Design an intelligent ML based flow eviction algorithm

Name : M Anand Krishna

Roll No: CS22MTECH14003

Contents

Background

▼ Introduction

Flow features play a crucial role in network security, as they help in identifying and characterizing network traffic. By analyzing these features, security analysts can detect anomalies, malicious activities, and potential threats in the network. Some of the important flow features include packet length, flags, and inter-packet timing frequency distributions.

Packet length provides information about the size of the packets transmitted within a network flow. Analyzing the distribution of packet lengths can reveal unusual traffic patterns or malicious activities, such as denial-of-service (DoS) attacks, covert attack etc. Flags, present

in the TCP header, can indicate the state of a connection or specific events, such as the initiation or termination of a connection. Examining flag patterns can help detect port scanning activities.

Inter-packet timing frequency distributions capture the time intervals between consecutive packets in a flow. These distributions can reveal patterns of network usage and help detect anomalies like network scans, botnet communications, or covert channels.

Machine learning (ML) techniques can be employed to develop efficient flow classification models based on these features. By training ML algorithms on labeled flow data, classifiers can be created to automatically categorize network traffic, enabling real-time detection of malicious activities and potential threats.

However, implementing ML-based flow classification techniques on programmable switches poses several challenges:

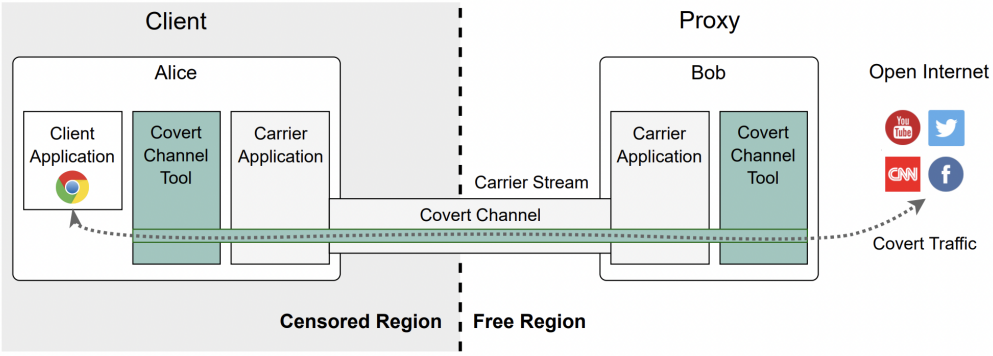
1. **Memory constraints:** Programmable switches have limited stateful memory (e.g., 100 MB SRAM), which restricts the amount of data that can be stored and processed in real-time. This limitation demands efficient data structures and algorithms that can work within the available memory while maintaining high accuracy.
2. **Processing constraints:** To ensure line-rate processing, switches have a fixed and short amount of time at each pipeline level for processing packets. This constraint limits the number and type of operations that can be executed within each stage, necessitating the design of highly efficient and optimized ML algorithms to work within these processing constraints.

Flow features are essential in network security, and ML techniques can offer efficient flow classification based on these features. However, challenges like memory and processing constraints on programmable switches need to be addressed to effectively implement such techniques in real-world network environments.

▼ Understanding Covert Channel Attack

- A covert channel is a technique used to transmit information between two parties in a way that avoids detection by security mechanisms, such as firewalls, intrusion detection systems, or network monitoring tools. The communication takes place over an existing communication channel that was not intended for this purpose, such as a multimedia stream.

- With regards to Internet censorship circumvention, the purpose of network covert channels is twofold: (i) to enable a client located within a censored region to communicate with services located in the free Internet while (ii) ensuring that the existence of such communication cannot be easily detected by an adversary(govt like).



Basic Idea

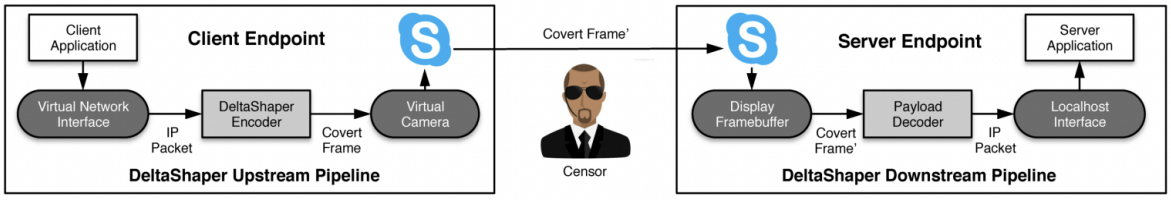
- The censored region is controlled by a state-level adversary (like Govt, which has censorship policies) which is able to observe, store, interfere with, and analyze all the network flows within its jurisdiction, and block the access to remote Internet services, such as CNN or Twitter, by the residents in the censored region.
- The free region consists of the Internet portion that is considered not to be under the control of the adversary or any other entity that aims to block Internet communications.
- Covert channels aim at enabling clients located within a censored region (e.g., Alice) to bypass the Internet communication constraints enforced by the adversary by leveraging the cooperation of a proxy located in the free region (Bob), and a carrier application featuring an encrypted communication protocol (e.g., Skype) whose traffic the adversary authorizes to cross the boundaries of the censored region.
- To create a covert tunnel through the data stream generated by the carrier application, Alice and Bob must run a special software on their local computers. This software will be responsible for embedding
  covert data within the seemingly legitimate data stream (e.g., by encoding covert data in images sent through a Skype video call). This tunnel will allow Alice to contact remote hosts on the open Internet, for instance, by tunneling the traffic generated by her web browser.

