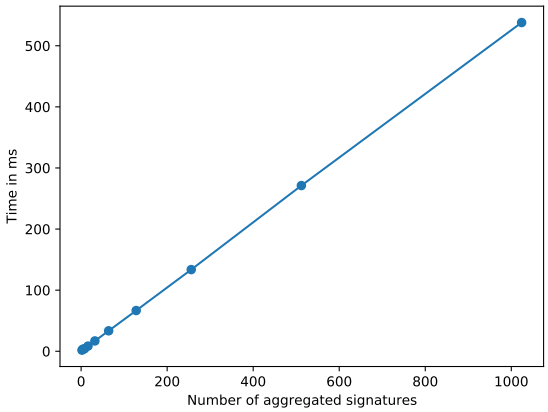


Figure 1.20: Signature Aggregation.

Computational Costs of Signature Aggregation:

Figure 1.20 shows the efficiency of signature aggregation. Aggregating the minimum number of signatures required by the framework (i.e. two signatures) takes 1.8 ms. Overall, the figure shows a linear relationship between the number of aggregated signatures and the computational time. In our framework, most aggregations that require more than two signatures can be computed offline since the UAV's flight plan will most likely include the special areas where the UAV is expected to reveal more than one Flight Certificate (FC) at the same time.

1.2.8 Drone Authentication Conclusion

In this module, we proposed a Privacy-Preserving Authentication Framework for UAS Traffic Management Systems (UTM) based on the FAA's Remote-ID module. This framework anonymously authenticates flying UAVs and at the same time verifies their flight permission without exposing their entire flight route. To achieve authentication anonymity, a flight plan slicing technique is introduced where the flight plan of a UAV is divided into blocks of flight zones which are bounded by a time and a location while each zone has its own public key. A unique structure of a Remote-ID message is also proposed where a UAV carries multiple

signatures and uses signature aggregation algorithm to send all required signatures as one signatures. A security analysis and a performance evaluation is also introduced.

1.3 Third Module: Localized Authentication

Autonomous vehicles such as self-operated drones and cars will soon enable new applications in different domains, such as autonomous transportation and drone delivery. However, safety-critical operations, such as object-detection and avoidance, are integral components of these applications and it would make them appealing targets for cybercriminals [21]. In the United States, various efforts from government agencies have been directed to set up the rules, regulations, and policies for managing the secure operations of future autonomous systems (i.e., vehicles and Roadside Units, or RSUs). For connected and self-driving cars, the U.S. Department of Transportation (USDOT) has adopted the Security Credential Management System (SCMS) [81] for handling secure vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communications. For Unmanned Aerial Vehicles (UAVs), the Federal Aviation Administration (FAA) and National Aeronautics and Space Administration (NASA) have jointly developed the Remote ID framework [41] which would set the foundation for the anticipated Unmanned Aircraft System (UAS) Traffic Management System (UTM) [40]. Unfortunately, these systems still lack effective security measures to defend against existing attacks [43][10].

In the academic community, several attacks have been demonstrated against autonomous systems such as GPS spoofing attacks [48], V2I attacks [47], and Sybil (vehicle ID duplication) attacks [45]. These attacks have different threat models and attack methodologies, and find different mitigation approaches. However, we contend that all these attacks stem from the same root: *the decoupling between the sender of a message and its transmitting location.* In other words, if receiver of a message cannot locate its transmitter, multiple threats are exposed.

In this module, we propose a Vision-Based Two-Factor Authentication & Localization Scheme for Autonomous Vehicles to pinpoint and authenticate the location of a message transmitter. In this scheme, we leverage the lighting source of the vehicle to create an Optical Camera

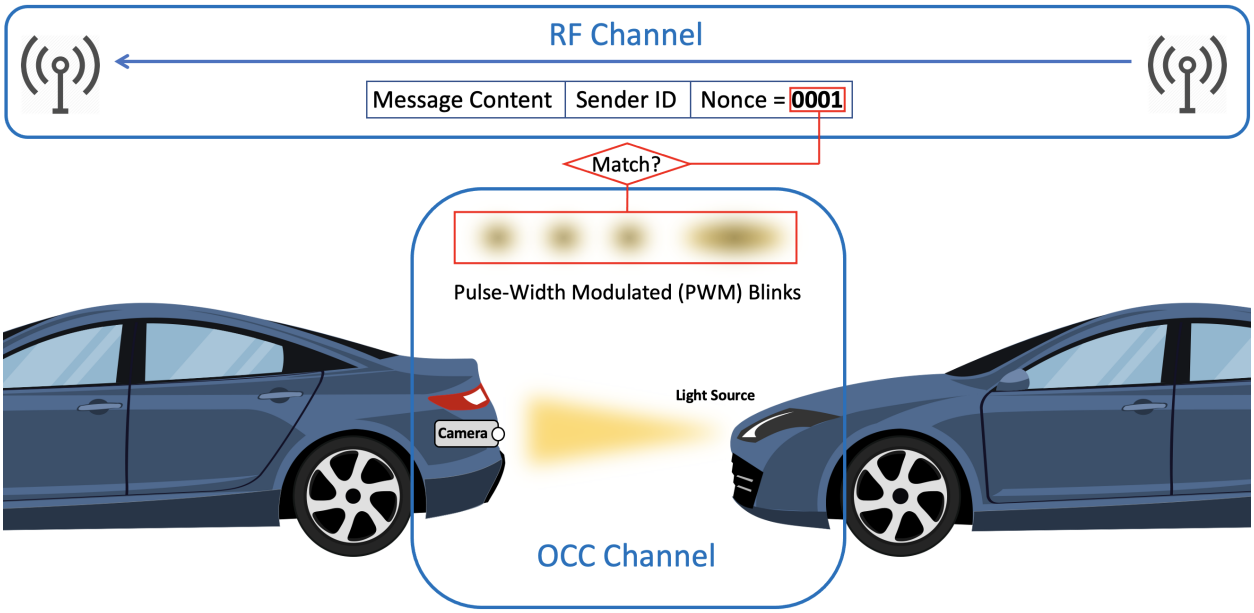


Figure 1.21: Scheme Overview: a vehicle sends an RF message and a visual nonce as optical Pulse-Width Modulated (PWM) symbols using its headlights

Communication (OCC) channel to send a random *nonce* – a randomly generated number that to be used only once – encoded as flashing blinks from the light source, while at the same time the sending vehicle includes the same nonce into the message to be sent over the Radio Frequency (RF) channel (figure 1.21). In this scheme, the receiver is equipped with an RF interface and a camera, so that it can receive the message via the RF channel while simultaneously use its camera to record the sender's blinks and decode the visually modulated nonce using computer-vision algorithms. The receiver then matches the nonces received on the two channels (RF and OCC). If the nonces match, then the vehicle is visually authenticated and localized and is indeed identified as the transmitter of the message. Note that our scheme can be used to authenticate each RF message as well as it can be used as a bootstrap to authenticate a vehicle's location once then establish a secure channel with the localized vehicle.

We summarize our contributions as follows:

- We propose a vision-based authentication scheme that pinpoints (authenticates and localizes) the actual transmitter of a message sent over the RF channel by utilizing

cameras and light sources to create an OCC channel.

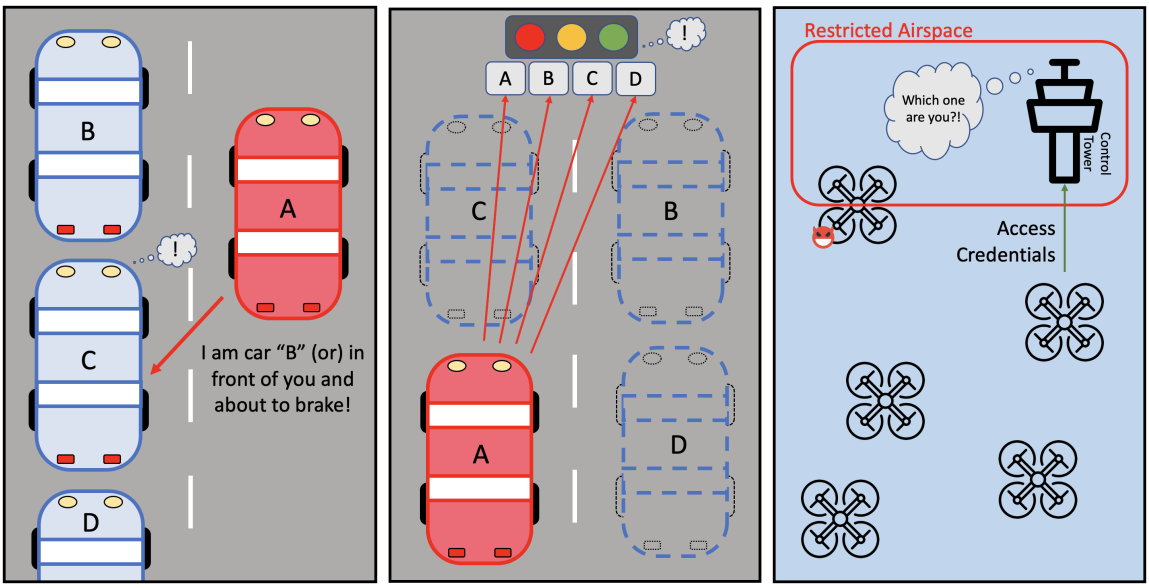
- We identify a possible attack (named a Copycat Attack) with different variants against the proposed scheme, where a malicious party mimics another legitimate transmitter's visual blinks to spoof its identity or location.
- We present mitigation approaches that circumvent all variants of the copycat attack.

In the next section, we present our threat model with three different attack scenarios. In section 1.3.2, we give a brief background about wireless communications that utilize optical techniques. In section 1.3.3, we provide an overview of our proposed scheme. In section 1.3.4, we evaluate the attack scenarios discussed in section 1.3.1 against our scheme presented in section 1.3.3. Section 1.3.5 demonstrates a new adaptive attack (named Copycat Attack) under multiple scenarios as well as mitigation approaches for each scenario. Related work is discussed in section 1.3.6 and section 1.3.7 concludes the module.

1.3.1 Threat Model

In our threat model (figure 1.22), a malicious party exploits the decoupling of a sent message from the location of its transmitter. Therefore, our main objective in this threat model is to identify and authenticate the physical location of a message's transmitter, whether the transmitter is an attacker or a legitimate vehicle. To illustrate further, three different attack scenarios are demonstrated as examples:

Location Spoofing: In this scenario, a compromised or a malicious vehicle fabricates fake GPS coordinates about its current location to cause accidents, alter a system's behavior, or earn unlawful privileges. For example in figure 1.22a, a platoon of vehicles (cars $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$) is already formed and a malicious adversarial vehicle $\mathcal{A}$ sends an emergency-braking message to vehicle $\mathcal{C}$ while pretending to be in the location of vehicle $\mathcal{B}$. Or if vehicle $\mathcal{A}$ is



(a) C cannot tell if the message came from A or B

(b) RSU cannot tell who actually sent the four messages

(c) Control tower cannot tell which is the legitimate UAV

Figure 1.22: Aerial view of three different attack scenarios

capable of compromising the identity of vehicle $\mathcal{B}$, $\mathcal{A}$ can pretend to be $\mathcal{B}$ itself without the need of fabricating any coordinates. Vehicle $\mathcal{C}$ would react to $\mathcal{A}$'s fabricated message and apply its brakes to avoid crashing into $\mathcal{B}$ but also it would cause $\mathcal{D}$ to crash into $\mathcal{C}$ if there are no implemented precautions against this scenario. In this scenario, the malicious vehicle does not need to wait for an opening to physically drive up in front of the targeted vehicle. Instead, the attack can be launched from faraway.

Identity Duplication: Identity Duplication attack, also known as Sybil Attack, is an attack carried out when a single malicious vehicle sends out multiple messages with multiple identities to give an impression of a congested road and alter the behavior of the infrastructure and/or other vehicles. For example, in figure 1.22b, the additional fake vehicles (ghost cars $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$) generated by $\mathcal{A}$ would cause the traffic light to turn green sooner in the attacker's lane and longer for other lanes in order to clear the congested lane.

Identity Confusion: In this scenario (figure 1.22c), a swarm of unmanned aerial vehicles (UAVs) are flying in close-proximity of each other. A legitimate UAV is supplementing its

credentials to the control tower for accessing the nearby restricted airspace. During that time, another intruding UAV enters the restricted airspace. Here the control tower cannot enforce the given access (e.g., take down the intruding UAV) since it cannot physically differentiate between the legitimate and the intruding UAV.

We can observe that each scenario represents different attacking capabilities. In the first scenario, the attacker has the ability to either *fabricate the content of its messages or use the identity of another legitimate vehicle*. In the second scenario, the attacker has the ability to *generate multiple identities of non-existing vehicles*. In the third scenario, the attacker is able to *passively listen to messages* in order to make an opportunistic attack (e.g. invasive access towards a restricted airspace). However, all three scenarios can be exploited due to the same reason; *the inability of the receiver to localize the transmitter of the messages*.

1.3.2 Background on Optical Wireless Communication

1.3.2.1 Overview

Optical Wireless Communication (OWC) is any communication channel that utilizes the terahertz band of the electromagnetic spectrum which includes the infrared and ultraviolet frequencies. Furthermore, an OWC channel utilizing the visible-light portion of the terahertz band is referred to as Visible Light Communication (VLC).

Optical communications require an imaging sensor for receiving the incoming light photons. That sensor is commonly known as a photodetector. A camera sensor such as Complementary Metal Oxide Semiconductor (CMOS) consists of a matrix of photodetectors, each represents a pixel which gets its color by measuring the intensity and frequency of the recorded photon by its corresponding photodetector. The main objective of a camera sensor is to create an image out of the CMOS output as a grid of colored pixels, and the rate of images

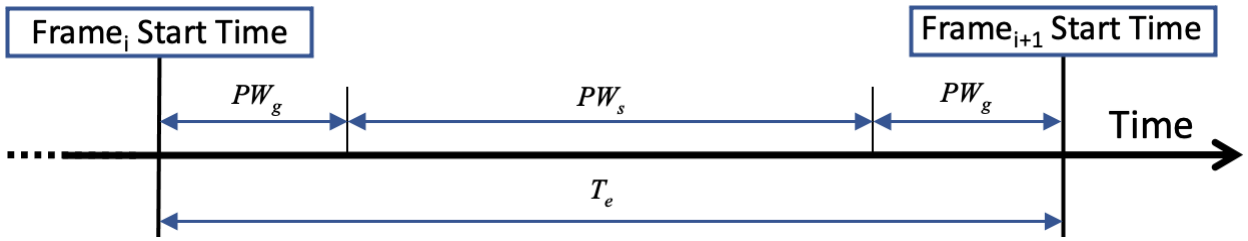


Figure 1.23: OCC Channel Model

created by the sensor is known as frame per second (FPS). However, since cameras have become a commodity hardware, their sensors have been repurposed for various applications such as decoding messages that are digitally modulated into images. This type of optical communication technique is commonly known as Optical Camera Communication (OCC).

