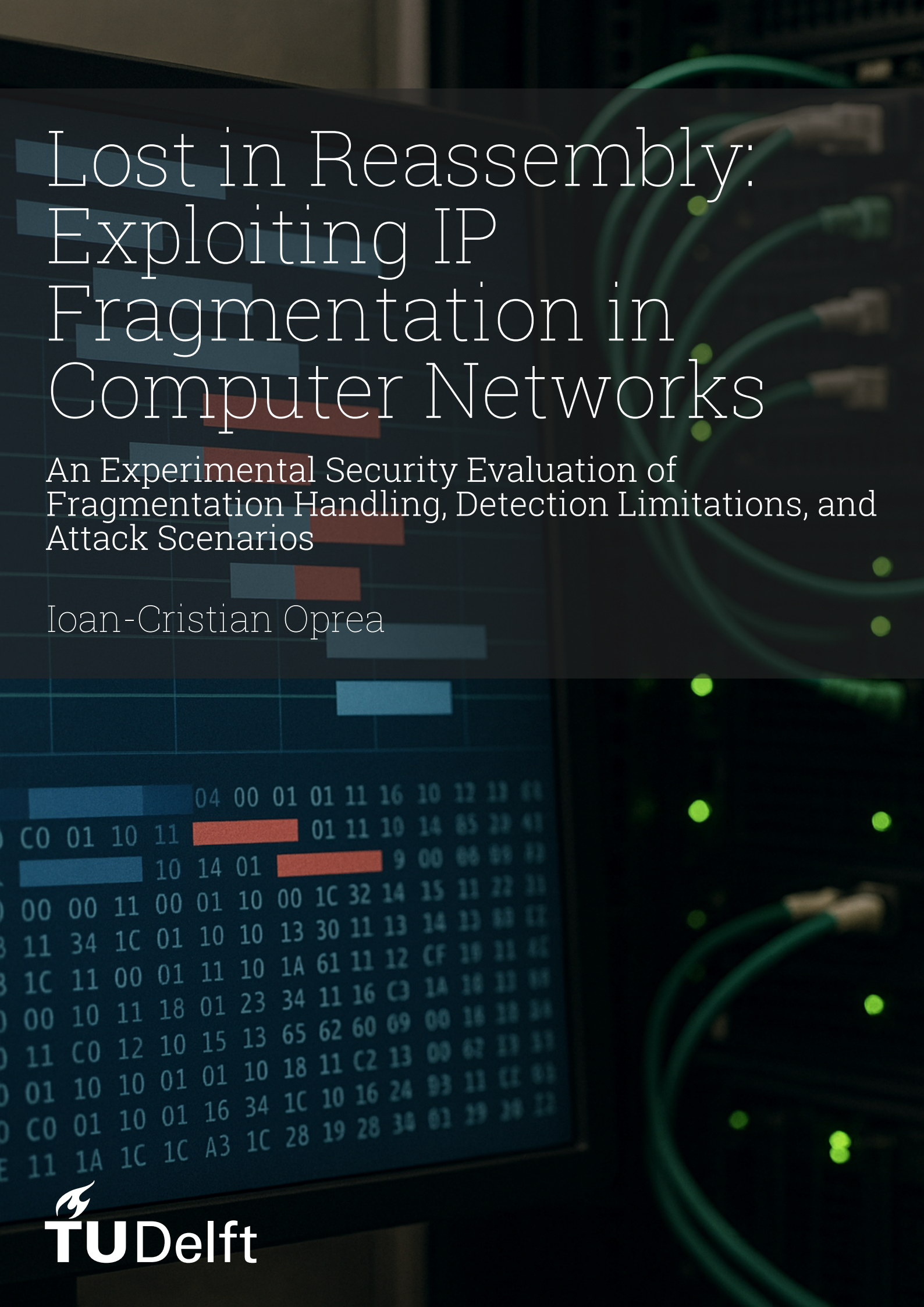

Lost in Reassembly: Exploiting IP Fragmentation in Computer Networks

An Experimental Security Evaluation of Fragmentation Handling, Detection Limitations, and Attack Scenarios

Ioan-Cristian Oprea

TU Delft

Lost in Reassembly: Exploiting IP Fragmentation in Computer Networks

An Experimental Security Evaluation of Fragmentation Handling, Detection Limitations, and Attack Scenarios

by

Ioan-Cristian Oprea

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Friday June 13, 2025 at 15:00.

Student number:	5214823	
Project duration:	October 14, 2024 – June 13, 2025	
Thesis committee:	Prof. dr. ir. G. Smaragdakis,	TU Delft, supervisor
	Dr. E. Bassetti,	European Space Agency - TU Delft, co-supervisor
	Dr. H. Griffioen,	TU Delft
	Dr. N. Mohan ,	TU Delft

TU Delft

Preface

I would like to express my profound appreciation to my supervisor, Prof. dr. ir. G. Smaragdakis, for his insightful guidance, constructive feedback, and constant encouragement throughout this research. His expertise and support were instrumental in shaping both the direction and quality of this work. I am equally grateful to my co-supervisor, Dr. E. Bassetti, for his valuable advice and critical review of my experiments and for sharing his extensive knowledge of network security, which greatly enriched the thesis.

I owe my deepest gratitude to my family, whose unwavering love and support have carried me through every challenge. To my father, Florian Oprea, my mother, Iuliana Oprea, and my brother, Constantin-Danut Oprea: thank you for standing by me not only during the writing of this master's thesis but throughout my life. Your encouragement has been my foundation.

I am also profoundly grateful to my partner, Ariadna Zara, for patiently listening to my late-night reflections and standing beside me through every frustration and breakthrough. Finally, I would like to acknowledge my friends: Codrin Socol, Alexandru Dumitriu, David Peta, Razvan Popescu, Andru Turcu, Ciprian Stanciu, Matei Galesanu, Sebastian Manda, and Teodor Oprescu, for their camaraderie and invaluable help during both my bachelor's and master's studies. Your friendship and support have meant the world to me.

Ioan-Cristian Oprea
Delft, June 2025

Abstract

IPv6 fragmentation remains a subtle yet impactful security concern in modern high-throughput, low-latency networks, where packet inspection is constrained by performance requirements and out-of-path monitoring architectures. This thesis investigates how discrepancies in IPv6 fragment reassembly behaviour between passive Intrusion Detection Systems (IDS) and endpoint hosts can lead to detection gaps, misinterpretation, and even exploitable evasion opportunities. Using a permutation-based testing model inspired by prior work, the study evaluates 720 overlapping fragment sequences across multiple operating systems and analyses the detection behaviour of Suricata deployed inside a 5G-simulated User Plane Function. The results reveal inconsistencies in reassembly policies, particularly under retransmission conditions, and demonstrate that alerts are not always semantically aligned with the payload seen by the host. A proof-of-concept exfiltration attack further illustrates how timing-based fragment delivery can bypass IDS inspection while reconstructing sensitive data at an attacker-controlled receiver. To mitigate these risks, the thesis proposes temporal IP ID tracking, overlap enforcement, and targeted traffic normalisation, especially in programmable or inline network functions. These findings highlight fragmentation as a persistent and under-addressed attack surface and call for more context-aware, timing-resilient detection strategies in next-generation networks.

Contents

Abbreviations

Abbreviation	Definition
5G	Fifth Generation Mobile Network
AF_PACKET	Linux kernel interface for high-performance packet capture
C2	Command and Control (attacker-controlled endpoint used for exfiltration or coordination)
DPI	Deep Packet Inspection (technique for analyzing packet payloads beyond headers)
EICAR	European Institute for Computer Antivirus Research (a standard non-malicious test string used to validate antivirus and IDS detection)
GTP-U	GPRS Tunneling Protocol - User Plane (carries user data within 5G and LTE networks)
ICMPv6	Internet Control Message Protocol for IPv6 (used for network diagnostics)
IDS	Intrusion Detection System (passive security system that monitors and alerts on traffic)
IPS	Intrusion Prevention System (active security system that can modify or block traffic)
IP ID	IP Identification field used to associate fragments with the same datagram
IPv6	Internet Protocol version 6
M=0	Fragmentation flag indicating the last fragment in an IPv6 packet
NFV	Network Function Virtualization (concept of implementing network functions in software)
OS	Operating System
PCAP	Packet Capture file format (used to store captured network traffic)
RFC	Request for Comments (formal documents describing Internet protocols and standards)
SDN	Software Defined Networking (approach that separates control and data planes in networks)
SMF	Session Management Function (a 5G core component handling session context)
TCP	Transmission Control Protocol (a reliable, connection-oriented transport protocol)
UDP	User Datagram Protocol (a lightweight, connectionless transport protocol)
UPF	User Plane Function (a 5G core network component)
UE	User Equipment (client device in 5G networks)
XDP	eXpress Data Path (a high-performance packet processing framework in the Linux kernel)

1

Introduction

1.1. Context and Motivation

IP fragmentation is a long-established method designed to facilitate the transmission of packets across networks with different Maximum Transmission Unit (MTU) sizes [1]. Despite improvements in protocol design between IPv4 and IPv6, fragmentation remains a cause of complexity and inconsistency [2, 3, 4], especially when traffic flows through several devices that must reconstruct the original datagram. In IPv6, fragment reassembly is generally performed at the receiving endpoint [1]. However, in security-sensitive environments, middleboxes such as firewalls or IDS/IPS systems may try to reassemble or inspect fragments mid-path. Because the RFCs leave the IP reassembly process underspecified and many implementers diverge from them, subtle discrepancies arise that adversaries can exploit to evade detection or inject malicious payloads.

In high-throughput, low-latency settings, such as 5G networks, where performance constraints limit the possibility of deep inspection or full user-space reassembly, these risks become increasingly important. To deal with increasing traffic rates and stricter latency requirements, current detection systems often delegate packet processing tasks to kernel-level functions [5, 6, 7] or rely on simplified processing pipelines. While these design decisions improve scalability, they may also introduce blind spots, particularly when fragmentation behaviour varies across operating systems, middleboxes and detection engines [4].

This thesis investigates fragmentation handling in such performance-constrained environments, focusing on the degree to which reassembly discrepancies can be exploited to avoid detection. While the findings have broad applicability, the evaluation is done in a representative environment: a simulated 5G network, whose constraints match the high-performance characteristics examined in this thesis.

The study focuses on scenarios in which an out-of-path Intrusion Detection System (IDS) relying on kernel-level IP defragmentation monitors fragmented traffic. This configuration is representative of high-performance networks where reducing system overhead is critical. The thesis explores whether discrepancies between kernel reassembly and endpoint behaviour can lead to attacks that remain undetected by passive monitoring tools, even when the host accepts the traffic.

1.2. Research Problem and Scope

Despite the theoretical security improvements of IPv6 over IPv4, real-world implementations still differ significantly in how fragmented packets are handled and reassembled. For example, even though RFC 5722 [8] mandates that hosts must silently drop packets with overlapping fragments, empirical studies show that most operating systems reconstruct these fragments instead [3], creating undesired behaviour that can be exploited by attackers. Each operating system also has its own reassembly policy, which could differ from that of middleboxes or security systems. Adversaries can use this lack of consistency to bypass security measures or fingerprint systems based on their response to fragmented traffic.

These inconsistencies are even more important in high-throughput, latency-sensitive environments such as 5G networks, where performance constraints make in-line security solutions like Intrusion Prevention Systems (IPS) impractical, unless correctly configured. Inline packet inspection can introduce processing delays and can become a bottleneck under high traffic levels, making it unsuitable for satisfying strict latency (e.g., sub-millisecond) and availability requirements. Academic and industry research has shown that IPS solutions risk adding unacceptable delay, particularly when deep inspection or reassembly is involved, and require extensive tuning or hardware acceleration to keep up with high traffic rates [9, 10]. As a result, many production deployments instead use out-of-path IDS that passively monitor traffic, which allows for non-intrusive, scalable traffic visibility without affecting live traffic flows [11].

To further reduce system overhead and match detection behaviour with that of the host OS, Suricata and similar IDS tools are usually run in AF_PACKET mode [12], configured with *defrag: yes* (default settings) to delegate IP fragment reassembly to the kernel. Since the IDS sees exactly what the kernel reconstructs and avoids user-space reassembly, this approach improves detection consistency as well as efficiency. This configuration is recommended in high-performance environments and has become a standard practice [5, 6, 7].

While this approach offers scalability, it also creates new attack surfaces: if an adversary can predict or control the kernel's reassembly behaviour, they may craft fragmented packets that evade detection by the IDS while being interpreted differently at the target host.

1.3. Research Questions

This thesis explores how variations in IP fragmentation handling affect the security of high-performance networks. It focuses on on the risks that arise when reassembly varies between systems in a passive monitoring setup.

The study is guided by the following main research question:

Main Question

How do inconsistencies in IPv6 fragmentation handling impact the security of computer networks, and how can these vulnerabilities be exploited or mitigated in a high-throughput environment (5G) using out-of-path Intrusion Detection Systems?

To answer this question, the following subquestions are explored:

1. What inconsistencies or vulnerabilities arise from IPv6 fragmentation and reassembly behaviour in high-throughput, low-latency networks?
2. How do different 5G network components, including User Equipment (UE), and the 5G External Data Network, handle fragmented IPv6 packets, and how does this affect packet reassembly behaviour?
3. What are the detection limitations introduced by deploying an out-of-path IDS (e.g., Suricata) that relies on kernel-level reassembly in high-throughput, low-latency environments?
4. Can fragmentation-based inconsistencies be exploited in practice for evasion, insertion, or exfiltration attacks, and under what conditions do they remain undetected by the IDS?
5. What configuration or architectural improvements can be made to reduce the risks posed by fragmentation-based attacks in high-throughput, low-latency networks?

1.4. Contributions

This thesis presents a structured and in-depth analysis of how IPv6 fragmentation affects detection visibility in high-performance networks. It presents a testing method based on permutation models of overlapping fragments. This method is used to examine how both the host and the IDS reassemble data in controlled and retransmission scenarios. Unlike previous research that focused only on endpoint acceptance, this work examines reassembly outcomes between the IDS and several operating systems, showing reassembly divergence.

An important contribution is the development and demonstration of a proof-of-concept exfiltration attack. This attack uses timing-based fragment delivery to bypass detection. It does so by taking advantage of the IDS's early reassembly completion and the attacker's ability to influence fragment reordering at the receiving endpoint. The thesis suggests an IP ID lifespan tracking system to make sure that reused fragment identifiers are no longer valid after reassembly. This lowers the possibility of late-arriving fragment insertion.

The thesis also contributes with an updated security assessment of passive detection systems in high-performance networks, highlighting the differences between alert presence and payload interpretation. It showcases useful insights into how IDS positioning, reassembly logic, and detection timing affect security in modern networks by testing how different operating systems handle fragments and mimicking high-throughput detection conditions.

1.5. Structure of the Thesis

The remainder of this thesis is structured as follows. Chapter 2 introduces the necessary background on IP fragmentation, reassembly policies, and the impact of middleboxes and 5G network architecture. Chapter 3 reviews related work on IDS evasion techniques, fragmentation-based attacks, and reassembly behaviour across systems. Chapter 4 describes the experimental methodology, including the design of the fragment permutation model and the classification criteria used in the analysis. Chapter 5 presents the experimental results, covering reassembly consistency, detection gaps, exploitability analysis, and a proof-of-concept exfiltration scenario. Chapter 6 proposes mitigation strategies, outlines the limitations of the study, and offers practical recommendations. Chapter 7 concludes the thesis with a summary of the findings and discusses directions for future work.

