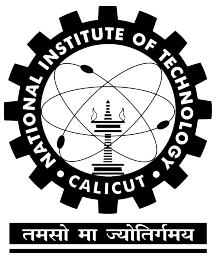
Implementation of Flow Radar in eBPF with Adversarial Attack Detection

CS4090 Project Final Report

Submitted by

Alen Antony	**Reg No: B200735CS**
Sreehari T Rajesh	**Reg No: B200754CS**
Gokul Praveen	**Reg No: B200740CS**

Under the Guidance of
Dr. Venkatarami Reddy Chintapalli

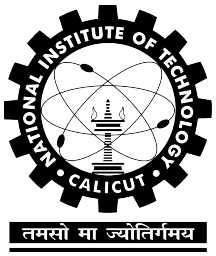


Department of Computer Science and Engineering
National Institute of Technology Calicut
Calicut, Kerala, India - 673 601

May 6, 2024

NATIONAL INSTITUTE OF TECHNOLOGY CALICUT, KERALA, INDIA - 673 601

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



2024

CERTIFICATE

Certified that this is a bonafide record of the project work titled

IMPLEMENTATION OF FLOW RADAR IN EBPF WITH ADVERSARIAL ATTACK DETECTION

done by

Alen Antony
Sreehari T Rajesh
Gokul Praveen

of eighth semester B. Tech in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering of the National Institute of Technology Calicut

Project Guide
Dr.Venkatarami Reddy
Chintapalli
Assistant Professor

DECLARATION

I hereby declare that the project titled, **Implementation of Flow Radar in eBPF with Adversarial Attack Detection**, is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or any other institute of higher learning, except where due acknowledgement and reference has been made in the text.

Place :NIT Campus,Kozhikode
Date :6/5/2024

Signature :
Name : Alen Antony
Reg. No. : B200735CS

Signature:
Name: Sreehari T Rajesh
Reg. No : B200754CS

Signature:
Name: Gokul Praveen
Reg. No.: B200740CS

Abstract

Network management tasks require the collection of large amounts of telemetry data through network monitoring systems. Flow Radar is one such network monitoring system that uses a bloom filter data structure which makes it space-efficient compared to alternatives like NetFlow. Flow Radar was implemented at the switch level using the P4 programming language [3]. The data collected would then be exported to a remote machine for further processing. This can have implications on latency and also we might require more fine-grained network data at the host level. Our project implements FlowRadar [4] at the host level using extended Berkeley Packet Filter (eBPF)[6] and eXpress Data Path (XDP)[7]. eBPF along with XDP allows fast processing of packets even before the packet reaches the kernel networking stack. Further, we also discuss various attacks possible on Flow Radar and propose features to detect these attacks. We are currently working on the detection mechanism which will be released later.

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to Dr.Venkatarami Reddy Chintapalli for their invaluable guidance, support, and expertise throughout this project. His insightful feedback, encouragement, and unwavering commitment have been instrumental in shaping the outcome of this work. We are deeply appreciative of his mentorship and the opportunities provided for personal and professional growth. We are also thankful to Harish S A(IIT Hyderabad) and Dr.Praveen Tammana(IIT Hyderabad) for their constant support and mentorship throughout this journey. We are also grateful to Dr. Subhasree M, Head of the Department of Computer Science and Engineering, whose contributions and encouragement have been invaluable in the completion of this project. Finally, we would like to acknowledge the support of our family and friends for their understanding, patience, and encouragement throughout this journey.

Contents

List of Figures

Chapter 1

Introduction

eBPF

eBPF (extended Berkeley Packet Filter) is an in kernel virtual machine implemented in the Linux kernel. eBPF programs are triggered on the basis of various hook points in the kernel. eBPF programs are loaded into the Linux kernel using the bpf system call in the form of bytecode. A compiler suite like LLVM is used to compile the pseudo-C code into eBPF bytecode.The eBPF program passes through 2 steps namely the Verification and JIT Compilation stage before it is attached to the requested hook.The verification stage validates the eBPF program based on several conditions and hence ensures that it is safe to run.The JIT Compilation stage translates the eBPF bytecode into machine specific instruction set.eBPF also provides the ability to store state and share the data between the eBPF programs as well as applications in user space with the help of eBPF maps via a system call.