1.3.2.2 OCC Channel Model

Our channel model (Figure 1.23) is based on two assumptions: **(i)** The modulation scheme used over the OCC channel is assumed to be optical On-Off Keying (OOK) since it only requires a single narrowband frequency (i.e., single color) which makes it applicable to any light sources. Note that visual symbols in figure 1.21 and 1.26 are pulse-width modulated (PWM) for ease of illustration only. **(ii)** We assumes the use of cameras that are operated using the global shutter mode where all camera pixels are scanned simultaneously. The rolling shutter mode (its counterpart) scans rows of pixel independently one after another. However, the security intuitions in this module still holds true for both assumptions and should be applicable to different imaging techniques.

In OCC systems, a communication channel using OOK modulation can be modeled as a rectangular waveform. Let T_e denote to the camera exposure time which defines how long a camera shutter stays open to capture a single frame. Therefore, T_e is the inverse of the camera sampling rate FPS such that $T_e = \frac{1}{FPS}$. Let PW_s denote the symbol pulse-width which defines how long a light source stays on during the T_e time frame. Intuitively, we have

$PW_s < T_e$. However, there is a minimum duty cycle $DC_{min} = \frac{PW_s}{T_e}$ for a given OCC system under certain conditions and SINR requirements. Furthermore, a guard width PW_g must be left unoccupied at the beginning and the end of the T_e time frame to prevent symbols from leaking into adjacent frames (similar to the guard bands that are added between RF channels to prevent inter-symbol interference). Finally, in our scheme, the OCC can be modeled as $PW_s + PW_g < T_e$ for $PW_s \geq DC_{min} \times T_e$. Note that the propagation delay is omitted from the channel model since it has a negligible effect especially for close-range communications such as in V2V and V2I. Furthermore, the guard band PW_g should mitigate the effects of propagation delay whenever it becomes critically high.

1.3.2.3 Challenges

There are several challenges when working with OCC systems such as: **(i)** unidirectional link - a lighting source can only be used for transmitting while a camera is only for receiving, **(ii)** highly directional communications - the camera and the lighting source need to face each other in a clear line-of-sight (LOS), and **(iii)** low bitrate - common camera sensors can sample up to 30 FPS (stand-alone photodetectors are capable of higher sampling rates with more complex modulation schemes, but cannot construct an image out of the received photons).

The requirements of our scheme are not affected by these shortcomings. For instance, the receiver in our scheme does not require a feedback channel for locating the sender, and whenever a feedback is required (e.g. loss in clock synchronization) the RF channel can be used. Also based on the threat model discussed in Section 1.3.1, the sender is expected to be in a clear view of the receiver which means that clear LOS is an inherent requirement for all the three attack scenarios explained in Section 1.3.1. Concerning the bitrate, the sender encodes only a short nonce into the OCC channel whereas the main message is sent over the RF channel. Therefore, a low OCC bitrate is sufficient to execute the authentication

scheme. Furthermore, lower bitrates can be modulated using longer symbol periods which inherently make it cover larger distances in the optical domain.

1.3.3 Scheme Overview

As discussed in the introduction of the localized authentication scheme, our scheme utilizes two different communication channels, an RF channel and an OCC channel. Both the sender and the receiver are equipped with an RF interface to implement an RF channel while the OCC channel requires the sender to be equipped with lighting source such as a light-emitting diode (LED), and the receiver to be equipped with a camera. We assume that these equipment are already available in most autonomous vehicles since we expect modern autonomous systems to have the minimum set of requirements to carry out safe and secure autonomous operations, which include: an RF interface (for wireless communications), a camera (for object detection and navigation), and a lighting source (for safety, illumination, and identification purposes) such as car's headlights/taillights and drone's anti-collision strobe lights.

Authentication & Localization Process: When a sender transmits a message over the RF channel, it includes a nonce in the message, while at the same time it encodes the same nonce as modulated blinks into the OCC channel using its light source. On the other hand, the receiver will continuously scan every captured camera frame for possible OOK modulated symbols. Here the receiver employs computer-vision algorithms and look at every two consecutive frames for a sudden change in pixel intensity. When a pixel intensity change is found, the algorithm locks on the object emitting the light source that caused the intensity change and extract the symbols encoded into each subsequent frame. The area of the locked object is called Region of Interest (ROI). Now whenever the receiver receives a message over the RF channel, it cross-references its nonce with the demodulated nonces emitted by the current ROI over the OCC channel at the time of message reception. To

illustrate, to transmit bit_i = "1" as an OOK modulated symbol, the sender sets its light source on high for PW_s seconds during the camera exposure time frame T_{e_i}. To transmit the next bit_{i+1} = "0", the sender sets its light source to off-state during the next transmission window $T_{e_{i+1}}$. The receiver would detect a transition period from T_{e_i} to $T_{e_{i+1}}$ time frames as a change in pixel intensity caused by bit_i and bit_{i+1} respectively. As a result, the receiver will lock on the ROI region of the captured picture that has a pixel intensity change. This process will be triggered from the very first transmitted bit_1 where a timer T_{OCC} will be set starting from the beginning of T_{e_1}. When the receiver receives a message from the RF channel that has the same nonce which is received from the OCC channel, the time difference between the two messages must satisfy the following condition: $|T_{OCC} - T_{RF}| < \theta$ where T_{RF} denote to the time where the first bit of the message was received over the RF channel and θ denote to a minimum threshold that can be used as an attack window. More details on the aforementioned attack will be discussed in Section 1.3.5.1.

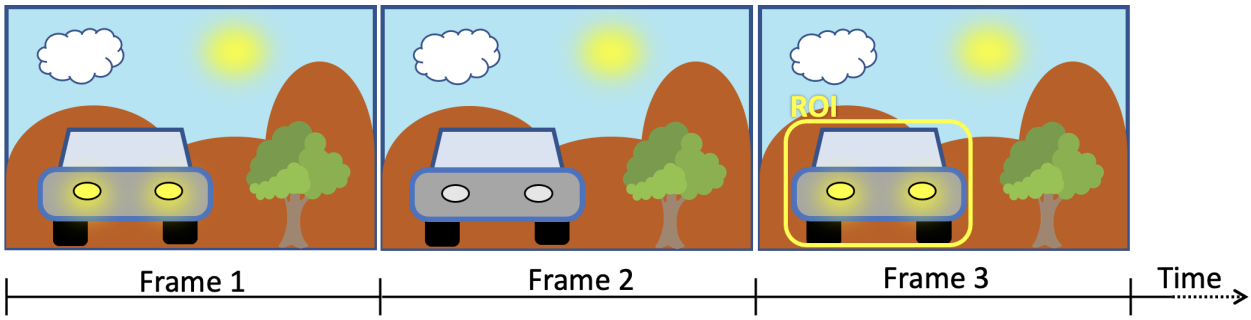


Figure 1.24: Synchronizing OCC Channel with Region of Interest (ROI)

Technical Considerations: To prevent accidental ROI locks that are caused by pixel intensity change due to random environmental factors (such as a vehicle turning on its headlights or an object moving in front of a light source which would be interpreted as an OOK modulated symbol), OCC systems would use a preamble with an alternating 0s and 1s to synchronize the sender with the receiver before locking on the ROI and recording the nonce (Figure 1.24). Another point to consider is that during an OCC transmission, the nonce encoded by the sender might include long runs of 0s or 1s (continuous repetitions of 0s or

1s) which would cause multiple consecutive frames to not have any intensity change across their transitions which in turn cause a loss in clock synchronization on receiver side (cannot distinguish whether the transmission has ended or a long run is being transmitted). Therefore, the sender would use Run Length Limited (RLL) codes such as Manchester encoding where a "1" is represented as "01" and "0" is represented as "10". In this case, the receiver would have at most two consecutive frames with the same modulated symbol (maximum possible run is two frames). Finally, the change in OCC channel state (i.e. blinking) should be *faster than the perception of the human eye* to prevent causing confusions to surrounding human drivers/pedestrians and also physiological effects such as nausea. The IEEE 802.15.7 standard [1] for Short-Range Optical Wireless Communications recommends the use of at least 200 Hz in light flickering frequency.

1.3.4 Scheme Defense Analysis

In this section, we evaluate our scheme (section 1.3.3) against the threat model (section 1.3.1).

Scenario A: in figure 1.22a, the malicious vehicle $\mathcal{A}$ attempts to deceive vehicle $\mathcal{C}$ by pretending to be vehicle $\mathcal{B}$ or by fabricating its location to appear as if $\mathcal{A}$ is in front of $\mathcal{C}$. With our scheme, whenever $\mathcal{C}$ receives a message pretending to be from $\mathcal{B}$, it checks the message's nonce with the visual nonce emitted by $\mathcal{B}$ which is physically in front of $\mathcal{C}$. If the two nonces from the two channels (RF and OCC) match, the message is indeed from $\mathcal{B}$. Otherwise, the message will be flagged as a spoofing attempt.

Scenario B: in figure 1.22b, the malicious vehicle attempts to deceive the traffic light into believing that the lane is crowded by broadcasting multiple messages with different IDs. Here the traffic light will use the OCC channel to match each message with its sender. Since the attacker is the sender of all the messages, all message IDs will be mapped to the same

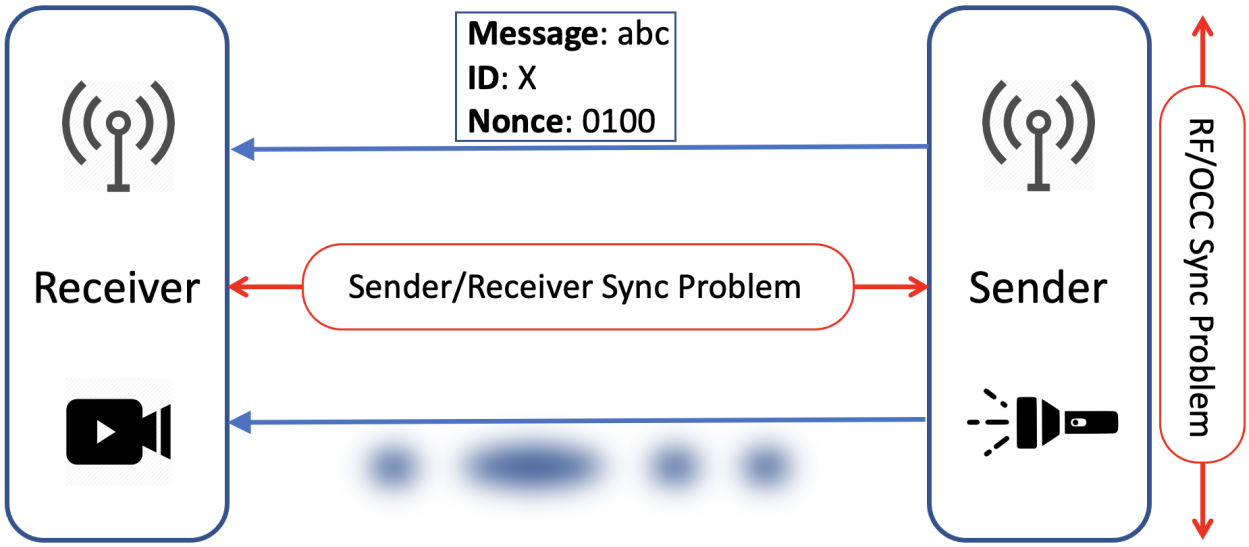


Figure 1.25: Sender/Receiver & RF/OCC Channel Synchronization Problem

vehicle which will trigger a spoofing attempt.

Scenario C: in figure 1.22c, a malicious unmanned aerial vehicle (UAV) attempts to access a restricted airspace by opportunistically waiting for a legitimate UAV to present its access credentials to the control tower. When the credentials are broadcasted, the malicious UAV flies to the restricted airspace knowing that the control tower cannot physically distinguish between the legitimate UAV from the malicious one. With our scheme, the legitimate UAV can physically present itself using OCC channel to distinguish itself from all other nearby UAVs.

1.3.5 Copycat Attack & Mitigation Methods

The threat model presented in section 1.3.1 represents three different attack scenarios that an attacker might attempt under different capabilities. However, we identified a possible attack against our authentication and localization scheme where the attacker adapts to our authentication strategy by recording visual nonces using its own camera then plays them back to mimic the legitimate sender (hence, the name Copycat), or fabricate new messages

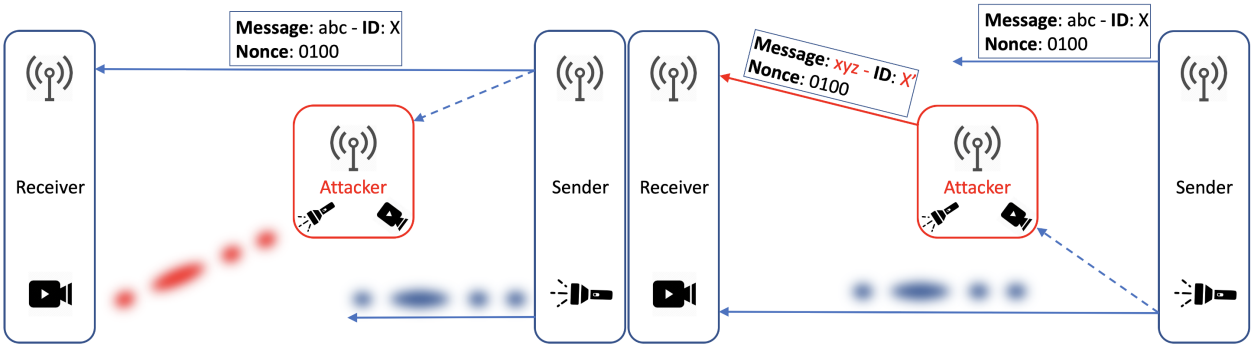
with someone else's OCC nonce (Figure 1.25).

Copycat attack can be executed under different scenarios. To mitigate its impact, we present the scenarios where this vulnerability might exist and discuss how to overcome it.