2

Background

This chapter provides the basic knowledge required to understand the problem of fragmentation-based attacks in the context of high-throughput, low-latency networks like 5G. It presents key networking concepts, analyses the evolution of packet fragmentation, and investigates the role of security devices and network functions in reassembly.

2.1. Internet Protocol Fragmentation

All current network communication is centred around the Internet Protocol (IP). It is a connectionless, stateless protocol that enables the transmission of packets between endpoints, abstracting away the underlying physical links and topologies. As the most used protocol in Layer 3 of the OSI model, IP enables heterogeneous networks to function as a unified internetwork by providing routing and addressing capabilities [13].

Modern networks use both IPv4 and IPv6, often at the same time. Though the two protocols differ in structure, such as in address size and header format, they both have the same basic duties: routing datagrams and managing packet lifetimes. However, an important difference is their approach to fragmentation.

Packet fragmentation is a core mechanism within the Internet Protocol that allows large IP datagrams to travel across networks with lower Maximum Transmission Units (MTUs). When a packet exceeds the MTU of a network segment, it must be split into smaller fragments to guarantee delivery. Each fragment is treated as a separate IP packet and contains metadata that indicates its position in the original datagram and whether more fragments are expected. Fragments are then reassembled into the original payload only at the destination host [14].

Fragmentation exists to ensure end-to-end delivery when different segments of the network impose varying MTUs, but IPv4 and IPv6 handle it differently. In IPv4, both the source and intermediate routers along the path can split datagrams, using header fields like Identification, Fragment Offset and the "More Fragments" flag to help reconstruct the original datagram [14]. This flexibility, however, burdens the core network with extra processing complexity [15], complicates error recovery [16] and opens the door to denial-of-service [17] and evasion exploits [18] when maliciously manipulating fragments.

On the other hand, IPv6 eliminates fragmentation at intermediate routers. Using a separate Fragment Extension header, IPv6 shifts the responsibility for fragmentation entirely to the sender. This header contains fields similar to those in IPv4 but is an optional metadata attached to the packet as opposed to being part of the base header. When routers come across oversized IPv6 packets, they must instead send an ICMPv6 "Packet Too Big" message back to the source, requiring the sender to adjust its packet size via Path MTU Discovery (PMTUD) [1].

The architecture's emphasis on service-based design and IP-layer tunnelling makes IP a central component in 5G networks. All user data in the 5G User Plane is encapsulated in GTP-U (GPRS Tunnelling

Protocol for User Data) packets that carry an inner IP payload. These IP packets may be further encapsulated or fragmented depending on MTU constraints between User Equipment (UE), gNB (next generation NodeB) and the 5G Core Network (5GC). However, 5G architecture will be presented in a subsequent section.

Fragmentation in 5G networks can take place at more than one point in the network. The IP payload encapsulated in a GTP-U tunnel may be fragmented at the inner IP layer, at the outer transport layer, or both [19]. Additionally, fragmentation could present poor interactions with tunnelling and security systems. For example, packets can be silently discarded at in-path devices not supporting GTP-U fragmentation, or it can bypass inspection when an IDS is not configured to reassemble fragments with the proper reassembly context (e.g., across tunnel boundaries) [20].

2.2. Security Concerns in IP Fragmentation

Even though packet fragmentation is necessary for end-to-end connectivity over heterogeneous networks, it presents several security issues. These are not only due to the increased complexity of reassembly but also due to ambiguities in implementation, lack of strict enforcement of protocol standards, and inconsistencies in how fragments are interpreted across different systems. In the past, attackers have exploited these weaknesses to create evasion methods that reduce intrusion detection and firewall policy enforcement [21, 22].

Fragment-Based Evasion and Insertion Attacks

The basis of most fragmentation-related exploits is the idea of interpretation disparity, where different network hosts reconstruct the packet differently because of inconsistencies in fragment handling logic. This leads to two methodologies [23].

Evasion is where the attacker constructs a series of fragments that the target host reconstructs into a complete, valid payload while the middlebox does not reassemble them or reconstructs a benign version. This allows malicious content to evade detection. For example, a signature may be divided over two fragments the middlebox can't match individually, but the host reconstructs it into an intact attack payload.

Insertion involves sending packets that are accepted and processed by the middlebox but discarded by the target system. This attack can be used to desynchronise the middlebox with the target system, tricking the middlebox into thinking that it has effectively monitored all traffic, while in reality, the malicious payload is hidden from the middlebox. This may involve exploiting variances in the middlebox or target system's protocol implementations or even different network stack behaviours.

Overlapping and Out-of-Order Fragments

When receiving fragments out of order, the order in which they are reassembled is important. The original IP specification does not indicate a common behaviour in cases of overlapping fragments (i.e., when two fragments overlap in the byte range but have different data).

RFC 5722 [8] clarifies that any packet containing overlapping fragments should silently be discarded to prevent ambiguity in reassembly. In reality, however, many systems ignore this requirement, do not comply [3], and apply heuristics such as "first fragment wins" or "last fragment wins" to deal with overlaps. These differences introduce a dangerous inconsistency: by carefully timing and crafting overlapping fragments, an attacker can manipulate which data the IDS inspects versus what the end host actually reconstructs.

Fragmentation Timeout and Reassembly Policy Discrepancies

Reassembling IP fragments requires keeping track of each flow's fragments over time, so both hosts and security systems must maintain per-flow state. That state is bounded by timeouts, finite buffer space and protocol-specific heuristics [24]. For example:

- **Timeout policies** specify how long a system should wait for missing fragments before they get discarded. If the timeout is short, the system might never process delayed fragments. If it is long, the system is more vulnerable to resource exhaustion.

- **The maximum number of fragments per packet** or **buffer size per reassembly context** is implementation-based as well. Exhausting these resources can lead to fragmentation-based Denial-of-Service attacks.

When different systems apply different policies, there are inconsistencies. For example, an IDS might time out and drop the reassembly state before the last fragment is received, but the end host still accepts it. Or a security system might enforce strict limits and drop the reassembly buffer, while the host with more relaxed settings completes the reconstruction [25].

Performance-Driven Shortcuts in Detection Systems

In high-performance networks, like 5G networks, line-rate requirements tend to result in simplifications of how fragments are processed within security systems. IDS software can disable reassembly completely for certain traffic types [26] or use kernel-level defragmentation rather than performing user-space reassembly [5, 6, 7]. This will be explored further in Section 2.3.

While these optimisations improve throughput and reduce CPU overhead, they also open the door to attackers who know how to manipulate fragment structure to remain undetected. When traffic is mirrored to an out-of-path IDS that uses kernel defragmentation, the system inherits all the quirks and limitations of the kernel's IP stack, including its fragment overlap policy, timeout behaviour, and handling of partial reassembly.

2.3. Middleboxes

In computer networks, middleboxes are intermediary devices that perform more than packet forwarding. While routers and switches focus on routing traffic and handling link-layer delivery, middleboxes enforce policies, inspect and filter data flows, and perform tasks like address translation. Common examples include: Network Address Translators (NATs), load balancers, and Intrusion Detection or Prevention Systems (IDS/IPS). In this section we will focus on the IDS/IPS category.

Middlebox Reassembly

When it comes to IP fragmentation, middleboxes play a crucial role. Since many of their security and performance features run at or above the transport layer, they need to be able to correctly decode and reconstruct fragmented packets before inspecting and enforcing any policies [24]. This creates a reliance on fragment reassembly logic and, with it, a wide attack surface when reassembly behaviour is inconsistent, simplified, or different than the one used in the endpoint device.

Unlike endpoints that typically depend on a specific operating system as well as a protocol stack to perform the reassembly of packets, middleboxes are usually implemented with a custom, proprietary logic due to performance or architectural reasons [2]. This means that the way fragments are reassembled can be different between multiple vendors as well as classes of devices.

Reassembling IP fragments at line rate is a resource-intensive task: each fragment must be matched to its reassembly context, stored in memory until the entire datagram arrives or a timeout occurs, and then checked for overlaps or malformed data before being reordered and reassembled into a complete packet.

This process can be weaponised in several ways. An attacker might launch a fragmentation-based denial-of-service by flooding the middlebox with incomplete or delayed fragments, exhausting its buffers and triggering connection drops or reassembly failures. They might also induce a state desynchronisation by crafting fragments that the middlebox accepts but the end host rejects (or vice versa), causing the IDS to see a different payload than the target system. Finally, adversaries can bypass security policies by exploiting offset methods or creating overlapping, ambiguous fragments that evade standard reassembly logic.

Middlebox Deployment Models

Another important aspect is the middlebox deployment model. **Inline devices** (e.g., IPS, firewall) sit directly in the path of the traffic and can drop, modify, or redirect packets. **Out-of-path devices** (e.g., IDS) receive a copy of the traffic and monitor without affecting forwarding.

In today's high-speed, low-latency networks, using in-line security tools like IPSes is not feasible unless they're carefully tuned. These systems inspect traffic as it flows, which becomes slow when they need to perform deep inspection or reassemble packets. That kind of delay is not accepted in such performance-heavy networks. Industry and academic research suggest that IPS solutions often struggle to keep up without dedicated hardware or a lot of fine-tuning [9, 10]. Because of this, many real-world setups use out-of-path IDS instead. These monitor traffic passively, usually through port mirroring or network taps, giving analysts visibility into the network without disrupting the traffic flows [11].

To further handle high traffic loads efficiently, many deployments configure IDS solutions like Suricata to operate in AF_PACKET mode [12], using the default *defrag: yes* setting to shift IP fragment reassembly to the kernel. By allowing the operating system to manage reassembly, the IDS avoids extra processing overhead in user space and evaluates packets exactly as the host sees them. This setup leads to faster performance and more reliable detection and has become a widely adopted strategy in performance-critical environments [5, 6, 7].

2.4. 5G Network Architecture

In this thesis, 5G serves as our evaluation environment, offering a timely case study for how IP fragmentation behaves in today's high-performance networks. Its modular design, encapsulated data plane and strict latency requirements introduce challenges for fragment reassembly and continuous security monitoring. In the following section, we'll describe 5G's core architecture, the flow of user traffic through the system and discuss how these characteristics influence the placement and effectiveness of intrusion-detection mechanisms.

2.4.1. Overview of 5G Network Functions

According to [27], fifth-generation (5G) networks adopt a modular, service-based architecture (SBA) that withdraws from the vertically integrated, monolithic designs used in previous generations. In this architecture, each logical element of the networks, known as a Network Function (NF), performs a distinct role and interacts with others via defined interfaces, often implemented as RESTful APIs. This division of tasks allows operators to deploy NFs as containerised or virtualised workloads across cloud, edge and on-premise environments, improving scalability and flexibility.

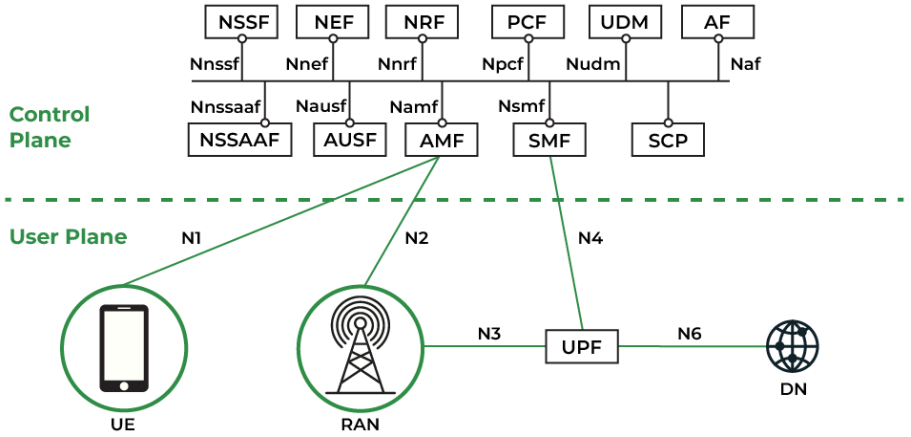

Figure 2.1: Figure illustrating the 5G network architecture [28]

The 5G Core (5GC) is built around a split between the **control plane** and the **user plane**. The control plane handles signalling, session establishment and subscriber management, while the user plane is responsible for forwarding the actual user data. Such control plane functions are:

- The **Access and Mobility Management Function (AMF)**, which manages User Equipment (e.g., mobile phones) registrations and mobility events
- The **Session Management Function (SMF)**, responsible for allocating IP address and configuring routing paths

- The **Authentication Server Function (AUSF)**, which authenticates users via credentials or certificates

The primary component in the user plane is the **User Plane Function (UPF)**. Acting as the gateway between the Radio Access Network (RAN) and external data networks, the UPF routes packets based on policies set by the control plane. As a performance-critical element, it must handle traffic at full line rate, enforce quality-of-service rules, and perform GTP-U encapsulation and decapsulation, all while avoiding any processing bottlenecks

The decoupling of the 5G architecture brings with it performance benefits, scaling capabilities, and security isolation. However, it also means that packet processing is distributed across heterogeneous platforms. Each function, including the UPF and surrounding network elements, may run different operating systems, kernel versions and processing pipelines. This heterogeneity makes consistent packet interpretation (especially for fragmented traffic) a non-trivial problem and forms the basis for this thesis' investigation.

2.4.2. Data Path and Encapsulation

In a standard 5G implementation, traffic from a mobile device (i.e., **UE**), is routed through multiple components before reaching external networks. At the radio level, the UE connects to the **gNodeB**(the RAN in Figure 2.1) , which serves as the access point to the core network. Data leaving the gNodeB is encapsulated in a **GPRS Tunnelling Protocol – User Plane (GTP-U)** tunnel. This encapsulation allows the network to separate each user's traffic, implement mobility management, and enforce isolation policies [19].

In a GTP-U tunnel, each IP packet is wrapped in a GTP-U header plus a UDP/IP outer header and sent through the core network to the UPF. There, it decapsulates the tunnel, retrieves the original IP packet and forwards it to the appropriate destination (e.g., a public website or service provider). When data returns to the user, the UPF re-encapsulates the packet in a new GTP-U tunnel, which carries it back through the core to the gNodeB and ultimately to the UE.

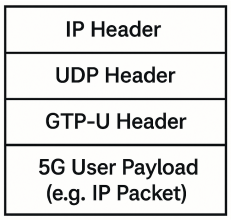

Figure 2.2: Figure illustrating the high-level packet structure of GTP-U

Encapsulating user traffic in GTP-U tunnels has multiple advantages: it makes it possible to apply per-user traffic control policies [19], enables mobility without disrupting end-to-end IP sessions [29], and allows multiple services or network slices to share the same network infrastructure [27]. However, encapsulation complicates deep packet inspection since any monitoring system must remove the outer headers to inspect the actual user payload. This becomes more important when the inner IP packets are fragmented, because each fragment is wrapped separately [30].