XDP

The eXpress data path (XDP) is a hook on the network interface to process packets before it reaches the user space.This helps in performing high-speed packet processing within BPF applications.

Bloom Filter

Bloom filter is a hash-based probabilistic data structure primarily used to check an element's membership in the set efficiently.It is implemented in the form of a bit array.A bloom filter includes k hash functions with which we hash the incoming values.The k hash outputs are the bit positions of the bloom filter which should be set.It is one of the most popular data structures due to its lower memory consumption and constant lookup time.

Invertible Bloom Lookup table

Invertible Bloom Lookup Tables (IBLTs)[5] are a data structure used for approximate set reconciliation. They are an extension of the classic Bloom filter, which is a probabilistic data structure used to test whether an element is a member of a set.In IBLTs, each entry consists of a key and a value, similar to a key-value pair in a hash table. However, unlike a hash table, IBLTs allow for efficient merging and subtracting operations.The main operations supported by IBLTs are insertion, deletion, and lookup. These operations have time complexities that are independent of the size of the set being represented, making IBLTs efficient for large datasets.Overall, Invertible Bloom Lookup Tables provide an efficient and scalable solution for approximate set reconciliation in distributed systems, making them valuable in scenarios where maintaining consistency between large sets of data is critical.

Chapter 2

Literature Survey

NetFlow[2] has been a widely used monitoring tool for over 20 years, which records the flows and their properties.When a flow finishes after the inactive timeout, NetFlow exports the corresponding flow records to a remote collector.NetFlow has been used for a variety of monitoring applications such as accounting network usage, capacity planning, troubleshooting, and attack detection. Despite having a wide range of applications the main challenge of implementing NetFlow in hardware was to maintain a data structure for storing an active set of flows in low time and space complexity. FlowRadar[4] is an improvement over Netflow in terms of memory efficiency. The key design of Flowradar is to identify the best division of labor between cheap switches with limited per-packet processing time and the remote collector with plenty of computing resources. Encoded flow sets with limited constant time instructions for each packet are thus easily implemented in hardware. Capture encoded flow counters with constant time for each packet at switches. They introduce encoded flowsets, which are an array of cells that encode the flows and their counters. Encoded flowsets ensure constant per-packet processing time by embracing hash collisions rather than handling them.

Flowset

The FlowSet has two components: FlowFilter and Counting Table

Flow Filter

Used to register new flows and identify subsequent packets that belong to the registered flows. A flow filter is implemented as a vanilla bloom filter. When a packet associated with flow f is present in the bloom filter we hash it using k hash functions H1, H2, ... Hk to set the bits in the flow filter, thus registering the flow as a 'new flow'. However, if the hashed locations are already set, then the flow is considered as an old flow.

Counting table

The counting table is a modified invertible bloom lookup table[5] used to capture and maintain further information about the flows. It is structured as an array of cells. Each cell contains three fields : FlowXOR, FlowCount, and PacketCount. FlowXOR holds FlowIDs, FlowCount holds the number of flows mapped to the same cell, and packet count holds the number of packets observed for each flow. Upon the arrival of a new flow, FlowXOR XORs any previous flowIDs, and the current FlowID, FlowCount, and Packet Counts are incremented at the indices at which Hi(flow) maps to. On arrival of an old packet, only the packet count fields are incremented

Flowradar Operations

Flowradar has two main operations: Single Decode and Counter Decode

Single Decode

Single Decode takes as input the counting table and outputs a set of flowIDs that can be used for CounterDecode operation, i.e. the FlowXORs values having flow count as 1. These cells which have FlowCount as 1 are known as pure cells

Counter Decode

The Counter Decode takes as input the set of Single Decoded flowIDs and the packet count field values of all the counting table entries and outputs their packet counts of each flow with a reduced error margin. It solves a system of linear equations using approximation methods like method of least squares.