1.3.5.1 RF/OCC Channel Synchronization

In this scenario, the attacker attempts to exploit the time difference between the transmission time over the RF channel and the transmission time over the OCC channel in order to inject a spoofed message/nonce. To illustrate, let us consider the scenario in figure 1.26a. The legitimate vehicle transmits a message over the RF channel then shortly later transmits the nonce over the OCC channel. Assuming the attacker is capable of reading the RF message, the attacker extracts the nonce from the message then reproduces it over the OCC channel using its own lighting source. Now the receiver will believe that the RF message was actually sent by the attacker since the receiver correctly received the attacker's OCC nonce first.

Similarly, in the scenario demonstrated in figure 1.26b, the legitimate vehicle first transmits the nonce over the OCC channel then afterwards it transmits the message over the RF channel. By assuming the attacker is capable of fabricating the legitimate vehicle's identity,



(a) Receiver thinks the attacker is the sender of the RF message

(b) Receiver thinks the sender has sent the fabricated RF message

Figure 1.26: RF/OCC Channel Synchronization Problem

the attacker can read the legitimate vehicle's visual nonce and then fabricate an RF message using that nonce. Now any receiver would believe the attacker's message was actually transmitted by the legitimate vehicle since the two nonces match (attacker's RF nonce and legitimate vehicle's OCC nonce).

To understand how to mitigate this attack, we need first to define the notion of "attacker response time". Let $AttRes$ denote to the attacker response time which defines the minimum time the attacker needs to react to a message transmission. In other words, $AttRes$ is the elapsed time from the moment the attacker detects a transmitted message to the time the attacker reacts to that message by sending a spoofed message on the opposite channel. By knowing or estimating the $AttRes$ time, we can formulate a minimum acceptable RF/OCC transmission time difference such that: $|T_{OCC} - T_{RF}| < AttRes$. In other words, when one of the channels (either RF or OCC) starts transmitting, the other channel should start transmitting no later than $AttRes$ seconds. Note that since we omitted the propagation delay in section 1.3.2.2, we re-used T_{RF} and T_{OCC} from section 1.3.3 to denote to transmitting time instead of receiving time.

1.3.5.2 Sender/Receiver Channel Synchronization

In the previous scenario, the attacker either listen to the RF channel then transmit over the OCC channel, or listen to the OCC channel then transmit over the RF channel. In this scenario, the attacker listen to the OCC channel then transmit over the OCC channel as well to mimic the legitimate vehicle's visual nonce and cause confusion at the receiver side. In other words, the receiver would receive a single RF message from the legitimate vehicle but at the same time it will receive the same visual nonce from two different vehicles. Therefore, the receiver cannot distinguish who is the actual transmitter since there are two different vehicles emitting the same visual nonce.

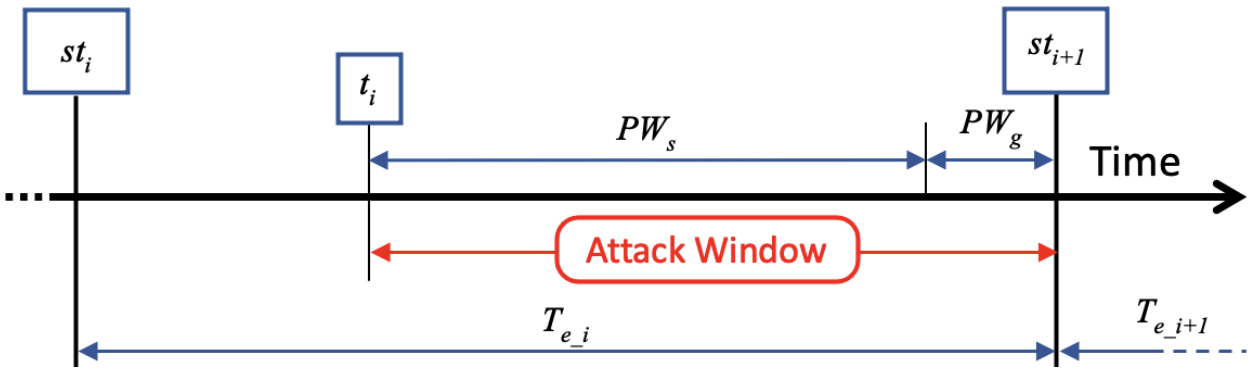


Figure 1.27: Sender/Receiver Channel Synchronization Problem

To mitigate the impact of this attack scenario, the camera frame captured by the receiver must only include the OCC symbol produced by the sender (Figure 1.27). To achieve this property, the sender needs to delay its OCC transmission to the end of the receiver's frame exposure time window T_e. This would cause the attacker's symbol to appear in the subsequent/different frame at the receiver side. To accomplish this, the sender needs to know the sampling rate of the receiver's camera (i.e. its FPS) as well as the receiver's clock time at which the camera starts and ends its shutter exposure period for each frame. Let st_i and st_{i+1} denote to the the start time of $frame_i$ and $frame_{i+1}$ respectively. If the sender knows st_i, then it can synchronize its clock with the receiver for all subsequent frame start times such that $st_{i+1} = st_i + T_e$. Now for the sender to transmit an OCC $symbol_i$ to be captured during $frame_i$, the time to transmit $symbol_i$ is at time $T_i = st_{i+1} - (PW_s + PW_g)$. This would cause $symbol_i$ to be transmitted as close as possible to the end of $frame_i$ but not too close to risk an inter-symbol interference. Furthermore, in section 1.3.2.2 we discussed that the symbol pulse-width must satisfy $PW_s \geq DC_{min} \times T_e$. In this section after defining the Sender/Receiver Channel Synchronization problem, we set $PW_s = DC_{min} \times T_e$ to minimize the attack window as much as possible. Therefore, for the OCC symbol transmission to be safely and successfully captured, the transmission time for $symbol_i$ is $T_i = st_{i+1} - (PW_s + PW_g)$ and the total pulse-width is $AttRes > (PW_s + PW_g) < T_e$ for $PW_s = DC_{min} \times T_e$.

1.3.6 Related Work

Most previous efforts related to vehicle authentication are directed toward authenticating their identities (e.g. through the use of digital certificates) [81]. However, such conventional authentication cannot be directly applied to many threat models as we have seen in section 1.3.1 where authenticating the location (i.e. "where you are") factor is also required. Mainly there are two localization techniques in the literature, RF-based and optical-based. Note that since we are interested in locating transmitters, techniques for localizing passive devices/objects such as computer-vision object detection and radar will not be discussed.

RF-Based Localization has abundant number publications. It mainly relies on Time/Difference of Arrival (ToA/TDoA), Angle of Arrival (AoA) and RSSI measurements techniques. For example [45] proposed an RSSI-based localization algorithm that depends on vehicles or RSUs as observers to triangulate the target vehicle. However, the dependence on other trusted vehicles sets a limitation to the threat model. Furthermore, the location of a target vehicle is approximated as an area (a common limitation for most RF-based localization schemes). That means a group of vehicles in close-proximity with each other (a typical road scenario) would have very similar RF transmissions in terms of timing and angle which makes it a challenging task to localize one of the vehicles among the other group members. Even a single attacker can manipulate its transmission power and timing to deviate the observers. In general, RF-based localization schemes tend to be complex, require special antenna designs, and suffer from multipath and interference phenomena [58].

Vision-Based Localization is a well studied research field but mostly in the context of using visual cues as beacons for enabling devices to localize themselves. For example, [58] uses projectors and photodetectors while [27] uses LEDs and cameras as indoor localization solutions. However, such solutions cannot be repurposed to our threat model since it either require the cooperation of the localized device or require a map reconstruction phase (also

found in RF-based techniques) prior to rolling out the localization system. In vehicle network, the use of VLC channels is gaining a great attention lately but mainly in the context of communication rather than localization which might make it inapplicable to our threat model. For example to increase the bitrate of the VLC channel, [62] sampled each row of a rolling shutter independently while [65] suggested the use of a standalone photodiode but both techniques make it impossible (or at least extremely difficult) to construct an image from the received VLC signal. [52] and [87] attempted to increase the VLC bitrate with a fully constructed image by utilizing visual-MIMO (Multiple Input Multiple Output) where multiple light sources act as multiple output and the individual pixels of the receiver's camera are considered the multiple inputs. However, visual-MIMO via OCC requires additional hardware and its bit error rate drastically increases with increased distances [55]. In an alternative approach, [68] implemented a visual localization scheme for drone swarms where a locator commands the target drone that need to be localized to perform a short flight maneuver such as a quick turn which would act as a visual cue for the locator who is equipped with a camera to distinguish the target drone among the other swarm members. However, the maneuver command cannot be performed until a safe distance from all surrounding objects is established. Furthermore, it is difficult for stationary vehicles to perform maneuvers such as cars stopped at a red light. Finally, accepting maneuver requests from other devices adds an additional attack surface to the autonomous system.

1.3.7 Conclusion & Future Work

We proposed a vision-based authentication and localization scheme for autonomous vehicles that employ localization as a form of authentication where we use visual nonces as a proof of RF message transmission. We also introduced a Copycat Attack that exploits our scheme's synchronization vulnerabilities as well as mitigation approaches for each vulnerability. In future work, we will design testbed to **(i)** numerically define $AttRes$ and PW_g then evaluate

if $AttRes > (DC_{min} \times T_e) + PW_g$ can be applied using commodity cameras with an acceptable SINR, **(ii)** implement an encoding scheme that utilizes a low frame rate (e.g. 30FPS) as a receiver and high flickering frequency (e.g. 200Hz) as a transmitter without introducing the sender/receiver sync problem, and **(iii)** demonstrate a dual-channel protocol that sync between the low bitrate of the OCC channel and high bitrate of the RF channel without introducing the OCC/RF sync problem.

Chapter 2

Toward Reliable Networked Systems of Connected & Autonomous Vehicles

The applications of Connected & Autonomous Vehicles (CAV) are designed to perform mission-critical tasks that are vital in many areas such as autonomous driving and search-and-rescue operations. However, whether the vehicle is only connected or has some degree of autonomy, mission-critical tasks most likely would require a reliable network support. For example, a drone performing environmental inspection tasks would typically require a stable network connection to enable a remote human operator to manually control the drone. Even if the drone operates autonomously, it might also require a network support for offloading sensor data and receiving updated mission instructions. However in both cases due to the latency-sensitive nature of such tasks, the underlying application requires a reliable network support to perform adequately.

One of the main challenges in designing reliable CAV applications is verifying the application performance under different network conditions and computational workloads. Usually during the development phase of the application, a handful of scenarios would be tested.

However, designing a more exhaustive set of scenarios would be a challenging task. In this chapter, we will discuss different tools and techniques for rapid and efficient testing of CAV applications. Furthermore, we will discuss how these tools can be used to support CAV applications to maintain an acceptable performance under uncertain conditions.

In the first Module, we will go through a traffic profiling technique that can be used to create different application traffic profiles to test CAV applications under different traffic generation loads. The second Module will focus on resource allocation tools for testing and supporting CAV applications using reserved resources. Namely, this Module will discuss network slicing and computational slicing techniques. In the third Module, we propose a state-recovery protocol that act as a feedback loop for recovering the state of the network after random network disconnections.

2.1 First Module: Traffic Profiling

Connected & Autonomous Vehicles (CAV) applications might generate different volumes of network traffic during different intervals. When testing CAV applications, it might be challenging to test many application scenarios. For example, an autonomous drone could be designed to offload video frames to edge servers for image processing tasks such as object detection and tracking. However, depending on the drone's dynamics and the degree of motion in the captured video, the video encoding algorithm would continuously alter the compression threshold of the video frames for encoding efficiency. Furthermore, the camera hardware used during testing could be -for example- configured to capture video frames at a rate of 30 frames per second (FPS), while in the production phase, different camera types with different configurations could be fitted into the drones. Therefore, it would be challenging to test the performance of the drone's application using different combinations of various hardware setups and drone motion scenarios.

In this Module, we propose a traffic profiling tool that can generate network traffic patterns to represent any application traffic scenario. The tool can be used -for example- to generate video traffic at different compression and frame rates given a desired traffic profile.

2.1.1 Tool Architecture

Any traffic profile can be represented by two parameters, packet size and packet inter-arrival (Figure 2.1). With every generated packet, its payload size is determined by the packetization policy of the underlying network, while the inter-arrival of the next packet is determined by the traffic profile of the desired application. Furthermore to represent a realistic application traffic behavior, the tool introduces the concept of "objects" where each object is a set of bytes (e.g. a single video frame) generated by the application in one burst to be transported

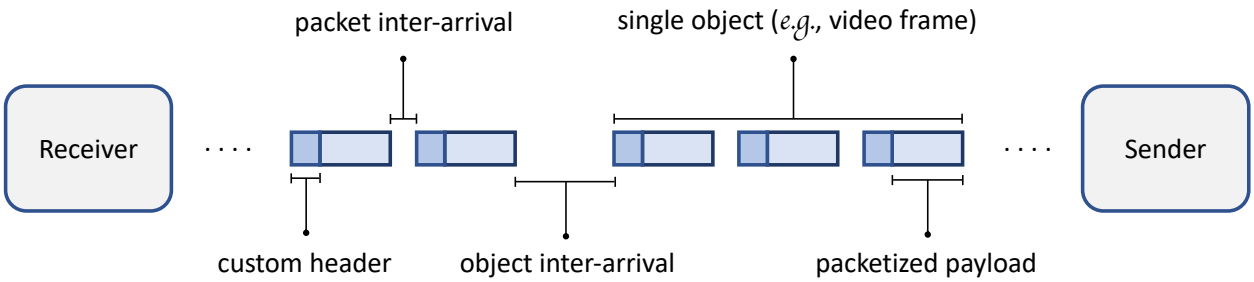


Figure 2.1: Traffic Profiling Architecture

over the network. The proposed tool also have control over the size and inter-arrival of each object. In other words, packets that belong to the same object are transmitted at the maximum network throughput while the inter-arrival between the last packet of object O_i and the first packet of object O_{i+1} is modeled by the object generation rate of the profiled application.

2.1.2 Header Design

Every transmitted packet is attached with a customized header that includes the following three fields:

- **packet_sequence_number:** a 2-bytes filed for tracking the sent and received packets.
- **object_sequence_number:** a 2-bytes filed for tracking objects that have been received successfully. Any packets that have the same object_sequence_number means that their payloads belong to the same object.
- **packet_generation_timestamp:** a 4-bytes filed for tagging the relative packet generation time in microseconds.