2.4.3. Performance Constraints in 5G

One of the main characteristics of 5G networks is their support for diverse use cases with strict performance requirements [31, 32]. For example, **Enhanced Mobile Broadband (eMBB)** aims for high data throughput (e.g., multi-gigabit connections), while **Ultra-Reliable Low-Latency Communication (URLLC)** applications require end-to-end latency on the order of **1 millisecond**. Additionally, **Massive Machine-Type Communications (mMTC)** use cases need high scalability for sensor-rich environments such as smart cities or industrial automation.

Meeting these requirements puts considerable stress on the data plane, where every microsecond counts. Processing packets through deep inspection layers, such as application-layer firewalls or traditional IPS, is not feasible when traffic is high [9, 10]. Any extra per-packet processing is likely to cause

delays that break SLAs and negatively affect the user experience.

To deal with this, 5G deployments increasingly depend on performance-optimised processing pipelines. These may include:

- **Kernel bypass technologies** (e.g. DPDK, XDP) to accelerate packet I/O [6, 7]
- **Offloading functions to SmartNICs or FPGAs** to reduce CPU load [33]
- **Delegating IP defragmentation to the kernel**, rather than performing it in user space [5]

2.4.4. Security Monitoring in 5G

To secure 5G networks, we need monitoring tools that accommodate the modular design and high-performance requirements [34]. In 5G, data is no longer routed through a single, central point instead, it is split into modules and encapsulated in multiple layers. Therefore, traditional in-line inspection tools are not suitable anymore. This means we need to adopt more adaptable and scalable intrusion-detection methods that can handle 5G's service-based architecture and massive data rates.

In a typical 5G system, we can deploy middleboxes at multiple network points. One common approach is to place the middlebox near the gNodeB, the base station, where it can inspect traffic as it enters the radio access network. This edge-based placement is especially useful in decentralised or mesh-style networks, where having threat detection close to the user is key [35, 36].

Middleboxes can also be deployed between the gNodeB and the core network, for example, on the N3 interface [37], or between the SMF and the UPF on the N4 interface [38]. This allows for partial inspection of encapsulated traffic, such as the GTP-U tunnel, but they still don't inspect the reassembled content of the final payload. Other designs go a step further by integrating the middlebox directly into core functions like the AUSF [39] or by running it as a separate network function that gathers and analyses aggregated traffic [40].

In this thesis, we focus on the placement of the middlebox directly inside the UPF, at the point where the user plane is decapsulated and forwarded to the external networks. This location brings three key benefits. First, it gives useful visibility into both uplink and downlink flows since the UPF handles all user data through the mobile core. Second, by being placed after GTP-U decapsulation, the IDS receives raw IP packets and avoids the need to remove tunnel headers or perform recursive parsing. And third, because the UPF sits at the border between the core network and external domains, it is the perfect spot for detecting attacks coming from either side.

Several academic and industry sources support the feasibility and effectiveness of UPF-integrated detection. For example, research has suggested placing IDS systems directly within or alongside the UPF [41], while commercial solutions such as Cisco's DPI-enabled UPF modules offer equivalent capabilities [42]. These designs are driven by the same trade-offs examined in this thesis: maximising packet visibility and detection accuracy without compromising processing efficiency.

Given the goal of examining how fragmentation impacts detection under high-throughput conditions, embedding the IDS in the UPF is both realistic and technically sound. In this position, it captures fragmented IP traffic immediately after GTP-U decapsulation, allows integration with kernel-level reassembly methods and reflects deployment patterns used in operational 5G core networks.

3

Related Work

IP fragmentation has long been recognised as a source of ambiguity in network security, especially when fragment reassembly behaviour is different across endpoints, middleboxes, and detection systems. These inconsistencies can be exploited by attackers to evade detection, mislead security tools, or bypass enforcement policies entirely. In this review, we'll look at both foundational and recent research on fragmentation-based evasion, methods for testing reassembly logic, and the specific challenges fragmentation poses for passive detection in high-performance environments such as 5G. A summary of the related work explored in this chapter can be viewed in Table 3.1.

Reference	Focus Area	Key Contribution	Relation to Thesis	Summary of Findings Related to Thesis
Ptacek & Newsham (1998)	Fragmentation-based evasion	Defined insertion/evasion model	Foundational Model for IDS reassembly mismatches	Showed attackers can evade IDS by exploiting fragment interpretation gaps
Atlasis (2012, 2014)	IPv6 fragmentation attacks	Practical IPS evasion via overlaps	Motivates IPv6-specific attack surface	Overlapping IPv6 fragments bypass commercial detection
Shankar & Paxson (2013)	Testing reassembly behaviour	Structured model for probing fragment policy	Methodological foundation for later automated testing	Demonstrated reassembly policy is measurable and differs by OS
Di Paolo et al. (2023)	IPv6 reassembly testing	Permutatio-based testing framework	Inspiration for test design	Found inconsistencies in OS reassembly
Pevma (SEPTun) (2016–2024)	IDS performance tuning	Guidelines for Suricata performance tuning	Guidelines for Suricata performance tuning	Kernel defragmentation improves performance but shifts trust to OS behaviour

Table 3.1: Related Work Summary Table.

3.1. Fragmentation-based evasion

In their work on IDS evasion, Ptacek and Newsham present [23] an attacker model that exploited ambiguities in the way packet fragments are interpreted by different network devices. They identify two core evasion strategies: insertion and evasion. In insertion attacks, a detection system incorrectly reassembles and analyses a packet that is ultimately discarded by the endpoint, resulting in a false alert. On the other hand, evasion attacks occur when the IDS fails to reconstruct a packet that is correctly accepted and processed by the target host, allowing malicious payloads to pass undetected.

Their experiment focused on TCP/IP traffic, particularly fragmented IP and out-of-order TCP segments, and they showed how detection systems that did not mirror the exact reassembly logic of the host could be easily evaded. The work emphasised that even minor implementation differences in how packets are reassembled, timeouts are handled, or overlaps are resolved could be used by attackers to their advantage.

This thesis extends Ptacek and Newsham's framework by investigating whether the same kinds of reassembly mismatches can be exploited in IPv6 fragmentation to bypass detection. While their original work examined TCP/IP and generic IDS-host discrepancies, here we focus on high-performance environments, where the demands of throughput and latency may increase the attack surface.

3.2. IPv6 fragmentation attacks

In his 2012 Black Hat Europe presentation [4], Atlasis demonstrated that, despite IPv6's more rigorously defined fragmentation model, many high-end IPS devices remain vulnerable to evasion via carefully crafted overlapping fragments and extension-header manipulation. He showed that discrepancies in how detection systems and operating systems resolve overlapping payloads create blind spots in deep packet inspection, allowing malicious traffic to pass undetected.

In 2014, Atlasis and Rey [43] released a detailed paper demonstrating that by combining IPv6 Routing Headers, Fragment Headers, and Destination Options, attackers could redirect packets or hide malicious payloads. Their evaluation of commercial IPS and firewall products revealed that even those configured to meet RFC standards were still vulnerable to fragmentation-based evasion. They highlighted that, in practice, vendors often implement only partial or divergent reassembly logic, especially when it comes to handling IPv6 extension headers, creating persistent blind spots in network defences.

While their work was important in revealing the practical risks of IPv6 fragmentation, it did not model reassembly behaviour across operating systems or test passive configuration. On the other hand, this thesis investigates non-inline IDS deployments that depend on kernel-level reassembly, examining whether such configurations remain vulnerable to overlapping-fragment attacks. Building on Atlasis's insights, we aim to show that even in modern, high-performance environments, fragmentation ambiguities continue to create exploitable gaps in security visibility.

3.3. Testing reassembly behaviour

In 2003, Shankar and Paxson [44] introduced Active Mapping, an empirical technique for uncovering exactly how a given host handles IP fragment reassembly. By sending carefully crafted probe fragments and observing which ones the target accepts or rejects, their method allows an IDS to "learn" the host's reassembly policy. This host-aware approach ensures that the IDS's own defragmentation logic mirrors that of the protected system, closing the gap that attackers might otherwise exploit through mismatched fragment handling.

Their methodology involved sending probe sequences of overlapping fragments to target systems and analysing their responses. By varying fragment order, overlap type, and timing, they could deduce whether a system followed a "first fragment wins" or "last fragment wins" strategy and how it dealt with partially reassembled packets. This allowed them to build reassembly profiles of different operating systems, which could then be used to tune detection rules in a way that matched the endpoint's behaviour.

The key contribution of their work is that reassembly is not only variable but also measurable and classifiable, enabling detection systems to adapt rather than rely on a fixed reassembly model. Their

findings laid the groundwork for newer models, such as those of Di Paolo et al. [3].

This thesis adopts the same core assumption, that reassembly behaviour varies across systems, but shifts evaluation from active probing of endpoints to passive observation within an IDS pipeline. Rather than deriving custom fragment-handling rules as Shankar and Paxson did, it evaluates whether simply relying on the operating system's native, kernel-based reassembly is enough for the IDS to faithfully match the host in complex fragmentation scenarios.

3.4. IPv6 Specific Reassembly Testing

In their recent work, Di Paolo, Bassetti, and Spognardi [3] introduced a framework for testing IPv6 fragment reassembly behaviour across operating systems. Their paper addresses the gap in empirical methodologies by proposing a permutation-based model that generates all meaningful combinations of IPv6 fragments under controlled assumptions. Their focus is on whether endpoint stacks reassemble packets in the presence of overlaps, out-of-order delivery and fragment retransmission.

A key strength of their approach is its meticulous test generation model, which enumerates input cases based on fragment metadata (offsets, lengths, overlaps). The authors built a testing pipeline capable of injecting crafted IPv6 fragments into real hosts and observing whether a complete packet is accepted or dropped. They applied this framework across major OS families (Linux, Windows, OpenBSD, FreeBSD) and discovered different reassembly behaviours.

Their work is relevant for this thesis for two main reasons. First, their permutation-driven testing framework, together with its binary "accept/reject" outcomes, is reused in the experiment architecture adopted in this thesis. Second, by demonstrating that even standardised systems display different fragmentation behaviours, they fuel our concern that passive IDS solutions, which rely on kernel reassembly, may not always reflect the true logic used by endpoints.

However, Di Paolo et al. focus only on endpoint reassembly, not detection systems or passive monitoring setups. This thesis extends their model into the IDS domain, applying similar permutation logic but observing not only whether the host accepts the packet but also how it reassembles it and whether an IDS detects it. It also considers traffic of high-performance networks, such as 5G networks.

3.5. Suricata Performance Tuning

Apart from academic research, real-world suggestions on deploying intrusion detection systems in high-speed networks have also influenced this thesis. One of those suggestions is the **SEPTun (Suricata Extreme Performance Tuning)** series by Manev (Pevma) [5, 6, 7]. These community-driven documents provide technical recommendations for configuring Suricata, a widely used open-source IDS, for line-rate traffic monitoring in multi-gigabit environments.

The SEPTun guides suggest the need for zero-copy packet capture, parallel processing and kernel-space optimisations. They recommend using AF_PACKET mode so that Suricata can tap directly into the kernel's memory-mapped ring buffers, cutting per-packet overhead and enabling multi-threaded analysis with little performance penalty. However, this configuration comes with a trade-off: Suricata must depend entirely on the kernel for IP defragmentation rather than using its own user-space reassembly engine.

This design choice introduces a dependency on the correctness and security of the kernel's reassembly logic. If the kernel reassembles fragments in a way that differs from the final destination host (due to differing OS versions, patches, or kernel parameters), the IDS may interpret the packet stream differently, recreating the exact *interpretation gap* that Ptacek and Newsham [23] warned about.

The performance-tuning setup recommended in SEPTun is exactly the model used in this thesis' testbed. It reflects real deployment constraints and justifies the relevance of evaluating whether kernel-based reassembly in passive IDS setups creates opportunities for evasion under fragmentation. Therefore, this thesis examines the security implications of a method typically regarded as a best practice for high-throughput detection environments.

3.6. IDS Deployment in 5G Core Networks

The increasing modularity of 5G networks has led to new design opportunities for placing security functions inside the core. Several studies have examined where intrusion detection systems can be deployed for maximum visibility with minimal performance impact.

Radoglou-Grammatikis et al. [38] explored deploying an IDS at the N4 interface, which connects the SMF to UPF. Their system used AI-based anomaly detection to monitor signalling traffic for suspicious behaviour. The placement was chosen to avoid interfering with user traffic while still observing control-plane interactions that might indicate compromise or misconfiguration. Although their work focused on detecting anomalies in control messages, their results demonstrate that modular, real-time IDS deployment within the 5G core is both practical and effective.

Le et al. [40] take a different track by virtualising the IDS as an independent network function within the 5G core. Their deployment model involved virtualising the IDS and orchestrating it using service-based management layers, ensuring flexibility, scalability, and NFV compatibility. This model supports decoupled monitoring and aligns with the general 5G principle of dynamic function chaining.

Both studies validate the feasibility of embedding IDS modules within the 5G core, either adjacent to or inside the UPF. However, they do not address fragmentation, reassembly behaviour, or packet-level evasion techniques. This thesis differentiates itself by focusing on the fragmentation-specific risks that arise when passive IDS systems are integrated into high-speed, encapsulated environments.

4

Methodology

In this chapter, we outline how we'll test the impact of IP fragmentation handling inconsistencies on security visibility within a high-performance network (i.e., 5G) monitored by an out-of-path IDS relying on kernel-level defragmentation. We start by generating carefully controlled sets of fragment sequences, then send them through GTP-U tunnels to mimic real 5G traffic. As each fragmented packet is decapsulated and reassembled by the kernel, we record both what the target system reconstructs and whether the IDS raises any alarms. By varying fragment sizes, offsets, overlaps and timing in a controlled way, we can map out where the IDS and the endpoint disagree in reassembly and thus how easy it is to perform insertion or evasion attacks.

4.1. Experimental Design

This thesis introduces an experimental framework to evaluate how IP fragmentation affects security in a high-performance network (i.e., 5G network) with an IDS built into the UPF. The framework uses a pipeline that transforms fragmented packet permutations into test observations and abstracts from the implementation-specific details. These specific details will be explored in Section 5.1

The focus is on identifying inconsistencies between packet reassembly at the monitoring IDS (which relies on kernel-level defragmentation) and the target host.

Pipeline Overview

The high-level evaluation pipeline consists of five main stages: (1) generating fragmented packet permutations, (2) transmitting the packets through a simulated 5G user-plane path using GTP-U encapsulation, (3) observing how the Linux kernel reassembles the fragments, (4) applying Suricata traffic inspection on reassembled packets, (5) collecting the host responses, packet traces and IDS alerts and (6) classifying the outcomes based on observed host behaviour and IDS alert generation. This pipeline can also be seen in Figure 4.1.