Adversarial attacks

Chosen Insertion Adversary

CIA works by generating flows that map to all unset bits in the flow filter. The flows generated by the CIA map to different locations in the bloom filter. This affects the pure cells of the counting table and increases the false positive rate. To perform this attack, the adversary needs to know the size of the bloom filter and the type and number of hash functions

Query Only Adversary

QOA works by generating flows that map to already set bits in the flow filter. The idea of QOA is to pollute the statistics collected behind the bloom filter. To perform a QOA attack the adversary needs to know the size, partial state, hash type, and the number of hash functions of the bloom filter

Chapter 3

Problem Definition

The existing implementation of FlowRadar[4] in switches using P4[3] has its limitations.P4 is tailored specifically for programming network switch data planes. Although it offers powerful features for packet processing and forwarding, it may lack flexibility and expressiveness. This could limit the range of algorithms that can be efficiently implemented within the constraints of P4. Secondly, P4 programs are often tightly coupled with the underlying hardware architecture of network switches. This means that a P4 program developed for one type of switch may not be easily portable to another switch model or vendor's hardware.This is where eBPF comes into play as it provides a more general-purpose environment compared to P4 and at the same time a reduced latency, as data collection and decoding is done in the host itself compared to switch-level implementation where the data collection is done at switches and decoding is done at a remote machine. In addition, eBPF programs run in the kernel space and can potentially be deployed across a wider range of systems without being as hardware-dependent. Also as it is run at the host level, we get more fine-grained data about the network flows. In this paper, we have discussed our implementation and testing methodology of FlowRadar in eBPF. We have also looked into various attacks possible on Flow Radar and features that can be used in the detection of these attacks.

Chapter 4

Methodology

Design

Our implementation of Flow Radar consists of two components - the user space program and the eBPF program running in the kernel space.

eBPF Program

The eBPF program consists of two flow sets and a flowsetID data structure implemented as eBPF array maps. The flowset consists of the flow filter and the counting table.The flowsetID data structure stores the ID of the flow set to which the packets are inserted currently.The eBPF program inserts packets to the flow sets based on the flowsetID. Insertions into the flow filter and counting table are done by using the Murmur3 hash function on the 5-tuple information of the packets.

User space program

The userspace program consists of a main thread, a flow switcher thread and a circular queue. The flow switcher thread copies the flowset from the eBPF map and enqueues it to the circular queue based on the current flowsetID every epoch. It also switches the flowset to which new insertions occur by inverting the flowsetID every epoch. The concurrent access and updation of

flowsetID are synchronised by an eBPF spinlock. The main thread consumes the flow sets from the circular queue and performs single decode and counter decode on the flow set. The flow IDs and their counts are saved to a CSV file.

Testing mechanism

We run our program in a QEMU/KVM virtual machine. The program listens on a virtual interface which is connected to a bridged network with the host. Packets are sent with tcpreplay/scapy to the bridge interface. The eBPF program saves the flow IDs and their counts to a CSV file after counter decode. This count is compared with the actual number of flows and we get the number of decodable and undecodable flows.