2.1.3 Tool Services

The proposed traffic profiling tool can provide the following services:

Customized Traffic Profiles: customized traffic profiles can be generated by the fine-grained control over the sizes and inter-arrivals of individual packets.

Customized Packetization Policy: packetization refers to the maximum data that is carried by each individual packet. If no policy is provided, the tool would attempt to fill each packet to its Maximum Transmission Unit (MTU).

Transporting Real Data Instead of Padded Data: Unlike most packet generation tools, our tool carries real data that are generated by real applications. For example, our tool can carry real video frames to edge servers to be processed for computer-vision tasks. However, video frames should be prepared offline in case different frame rates are to be tested. Nevertheless, the option of carrying padded data is also available.

Transporting Data via TCP and UDP: Our traffic profiling tool is mainly implemented over UDP to have a fine-controlled packet injection rate. As opposed to UDP, TCP has error-control and congestion-control services which would decrease the packet injection rate in case of a packet loss. However, out tool can run over both transport protocols.

Concurrent Multiple Connections: Many application scenarios require multiple devices connecting to a central service hosted on a server. Our tool can spawn multiple application processes from multiple devices and connect them to a single server. Furthermore, it support multi-client and multi-server model where a single client might connect to multiple servers for redundancy or load-balancing.

Transporting Data via Uplink and Downlink: Besides running the profiled application on client devices, the application can also be run on the server where packets are injected in

the downlink channel as well.

Dataset Generation: For every transmitted packet, the receiver logs the following features:

- **packet_size:** the size of the packet payload which is determined by the packetization policy at the sender side.
- **object_size:** the size of the entire object that this packet is part of.
- **packet_generation_timestamp:** the relative timestamp of packet generation time.
- **packet_reception_timestamp:** the relative timestamp of packet receiving time.
- **transmission_delay:** the time difference between packet_reception_timestamp and packet_generation_timestamp.
- **packet_sequence_number:** sequence number of the packet.
- **object_sequence_number:** sequence number of the object. Any packets that have the same object_sequence_number indicate that their payloads belong to the same object.
- **lost_packets:** the number of lost packets since last received packet (the difference between two consecutive packet_sequence_number).
- **corrupted_objects:** the number of corrupted objects since last received object (object segments less than object size).

2.1.4 Traffic Sample

Figure 2.2 shows a sample of a traffic profile generated by our tool. We performed the experiment over an LTE network built on the Colosseum testbed; a large network of software-defined radios (SDRs) that can be used to build various types of networks (a detailed discussion about the testbed setup will be outlined later in Section 2.2.2.1). In the experiment, we

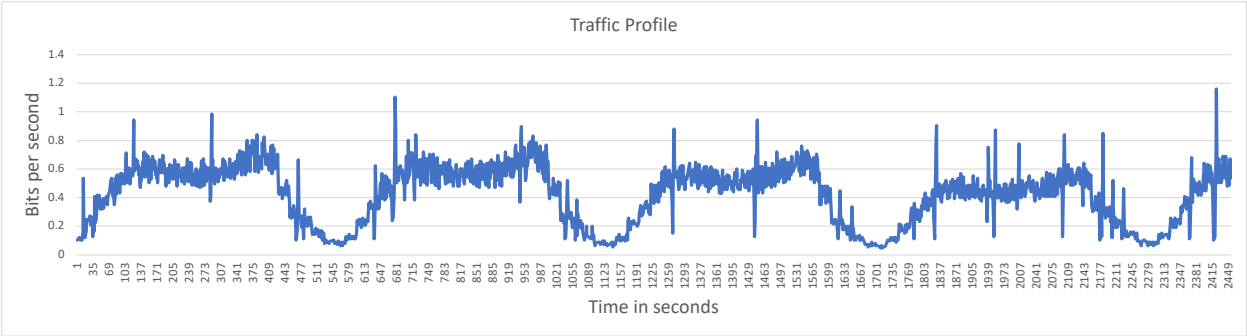


Figure 2.2: A sample of our Traffic Injection tool over an LTE network.

configure the injection rate to cycle through several periods between 100Kpbs and 800Kbps with two different peaks, one short peak at the beginning of the cycle and another longer peak at the end of it. From the experiment, we can notice the traffic spikes that are caused by the LTE HARQ protocol which is performed at the LTE physical layer.

2.2 Second Module: Resource Allocation

One of the main techniques that are used to support reliability requirements for networked applications is allocating dedicated resources to the application. This process is commonly known as "Slicing" where a group of applications or devices are allocated a dedicated amount of resources, whether network or computing resources, to ensure a stable performance of the application under different network and computational loads.

When comparing resource slicing to resource sharing, resource slicing would reserve specific amount of resources to a special set of devices or applications. However when these applications reach the maximum resource utilization of their slice, they won't get any additional resources even if other neighboring slices has unused resources. As an optimization for this problem, a "resource borrowing" technique could be used which would allow the fully utilized slice to borrow unused resources from other slices. However, these borrowed slices would be returned to its original owner when needed even if the borrowing slice is still using the borrowed resources.

In this Module, we propose two different resource allocation tools. The first tool is designed for computational slicing while the second tool is designed for network slicing.

2.2.1 Computational Slicing:

Computational slicing refers to the process of allocating a dedicated amount of CPU cycles to a specific application or a set of applications. One of the technical challenges in designing computational slicing schemes is the reliance on the hardware support; the underlying microcontroller and its operating system must have some form of resource allocation support in order to develop a computational slicing scheme on a given hardware platform. In this

Section, we propose an abstraction layer for allocating computational resources to individual applications. Our approach toward building the resource abstraction layer is by controlling the input of bytes into the CPU for individual slices. In other words, the computational slicing abstraction layer would limit the rate at which the received bytes are passed into the CPU.

To implement the computational slicing abstraction layer, the received bytes will first be placed in a buffer where a scheduler would dequeue the bytes from the buffer and pass it to the receiving application (i.e. the CPU) for processing. Every slice will have its own dedicated dequeue rate and all applications of the same slice would share the dequeue rate. This technique is similar to the Network Traffic Shaping principal which is used to smooth-out bursty traffic flows going though the network.

2.2.1.1 Design Options:

There are two options in which the abstraction layer could be implemented, either by assigning a dedicated buffer for each slice or a dedicated buffer for each device/application. In the first option as shown in Figure 2.3, all devices that belong to the same slice would share the slice buffer. While in the second option as shown in Figure 2.4, every device has a dedicated buffer but all devices that belong to the same slice would share the slice dequeue rate. The main difference between the two options is that in the first option, if the total packet injection rate of all devices in the same slice surpasses the slice dequeue rate, then the effective throughput of each device would be the slice dequeue rate multiplied by the ratio of the device injection rate to the total injection rate. On the other hand in the second option, if the total packet injection rate surpasses the slice dequeue rate, the dequeue rate will be divided between the slice devices evenly (assuming that the local scheduler of the slice gives no priority to any device).

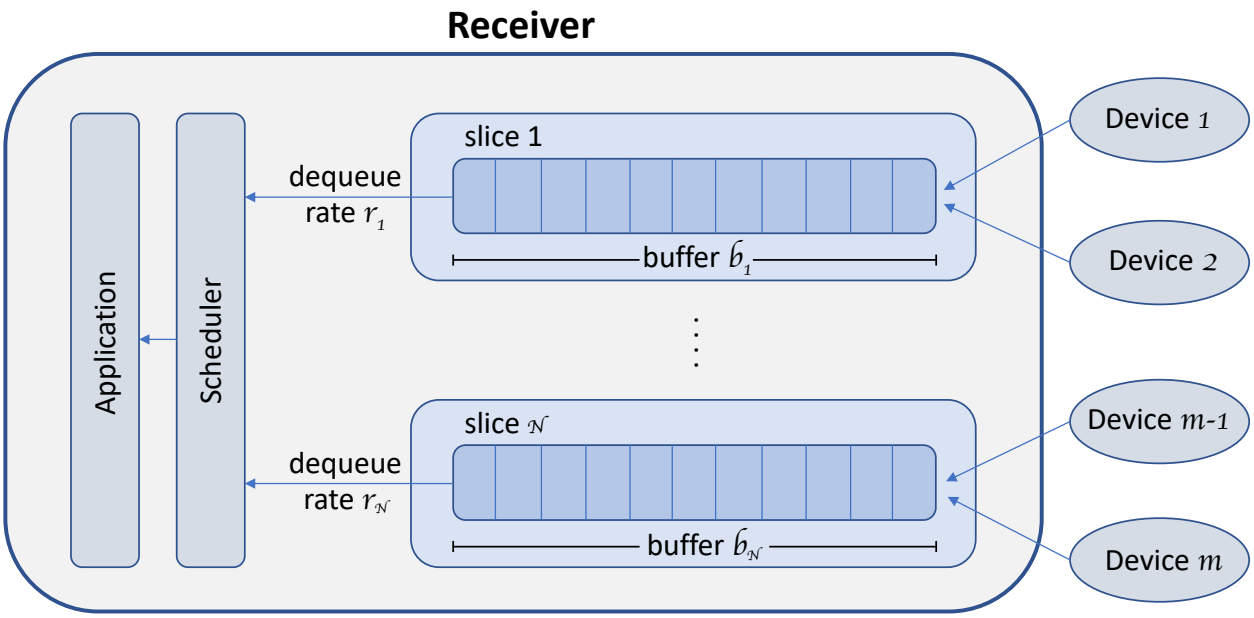


Figure 2.3: Design Option 1 - Dedicated Buffer Per Slice

Figure 2.5 shows a simple computational slicing experiment performed using design option 1. The experiment is performed over an LTE network built on the Colosseum testbed; a large network of software-defined radios (SDRs) that can be used to build various types of networks (a detailed discussion about the testbed setup will be outlined later in Section 2.2.2.1). In the experiment, we initiate a slice with a 250KBytes buffer and 2Mbps dequeue rate. Then we initiate two devices, A and B. Both devices are attached to the slice. In the beginning of the experiment, device A steadily injects 1Mbps of traffic into the slice. After 10 seconds, device B start injecting 4Mbps of traffic into the slice. From the results in Figure 2.5, the effective throughput of device A is 0.4Mbps while 0.6Mbps of the device traffic is dropped. For device B, 1.6Mbps pass through the slice and 2.4Mbps gets dropped. In other words, both devices have 40% of their traffic dropped and 60% of their traffic pass through the slice into the CPU. Also we can notice how the two devices compete over the slice buffer between the 10 to 22 seconds window. Then around the 22nd second, device A stops injecting any traffic into the slice which in turn would leave the entire slice resources available for device B. Here we can notice that device B traffic pattern becomes stable even

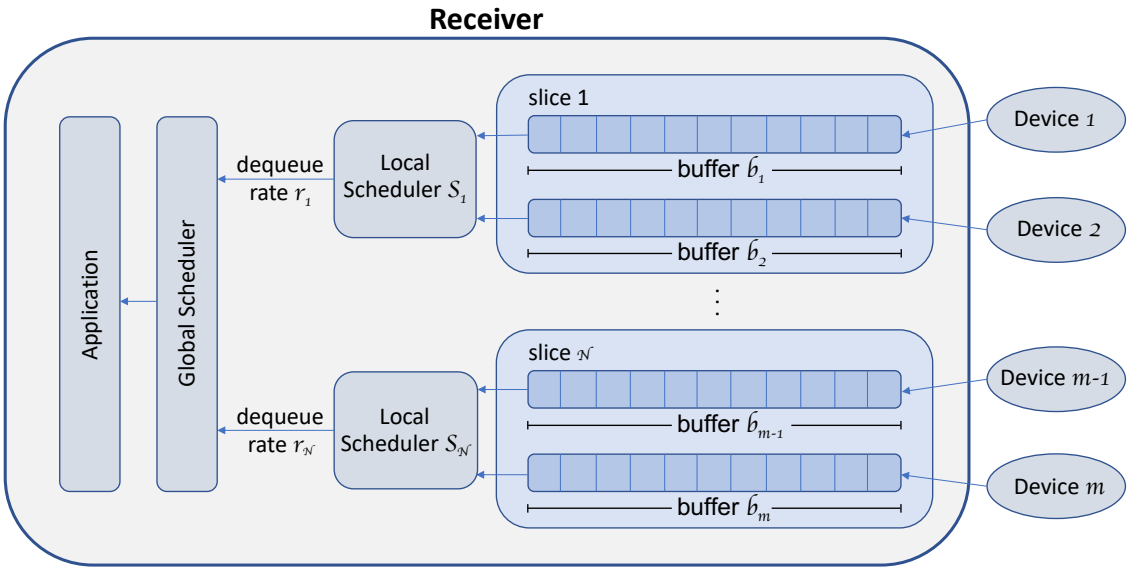


Figure 2.4: Design Option 2 - Dedicated Buffer Per Device

though 50% of the injected traffic is being dropped since it exceeds the slice dequeue rate. If we choose to run the same experiment using the second design option with a fairly weighted round-robin configured as the local slice scheduler, and both devices are injecting at the same time, 100% of device A's injected traffic (1Mbps) would pass through the slice while only 25% device 2 injected traffic (1Mbps) passes through the slice, and the remaining 3Mbps would get dropped. In other words when using the second option, if a device injection rate surpasses its share of the slice dequeue rate, only that device would experience packet drop and all other devices in the slice would remain unaffected.

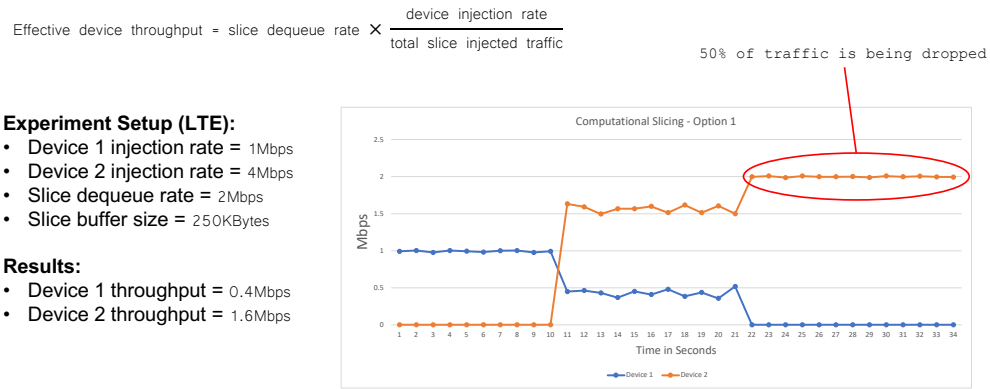


Figure 2.5: Computational Slicing experiment performed on an LTE network using design option 1

2.2.1.2 Configurable Parameters & Collected Dataset:

Our computational slicing tool provides the following options to control different slicing parameters. Namely, the main configurable parameters are:

- The choice of two different design options as shown above.
- The ability to initiate any number slices.
- For each slice, any number of devices can be initiated and associated with the slice.
- For each slice, a customized dequeue rate can be configured.
- Configuring a customized global scheduler for dispatching bytes to applications for processing based on a configurable policy.