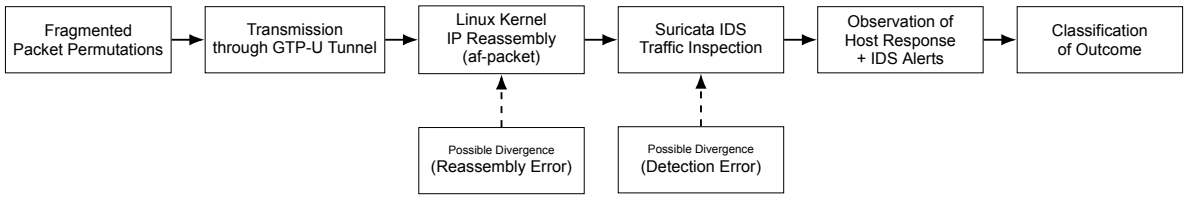

Figure 4.1: Pipeline Overview: Fragmented Packet Processing

Using the output of this pipeline, several real-life test scenarios were designed to test the security and resiliency of the 5G system against IP fragmentation attacks. These scenarios are discussed in more detail in Chapter 5.

Generation: Fragment Permutations

The input to the experiment consists of systematically generated fragment permutations, representing different fragmentation strategies. The permutations use a combination of the following variables:

- **Fragment ordering**: Fragments are transmitted out of sequence to test reassembly ordering robustness.
- **Overlap patterns**: Fragments overlap partially or fully, creating ambiguity during reassembly.
- **Retransmission**: Fragments are retransmitted multiple times to test the reassembly synchronisation between the IDS and end-host
- **IP Identification manipulation**: Fragment sets share or differ in IP ID fields to test context association.

Processing and Transmission

For each generated fragment permutation, we encapsulate the fragments within a GTP-U tunnel and forward them through the 5G core towards our emulated end host, passing directly through the UPF. This setup reproduces the user-plane behaviour of an operational 5G network, ensuring that every test packet must be decapsulated and reassembled by the kernel before it reaches the server.

At the UPF, the IDS passively captures a copy of the traffic using *AF_PACKET* mode. As fragments arrive, the Linux kernel performs IP reassembly, merging them into complete packets. Those reconstructed packets are then forwarded to Suricata, which examines them for anomalies, signature matches, and policy violations. Meanwhile, the UPF continues to forward the original traffic toward its intended destination.

Observation and Classification

For every test case, we log how the target host handles the reassembled packet, whether it accepts it and generates a response or rejects it. Additionally, we note any alerts raised by Suricata, such as overlap warnings or rule matches. At the same time, we capture packet traces at critical points in the path (the UPF, the end host, and the Linux bridge between containers or VMs) to support a detailed comparison of what each component actually saw.

Afterwards, each permutation is analysed and classified based on the following outcomes:

- **Exploitable and Undetectable:** A reassembly discrepancy occurs between the UPF and the end host, the end host accepts the packet, and the IDS does not generate any fragment overlap or related alerts.
- **Detectable Exploitation:** A reassembly discrepancy occurs between the UPF and the end host, the end host accepts the packet, but the IDS generates fragment overlap or related alerts.
- **Non-Exploitable:** There is no reassembly discrepancy between the UPF and the end host, the end host rejects the packet, or reassembly fails.

Pseudocode for Experimental Evaluation

The overall logic of the experimental process can be expressed as pseudocode. This pseudocode abstracts the experiment's control flow and highlights the decision points used in analysing the outcomes.

Algorithm 1 Fragment Permutation Testing

```
Fragment_Permutations ← 720 permutations representing all possible transmission orders of 6 fragments
for each permutation in Fragment_Permutations do
    Create a fragmented ICMPv6 Echo Request
    Encapsulate each fragment inside GTP-U
    if test_1 then
        Send each fragment once
    else
        Send each fragment 5 times with the same IP ID
    end if
    Monitor the target host:
    if host reassembles and responds then
        Host_Result ← "Accepted"
        Record how the host reassembled the packet (Host_Packet)
    else
        Host_Result ← "Rejected"
    end if
    Monitor Suricata IDS alerts:
    IDS_Result ← "No Alert"
    if an alert is triggered then
        Store the alert and correlate it with the current permutation
        IDS_Result ← "Alert"
    end if
    Monitor and save the network traffic:
        - At the UPF (UPF_Packet represents how the UPF reassembled the packet)
        - At the Linux bridge
        - At the end-host
    Classify permutation outcome:
    if Host_Result == "Accepted" and IDS_Result == "No Alert" and Host_Packet ≠ UPF_Packet then
        Mark as "Exploitable and Undetectable"
    else if Host_Result == "Accepted" and IDS_Result == "Alert" and Host_Packet ≠ UPF_Packet then
        Mark as "Detectable Exploitation"
    else
        Mark as "Non-Exploitable"
    end if
end for
```

4.2. Necessary and Sufficient Conditions for Fragmentation Attacks

In this section, we define the criteria that make fragmentation-based attacks possible by distinguishing between two sets of conditions. Necessary conditions are the minimal preconditions for a potential attack to be feasible. Sufficient conditions, on the other hand, are the observable outcomes that indicate a successful attack.

Necessary Conditions

The following conditions must hold for a fragmentation-based attack to be possible:

1. **Reassembly must occur at the host**: The target system must successfully reassemble the fragments into a valid payload and accept the packet at the network stack.
2. **The attacker must control fragmentation behaviour**: The attacker must be able to manipulate fragmentation variables, including fragment offsets, ordering, overlap, and retransmission.
3. **The IDS must inspect reassembled packets**: Detection uses a passive IDS that relies on kernel defragmentation
4. **Fragmentation must preserve transport-layer coherence**: The fragments must be valid in structure and not rejected due to protocol-level constraints (e.g., fragment length alignment, header corruption).

Sufficient Conditions

An attack is considered successful if, in addition to satisfying all necessary conditions, the following hold:

1. **Host accepts the reassembled packet**: When the endpoint correctly reassembles and acts on the payload, for example, replying to an ICMP echo or completing a TCP handshake, demonstrating that the fragmented data was interpreted exactly as intended.
2. **IDS fails to detect the payload or anomaly**: The IDS does not raise an alert for known malicious signatures (e.g., EICAR) or for fragmentation policy violations (e.g., overlap), indicating that the detection logic failed to trigger.

Together, these criteria establish whether a fragmentation-based exploit is truly feasible. An attack succeeds only when the endpoint reassembles and acts on the malicious payload while the IDS remains silent. If the host accepts the packet but Suricata raises an alert, the exploit could still reach its target but would be detected. If the host rejects the reassembled packet, the attack cannot be exploited, regardless of whether the IDS spots it.

4.3. Metrics and Evaluation Criteria

This section defines the evaluation criteria for each fragmentation test case, focusing on the visibility, correctness, and exploitability of the fragmented traffic, as observed by both the IDS and the target host. These metrics allow for measuring IDS effectiveness, understanding OS-specific reassembly behaviour, and analysing fragmentation inconsistencies and their exploitability.

Reassembly Consistency

Definition: A binary indicator of whether the payload reconstructed by the IDS matches exactly the payload reconstructed by the endpoint OS.

Purpose: By comparing IDS and host reassembly outputs, this metric reveals vulnerabilities from IP fragmentation in high-throughput, low-latency networks (RQ1) and shows how different 5G components perform reassembly. Consistency means detection gaps lie in signature coverage and are not because of the reassembly behaviour of the systems, while inconsistency suggests an evasion surface. This approach follows Ptacek & Newsham's [23] insertion/evasion framework and Shankar & Paxson's [44] active mapping technique for quantifying host/IDS reassembly discrepancies.

Evaluation Metric: A binary metric indicating

- **Consistent**: Both host and IDS reconstruct identical payloads.
- **Inconsistent**: Different reassembly or silent packet rejection.

Detection Visibility

Definition: The percentage of all sent fragment permutations that trigger at least one IDS alert, either for fragmentation anomalies (e.g., overlapping offsets) or for payload signatures (e.g., EICAR).

Purpose: Detection rate is a well-known metric in IDS evaluations [45], measuring the proportion of malicious or anomalous events that generate alerts. Our Detection Visibility metric applies this concept to fragmented traffic, directly addressing RQ3 by quantifying how often an IDS flags suspicious fragment behaviour. A high alert rate on host-accepted permutations indicates good coverage, while a low rate reveals blind spots where stealthy payloads may slip through.

Evaluation Metric:

$$\text{Detection Rate} = \frac{\text{Alerts Triggered}}{\text{Total Permutations Tested}} \times 100\%$$

Host Acceptance

Definition: The number of sent fragment permutations that the endpoint OS successfully reassembles into a valid packet, as evidenced by a proper ICMPv6 Echo Reply.

Purpose: Host Acceptance determines which fragment sequences can actually deliver payloads to the host, directly addressing RQ1 by identifying real-world vulnerabilities from IP fragmentation and RQ2 by revealing how different 5G components influence reassembly outcomes. Only permutations that the host accepts are exploitable, making this metric the essential first filter for both evasion and insertion attacks. Measuring host acceptance is a standard practice in fragmentation-evasion research, as seen in Di Paolo et al.'s [3] permutation-based evaluation of IPv6 fragmentation behaviour.

Evaluation Metric: Binary outcome

- **Accepted:** Host processed packet and responded.
- **Rejected:** No response, packet reassembly failed or was dropped.

Permutation Filtering and Classification

Definition: Permutations are classified based on the combined outcome of host acceptance and IDS alerting.

Purpose: By filtering all tested permutations through this framework, we are able to find the "window" in which an attacker can both deliver a payload to the host and evade detection, thus addressing RQ4 and guiding towards necessary solutions addressed in RQ5 .

Evaluation Metric:

Host Accepted	IDS Alert	Category
Yes	No	Exploitable and Undetectable
Yes	Yes	Detectable Exploitation
No	Yes/No	Non-Exploitable

Table 4.1: Classification of Exploitation Based on Host and IDS Response

Evaluation of Real-Life Fragmentation Scenarios

Definition: The evaluation of a separate subset of tests based on the classification outcome of the previous step that tests the detection of malicious vectors used in real-life testing (i.e., EICAR test string).

Purpose: By testing the practicality and effectiveness of the exploitable permutations with real-life malicious vectors, we are directly addressing RQ4. Proof-of-concept attacks are a standard practice in cybersecurity research, as seen, for example, in Atlasis' work [4, 43].

Evaluation Metrics:

- **Exploitable:** The fragmented payload was accepted by the host and triggered an expected response (e.g., ICMP reply, connection establishment).
- **Detectable:** The IDS generated an alert for the fragmented payload (e.g., signature match, anomaly detection).
- **Stealthy:** The IDS did not generate an alert for the fragmented payload.

5

Experimental Results

This chapter presents the results of a set of tests that were created to evaluate how typical operating systems and a passive IDS respond to IPv6 fragmentation under controlled experiment conditions. By analysing detection behaviour, consistency of reassembly, and semantic interpretation of different IPv6 fragments, the study discovered differences between what the IDS inspects and what the host processes. It was found that reassembly policies, timing variances and fragment structure can allow attackers to bypass detection. Moreover, the findings enable us to evaluate the risks and possible exploitation in the real world.

5.1. Experimental Setup

This section describes the design and implementation of a controlled testing environment intended to evaluate how IP fragmentation discrepancies impact detection accuracy in high-throughput networks. We deployed a realistic 5G network using OpenAirInterface[1], and employed passive monitoring with kernel-level packet reassembly. Multiple endpoint operating systems were tested to compare their respective reassembly behaviours. The overall setup is designed to reflect a real-world deployment scenario where performance considerations motivate out-of-path monitoring and reassembly is delegated to the Linux kernel.

5.1.1. 5G Testing Environment

The testing environment consists of four logical nodes: a User Equipment simulator, a virtualised gNodeB, a UPF node where Suricata is deployed as the IDS, and a target host. All the subsystems are deployed inside separate Linux containers or virtual machines and connected via Docker networks and virtual bridges.

The **User Equipment** was implemented using a Docker container based on the official Docker image *oaisoftwarealliance/oai-nr-ue:develop*. This container was configured with custom packet generation scripts written in Python 3.10[2] using *scapy*[3] to craft and send specific IPv6 fragments. The scripts support full control over fragment offset, ordering, payload content, and retransmission logic. Traffic from this container is routed through a common Docker bridge that connects it to the gNodeB

The **gNodeB** was simulated using the official Docker image *oaisoftwarealliance/oai-gnb:develop*, configured in standalone mode. While not a full radio layer simulation, the container provides the necessary encapsulation and GTP-U setup to forward user-plane packets from the UE to the core. The gNodeB is also connected to the common Docker bridge and forwards packets between the User Equipment and the User Plane Function.

The **User Plane Function** was deployed on a Docker container based on the official Docker image

[1] https://openairinterface.org/
[2] https://www.python.org/
[3] https://scapy.net/

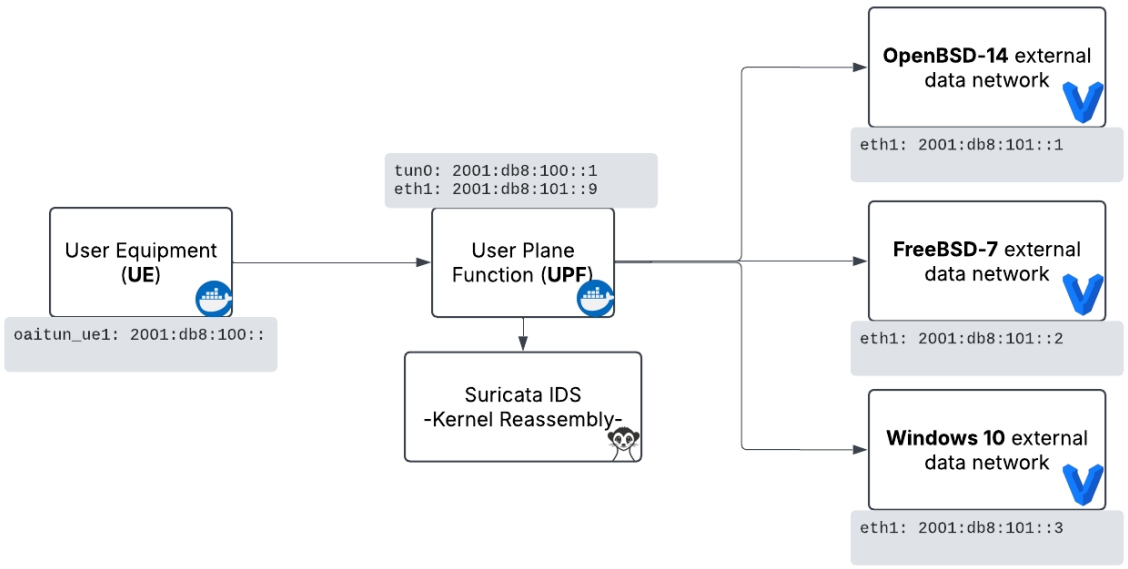

Figure 5.1: Test Environment Architecture

oaisoftwarealliance/oai-upf:develop. Inside the UPF, Suricata[4] was deployed in *AF_PACKET* mode using version 6.0.9, with kernel-based defragmentation enabled via *defrag: yes*. This UPF hosts the IDS, which inspects the traffic after GTP-U decapsulation. The gNodeB is also connected to the common Docker bridge and forwards packets between the gNodeB and the destination host.

The final component, **the destination host**, acted as the endpoint responsible for replying to ICMPv6 Echo Requests. This node was virtualised with *Vagrant by HashiCorp*[5], and it provided the "ground truth" for determining which reassembled packets the OS accepted and processed. This component uses the same common Docker bridge to receive and send packets. ICMPv6 Echo Replies were used to confirm packet acceptance.