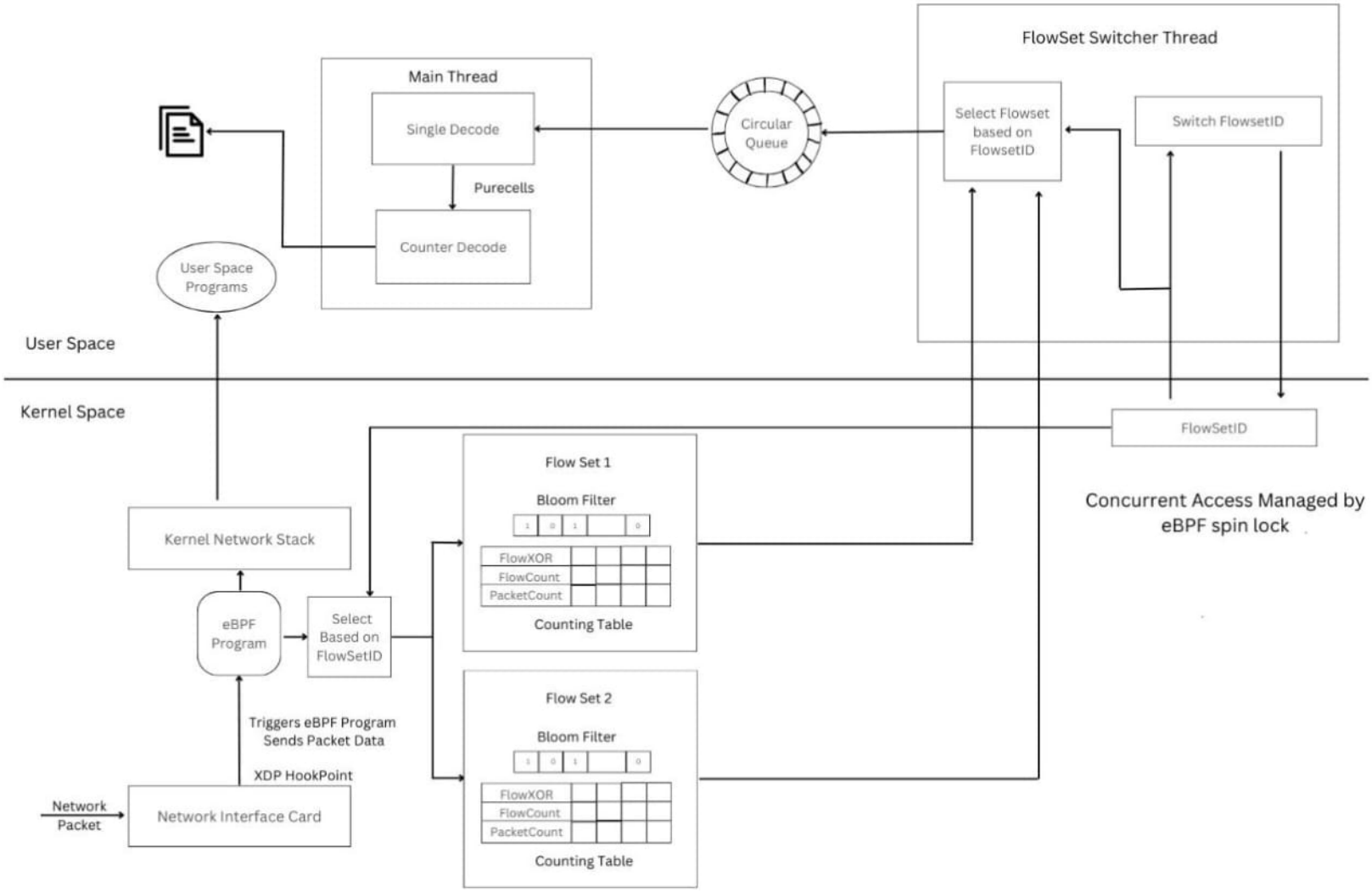


Figure 4.1: System design of eBPF Flowradar

Chapter 5

Results

We tested our implementation using the 24k flows CAIDA dataset. We used a flow filter size of 0.24 million cells and a counting table size of 0.03 million cells as per the values given by [1]. We also tested our implementation with various percentages of malicious flows by the CIA and QOA attack and also with random flows. For CIA and QOA there was a drastic rise in the number of un-decodable flows at 0.3% of malicious flows and for random flows at 1%. These results matched with the simulation study conducted by [1]