In addition to main parameters, each design option has additional specific parameters that are tied to that option. For option 1, each slice can have a dedicated buffer with a customizable buffer size. For option 2, every device can have a dedicated buffer with a customizable buffer size, while each slice can have its own local scheduling policy for dispatching received bytes from devices' buffer toward the global scheduler. Note that all of these parameters can be changed at runtime.

One of the main advantages of using a local slice scheduler in option 2 is that in conventional CPU resource allocation, the computing resources are allocated based on CPU cycles. However our proposed computational slicing tool allocates computing resources based on the number of processed bytes. These two allocation approaches are not necessarily the same since some tasks are more computationally extensive than other, regardless of the size of the task input. For example, a video frame sent for an object recognition task would consume less CPU cycles if the same video frame is sent to an object segmentation task. Therefore,

abstracting the CPU cycles as a number of processed bytes might not always represent accurate computational slicing. To mitigate this disparity, the local scheduler of the slice can assign different weights to different applications where each weight represent the estimated additional overhead of its underlying task.

2.2.1.3 Collected Dataset:

In addition to the datasets collected by our traffic profiling module, if our computational slicing tool is enabled as well, additional features will be collected. These additional features are:

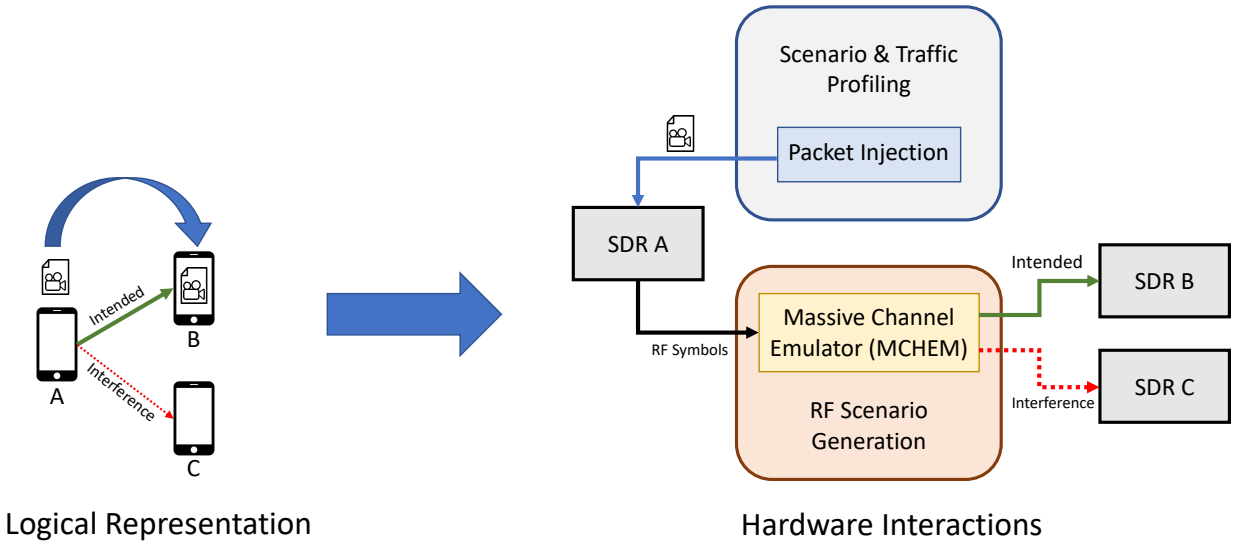
- **slice_id:** a numerical identifier of the slice which houses the device that sent the packet.
- **dequeue_rate:** the current dequeue rate assigned to the slice.
- **slice_buffer_utilization:** the current utilization ratio of the slice buffer (option 1 only).
- **slice_buffer_size:** the maximum buffer size currently assigned to the slice (option 1 only).
- **device_buffer_utilization:** the current utilization ratio of the device buffer (option 2 only).
- **device_buffer_size:** the maximum buffer size currently assigned to the device (option 2 only).
- **dispatch_timestamp:** the timestamp of the packet leaving the computational slicing layer.
- **buffering_delay:** the time difference between the packet receive time and the dispatch time.

2.2.2 Network Slicing:

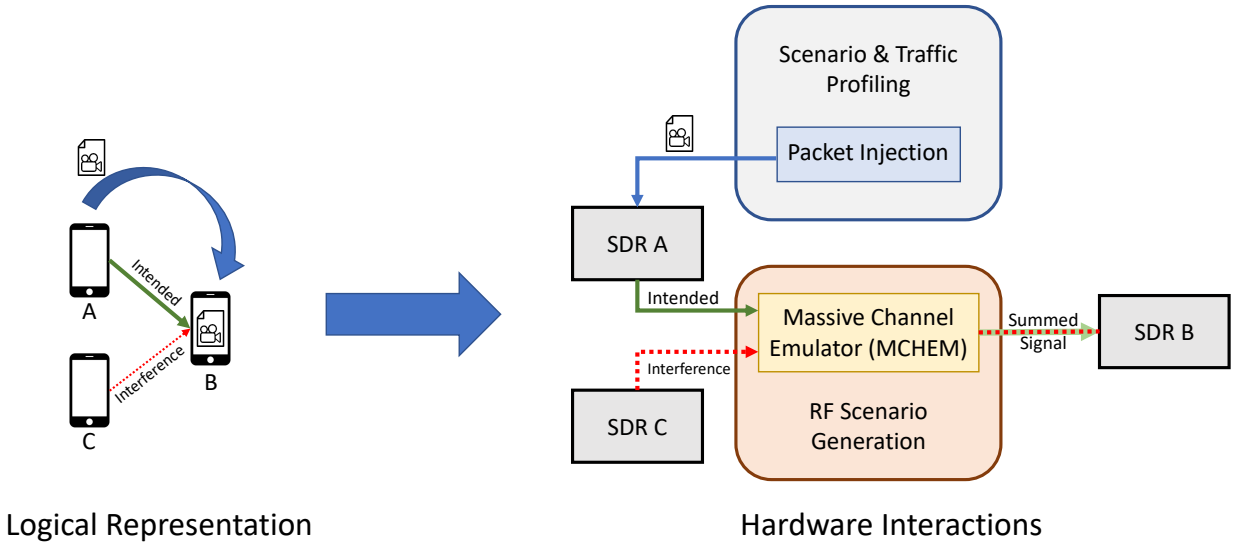
Network slicing refers to the process of allocating a dedicated amount of physical-layer network resources (e.g. bandwidth) to a specific group of devices or applications. However, it is not feasible to abstract the network resources without a form of hardware support. In this Section, we propose our network slicing tool that is built on top of Colosseum [18], world's largest RF emulator.

2.2.2.1 Colosseum Testbed Overview:

Colosseum is a 100 millions dollar network of high-end servers and software-defined radio (SDR) devices. The SDR hardware used in the Colosseum is USRP X310, a high-end SDR device with a 200 MHz bandwidth per channel. The Colosseum testbed is composed of 256 SDRs connected to another 256 SDRs. This configuration is capable of outputting over 65K independent RF channels. Although the testbed is built for wireless experiments, all SDRs are connected via coaxial cables. To emulate the wireless environment, every SDR-generated signal goes through a Massive Channel Emulator (MCHEM) which is composed of 512 complex-valued FIR taps that are used to apply environmental conditions to the signals such as fading, path-loss, and interference. For example as shown in Figure 2.6a, when SDR_A sends a signal towards SDR_B, SDR_C would receive a distorted copy of the signal as interference based on the used RF scenario (e.g. applying node mobility or multi-path effects). Similarly in Figure 2.6b, when SDR_A sends a signal to SDR_B while having SDR_C is transmitting in the background, SDR_B would receive the summed signal of SDR_A and SDR_B transmissions.



(a) Implementing downlink channel on Colosseum



(b) Implementing downlink channel on Colosseum

Figure 2.6: Colosseum Hardware Setup

2.2.2.2 Testbed Setup:

Our testbed setup as shown in Figure 2.7 starts with srsRAN (formally known as srsLTE, an open-source implementation of the LTE and 5G standards) which runs on top of the SDR device. Then within the srsRAN stack runs SCOPE [17], a physical-layer slicing framework that allocates the Physical Resource Blocks (PRB) of the LTE channel into separate slices. By the LTE standard, a PRB is composed of 12 subcarriers, each with a bandwidth of 15 KHz with a duration of 0.5ms. Therefore, the total bandwidth of a single PRB is 180 kHz wide in frequency with a duration of 0.5ms. When using SCOPE framework, we can dynamically allocate a dedicated amount of PRBs to separate network slices. On top of SCOPE, we implement our network slicing tool which is controlled by an optimization policy such as a Deep Reinforcement Learning (DRL) agent.

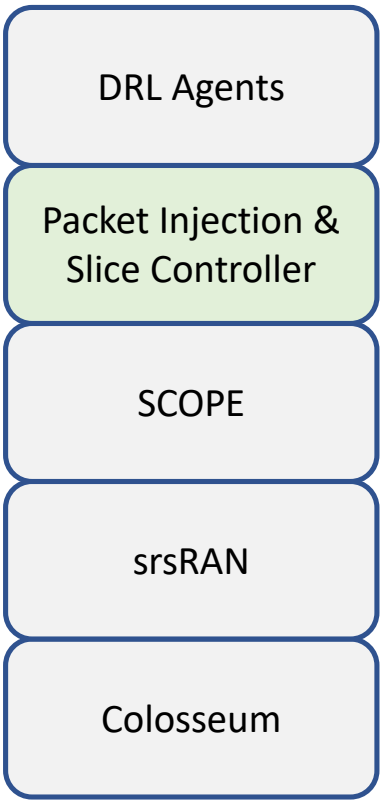


Figure 2.7: Colosseum Testbed Setup

2.2.2.3 Proposed Slicing Tool:

Our proposed network slicing tool further extends the capabilities of SCOPE by attaching a customized version of our traffic profiling tool along with a Slice Controller module. To test and demonstrate the capabilities of our tool, we implement a DRL agent as the user of the slicing tool (Figure 2.8).

SCOPE has a resolution of 250ms which means the system updates its state four times every second. Therefore, the traffic profiling tool was repurposed to update the traffic injection parameters every 250ms instead of after every packet since SCOPE system update would be the composition of all fine-grained individual traffic updates (different packet sizes and inter-arrivals). Therefore, our slicing tool is designed to sync its system state with SCOPE state. Another limitation imposed by SCOPE is that PRB allocation can take place in downlink channel only (i.e. from the LTE base station to the LTE devices).

2.2.2.4 Network Slicing Tool Functions:

For our slicing tool to work properly, SCOPE source code has been modified to automatically configure and set up the experiment parameters of SCOPE and Colosseum. When the slicing tool is invoked, it spawns eNodeBs (LTE base station) and UEs (User Equipment, a term for LTE mobile devices). Then every UE attaches itself to its assigned eNodeB via the LTE network (i.e. through the SDR interface) and every eNodeB initiate the network slices with its assigned UEs, as dictated by the experiment initial configurations. Here each slice will have a dedicated slice controller that can be controlled by a dedicated DRL agent. Once a UE connects to its eNodeB, the UE then opens another connection with the eNodeB but via the Colosseum's internal network. This additional side channel will be used to report the packet delivery statistics (e.g. delay) back to the slice controller without using any LTE resources.

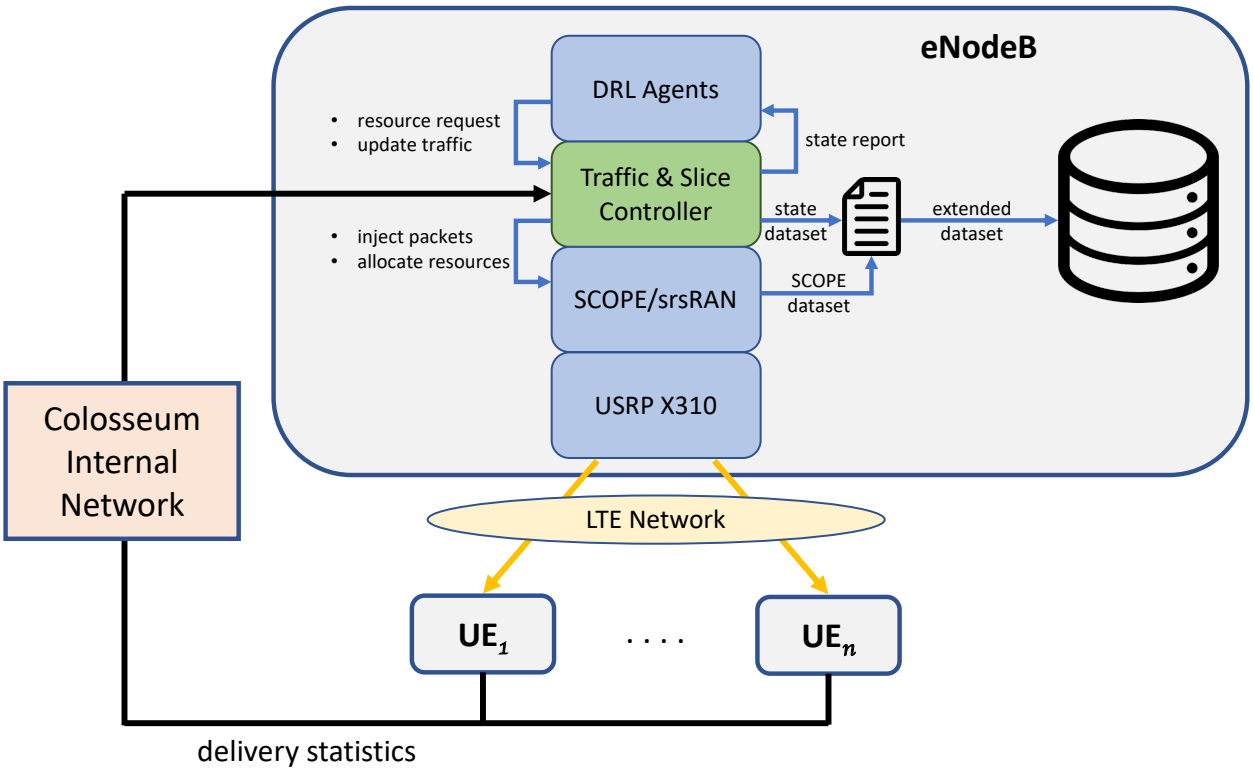


Figure 2.8: Network Slicing Architecture

2.2.2.5 Runtime Operations:

Since the slicing tool uses our traffic profiling tool, the injected traffic can be carried by either TCP or UDP. Also UEs can be added or removed at runtime. For each sent packet by the eNodeB, UE would send packet statistics back to its eNodeB. Once a report is arrived, it will be re-formatted and passed to the DRL agent for fine-tuning the model. Then at every new system state, the DRL agent would send network traffic updates and PRB allocation requests to the slicing tool. Then the slicing tool would update the injection parameters of the used traffic profile as well re-allocating the PRBs among the slicing based on the agent's request. Finally at the end of every system state, SCOPE would output the statistics of that state. Here the traffic profiling tool will merge its own statistics with SCOPE's such as the total amount of PRBs that were used in the last 250ms.