5.1.2. IDS Configuration

The IDS was deployed inside the UPF node using Suricata version 6.0.9. Suricata was configured in *AF_PACKET* mode to capture traffic via memory-mapped ring buffers with kernel-level zero-copy. The interface receiving mirrored post-decapsulation traffic was explicitly bound to Suricata, and traffic from the GTP-U output was tapped directly.

Suricata was configured with kernel-level defragmentation enabled (*defrag: yes*). This ensures that the IDS inspects packets that the Linux kernel has already reassembled. While efficient, this design introduces a dependency on the kernel's interpretation of fragmented data, an assumption tested in this thesis. Suricata's rule set was kept minimal and focused on three criteria: detection of the EICAR test string[6] as a known benign signature trigger, identification of generic ICMPv6 traffic for validation purposes, and logging of any fragmentation-related errors (such as invalid offsets or overlapping fragments).

5.1.3. Operating Systems Under Test

To examine how different operating systems interpret fragmented traffic, four target platforms were selected: Windows 10, Ubuntu Linux, FreeBSD, and OpenBSD. These were chosen for their known differences in RFC compliance [3]. On each system, we ran tcpdump to record both incoming fragments and outgoing Echo Replies. These packet traces enabled us to verify exactly which fragments arrived and whether the host responded, providing a clear "ground truth" for each operating system's

[4] https://suricata.io/
[5] https://developer.hashicorp.com/vagrant
[6] https://www.eicar.org/download-anti-malware-testfile/

reassembly behaviour.

Operating Systems	Kernel Version	Vagrant box version
GNU/ Linux Ubuntu 22.04	6.8.0-59-generic	-
OpenBSD	7.4 GENERIC.MP#1397	4.3.12
FreeBSD	14.0-RELEASE	4.3.12
Microsoft Windows 10	10.0.20348.2113	2102.0.2409

Table 5.1: Operating system, kernel versions and Vagrant box versions used for generating the test results.

5.1.4. Fragmentation Permutation Design

The input model used for this experiment is adopted from the permutation framework proposed by Di Paolo et al. [3], developed to test how IPv6 implementations handle incomplete or ambiguous fragment reassembly. We constructed a consistent set of fragments, each defined by its offset, length, and payload, and then varied the order in which they arrived. By observing how each target system handles these permutations, we can compare reassembly policies (for example, first-fragment-wins, last-fragment-wins, or strict RFC 8200 [1] compliant rejection) when dealing with incomplete or conflicting fragment data.

Each test case consists of a permutation of six fragments, labelled A through F (Figure 5.2). Each fragment contributes a slice of payload data and includes a fixed offset and length. Certain combinations (mainly, ABCF or ABEF) align perfectly to cover the entire payload without any gaps or overlaps, resulting in an unambiguous reconstruction. Other fragment combinations leave gaps in the byte range and are not accepted.

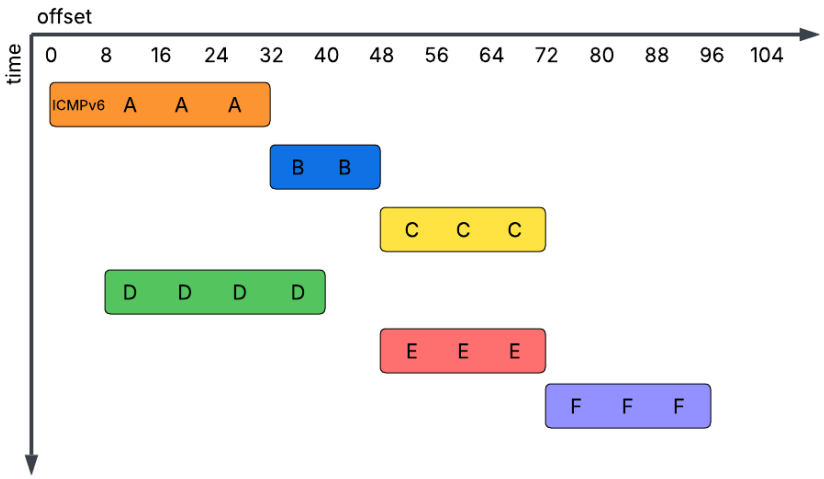

Figure 5.2: Testing model presented in paper [3]. The packet consists of 6 overlapping fragments used to uncover reassembly inconsistencies.

Each fragment in our test suite carries two different payloads (Table 5.2), selected dynamically to prevent checksum errors while maximising variation in reassembly behaviour. Specifically, if a fragment appears at an even index, it uses one version of the data. If it appears at an odd index, it uses the alternate version. Both variants are created such that, regardless of which fragments and in what order they arrive, the final ICMPv6 checksum remains valid. By eliminating checksum errors as a confounding factor, this scheme lets us focus on how different kernel and OS reassembly policies drive divergent outcomes.

All fragments within a permutation share the same IPv6 Fragment Header, including a common IP Identification value. The headers are individually valid and adhere to RFC 8200 [1], but the full set of fragments often does not reassemble into a complete, unambiguous packet. This is by design: the goal is to reveal whether the receiving system attempts to reassemble such ambiguous combinations

and, if so, how it resolves them, or whether it follows the stricter behaviour recommended by RFC 5722 [8] (i.e., discarding all fragments on detection of overlap).

Fragment	Odd packet	Even packet
A	11223344	44113322
B	11332244	44331122
C	22113344	44332211
D	22331144	11224433
E	33112244	11334422
F	33221144	22114433

Table 5.2: Payload data for each fragment and index [3]

Each permutation is transmitted sequentially using Scapy[7], and all permutations are logged and evaluated independently. In a second set of experiments, the same permutation set is used, but with fragment retransmissions enabled: each fragment is sent five times, using the same offset and IP ID, simulating lossy or unstable network conditions. This tests whether repeated delivery of fragments influences the reassembly strategy, for instance, by overwriting existing fragments or triggering early timeouts.

This design enables classification of system behaviour according to reassembly success, IDS alerting, and policy enforcement. It also reflects realistic evasion scenarios in which an attacker may send incomplete or deliberately ambiguous fragment sets to manipulate how detection systems and endpoint hosts reconstruct packets. The results allow for a direct comparison between kernel-level reassembly in the IDS and user-level reassembly in the host, under a diverse range of input conditions.

5.2. Reassembly Behavior Across Systems

This section explores Sub-Research Questions 1 and 2 by comparing how different endpoint operating systems and the passive IDS within the UPF reconstruct IPv6 fragments. It begins by detailing each system's approach to handling overlapping fragments and identifying whether they follow a "First-In" or "Last-In" policy. Next, it highlights any inconsistencies between the host's native reassembly and the kernel-based process used by the IDS. We ran two experiments, first without any fragment retransmissions to observe deterministic behaviour in an ideal, low-noise environment, and then with retransmissions to determine how fragment timing or redundancy affects reassembly logic and alignment between IDS and host.

5.2.1. No Retransmission Case

In the first series of tests, each of the 720 permutations was sent exactly once, with no retransmissions and consistent timing across fragments. By design, every permutation included overlapping fragments that, under strict RFC 5722 [8] compliance, should cause the entire packet to be dropped. This RFC mandates that any detection of overlapping fragment offsets should lead to complete reassembly failure and packet discard. However, the results from this baseline experiment show that many widely used operating systems, including the Linux kernel used by Suricata, deviate from this policy.

OpenBSD demonstrated the most conservative behaviour, accepting only **94 permutations**. This strongly suggests that OpenBSD enforces strict drop-on-overlap logic, as defined by RFC 5722 [8]. Fragments with conflicting offsets appear to be detected early in the reassembly process and result in packet discard. **FreeBSD** and **Windows 10**, by contrast, accepted **168** and **240 permutations**, respectively, a clear indication that they implement a more permissive reassembly strategy that tolerates some degree of offset reuse or overlap.

On the **IDS** side, the Suricata deployment reassembled all **720 permutations**. Moreover, a significant number of these (**120**) were reconstructed using **"Last-In"** behaviour, meaning that the most recently received fragment for a given offset was used in the final reassembled payload. The remaining **600**

[7] https://scapy.net/

permutations were reassembled with **“First-In”** behaviour, similar to the defragmentation strategy of the other operating systems.

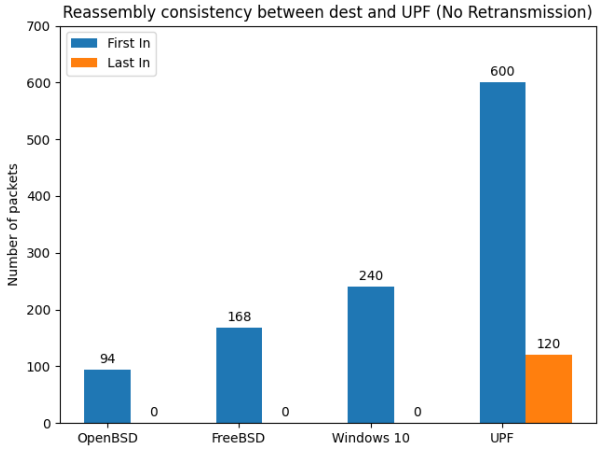

Figure 5.3: Fragment reassembly acceptance across platforms. OpenBSD’s strict drop-on-overlap accepts 94/720 permutations. FreeBSD and Windows 10 accept 168 and 240, respectively. Suricata on Linux’s AF_PACKET reassembles all 720 (600 first-in, 120 last-in).

This divergence shows an inconsistency in how fragmented IPv6 traffic is interpreted. Within the IDS, Suricata delegates reassembly to the Linux kernel, choosing performance over strict compliance, which allows later fragments to overwrite earlier data and complete the packet. Endpoint operating systems, on the other hand, apply stricter policies or default to “First-In” logic, effectively locking out any subsequent fragments once the reassembly buffer has been populated. As a result, the IDS and the host could potentially analyse different “final” payloads, thus affecting the reliability of out-of-path monitoring in security-sensitive environments.

Although no explicit alert bypass was observed in this baseline test (explored in Section 5.3), the inconsistency in reassembly behaviour is itself a cause for concern. It demonstrates that even in low-noise conditions, a passive IDS cannot be assumed to share a common packet view with the endpoint.

5.2.2. Retransmission Case

In the second test group, each permutation was transmitted five times, maintaining identical fragment offsets and IP IDs across each retransmission. This scenario simulates real-world conditions such as packet loss or link-layer retries. The goal was to examine how each system resolves duplicate fragments and whether reassembly strategies remain stable under lossy networks.

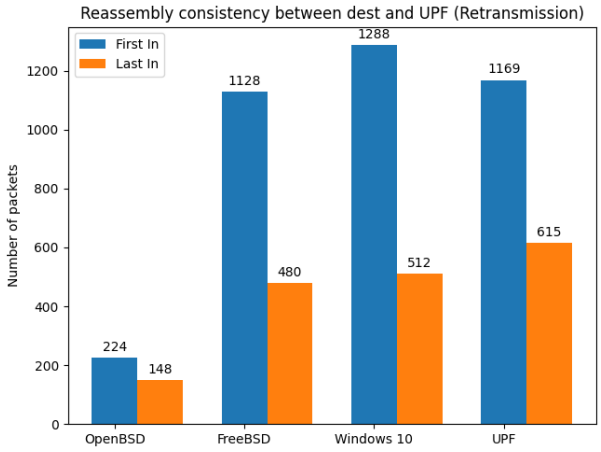

Figure 5.4: Accepted fragment permutations and “Last-In” defragmentation cases by system. OpenBSD: 372 accepted (148 Last-In), FreeBSD: 1,608 accepted (480 Last-In), Windows 10: 1,800 accepted (512 Last-In), IDS: 615 Last-In cases.

The results showed a dramatic shift across all tested systems. **OpenBSD**, which had previously accepted only 94 permutations, now accepted **372**, a 4 times increase. **FreeBSD** accepted **1,608 permutations** (up from 168), and **Windows 10** accepted **1,800 permutations** across all test runs.

Across all endpoints, the “Last-In” policy became far more common: **OpenBSD** applied this strategy to **148 cases**, **FreeBSD** to **480**, and **Windows 10** to **512**. Similarly, the **IDS** followed the same trend, with **615** of its defragmentations using the most recently received fragment when conflicts arose.

These results indicate that retransmissions weaken the determinism of reassembly behaviour, likely due to factors such as fragment cache replacement, buffer invalidation, or race conditions in timeout management. When the same fragments are resent, the reassembly engine desynchronises, and what was supposed to be a “First-In” behaviour by design, in reality becomes a “Last-In” behaviour due to the desynchronisation.

From a detection point of view, this is highly problematic. The more permissive or inconsistent a system becomes under retransmission, the more likely it is that host and IDS views of the packet will be different. Additionally, if the IDS completes reassembly based on early fragments and discards late-arriving overlapping ones, it may never see the data that the host ultimately processes. This issue is especially problematic in high-throughput or noisy environments, such as 5G networks, where retransmissions are frequent.

5.3. Evasion and Exploitability Analysis

This section addresses Sub-Research Question 3 by evaluating whether mismatches in fragment reassembly and IDS detection can be exploited to craft stealthy or misleading traffic. Specifically, we investigate whether any fragment permutations are accepted by the endpoint yet go unnoticed by the IDS, whether the IDS and host reconstruct divergent payloads, and whether those cases trigger alerts or remain silent under both no-retransmission and retransmission conditions. By applying our three-category classification framework to each permutation, we analyse detection outcomes, reassembly divergence, and their practical exploitability.

5.3.1. No Retranmission Case

In the baseline experiment, the number of reassembly discrepancies between IDS and the host was low. Only **18 permutations**, all observed on **Windows 10**, showed a mismatch in reassembly strategy, that is, the IDS used a “First-In” policy while the host used “Last-In”, or vice versa. This definition of exploitability reflects a condition in which the two systems reconstruct different payloads from the same fragment set. These permutations are thus labelled exploitable, since an attacker could leverage them to embed content that the IDS either ignores or misinterprets.

However, while these 18 cases represent reassembly divergence, they did not go undetected. Each of them triggered an alert in Suricata for overlapping fragments. These alerts do not reflect the content of the packet. Suricata may have seen only a benign reassembly, but the alert is still triggered on the packet. Therefore, the 18 exploitable permutations observed here are detectable, but only because they trigger an overlapping fragment alert. The permutations that did not trigger alerts are a different concern. Suricata **failed to generate alerts** for **96 permutations** out of 720, which we will further explore in Section 5.4. The alert rate can be seen in Figure 5.5.

When we combine alert status with exploitability, we can construct a filtering pipeline that quantifies how many fragment permutations could trigger a “silent” attack on the end host. Figure 5.6 visualises this as a Sankey diagram: all permutations first branch on whether Suricata raised an alert, then each branch splits on whether the remaining permutations are exploitable. As previously presented, in the **“alerted”** branch, only **18** of the Windows 10 permutations remain exploitable. In the **“unalerted”** branch, **96 permutations** triggered no alerts, and, in this scenario, none proved exploitable. Nevertheless, as we show in Section 5.4, those same 96 silent permutations can enable an attack under a different attacking setup.