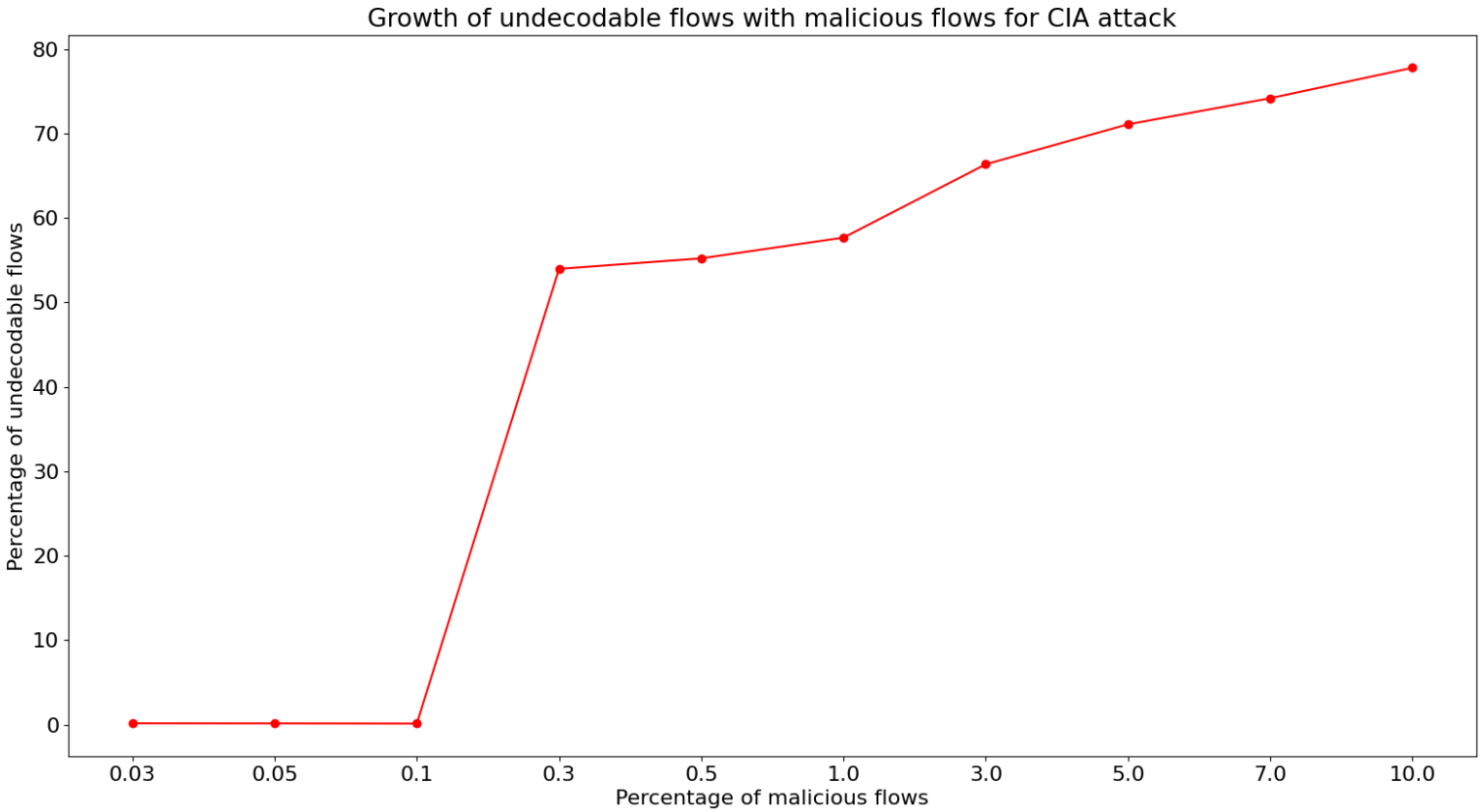


Figure 5.1: Undecodable flows in CIA

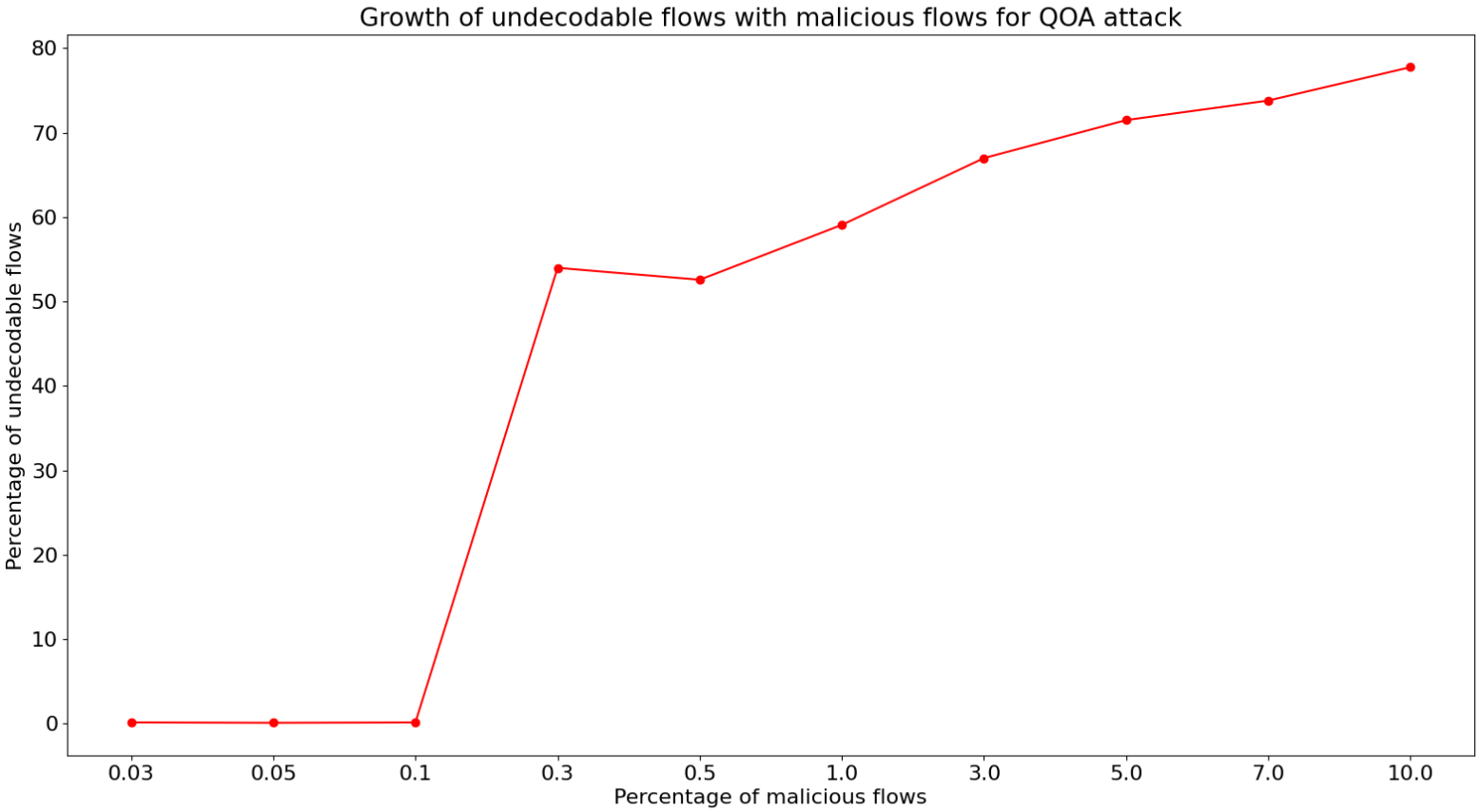


Figure 5.2: Undecodable flows in QOA

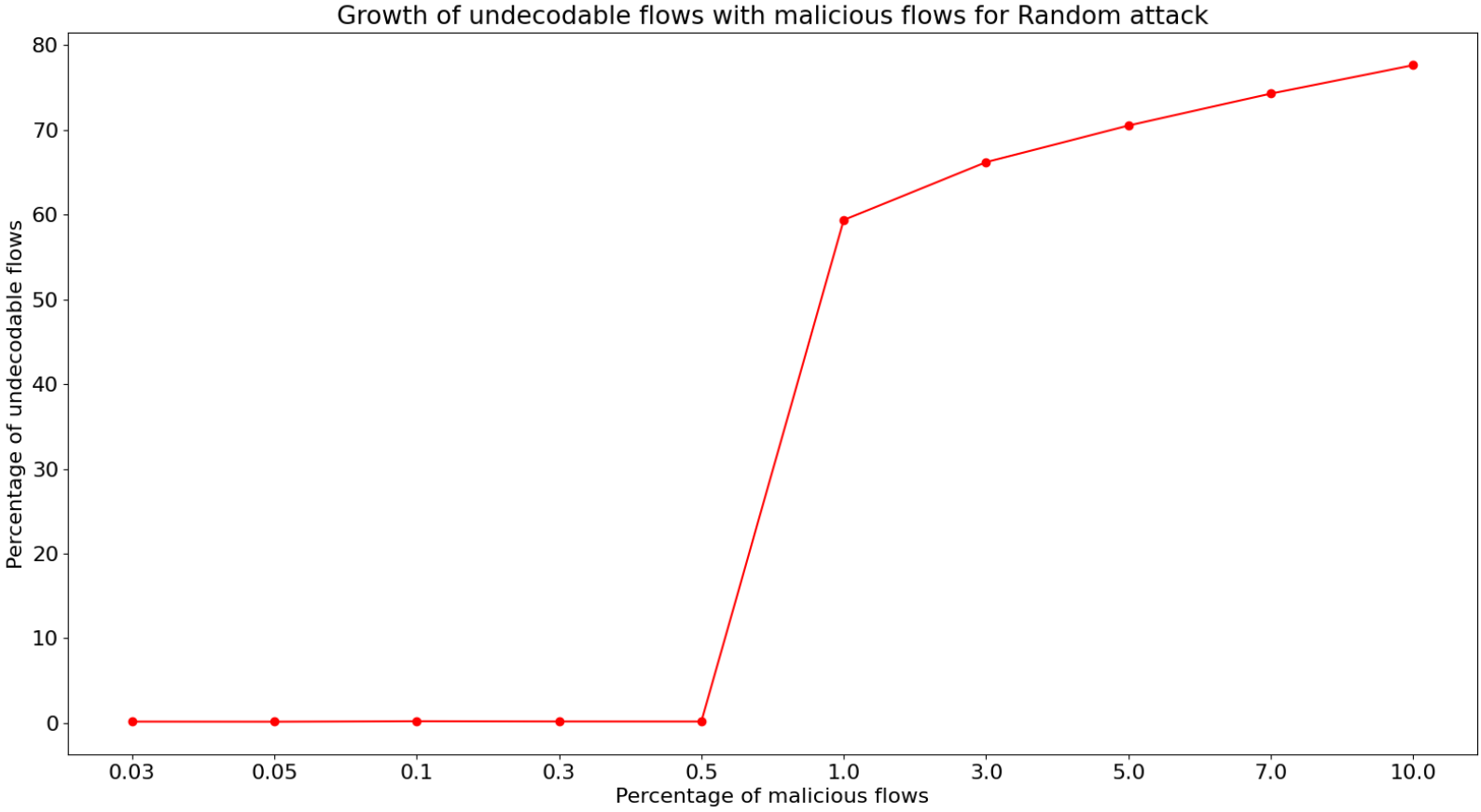


Figure 5.3: Undecodable flows for random flows

Chapter 6

Conclusion and Future work

We have successfully implemented Flowradar[4] in eBPF and tested our implementation with attacks(QOA and CIA). We also collected some features based on which an attack can be detected. For QOA, the features include the growth of pure cells, the number of hash collisions, and the number of incoming flows that collide at all of its hash indices. For CIA, the features include the growth of pure cells, the number of hash collisions, and the number of incoming flows that set all