2.2.2.6 Collected Dataset:

The Slice Controller collects additional features which are generated every 250ms:

- **slice_prb:** current Physical Resource Blocks (PRB) assigned to the sender's slice.
- **requested_slice_resources:** the amount of slice resource requested by the DRL agent.
- **dl_buffer:** the number of bytes in the slice downlink buffer.
- **tx_brate:** the rate of bytes passing through the slice buffer in Mbps.
- **tx_packets:** total number of packets passed through the slice buffer since the last 250ms.
- **dl_buffer:** the number of bytes in the slice downlink buffer.
- **sum_requested_prbs:** total number of PRBs requested by the DRL agent since the last 250ms.
- **sum_granted_prbs:** the actual number of assign PRBs to slice since the last 250ms.
- **lost_packets_per_250ms:** total number of lost packet since the last 250ms.
- **ul_sinr:** the signal-to-interference-plus-noise ratio (SINR) of the up-link channel.
- **ul_rssi:** the received signal strength indicator (SINR) of the up-link channel.
- **action:** the action taken by the machine learning model in the current 250ms time slot.
- **action_values:** the possible actions for the DRL agent.
- **agent_reward:** the DRL reward for the taken action.
- **agent_loss:** the value of the DRL loss function.

2.3 Third Module: Active Feedback Loop & State-Recovery

Continuing the discussion on reliability of cellular networks, it is imperative to discuss about the 5G's URLLC service class and what type of support such class of applications need.

5G Systems (5GS) are expected to bring substantial improvements over its predecessor, 4G LTE. These improvements are designed to be utilized by three main classes of applications, namely Enhanced Mobile Broadband (eMBB), Massive Machine-Type Communications (mMTC), and Ultra-Reliable Low-Latency Communications (URLLC). The main objective of eMBB is to provide very high data rates for users (100Mbps per user and up to peak rates from 10 to 20 Gbps). mMTC on the other hand focuses on providing low-power wide-area (LPWA) connectivity for area coverages with a high density of devices such as a wireless network of low-cost sensors. For supporting mission-critical applications (*e.g.,* connected and autonomous vehicle applications), URLLC service class is used. In URLLC, the end-to-end latency requirement is assumed to be no more than 5ms while the reliability of packet delivery is set to be no less than 99.999%. These two requirements are conflicting in nature and are difficult to be jointly achieved by the current protocols. For example, the packet retransmission scheme used to support the reliability guarantee service of the Transmission Control Protocol (TCP) increases the latency of delivered packets. On the other hand, while User Datagram Protocol (UDP) does not induce any additional latency on transmitted packets, it does not guarantee transmission reliability. Other transport protocols attempt to combine the best features of both TCP and UDP. However, often these protocols are designed with a specific application in mind such as Real-Time Transport Protocol (RTP) which is designed for the synchronization of real-time streaming of audio and video, but fail to meet the reliability guarantee of URLLC applications.

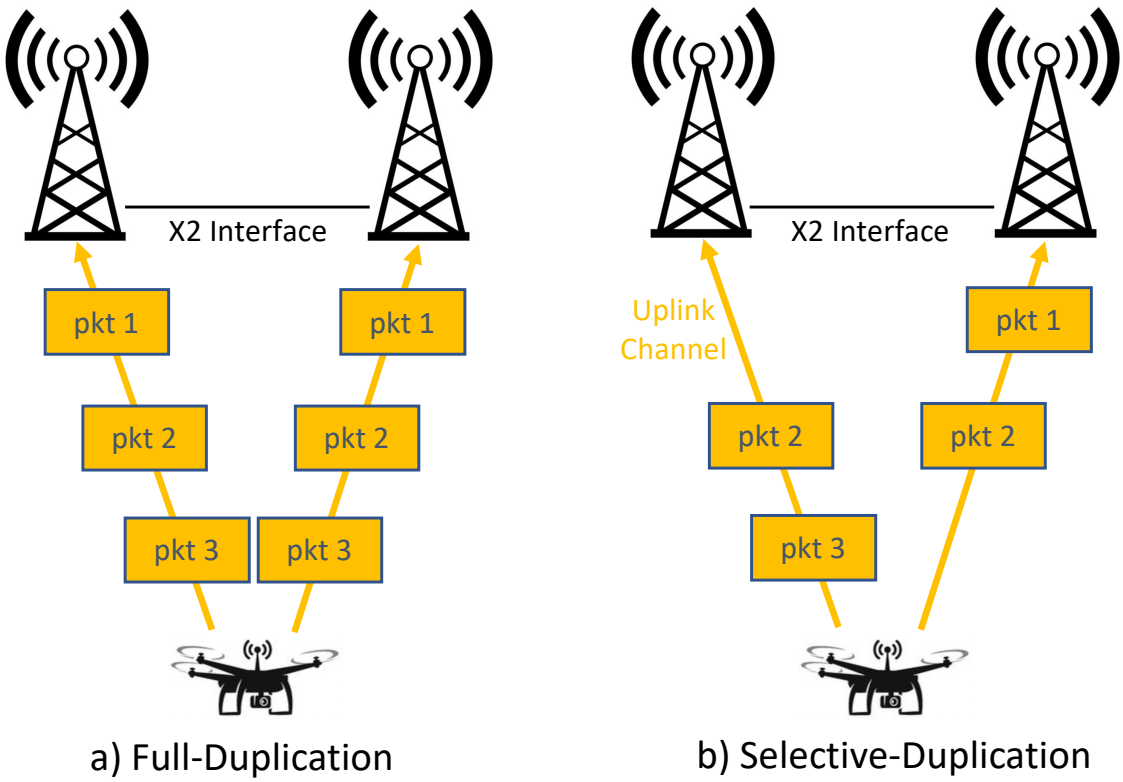


Figure 2.9: Packet duplication schemes

In an attempt to combine these two strict and conflicting requirements into the same URLLC service class, the system architecture draft for the 5G System (version 16.6.0 release 16) released by the 3GPP group [5] suggests that URLLC applications could establish two redundant data sessions over the 5G network. This redundant connectivity scheme is referred to as Dual-Connectivity (DC) in the literature, which is part of the general Multi-Connectivity (MC) scheme. These two sessions can be established over two different channels toward either the same base station (BS) or two different BSs. Importantly, URLLC data is not split over the two channels to increase data rate, but for the purpose of packet duplication. That is, each packet generated by a URLLC application is sent twice, once over each channel. This scheme is known as Packet-Duplication (PD). Using the PD/MC scheme ensures that a packet lost over one channel can be recovered by the other channel without any packet retransmission. This results in an improved packet latency and reliability, at the price of a doubled bandwidth usage.

2.3.1 Related Work

Different research efforts have addressed various issues related to URLLC applications. Research in [30] and [66] investigate the effect of correlated channels on URLLC applications, as well as approaches to find the least correlated channels. As correlation in interference and path-loss decreases diversity, a packet duplicated on two distinct but correlated channels is likely to have a similar outcome on both channels. The authors in [29] extend the channel correlation problem and investigate the effect of correlated queues, i.e., the queues of different BSs with similar buffering loads. Other contributions study the minimization of PD/MC overhead on 5GS caused by URLLC applications. For example, in [35] a technique is provided to optimally decide which URLLC users are qualified to obtain an MC privilege – and thus increase reliability – and which users can use PD on top of MC. Other contributions, such as [24] and [71], study a PD scheme, called selective-duplication, where only packets that meet a set of criteria, such as poor channel quality at the time of transmission, are tagged for duplication. The main purpose of this scheme is to lower the URLLC emitted traffic, which is inversely proportional to the overall number of admissible URLLC users.

2.3.2 Problem Formulation & Contribution

As discussed earlier, a selective-duplication scheme can reduce the bandwidth usage of URLLC applications. Figure 2.9 compares a full-duplication scheme (2.9a) with its selective-duplication counterpart (2.9b). In Figure 2.9b, it is assumed that the delivery probability of packet 2 is below the URLLC threshold of 99.999%, while packets 1 and 3 meet the requirement. Figure 2.9a shows that the full-duplication scheme duplicates all packets regardless of the specific network conditions. On the other hand, the selective-duplication scheme only duplicates packet 2. The latter approach reduces bandwidth usage for the incoming packets. However, the duplication decision (whether it is optimization-based [35] or machine

learning-based [71]) relies on channels' feedback and overall state (both current and previous states) of the network such as current channel conditions and delivery statistics of previously transmitted packets.

Different network statistics, such as signal-to-noise ratio, can be acquired directly by the transmitter, but other important statistics can only be reported back by the receiver in the form of direct acknowledgments (ACKs). However, the ACK message itself could be lost and thus, the state of the transmitted packet would be lost as well. This in turn would likely degrade the performance of selective duplication decisions for the next packets. In other words, it is difficult to precisely determine whether the packet or its ACK is lost - i.e., the so-called lost acknowledgments problem (LAP). Furthermore, the retransmission of packets (or their ACKs) is not feasible for URLLC applications due to their strict delay requirement.

There is a plethora of research efforts [51, 73, 25] addressing the LAP with the main focus on enhancing the reliability of packet delivery or the throughput using packet retransmission. Nonetheless, the objectives of acknowledgments in URLLC applications are different (deriving the state of the transmitted packets and its underlining network) than the mainstream networked applications, where delivery guarantee comes at the expense of a larger delay.

In this module, we propose a state-recovery protocol for URLLC applications. The main motivation behind the proposed protocol is to aid and enhance the selective-duplication performance by identifying whether the loss is related to the transmitted packet or its ACK. We implemented our protocol on the Colosseum lab [18]. The results reveal that the proposed protocol enables a perfect selective-duplication scheme to outperform the full-duplication scheme by enhancing channel efficiency up to 95%. The protocol proposed in this module is designed to support selective duplication in the uplink (UL). However, the protocol can be repurposed to support downlink (DL) as well. Importantly, our protocol improves the efficiency of the existing Selective-Acknowledgment (SACK) schemes [16] used by modern TCP implementations. SACK requires 8-bytes to represent a single contiguous gap (maxi-

mum of four gaps) of an unacknowledged stream of bytes, while our protocol represents an entire unacknowledged packet with a single bit. Furthermore, our protocol offers additional advantages such as indicating the delivery deadline state for previous packets and combining multiple acknowledgments from multiple BSs.

2.3.3 State-Recovery Protocol

The proposed state-recovery protocol enables the differentiation of the loss of a packet and its acknowledgment. To this end, the protocol recovers the state of a transmitted packet in the case that its ACK is not received. The state of the packets can then be used to aid the duplication decision process of the next packet. The design and implementation of such a process can take the form of either an optimization-based or machine learning-based algorithm. However, the details of this process are out of the scope of the module.

2.3.3.1 Protocol Design Overview

We consider a system including one URLLC device, simply called device, and M base stations, where $1 \leq m \leq M$ indexes the m-th BS. There are a total number of M channels in the system each assigned to one BS. $s_i = \{r_i, d_i\}$ is the state of packet i (pkt_i), where both r_i and d_i are binary variables. r_i indicates the packet delivery status, where r_i is equal to "1" if pkt_i is successfully received by the receiver, and "0" otherwise. On the other hand, d_i represents the packet deadline status, based on the relationship between the packet's delivery delay (sum of the transmission and queuing delays) dl_i and a predefined deadline parameter d_p. Specifically, d_i is equal to "1" if $dl_i \leq d_p$ and "0" otherwise. These two binary variables are attached to each acknowledgment ACK_i. Moreover, the receiver tracks the states of the last n packets. Then, with each sent out acknowledgment ACK_i, the receiver attaches the state of the last n packets to the acknowledgment, *i.e.,* $ACK_i = s_i + \{s_{i-1}, \ldots, s_{i-n}\}$.

Lastly, for each BS_m, a binary status $b_{m,i}$ is defined, where $b_{m,i}$ is mapped to "1" if BS_m has received a copy of pkt_i and "0" otherwise. Then, $b_{m,i}$ is attached to ACK_i such that $ACK_i = s_i + \{s_{i-1}, \ldots, s_{i-n}\} + \{b_{1,i}, \ldots, b_{M,i}\}$. Notably, it is assumed that all BSs can communicate via a reliable channel such as the X2 interface in LTE networks.

2.3.3.2 ACK Header Structure

Figure 2.10 depicts the proposed ACK header, which consists of five fields: 1-byte sequence number (sn), 1-byte packet delivery delay (dl), n-bits ACK array (ar), n-bits deadline array (dr), and m-bits BS array (br). For each pkt_i, $sn = i$ is simply the sequence number of the packet successfully received at the BS, and dl_i shows the total delivery delay in milliseconds. Both fields can take any number ranging between [0, 255]. ar_i and dr_i on the other hand are the delivery state and the deadline state (since the reception of pkt_i) of the last n packets, respectively. For instance, for pkt_{i-2}, we have $r_{i-2} = ar_{i-2}[-2]$[1] and $d_{i-2} = dr_{i-2}[-2]$, where the position of each bit in the array represents the position of pkt_{i-2} with respect to $sn = i$ within a series of n packets. Each bit in br_i is mapped to "1" if a BS has successfully received pkt_i and "0" otherwise.

2.3.3.3 Acknowledgments Duplication

The selective-duplication process requires an immediate ACK response to determine the duplication decision for the next packet. If an ACK is lost, the channel recovery must be postponed until the next ACK is successfully received. This waiting time would force the selective-duplication process to either rely on outdated information from the last received ACK or simply consider the previously sent packet as lost. To mitigate this issue, every ACK sent on one channel is duplicated over all the other channels, regardless of whether the

[1] A negative index indicates indexing from the array tail where [-1] is the last element in the array while [-2] is the second to last element, and so on.