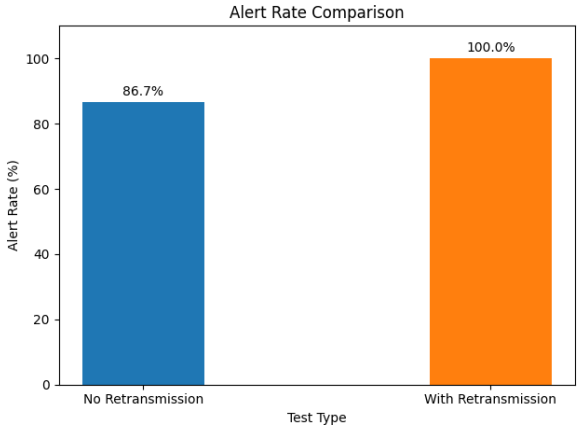

Figure 5.5: Impact of retransmission on alert delivery success: the alert rate increases from 86.7% without retransmission to 100% when retransmission is enabled.

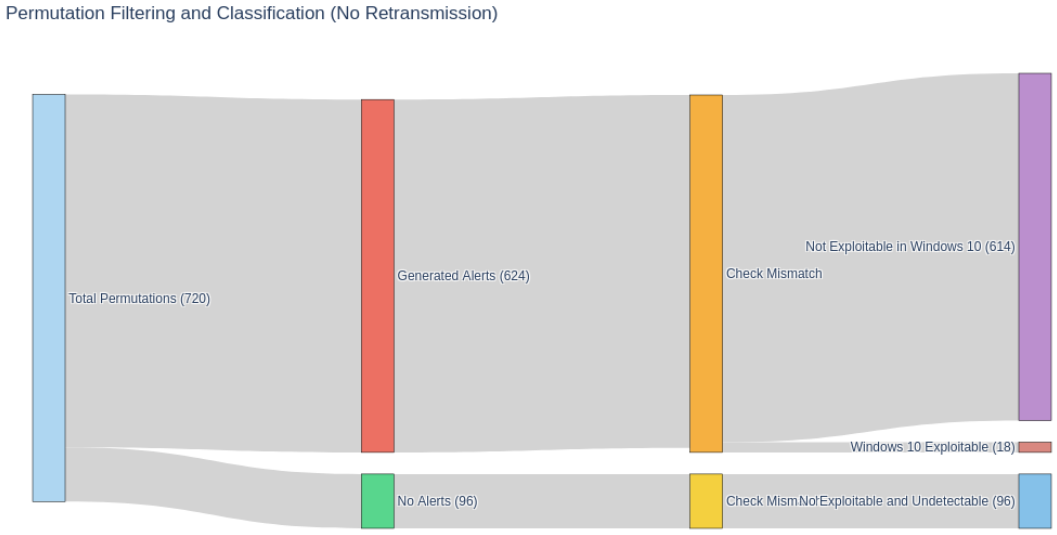

Figure 5.6: Sankey diagram of the no-retransmission permutation analysis. Out of 720 total permutations, 624 generated alerts (leading to 18 Windows 10-exploitable and 614 non-exploitable cases) while 96 produced no alerts (all deemed not exploitable and undetectable).

5.3.2. Retransmission Case

The retransmission experiment introduced a more aggressive scenario. Under these conditions, the number of reassembly mismatches between IDS and hosts increased drastically. **Windows 10** and **FreeBSD** each showed around **140 permutations** where the IDS used a different reassembly policy than the host. **OpenBSD**, while stricter, still showed such cases. In each of these, the reconstructed payloads differed, creating the potential for evasion: the IDS sees one benign payload and does not raise an alert, while the host sees and processes a malicious payload.

Thankfully, every permutation in the retransmission case triggered at least one alert in Suricata, as seen in Figure 5.5. In all cases, the alerts were related to fragment overlap or duplicate detection. These alerts are superficial: they flag the structure of the packet, but not the content. The important takeaway is that while detection is complete in quantity, it may be incomplete in meaning. Alerts exist, but they don't always reflect the final packet seen by the host.

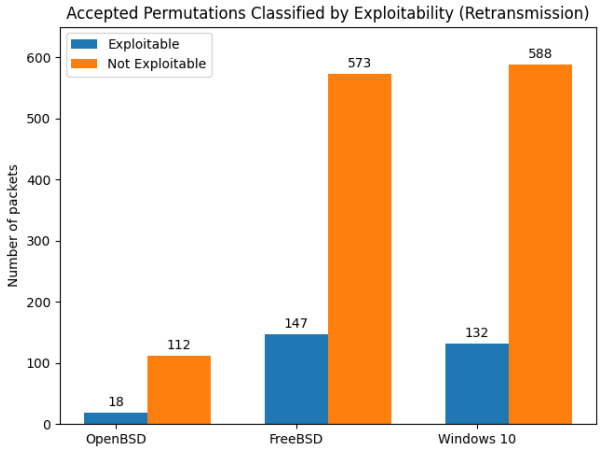

Figure 5.7: Distribution of accepted packet permutations by exploitability under retransmission: OpenBSD saw 18 exploitable versus 112 non-exploitable packets, FreeBSD had 147 exploitable versus 573 non-exploitable, and Windows 10 showed 132 exploitable against 588 non-exploitable packets.

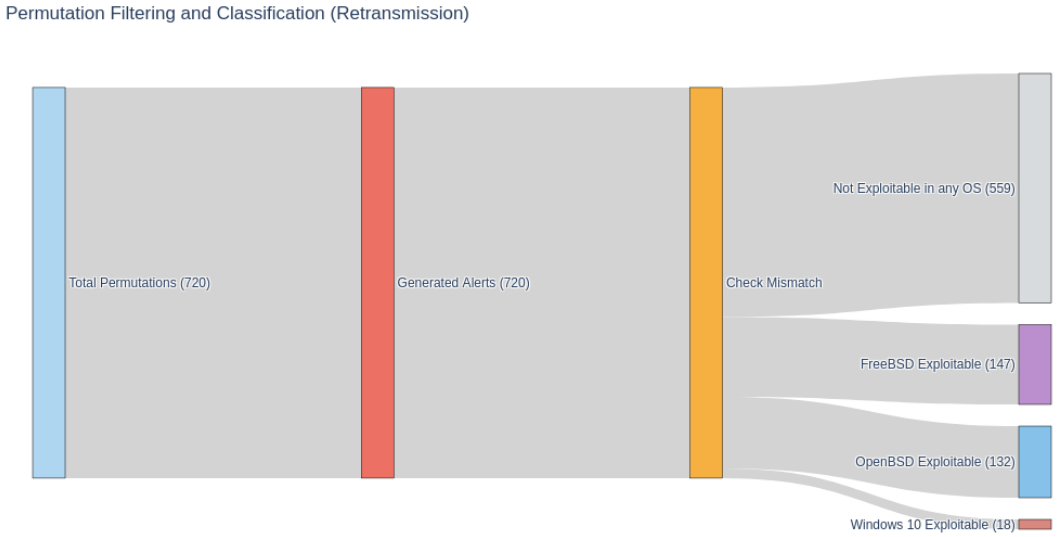

Figure 5.8: Sankey diagram of the retransmission permutation analysis: all 720 total permutations generated alerts, and after mismatch checks, 559 were not exploitable in any OS, 147 were FreeBSD-exploitable, 132 OpenBSD-exploitable, and 18 Windows 10-exploitable.

When combining both the alert rate and exploitability, we get the Sankey diagram in Figure 5.8. Unlike the baseline case, there are no permutations without alerts. Every path leads through Suricata detection, but from there, the permutation may be exploitable or not, depending on how the host reassembled it.

This diagram presents an important point: in the retransmission scenario, the attacker no longer has a stealth channel to conduct an attack, but they now have multiple divergent interpretations that they can use to manipulate the visibility of the IDS. While every permutation is flagged, the alert content may be generic, and the host may still reconstruct a payload that the IDS didn't actually see. As a result, retransmissions become a powerful tool for obfuscating payloads, splitting signatures, or delivering content selectively, all while out-of-path monitoring remains active.

Combing back to the first 3 research questions, both the baseline and retransmission tests underscore the same key finding: overlapping fragments can generate reassembly inconsistencies, creating sce-

narios where the endpoint reconstructs a packet the IDS never actually inspected. In our experiments, a correctly configured IDS, one that flags overlapping fragments, still catches these cases. However, if an IDS were improperly configured and lacked signatures for overlapping fragments, the security implications would be severe. The next section explores an even more covert attack technique, one that remains invisible to the IDS under any signature configuration, not just those with misconfigurations.

5.4. Data Exfiltration through Reassembly Mismatch

This section presents a realistic and undetectable data exfiltration technique that exploits the asynchronous handling of overlapping IPv6 fragments by a passive IDS, thus addressing research question 4. It builds upon the prior analysis of detection gaps and “No Alert” permutations by shifting the threat model: rather than bypassing detection from an external position, the attacker is already present inside the network and aims to exfiltrate sensitive data from the compromised target host to a controlled Command-and-Control (C2) server, typically positioned as a UE in the simulated environment. Crucially, because the attacker controls the receiver, they can define the reassembly policy used to interpret ambiguous fragment sequences, which allows them to reconstruct payloads undetected by the IDS, as described in this section.

5.4.1. Timing Overlap Techniques for Silent Exfiltration

As shown in Section 5.3, there were **96 permutations** in the baseline dataset that did not trigger any alerts in Suricata, despite containing overlapping fragments. Upon closer inspection, many of these permutations were structured in such a way that the overlapping fragments arrived after the kernel had completed reassembly. This phenomenon is illustrated in Figure 5.9, which depicts the packet timeline and reassembly window.

When a fragment marked as the final fragment arrives and completes the byte coverage of a datagram, the kernel delivers the reassembled packet to Suricata and terminates the associated reassembly state. Any additional fragments that arrive afterwards, even if they overlap with previously buffered offsets, are not associated with the already processed datagram. Instead, they are either dropped or treated as unrelated. This behaviour comes from performance-orientated reassembly implementations that prioritise quick forwarding over full defragmentation delay windows.

An important consequence of this behaviour is that any fragments carrying sensitive or malicious payloads can evade IDS inspection if they arrive after the kernel has already completed reassembly. As a result, the packet viewed by the IDS appears entirely benign, while the host’s own reconstruction of the same fragment stream may reveal the hidden content.

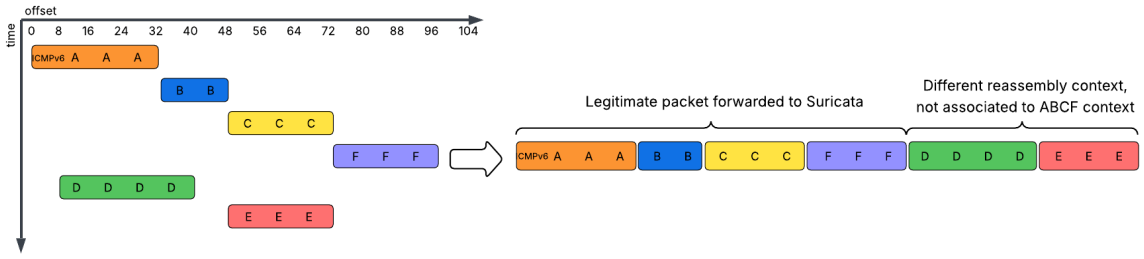

Figure 5.9: Packet arrival timeline showing that once the final fragment completes byte coverage, the kernel immediately reassembles and delivers the datagram to Suricata, purging its reassembly state and causing any later-overlapping fragments to be dropped (and thus bypass IDS inspection).

5.4.2. Attacker Strategy

Given this behaviour, an attacker can explicitly exploit the timing of fragment delivery to control what content is seen by the IDS versus what is reconstructed at the receiver. In this scenario, the attacker has already compromised the target host and wants to leak information to an external C2 node, which they fully control. This control extends to the reassembly logic at the receiver, allowing the attacker to ignore the M=0 completion rule and reconstruct any desired permutation of fragments.

The attacker constructs overlapping fragments such that an early subset of fragments (e.g., A, B, C, F) forms a syntactically correct, benign packet. This subset is designed to reach the IDS first, forcing early reassembly and preventing later fragments from being associated with the same datagram. The later-arriving fragments (e.g., D, E), which overlap with existing offsets, contain sensitive content. These are ignored by the kernel but preserved and used by the C2 node, which instead reconstructs ABEF, producing a different payload that may include sensitive data. In our test case, the sensitive data is an EICAR test string which is used as a known benign signature trigger.

The effectiveness of this strategy depends not on reassembly divergence between host and IDS but on the split in timing and reassembly ownership. The IDS's first-in-first-out behaviour is weaponised to exclude meaningful data from analysis, while the attacker ensures correct reconstruction at the destination.

5.4.3. Payload Design and Execution

To demonstrate this attack, a permutation from the 96 "No Alert" cases, specifically ABCFDE, was selected. The fragments were crafted using the position-dependent model described in Section 5.1.4. In this permutation, fragments A, B, C, and F together formed a benign ICMPv6 Echo Request, which was successfully reassembled and passed to Suricata without triggering any alert. Fragments D and E, which carried part of the sensitive content, were sent shortly after the final fragment (F), ensuring that they arrived outside the active reassembly window. In reality, the C2 server reassembles the packet as ABEF, which results in the sensitive payload the attacker wanted to transmit.

This ensured that Suricata completed reassembly based only on the early fragments and excluded the critical data embedded in the later ones. The attack's success depended on precise fragment timing and the exploitability of kernel behaviour, not on malformed headers or protocol violations.

To confirm that Suricata was capable of detecting the EICAR string under normal conditions, an unfragmented ICMPv6 Echo Request containing the complete EICAR string was also sent as a control. As expected, Suricata generated **two alerts**: one for the **outgoing Echo Request** and another for the **inbound Echo Reply** from the target. This validated that the detection rule was active, functional, and capable of matching the EICAR string in ICMPv6 traffic. Therefore, the lack of detection in the fragmented exfiltration case was not due to a faulty signature, obfuscation, or payload formatting issue, it was purely the result of the fragmentation structure and timing.

5.4.4. Proof-of-Concept Packet Trace Analysis

Figures 5.11 and 5.12 provide a detailed packet capture comparison between the IDS and the attacker-controlled receiver. The first two packets captured in both traces represent the control scenario, an unfragmented ICMPv6 Echo Request containing the EICAR string and the corresponding Echo Reply. These packets were successfully flagged by Suricata, generating the expected alerts and confirming baseline detection capability, which can be seen in Figure 5.10.

```
1 04/30/2025-20:54:29.244662  [**] [1:9990001:1] CUSTOM EICAR ICMPv6 Detected [**] [Classification: Potentially Bad Traffic] [Priority: 2] {IPv6-ICMP}
  2001:0db8:0101:0000:0000:0000:0000:0001:128 -> 2001:0db8:0100:0000:0000:0000:0000:0000:0
2 04/30/2025-20:54:29.251567  [**] [1:9990001:1] CUSTOM EICAR ICMPv6 Detected [**] [Classification: Potentially Bad Traffic] [Priority: 2] {IPv6-ICMP}
  2001:0db8:0100:0000:0000:0000:0000:0000:129 -> 2001:0db8:0101:0000:0000:0000:0000:0001:0
```

Figure 5.10: Suricata console output displaying two CUSTOM EICAR ICMPv6 Detected alerts for the unfragmented Echo Request (first line) and its corresponding Echo Reply (second line).