new cells. While testing our implementation with QOA and CIA attacks, we observed a steep hike in the number of undecodable flows when 0.3 percent of malicious flows were inserted. In the case of random attacks, we observed a steep hike in the number of undecodable flows when 0.5 percent of malicious flows were inserted. Our future work involves completing the detection mechanism for adversarial attacks, collecting features every time window in an epoch, and proposing a statistical/machine learning model to detect attacks. We are also planning to add IPv6 support, improve counter-decode implementation, and work on proper error handling at runtime and the documentation part as well.

References

[1] H. S A, K. S. Kumar, A. Majee, A. Bedarakota, P. Tammana, P. G. Kannan, and R. Shah, "In-network probabilistic monitoring primitives under the influence of adversarial network inputs," in *Proceedings of the 7th Asia-Pacific Workshop on Networking*, APNET '23, (New York, NY, USA), p. 116–122, Association for Computing Machinery, 2023.

[2] *NetFlow. https://www.ietf.org/rfc/ rfc3954.txt.*

[3] *Flowradar implementation in p4. https://github.com/USC-NSL/FlowRadar-P4.*

[4] Y. Li and M. Y. Rui Miao, Changhoon Kim, "Flowradar:a better netflow for data centers," *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16).*, March 16–18,2016.

[5] M. T. Goodrich and M. Mitzenmacher., "Invertible bloom lookup tables.," *In arXiv:1101.2245v2*, 2011.

[6] *eBPF. https://ebpf.io/.*

[7] *XDP. https://www.iovisor.org/technology/xdp.*