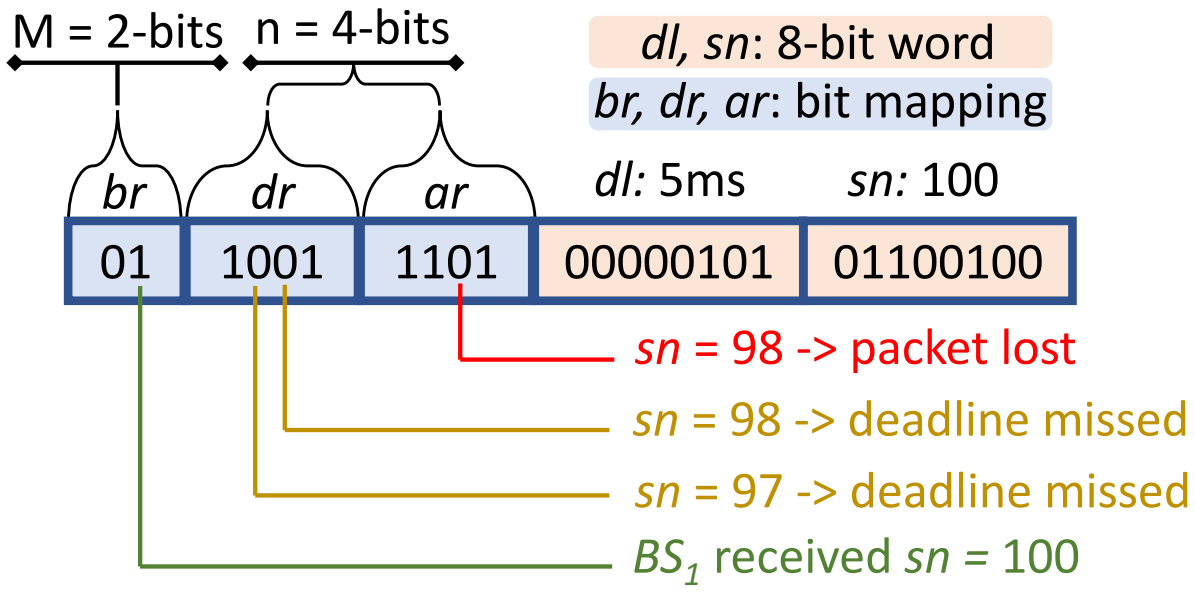


Figure 2.10: An example of the proposed ACK header

packet was duplicated or not. When a BS receives a packet from the UL channel, it first duplicates the ACK to all other BSs over a reliable backbone network, *e.g.,* X2 interface in LTE networks. Upon receipt of an ACK from the backbone network, the BS relays the ACK toward its own DL channel. To prevent sending the same ACK multiple times, the BS backs off for d_p seconds when it receives a UL packet in order to collect ACKs from all other BSs and fills up br before forwarding the ACK to the DL channel.

2.3.3.4 Protocol Scenarios via Sequence Diagram

The proposed protocol can recover the state of packets under different scenarios. Figure 2.11 shows that the recovery starts right after successfully transmitting four packets. Note that, dl and dr were omitted from the ACK header in Fig. 2.11 due to a space limitation. The first transmitted packet in Fig. 2.11 starts with pkt_4 ($sn = 4$) and is duplicated on both channels. The packet and its ACK are lost in $channel_2$ and $channel_1$ respectively. However, ACK_4 is recovered over $channel_2$ by BS_2. From $br = [01]$ of the received ACK, the device

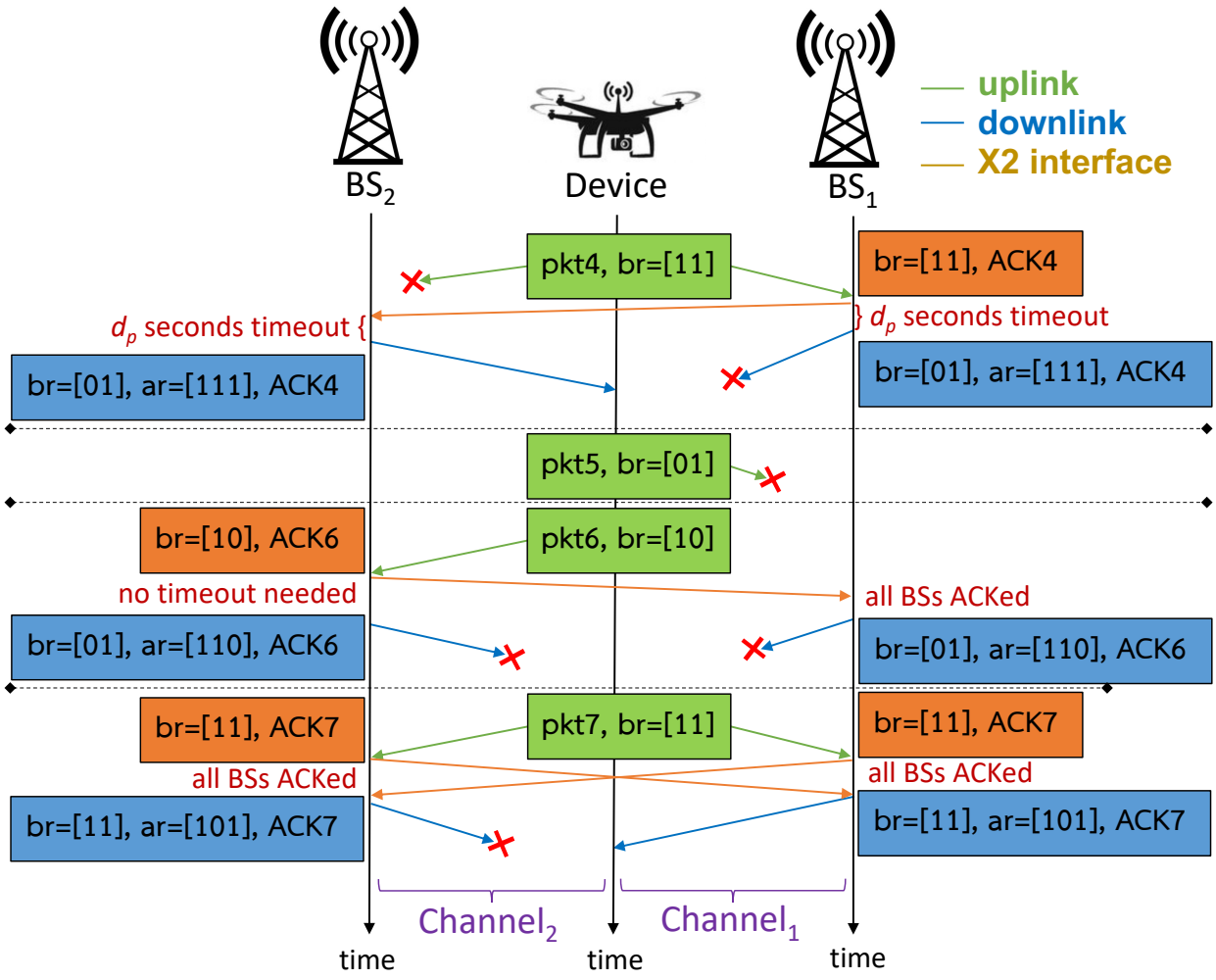


Figure 2.11: Sequence diagram of the proposed protocol

infers that the packet sent to BS_2, and the ACK sent by BS_1, are lost. Note how BS_1 backed off for d_p seconds waiting for a duplicated ACK from BS_2 to fill its *br* before sending its own ACK over $channel_1$. Similarly, BS_2 backed off for d_p seconds waiting for the UL packet to arrive. Thereafter, the device attaches its own *br* before sending the packet. This *br* is for optimization purposes to inform the BS about what other BSs should have received a packet duplicate. Hence, an ACK duplicate from other participating BSs is also expected over the X2 interface. Next, pkt_5 is sent over a single channel, but is lost. Then, pkt_6 is sent over a single channel as well, but all its ACKs (original and duplicates) are lost. The loss of all packet duplicates and/or its ACKs is the worst-case scenario for the state-recovery protocol

as there is no feedback from the network. However, after sending pkt_7, the device is able to recover the state of the previous two packets. Also, neither of the BSs backed off for d_p seconds since all duplicate ACKs were received. Interestingly, the device can infer from ACK_7 that pkt_7 was received by both BSs (by looking at 1s in $br = [11]$) and also, ACK_7 from BS_2 was lost. However, if ACK_7 from both BSs are lost, the next successfully received ACK (*e.g.,* ACK_8) can only predict that pkt_7 was successfully received by *some* BSs yet it cannot identify which one actually received the packet. This limitation is determined by the need to compressed information in the ACK header. The knowledge of previously used channels, *i.e.,* which BSs exactly received the previous packet, increases the header size w.r.t br and ar from $m + n$ to $m \times n$ bits.

2.3.4 Implementation & Evaluation:

A prototype of the protocol is implemented in the Colosseum [18], which emulates RF signals using a network of high-performance Software-Defined Radios (SDR), specifically, X310 USRP radios. The protocol is built on top of srsRAN, formerly known as srsLTE – an open-source implementation of the LTE stack, where User Equipment (UE) devices connect to and exchange traffic with BSs. We design an experiment where a single device is connected to two BSs via LTE. The device is configured to transmit a series of 1000-byte packets on UL for five minutes. The inter-departure of the packets is set to 8ms, which corresponds to a network traffic of $\approx$ 1 Mbps. During the experiment, the UE duplicates each packet toward each BS. Then, both BSs are connected via the Colosseum's internal cabled network to emulate a reliable LTE's X2 interface for the BSs. The implementation evaluates the effectiveness of the proposed protocol under different channel conditions. Specifically, we test the protocol under two experiments that use different signal decibel Full Scale (dBFS) receive powers: -50 dBFS and -70 dBFS. For each experiment, we compare the efficiency of the proposed protocol against the efficiency of the full-duplication scheme. The former is

Table 2.1: Experiment Results

	Experiment 1: -50dBFS	**Experiment 2: -70dBFS**
Total # of packets	37451	37459
# of packets received on both channels	35350	29695
# of packets received on one channel	2073	7620
# of packets lost in UL	28	144
# of ACKs lost in DL	0	12
Max # of continuous lost packets	2	3

defined as the percentage of packets successfully received on both channels, while the latter is expressed as the percentage of packets successfully received on only one channel. In this regard, we collect the number of packets lost in the UL, the number of ACKs lost in the DL, and the number of packets delivered via each channel.

Table 2.1 lists the results of both experiments. In the first experiment, the total number of packets successfully received is 37423, where 35,350 packets are received on both channels and 2073 ones are received on only one channel. This results in an efficiency of 94.5% and 5.5% for the protocol-assisted selective-duplication and the full-duplication strategy, respectively. In the second experiment, the efficiency of the full-duplication strategy increases to 20.4%, and our protocol improves efficiency up to 79.6%. This observation further motivates the usefulness of the selective-duplication scheme, and thus, the need for a state-recovery protocol to support an efficient duplication decision strategy.

In the first experiment, 28 packets are lost on both channels. The missing ACKs of these 28 packets provide an indication that the packets are lost. However, the UE device cannot determine with certainty whether the packets or its ACKs were lost in the absence of additional feedback enabling the recovery of the actual states of these missing packets. In the second experiment, 156 packets have missing ACKs, even though only 12 packets were

actually lost. To recover the past packet and network states, n should be selected carefully as it indicates the maximum number of packets that are expected to be lost in a sequence. Moreover, the maximum number of contiguous packets lost in experiments 1 and 2 are 2 and 3 packets, respectively.

2.4 Conclusions and Future Work

In this Chapter, we discussed different reliability approaches related to the networks of Connected Autonomous Vehicles (CAV). We started the discussion with the traffic profiling of CAV applications and how the process of dynamically profiling various CAV applications helps in accelerating the testing of reliability solutions. Then we proposed a computational slicing technique that abstracts the allocation of computational resources by controlling the flow of bytes into the CPU. We demonstrated two different computational slicing designs, the first one shares the slice buffer with all devices under the same slice which in turn leads to buffer contention when the total bytes injected by the slice devices exceeds the maximum slice dequeue rate. The second design allocates a dedicated buffer space to each device instead of having a shared slice buffer on the expense of added complexity. Next we discussed network slicing and the allocation of physical resource blocks (PRB) of LTE networks. We proposed a Slice Controller agent that runs on top of SCOPE, physical-layer slicing framework for cellular networks. The slicing framework is implemented on top of srsRAN, an open-source implementation of the LTE and 5G standards. A prototype of the slice controller agent has been demonstrated on the Colosseum testbed, world's largest RF emulator which is comprised of a large network of Software-Defined Radios (SDRs). Finally, we proposed a State-Recovery Protocol for aiding selective-duplication frameworks which are used to lower the consumed bandwidth for 5G's URLLC applications. The protocol helps selective-duplication frameworks by recovering the state of lost packets/acknowledgements

in the network which is essential information for making selective-duplication decisions. The protocol has the potential to be extended for other applications such as load-balancing, split/edge computing, and network slice management. There are different improvements for the protocol that can be utilized by the aforementioned applications such as 1) embedding the reading of the current buffer size of network slices, 2) dedicating a field in the header for reporting individual channel states, 3) introducing an adaptive header size for adding extra network statistics in the case of poor channel conditions, 4) using ACK timeouts, 5) implementing continuous/triggered ACKs, and 6) involving the sender in the message exchange. The latter is beneficial to report the scheduled transmission time of the next packet, whereby the receiver can reply with a triggered ACK if no packet is received within the scheduled time limit.

Bibliography

[1] *IEEE Standard for Local and metropolitan area networks.* IEEE 802.15.7-2018.

[2] Scapy, 2020-03-13.

[3] Wifi analyzer, 2020-03-13.

[4] Wireshark, 2020-03-13.

[5] 3GPP. System architecture for the 5g system (5gs). *3rd Generation Partnership Project (3GPP), Technical Specification (TS) 23.501*, 2020.

[6] AirMap. *AirMap, Wing, and Kittyhawk.io Demonstrate InterUSS Network-Based Remote ID Application*, 2020 (accessed July 5, 2020). `https://www.airmap.com/airmap-wing-kittyhawkio-demonstrate-network-based-remote-identification-interuss-platform/`.