In the UPF-side trace (Figure 5.11), packets 3 and 6 carry the control Echo Request and its corresponding Echo Reply. Next, fragments A (packet 4), B (5), C (7) and F (8) arrive and are reassembled with the same payload as packet 10 (highlighted blue packet). This reassembled packet, whose payload Suricata inspects for signature matching, contains the inserted "FILLER" text (red circle on the bottom), which disrupts the EICAR signature and prevents any alert from being triggered. Subsequently, fragments D (packet 9) and E (11) arrive too late to be merged with the reassembled datagram and therefore are never inspected alongside the earlier fragments. Notably, fragment E holds the final bytes that would complete the EICAR test string if it had been combined with fragments A, B and F, but by then the inspection window has already closed.

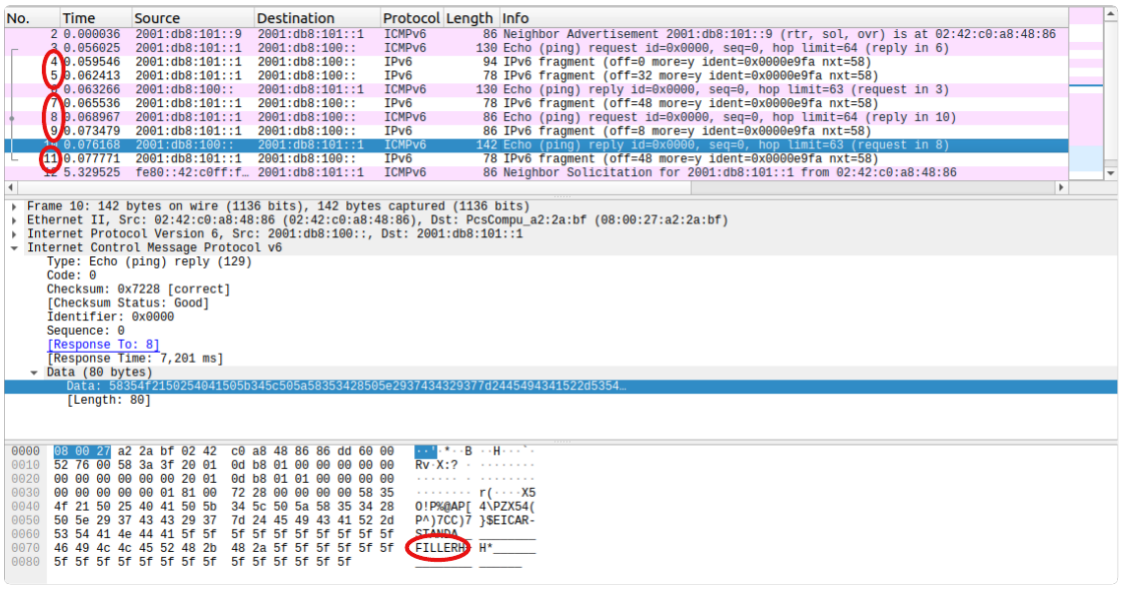

Figure 5.11: UPF-side Wireshark trace showing IPv6 fragments A (pkt 4), B (5), C (7) and F (8) reassembled into the highlighted reply: the injected “FILLER” text (circled) breaks the EICAR signature and evades Suricata inspection. Late arrivals D (9) and E (11), with the remaining EICAR tail, come after reassembly and are never checked.

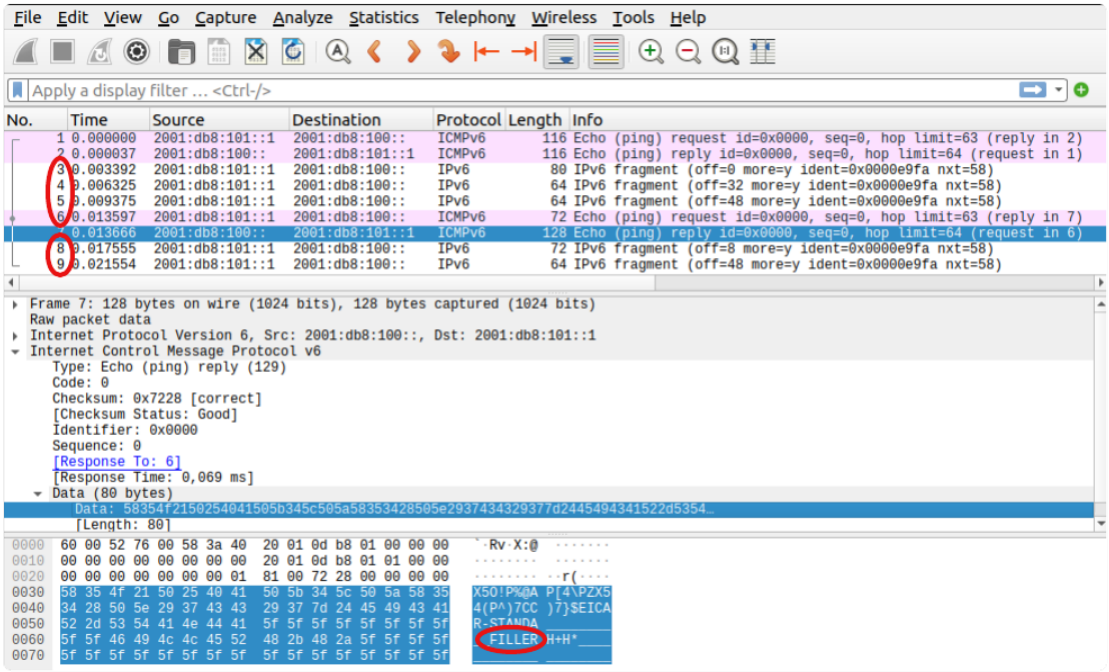

Figure 5.12: UE-side Wireshark trace showing all six fragments (A–F, pkts 3–9) arriving at the attacker’s C2 server. Although not performed here, substituting fragment E for fragment C during reassembly would restore the full EICAR string (highlighted), undoing the “FILLER” disruption seen by Suricata and enabling undetected exfiltration via the semantic split.

In the UE-side trace (Figure 5.12), all six fragments, A (packet 3), B (4), C (5), F (6), D (8) and E (9), are delivered intact to the attacker’s C2 server. By changing its reassembly logic to swap in fragment E for fragment C, the server reconstructs the full EICAR byte sequence, effectively removing the “FILLER”

disruption that Suricata saw. Because the IDS's data-loss-prevention signatures only inspect the original reassembly, this manipulated payload slips past undetected, demonstrating a clean semantic split and a reliable exfiltration channel for sensitive data (with the EICAR string as the payload and its signature match as the DLP rule). Note that we did not perform this fragment swap in our proof of concept, as it is trivial to implement.

5.5. Risk Analysis

The experimental results presented in this chapter reveal several risks in how passive IDS systems handle fragmented IPv6 traffic, especially in high-throughput, low-latency environments like 5G. These vulnerabilities do not come from strongly malformed packets or protocol violations but rather from the unexpected behaviours of the network under load, timing fluctuations, and optimisations made for performance reasons. This section combines the findings from reassembly behaviour and exploitability to assess the real-world implications of those gaps.

5.5.1. Ambiguity-Driven Attack Vectors

One of the most important findings of the experiments is that ambiguity in fragment reassembly creates an exploitable attack surface. The test environment revealed multiple permutations that led to inconsistent payload interpretation between the IDS and the host. On the upside, they were all detectable by the IDS, as they all triggered an overlapping alert.

The fact that these results came from valid IPv6 headers, using standard fragmentation extension headers, well-formed checksums, and legal lengths, underscores the severity of the risk. While the overlapping offsets technically violate RFC 5722 [8] and should trigger reassembly rejection, in practice, many systems do not enforce this policy strictly, leading to inconsistent handling of what should be considered malformed input.

The results show that even mainstream, updated systems like Windows 10, FreeBSD, and OpenBSD diverge under certain fragmentation patterns, especially when retransmission is involved. These differences in how fragment buffers are managed and completed create opportunities for attackers to craft packets that are not misrouted or dropped, but misinterpreted.

Another key finding is that alert presence does not imply alert correctness. In the retransmission test, Suricata raised alerts for 100% of permutations, yet many of those alerts were structural, triggered on duplicate fragment offsets or overlaps. These alerts do not reflect what was actually reassembled, nor do they always correlate to whether the payload was malicious. Therefore, in misconfigured IDS scenarios, these permutations could lead to an attack because the IDS alerts do not get triggered on the content but rather on the structure of the packet.

5.5.2. Reassembly Divergence and Detection Blind Spots

In scenarios where an IDS and a host reconstruct different payloads from the same fragments, detection becomes inconsistent. The IDS may flag a benign-looking payload while the host processes something else entirely, or vice versa. This gap undermines the core security goal of an IDS: to reflect what the host is seeing and prevent malicious data from slipping through.

The risk becomes more dangerous in high-performance deployments, such as 5G user planes, where low latency and high throughput constrain how much time an IDS can afford to spend reassembling traffic. In such environments, packet analysis may prioritise speed over completeness, causing the system to commit to a reassembly decision prematurely and ignore fragments that arrive later, as shown in Section 5.4.

This behaviour creates blind spots that are hard to detect and even harder to reason about. If an attacker understands the reassembly cutoff, for example, by knowing that the final fragment triggers reassembly, they can deliberately send benign fragments first, followed by overlapping malicious ones that are ignored by the IDS. At the host or attacker-controlled endpoint, where timing is fully under the attacker's control, the full payload can be reconstructed correctly.

This is not a parsing bug or a memory corruption vulnerability. It is a gap in interpretive context, and one that traditional IDS tools are not well-equipped to reason about, especially when visibility is limited

to a single passive tap.

Lastly, while this thesis focuses on ICMPv6, the results can be generalised to any protocol that permits large payloads or fragmented delivery. This is because IP fragmentation, and as such the detected vulnerabilities, happen below the transport layer. Moreover, many cloud-native workloads and edge-deployed services operate on virtualised stacks with fragmented internal communication. If an IDS is not positioned to match every point where reassembly occurs, those payloads can change in structure as they travel, effectively bypassing detection.

6

Mitigation Strategies and Limitations

This chapter outlines practical mitigation strategies and limitations encountered during the research, in the process answering research question 5. Although the experiments were done in a 5G testing environment, the results can be used in any high-throughput, low-latency environment where fragmentation reassembly is sensitive to timing, buffer policies, or detection system visibility constraints. The goal of this thesis is to close the gaps in detection and help design intrusion detection systems that are more reliable and consistent in fragment reassembly.

6.1. Mitigation Strategies

The biggest problem this thesis uncovered is that there are semantic differences between what a passive IDS analyses and what the host actually reassembles. These inconsistencies happen even with structurally legitimate but overlapping IPv6 fragments. These fragments are theoretically not compliant with RFC 5722 [8]. However, many operating systems still accept them. Chapter 5 showed that these differences in reassembly can be used to avoid detection, especially in passive monitoring setups.

One of the main ways to reduce the risk is to introduce temporal statefulness in IP ID handling. In the stealth exfiltration scenario in Section 5.4, overlapping fragments were injected after the IDS had already finished reassembly and cleared its buffer. IP ID should usually be random, but in this case, the IP ID had to stay the same such that the destination could correctly reassemble the data. Despite this, the IDS still had no context left to associate these new fragments with the completed datagram, and no alert was raised. To prevent this from happening, the IDS or even an operating system may add a stateful mechanism that labels IP IDs as recently used after a reassembly is done and keeps that identification in a cache for a short window. Any other fragments that arrive with the same IP ID within that time should be dropped or flagged as suspicious. This would prevent post-reassembly overwrite attempts from being silently ignored and protect against the scenario highlighted in Section 5.4. The cost of keeping such a short-term cache is quite low relative to how much it helps the detection pipeline.

Another mitigation includes enforcing stricter policies on overlapping fragment acceptance. RFC 5722 [8] says that overlapping fragments should be dropped at the kernel level. Therefore, none of the permutations should have been accepted by the operating systems under test if they were RFC compliant. This is a prevalent issue, and unless addressed and patched in future kernel versions, we recommend overwriting the kernel rule and silently dropping overlapping packets as mandated by the RFCs.

Although not a general solution for high-performance networks, placing inline security systems like IPS devices at low-bandwidth choke points in the network can completely eliminate reassembly ambiguity. Inline deployments have the benefit of actively reassembling packets and then re-fragmenting them before sending them to their final destination. Since only the fragments used in the initial reassembly are ever transmitted onwards, any delayed or overlapping fragments sent by an attacker are effectively discarded. This ensures that both the endpoint and the IPS will interpret the same packet, therefore eliminating reassembly discrepancies.

For high-throughput mobile networks like 5G, a different alternative is to normalise traffic at strategic network functions, for example, at the gNB or UPF. This means reconstructing fragmented packets early in the pipeline, getting rid of any overlaps or ambiguities, and then, if necessary, re-fragmenting the payload again, similar to what the previous solutions did. This method adds extra processing time and may lower the maximum throughput. However, in the networks that allow for this reduction in performance, it has security benefits because it makes sure that only clean traffic reaches the 5G core. This trade-off might be acceptable in this context because the UPF might already be programmed to have deep packet inspection features.

In the end, mitigation strategies must align with the network's architectural limitations and security posture. Not every environment can afford deep normalisation, and inline devices cannot always be deployed at scale. However, by combining reassembly context tracking using the IP ID, targeted deployment of inline defences, and packet normalisation in critical network functions, it is possible to reduce the risk surface exposed by fragmentation-based evasion and misinterpretation.

6.2. Limitations of Research

The research in this thesis was conducted in a controlled setting, using a virtualised testing environment that followed the behaviour of real systems. Even though we tried to make the deployment models as close to real life as possible, there are some limitations that need to be addressed.

First, all tests used ICMPv6 Echo Requests as the transport mechanism. We chose this protocol because of its simplicity and transparency. It guarantees a response if the reassembly is successful and has limited processing variability. However, in higher-layer protocols such as TCP or UDP, the fragment configuration and attack execution could differ or be more complicated, possibly resulting in different outcomes.

Second, the thesis only evaluated reassembly behaviour on three operating systems: Windows 10, FreeBSD, and OpenBSD. These systems show different implementation strategies and policy compliance levels, but they do not represent all the hosts that can be found in real networks. Embedded Linux variants, mobile OSes, container runtimes, and network middleboxes may handle fragments differently, especially when using custom stacks or legacy kernel versions. Therefore, more testing is needed to find out if the reassembly differences and exploitability generalise for a wide range of devices.

Third, the experiments were performed on a virtualised open-source 5G simulation platform instead of a real 5G network or dedicated hardware deployment. This gave us precise control over traffic timing, interface visibility, and component placement, but it did not fully recreate the concurrency, timing variability, and performance limits that are present in real-world deployments.

Finally, the tests in this thesis did not include any situations where encrypted transport or application-layer protocols were used. Encryption might make it more difficult for an attacker to craft valid overlapping fragments that preserve semantic correctness. For overlapping fragments to work, their content and structure must match, and encryption makes byte locations and cryptographic consistency closely linked. Changes or a reordering of small parts of the encrypted payload might cause authentication problems or dropped packets at the endpoint. Therefore, the attack surface demonstrated in this thesis may be less effective in fully encrypted environments, where attackers have limited control over how fragments are decrypted and reassembled. More research is needed to discover whether reassembly-based evasion is successful in these situations.

Even with these problems, the experimental framework is still solid and reproducible. The data strongly supports the identified risks, especially reassembly mismatches and stealth exfiltration. These risks provide a strong basis for more research in both academic and operational settings.

7

Conclusion

7.1. Summary of Findings

This thesis looked into the security risks of IPv6 fragmentation in high-throughput, low-latency networks, focusing on passive intrusion detection systems that operate in environments with performance constraints. While the experimental environment was based on a simulated 5G network, the findings apply more broadly to any network where traffic inspection must operate at line rate and reassembly decisions are delegated to the kernel or other time-sensitive subsystems.

The thesis revealed that even syntactically correct fragment sequences that follow baseline IPv6 standards can cause reassembly conflicts between the IDS and end host. This can happen depending on the reassembly strategy, the timing of the fragments and the methodologies used to resolve overlaps. Using a permutation-based testing model inspired by earlier work, the experiments showed a class of fragment sequences that are either silently accepted by the host but ignored by the IDS or interpreted differently by the two systems due to differences in reassmbly policies.

The results show that these kinds of inconsistencies are not just rare events, but they are behaviours that can be observed and reproduced. In the retransmission scenario, where fragments were duplicated to simulate noisy networks, the number of permutations displaying reassembly divergence increased drastically across all tested operating systems. It was even more concerning that some of these permutations, while triggering alerts in the IDS, were interpreted semantically differently than what the host actually processed. In other words, alerts were present, but the accuracy of detection was not as good.

Building on this insight, the thesis provided a proof-of-concept stealth exfiltration scenario in which a compromised host sent overlapping fragments to an attacker-controlled receiver. The attacker could bypass inspection without breaking protocol rules by making sure that only benign fragments arrive to the IDS in time to be reassembled. The actual sensitive payload was reconstructed at the C2 endpoint using fragments that arrived later. This confirmed that the combination of kernel-level reassembly, fragment timing, and passive monitoring can be leveraged to create packets that have two meanings.

The thesis suggested a number of mitigation strategies to deal with these concerning results. These included tracking IP Identification fields after reassembly, enforcing correct handling of fragment overlaps and placing inline analysis systems at choke points where bandwidth constraints allow. Additionally, in 5G environments, normalisation at programmable network functions like the gNB or UPF is a useful solution to clean up fragmented traffic early in the processing pipeline. These countermeasures are meant to reduce reassembly ambiguity, block late-arriving overwrite attempts, and restore alignment between detection systems and endpoint behaviour.

7.2. Future Work

A key area for future research will be to investigate a larger range of operating systems and network stacks. This thesis looked at three platforms: Windows 10, FreeBSD, and OpenBSD. However, it did not look into embedded devices or mobile operating systems, which could handle fragments differently because of differences in kernel versions or buffer management.

Another important area to investigate is how encryption affects fragmented traffic. This thesis did not include encryption because it makes it harder to see fragments and reconstruct a semantically sound payload. Encryption can also reduce the attack surface by making it harder for an adversary to construct fragments with predictable offsets or valid ciphertext. It is still an ongoing and important research subject to discover if and how reassembly mismatches manifest in protocols like IPsec or TLS.

More work is also needed to see how well the proposed mitigation strategies perform and how they affect operations. Testing IP ID lifetime enforcement, overlap drop policies, and fragment normalisation on real-world networks with real traffic would help understand how useful and expensive they are. Also, monitoring how these changes affect detection performance, such as false positive rates and throughput reduction, will help people decide when to use them in real-world situations.

Finally, some detection engines also have user-space reassembly modes that try to mimic host-specific behaviour. However, most of the time, these implementations are based on outdated studies or legacy RFC interpretations. As operating systems change how they handle overlapping fragments, host-based reassembly models should be re-evaluated and updated to reflect modern endpoint behaviour more accurately.

In summary, this thesis demonstrates that fragmentation remains a subtle but important security issue in high-performance networks. It shows how reassembly inconsistencies can be exploited and it provides both a guide for future study and a call of action to design detection systems that are more aware of reassembly inconsistencies and more resistant to timing issues.

References

[1] Dr. Steve E. Deering and Bob Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 8200. July 2017. DOI: 10.17487/RFC8200. URL: https://www.rfc-editor.org/info/rfc8200.

[2] Ron Bonica et al. *IP Fragmentation Considered Fragile*. RFC 8900. Sept. 2020. DOI: 10.17487/RFC8900. URL: https://www.rfc-editor.org/info/rfc8900.

[3] Edoardo Di Paolo, Enrico Bassetti, and Angelo Spognardi. "A new model for testing IPv6 fragment handling". In: *European Symposium on Research in Computer Security*. Springer. 2023, pp. 277–294.

[4] Antonios Atlasis. "Attacking ipv6 implementation using fragmentation". In: *Blackhat europe* (2012), pp. 14–16.

[5] Pevma. *Suricata Extreme Performance Tuning: SepTun Mark I*. https://github.com/pevma/SEPTun. Accessed: 2025-04-15. 2016.

[6] Pevma. *Suricata Extreme Performance Tuning: SepTun Mark II*. https://github.com/pevma/SEPTun-Mark-II. Accessed: 2025-04-15. 2018.

[7] Peter Manev and Andreas Herz. *Suricata Extreme Performance Tuning: SepTun Mark III*. Presented at SuriCon 2024. Available at https://suricon.net/wp-content/uploads/2024/12/SuriCon2024-Peter-Manev_Andreas-Herz_Suricata-Extreme-Performance-Tuning-SepTun-Mark-III.pdf, Accessed: 2025-04-15. 2024.

[8] Suresh Krishnan. *Handling of Overlapping IPv6 Fragments*. RFC 5722. Dec. 2009. DOI: 10.17487/RFC5722. URL: https://www.rfc-editor.org/info/rfc5722.

[9] Sebastian Gallenmüller et al. "5G URLLC: A case study on low-latency intrusion prevention". In: *IEEE Communications Magazine* 58.10 (2020), pp. 35–41.

[10] Joshua S White, Thomas Fitzsimmons, and Jeanna N Matthews. "Quantitative analysis of intrusion detection systems: Snort and Suricata". In: *Cyber sensing 2013*. Vol. 8757. SPIE. 2013, pp. 10–21.

[11] Razvan Bocu and Maksim Iavich. "Real-time intrusion detection and prevention system for 5G and beyond software-defined networks". In: *Symmetry* 15.1 (2022), p. 110.

[12] Qinwen Hu, Se-Young Yu, and Muhammad Rizwan Asghar. "Analysing performance issues of open-source intrusion detection systems in high-speed networks". In: *Journal of Information Security and Applications* 51 (2020), p. 102426. ISSN: 2214-2126. DOI: https://doi.org/10.1016/j.jisa.2019.102426. URL: https://www.sciencedirect.com/science/article/pii/S2214212619306003.

[13] Michael A. Gallo and William M. Hancock. "Chapter 3 - The Internet and TCP/IP". In: *Networking Explained (Second Edition)*. Ed. by Michael A. Gallo and William M. Hancock. Second Edition. Woburn: Digital Press, 2002, pp. 55–138. ISBN: 978-1-55558-252-4. DOI: https://doi.org/10.1016/B978-155558252-4/50029-8. URL: https://www.sciencedirect.com/science/article/pii/B9781555582524500298.

[14] *Internet Protocol*. RFC 791. Sept. 1981. DOI: 10.17487/RFC0791. URL: https://www.rfc-editor.org/info/rfc791.

[15] Christopher A Kent and Jeffrey C Mogul. *Fragmentation considered harmful*. Vol. 17. 5. Citeseer, 1987.

[16] *IP datagram reassembly algorithms*. RFC 815. July 1982. DOI: 10.17487/RFC0815. URL: https://www.rfc-editor.org/info/rfc815.

[17] Amir Herzberg and Haya Shulman. "Fragmentation considered poisonous, or: One-domain-to-rule-them-all. org". In: *2013 IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2013, pp. 224–232.

[18] Darren Reed, Paul S. Traina, and Paul Ziemba. *Security Considerations for IP Fragment Filtering*. RFC 1858. Oct. 1995. DOI: 10.17487/RFC1858. URL: https://www.rfc-editor.org/info/rfc1858.

[19] *3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals; GPRS Tunnelling Protocol User Plane (GTPv1-U); (Release 19)*. Tech. rep. TS 29.281. Version 19.1.0. Available from 3GPP: https://www.3gpp.org/ftp/Specs/. 3rd Generation Partnership Project (3GPP), Dec. 2024. URL: https://www.3gpp.org/ftp/Specs/archive/29_series/29.281/29281-j10.zip.

[20] *CVE-2021-45462: Open5GS 2.4.0 crafted packet from UE can crash SGW-U/UPF*. https://nvd.nist.gov/vuln/detail/CVE-2021-45462. National Vulnerability Database, NIST. 2021. (Visited on 04/23/2025).

[21] Tsung-Huan Cheng et al. "Evasion techniques: Sneaking through your intrusion detection/prevention systems". In: *IEEE Communications Surveys & Tutorials* 14.4 (2011), pp. 1011–1020.

[22] Jonathan AP Marpaung, Mangal Sain, and Hoon-Jae Lee. "Survey on malware evasion techniques: State of the art and challenges". In: *2012 14th International Conference on Advanced Communication Technology (ICACT)*. IEEE. 2012, pp. 744–749.

[23] Thomas H Ptacek and Timothy N Newsham. *Insertion, evasion, and denial of service: Eluding network intrusion detection*. Tech. rep. Technical report, Secure Networks, Inc, 1998.

[24] Fernando Gont. *Security Assessment of the Internet Protocol Version 4*. RFC 6274. July 2011. DOI: 10.17487/RFC6274. URL: https://www.rfc-editor.org/info/rfc6274.

[25] Sumit Siddharth. "Evading NIDS, revisited". In: *Symantec Connect Community* (2005), pp. 1–5.

[26] George Varghese, J Andrew Fingerhut, and Flavio Bonomi. "Detecting evasion attacks at high speeds without reassembly". In: *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*. 2006, pp. 327–338.

[27] 3GPP. *System architecture for the 5G System (5GS)*. Tech. rep. TS 23.501, V17.7.0, Release 17. 3rd Generation Partnership Project (3GPP), Jan. 2023. URL: https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/17.07.00_60/ts_123501v170700p.pdf.

[28] GeeksforGeeks. *5G Network Architecture*. Diagram (Fig. 1) from GeeksforGeeks. 2025. URL: https://www.geeksforgeeks.org/5g-network-architecture/ (visited on 05/08/2025).

[29] David B. Johnson, Jari Arkko, and Charles E. Perkins. *Mobility Support in IPv6*. RFC 6275. July 2011. DOI: 10.17487/RFC6275. URL: https://www.rfc-editor.org/info/rfc6275.

[30] Dr. Steve E. Deering and Alex Conta. *Generic Packet Tunneling in IPv6 Specification*. RFC 2473. Dec. 1998. DOI: 10.17487/RFC2473. URL: https://www.rfc-editor.org/info/rfc2473.

[31] Mansoor Shafi et al. "5G: A tutorial overview of standards, trials, challenges, deployment, and practice". In: *IEEE journal on selected areas in communications* 35.6 (2017), pp. 1201–1221.

[32] Russell Ford et al. "Achieving ultra-low latency in 5G millimeter wave cellular networks". In: *IEEE Communications Magazine* 55.3 (2017), pp. 196–203.

[33] Elie F Kfoury et al. "A comprehensive survey on smartnics: Architectures, development models, applications, and research directions". In: *IEEE Access* (2024).

[34] Ijaz Ahmad et al. "Security for 5G and Beyond". In: *IEEE Communications Surveys & Tutorials* 21.4 (2019), pp. 3682–3722. DOI: 10.1109/COMST.2019.2916180.

[35] Badre Bousalem et al. "DDoS attacks detection and mitigation in 5G and beyond networks: A deep learning-based approach". In: *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE. 2022, pp. 1259–1264.

[36] Lorenzo Fernández Maimó et al. "A Self-Adaptive Deep Learning-Based System for Anomaly Detection in 5G Networks". In: *IEEE Access* 6 (2018), pp. 7700–7712. DOI: 10.1109/ACCESS.2018.2803446.

[37] Ye-Eun Kim, Yea-Sul Kim, and Hwankuk Kim. "Effective feature selection methods to detect IoT DDoS attack in 5G core network". In: *Sensors* 22.10 (2022), p. 3819.

[38] Panagiotis Radoglou-Grammatikis et al. “5GCIDS: An Intrusion Detection System for 5G Core with AI and Explainability Mechanisms”. In: *2023 IEEE Globecom Workshops (GC Wkshps)*. 2023, pp. 353–358. DOI: 10.1109/GCWkshps58843.2023.10464667.

[39] Keshav Sood et al. “Intrusion Detection Scheme With Dimensionality Reduction in Next Generation Networks”. In: *IEEE Transactions on Information Forensics and Security* 18 (2023), pp. 965–979. DOI: 10.1109/TIFS.2022.3233777.

[40] Tan Nhat Linh Le et al. “5G-IoT-IDS: Intrusion Detection System for CIoT as Network Function in 5G Core Network”. In: *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*. 2023, pp. 4773–4778. DOI: 10.1109/GLOBECOM54140.2023.10437158.

[41] Aristide Tanyi-Jong Akem and Marco Fiore. “Towards Real-Time Intrusion Detection in P4-Programmable 5G User Plane Functions”. In: *2024 IEEE 32nd International Conference on Network Protocols (ICNP)*. 2024, pp. 1–6. DOI: 10.1109/ICNP61940.2024.10858543.

[42] Cisco Systems, Inc. *Cisco UCC 5G-UPF Configuration and Administration Guide, Chapter: 5G-UPF Overview*. 2022-01. Cisco Systems, Inc. 2022. URL: https://www.cisco.com/c/en/us/td/docs/wireless/ucc/upf/2022-01/config-and-admin/b_ucc-5g-upf-config-and-admin-guide_2022-01/m_upf-overview.pdf.

[43] Antonios Atlasis and Enno Rey. “Evasion of high-end IPS devices in the age of IPv6”. In: *BlackHat EU* 2015 (2014).

[44] Umesh Shankar and Vern Paxson. “Active mapping: Resisting NIDS evasion without altering traffic”. In: *2003 Symposium on Security and Privacy, 2003.* IEEE. 2003, pp. 44–61.

[45] Richard P Lippmann et al. “Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation”. In: *Proceedings DARPA Information survivability conference and exposition. DISCEX’00*. Vol. 2. IEEE. 2000, pp. 12–26.

A

Repository

The full source code and supplementary materials for this thesis are available at:

`https://github.com/OpreaCristian2002/reassembly-policies-5g-using-ipv6`