[7] AirMap. *Deploy Digital Airspace Automation For UAS Traffic Management*, 2020 (accessed July 5, 2020). `https://www.airmap.com/authorities/`.

[8] Alipour-Fanid et al. Machine learning-based delay-aware uav detection and operation mode identification over encrypted wi-fi traffic. *IEEE Transactions on Information Forensics and Security*, 15:2346–2360, 2019.

[9] A. Alsoliman, M. Levorato, and A. Chen. Vision-based two-factor authentication & localization scheme for autonomous vehicles. In *Third International Workshop on Automotive and Autonomous Vehicle Security (AutoSec) 2021 (part of NDSS)*, 2021.

[10] A. Alsoliman, A. B. Rabiah, and M. Levorato. Privacy-preserving authentication framework for uas traffic management systems. In *2020 4th Cyber Security in Networking Conference (CSNet)*, pages 1–8. IEEE, 2020.

[11] ASTM. *Standard Specification for Remote ID and Tracking*, 2020 (accessed July 5, 2020). `https://www.astm.org/Standards/F3411.htm`.

[12] S. Benzarti, B. Triki, and O. Korbaa. Privacy preservation and drone authentication using id-based signcryption. In *SoMeT*, pages 226–239, 2018.

[13] S. Birnbach, R. Baker, and I. Martinovic. Wi-fly?: Detecting privacy invasion attacks by consumer drones. 2017.

[14] Bisio et al. Improving WiFi Statistical Fingerprint-Based Detection Techniques Against UAV Stealth Attacks. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2018.

[15] Bisio et al. Unauthorized Amateur UAV Detection Based on WiFi Statistical Fingerprint Analysis. *IEEE Communications Magazine*, 56(4):106–111, 2018.

[16] E. Blanton et al. A conservative loss recovery algorithm based on selective acknowledgment (sack) for tcp. Technical report, 2012.

[17] L. Bonati, S. D'Oro, S. Basagni, and T. Melodia. SCOPE: An Open and Softwarized Prototyping Platform for NextG Systems. In *Proc. of ACM Intl. Conf. on Mobile Systems, Applications, and Services (MobiSys)*, Virtual Conference, June 2021.

[18] L. Bonati et al. Colosseum: Large-Scale Wireless Experimentation Through Hardware-in-the-Loop Network Emulation. In *Proc. of IEEE Intl. Symp. on Dynamic Spectrum Access Networks (DySPAN)*, December 2021.

[19] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 416–432. Springer, 2003.

[20] Busset et al. Detection and Tracking of Drones using Advanced Acoustic Cameras. In *Unmanned/Unattended Sensors and Sensor Networks XI; and Advanced Free-Space Optical Communication Techniques and Applications*, volume 9647, page 96470F. International Society for Optics and Photonics, 2015.

[21] Y. Cao et al. Adversarial sensor attack on lidar-based perception in autonomous driving. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019.

[22] L. Casado and P. Tsigas. Contikisec: A secure network layer for wireless sensor networks under the contiki operating system. In *NordSec*. Springer, 2009.

[23] E. E. Case, A. M. Zelnio, and B. D. Rigling. Low-cost Acoustic Array for Small UAV Detection and Tracking. In *2008 IEEE National Aerospace and Electronics Conference*, pages 110–113. IEEE, 2008.

[24] M. Centenaro et al. System-level study of data duplication enhancements for 5g downlink urllc. *IEEE Access*, 8:565–578, 2019.

[25] S. Chandra et al. Hybrid buffer-based optical packet switch with negative acknowledgment for multilevel data centers. *Journal of Optical Communications*, 2020.

[26] M. CHAVERS. *It's Privacy, Stupid. FAA Drone Remote ID Rule Comments Open Tomorrow*, 2020 (accessed May 5, 2020). `https://www.newsledge.com/its-privacy-stupid-faa-drone-remote-id-rule/`.

[27] P. Chavez-Burbano et al. Optical camera communication system for three-dimensional indoor localization. *Optik*, 192:162870, 2019.

[28] C.-L. Chen, Y.-Y. Deng, W. Weng, C.-H. Chen, Y.-J. Chiu, and C.-M. Wu. A traceable and privacy-preserving authentication for uav communication control system. *Electronics*, 9(1):62, 2020.

[29] C.-Y. Chen and H.-Y. Hsieh. Does queue correlation matter in 5g multi-connectivity with packet duplication? *IEEE Wireless Communications Letters*, 2022.

[30] Y. Chen et al. Impact of correlated fading on multi-connectivity. *IEEE Transactions on Wireless Communications*, 20(2):1011–1022, 2020.

[31] Y. Cheng, X. Ji, T. Lu, and W. Xu. Dewicam: Detecting Hidden Wireless Cameras via Smartphones. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 1–13, 2018.

[32] M. Conti, G. Rigoni, and F. Toffalini. Asaint: a spy app identification system based on network traffic. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, pages 1–8, 2020.

[33] DJI. Dji aeroscope, 2021-05-21.

[34] J. D'Onfro. *Amazon's New Delivery Drone Will Start Shipping Packages 'In A Matter Of Months'*, 2019 (accessed May 5, 2020). `https://www.forbes.com/sites/jilliandonfro/2019/06/05/amazon-new-delivery-drone-remars-warehouse-robots-alexa-prediction/#3bce6a4145f3`.

[35] J. Elias et al. Multi-connectivity in 5g new radio: Optimal resource allocation for split bearer and data duplication. *Available at SSRN 4102694*.

[36] M. Ezuma, F. Erden, C. K. Anjinappa, O. Ozdemir, and I. Guvenc. Micro-UAV Detection and Classification from RF Fingerprints Using Machine Learning Techniques. In *2019 IEEE Aerospace Conference*, pages 1–13. IEEE, 2019.

[37] FAA. *Advisory and Rulemaking Committees - UAS Identification and Tracking ARC*, 2017 (accessed May 5, 2020). `https://www.faa.gov/regulations_policies/rulemaking/committees/documents/index.cfm/document/information/documentID/3302`.

[38] FAA. *Remote Identification of Unmanned Aircraft Systems*, 2020 (accessed May 5, 2020). `https://www.regulations.gov/docket?D=FAA-2019-1100`.

[39] FAA. *UAS Data Exchange (LAANC)*, 2020 (accessed May 5, 2020). `https://www.faa.gov/uas/programs_partnerships/data_exchange/`.

[40] FAA. *Unmanned Aircraft System Traffic Management (UTM)'*, 2020 (accessed May 5, 2020). `https://www.faa.gov/uas/research_development/traffic_management/`.

[41] FAA. *UAS Remote Identification*, 2023 (accessed March 5, 2023). `https://www.faa.gov/uas/getting_started/remote_id`.

[42] U. D. o. T. Federal Aviation Administration. Uas remote identification overview, 2021-03-09.

[43] M. D. Furtado et al. Threat analysis of the security credential management system for vehicular communications. In *2018 IEEE International Symposium on Technologies for Homeland Security (HST)*. IEEE, 2018.

[44] S. R. Ganti and Y. Kim. Implementation of Detection and Tracking Mechanism for Small UAS. In *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1254–1260. IEEE, 2016.

[45] M. T. Garip et al. Interloc: An interference-aware rssi-based localization and sybil attack detection mechanism for vehicular ad hoc networks. In *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–6. IEEE, 2017.

[46] J. Hegranes. *FAA's proposed remote ID rules should make compliance easy*, 2020 (accessed May 5, 2020). `https://techcrunch.com/2020/02/12/faas-proposed-remote-id-rules-should-make-compliance-easy/`.

[47] S. E. Huang et al. Impact evaluation of falsified data attacks on connected vehicle based traffic signal control. *arXiv preprint arXiv:2010.04753*, 2020.

[48] X. Huang et al. Exposing spoofing attack on flocking-based unmanned aerial vehicle cluster: A threat to swarm intelligence. *Security and Communication Networks*, 2020, 2020.

[49] InterUSS. *The InterUSS Project enables trusted, secure and scalable interoperability between UAS Service Suppliers (USSs) to further safe, equitable and efficient drone operations.*, 2020 (accessed July 5, 2020). `https://interussplatform.org/`.

[50] I. Kershner. Israel builds a laser weapon to zap threats out of the sky, 2022-06-03.

[51] B. Kim and J. Lee. Retransmission loss recovery by duplicate acknowledgment counting. *IEEE Communications Letters*, 8(1):69–71, 2004.

[52] J.-E. Kim et al. Color-space-based visual-mimo for v2x communication. *sensors*, 16(4):591, 2016.

[53] Kittyhawk. *Remote ID & Commercial Drones*, 2020 (accessed July 5, 2020). `https://kittyhawk.io/remote-id/`.

[54] L. Kolodny. *Zipline, which delivers lifesaving medical supplies by drone, now valued at $1.2 billion*, 2019 (accessed May 5, 2020). `https://cnb.cx/2EKnGHa`.

[55] N. T. Le et al. A survey of design and implementation for optical camera communication. *Signal Processing: Image Communication*, 53:95–109, 2017.

[56] Li et al. Drone Profiling through Wireless Fingerprinting. In *2017 IEEE 7th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 858–863. IEEE, 2017.

[57] T. Liu, Z. Liu, J. Huang, R. Tan, and Z. Tan. Detecting Wireless Spy Cameras via Stimulating and Probing. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 243–255, 2018.

[58] S. Ma et al. Foglight: Visible light-enabled indoor localization system for low-power iot devices. *IEEE Internet of Things Journal*, 5(1):175–185, 2017.

[59] M. McNabb. *Chris Korody Speaks Out on Remote ID for Drones: "A Billion Dollar Solution to a Non-Existent Problem"*, 2020 (accessed May 5, 2020). `https://dronelife.com/2020/02/26/chris-karody-speaks-out-on-remote-id-for-drones-a-billion-dollar-solution-to-a-non-existent-problem/`.

[60] Meduim. *Wing Launches America's First Commercial Drone Delivery Service to Homes in Christiansburg, Virginia*, 2019 (accessed May 5, 2020). `https://bit.ly/3n2wGJ5`.

[61] Nassi et al. Drones' Cryptanalysis-Smashing Cryptography with a Flicker. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1397–1414. IEEE, 2019.

[62] T. Nguyen et al. High-speed asynchronous optical camera communication using led and rolling shutter camera. In *2015 Seventh International Conference on Ubiquitous and Future Networks*. IEEE, 2015.

[63] P. Nguyen et al. Matthan: Drone Presence Detection by Identifying Physical Signatures in the Drone's RF Communication. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, pages 211–224, 2017.

[64] OpenDroneID. *Welcome to opendroneid.org*, 2020 (accessed July 5, 2020). `https://www.opendroneid.org/`.

[65] P. H. Pathak et al. Visible light communication, networking, and sensing: A survey, potential and challenges. *IEEE communications surveys & tutorials*, 17(4):2047–2077, 2015.

[66] J. Rao and S. Vrzic. Packet duplication for urllc in 5g: Architectural enhancements and performance analysis. *IEEE Network*, 32(2):32–40, 2018.

[67] A. Rozantsev, V. Lepetit, and P. Fua. Flying Objects Detection From a Single Moving Camera. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4128–4136, 2015.

[68] C. Ruiz et al. Idrone: Robust drone identification through motion actuation feedback. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(2):1–22, 2018.

[69] Sciancalepore et al. Detecting Drones Status via Encrypted Traffic Analysis. In *Proceedings of the ACM Workshop on Wireless Security and Machine Learning*, pages 67–72, 2019.

[70] Sciancalepore et al. Picking a Needle in a Haystack: Detecting Drones via Network Traffic Analysis. *arXiv preprint arXiv:1901.03535*, 2019.

[71] D. Segura et al. Dynamic packet duplication for industrial urllc. *Sensors*, 22(2):587, 2022.

[72] G. Singh, H. Fulara, D. Jaisinghani, M. Maity, T. Chakraborty, and V. Naik. CRAWDAD dataset iiitd/wifiactivescanning (v. 2019-06-05). Downloaded from `https://crawdad.org/iiitd/wifiactivescanning/20190605`, Mar. 2022.

[73] B. Sinopoli et al. Optimal linear lqg control over lossy networks without packet acknowledgment. *Asian Journal of Control*, 10(1):3–13, 2008.

[74] M. Sorgente. *Chia-Network/bls-signatures*, 2020 (accessed July 5, 2020). `https://github.com/Chia-Network/bls-signatures/tree/master/python-bindings`.

[75] SpotterRF. Drone detection system, 2020-03-13.

[76] R. Staff. *Airobotics Creates In-House Drone Payload for Inspection, Safety Uses*, 2019 (accessed May 5, 2020). `https://bit.ly/3i6yvAU`.

[77] J. Sun, W. Wang, L. Kou, Y. Lin, L. Zhang, Q. Da, and L. Chen. A data authentication scheme for uav ad hoc network communication. *The Journal of Supercomputing*, pages 1–16, 2017.

[78] R. R. Systems. Robin radar systems, 2021-05-21.

[79] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic. Robust Smartphone App Identification via Encrypted Network Traffic Analysis. *IEEE Transactions on Information Forensics and Security*, 13(1):63–78, 2018.

[80] Y. Tian, J. Yuan, and H. Song. Efficient privacy-preserving authentication framework for edge-assisted internet of drones. *Journal of Information Security and Applications*, 48:102354, 2019.

[81] USDOT. *Security Credential Management System (SCMS)*, 2021 (accessed January 5, 2020). `https://www.its.dot.gov/resources/scms.htm`.

[82] J. Valente and A. A. Cardenas. Understanding Security Threats in Consumer Drones through the Lens of the Discovery Quadcopter Family. In *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy*, pages 31–36, 2017.

[83] K. Wagstaff. Tokyo police to deploy net-carrying drone to catch rogue drones, 2015-12-11.

[84] B. Warner. *python-ecdsa*, 2020 (accessed July 5, 2020). `https://github.com/warner/python-ecdsa`.

[85] M. Wazid, A. K. Das, N. Kumar, A. V. Vasilakos, and J. J. Rodrigues. Design and analysis of secure lightweight remote user authentication and key agreement scheme in internet of drones deployment. *IEEE Internet of Things Journal*, 6(2):3572–3584, 2018.

[86] Wing. *Wing delivery is easy to use*, 2020 (accessed July 5, 2020). `https://wing.com/how-it-works/`.

[87] W. Yuan et al. Computer vision methods for visual mimo optical system. In *CVPR 2011 workshops*, pages 37–43. IEEE, 2011.