▼ Identifying covert channels - General Approach

- The first technique described is deep packet inspection, which involves analyzing the contents of packets sent over a network to identify any suspicious traffic that may be indicative of a covert channel. Deep packet inspection can be used to identify traffic indicators that lead to the identification of a covert channel, such as non-uniform inter-packet delays or unusual packet sizes.
- The second technique is statistical traffic analysis, which involves analyzing traffic patterns over time to identify any anomalous traffic that may be indicative of a covert channel. Statistical traffic analysis can help the adversary identify covert channels that may be operating within the network and identify the characteristics of the traffic that may be used to hide the covert channel
- The third technique is the use of active network attacks, which involves sending network traffic to perturb the behavior of the covert channel, and potentially reveal its existence. The adversary may also launch indiscriminate active network attacks aimed at disrupting seemingly legitimate carrier streams while ensuring that legitimate streams maintain a reasonable quality. These attacks could include injecting traffic into the network or manipulating network traffic to trigger a response from the covert channel.

▼ Delta Shaper Tool

DeltaShaper is a censorship-resistant tool that supports bi-directional TCP/IP tunneling over videoconferencing Skype streams.



Architecture of Deltashaper

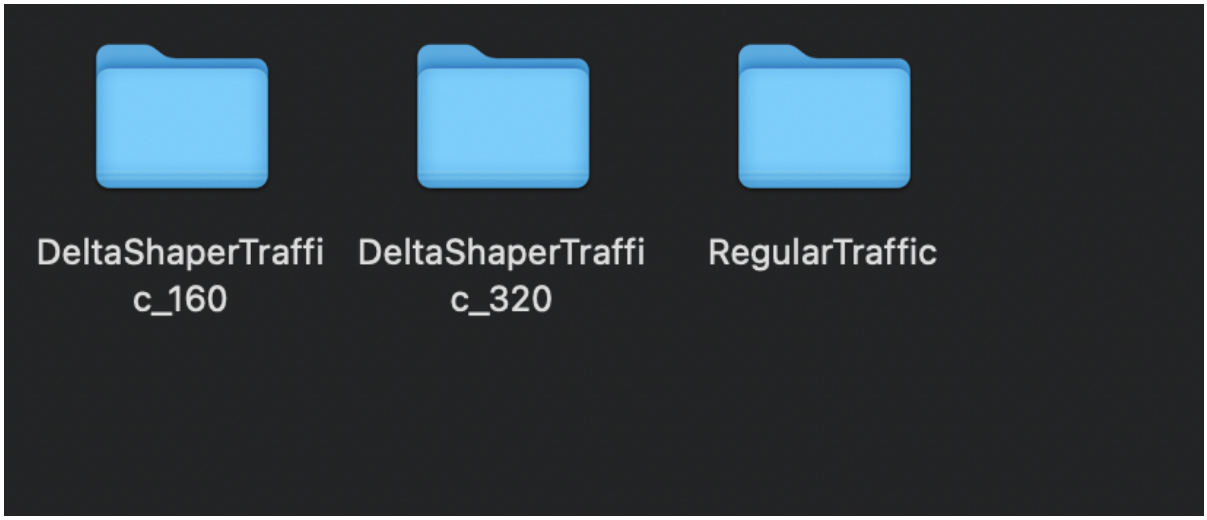
- On the sending side, Tool’s transmitter receives the payload and encodes it into colored matrices (payload frames) that are overlayed on top of a video stream that is fed to Skype using a virtual camera interface.

- Skype transmits this video to the remote Skype instance and the received stream is captured from the Skype video buffer.
- A decoder then extracts the payload from the video stream and delivers it to the application.

About Flow Lens paper's dataset creation using **Deltashaper** for FlowLens Paper

1. He emulated 300 legitimate bi-directional Skype calls by streaming a subset of our legitimate Skype video dataset. Author of FlowLens paper gathered DeltaShaper traffic samples by establishing a DeltaShaper connection between the Skype endpoints installed in both VMs. He gathered data for two DeltaShaper configurations, found to provide traffic analysis resistance guarantees, and which respected the tuple (payload frame area, cell size, number of bits, frame rate).

▼ About Labels in Delta Shaper:



Malicious packets in **DeltaShaperTraffic_160 and DeltaShaperTraffic_320 folders** : These were comprised by the (320 × 240, 8 × 8, 6, 1) and (160 × 120, 4 × 4, 6, 1) tuples**.** The tuple is (payload frame area, cell size, number of bits, framerate) .

Benign packets in **RegularTraffic Folder**

Problem Statement

A system is already there that successfully categorises malicious flows, obtaining high recall rates, especially when the training dataset contains much fewer malicious flows than benign flows. This functionality is essential for network security applications since it allows for the rapid and precise detection of malicious behaviour. However, the performance of the current system may suffer if the percentage of malicious flows in the training dataset rises to 50% or higher of all the benign flows. In certain cases, the dataset may become unbalanced due to the increased occurrence of malicious flows, which could make it harder for the classifier to distinguish between benign and malicious traffic.

Although the system has performed better than competing approaches, it has not achieved optimal recall and accuracy rates when dealing with a dataset containing a large number of malicious flows. This problem is being addressed by the AdaFlow system. In this project, we will also see the effectiveness of the AdaFlow system for detecting malicious network flows in the Covert dataset will be investigated.

Work done so far

▼ Understanding draft Adaflow paper

- **Utilising high-speed programmable switches**: AdaFlow efficiently gathers telemetry data by leveraging the processing capabilities and flexibility provided by modern programmable switches.
- **AdaFlow cache primitive:** A critical component of the system that is specifically designed to operate within the limits of the switch data plane, allowing for effective data collecting and processing.
- **Overcoming implementation challenges**: AdaFlow addresses the issues of limited stateful memory and strict per-packet operation limitations, assuring optimal performance.
- **Tofino switch prototype**: A practical version of AdaFlow has been constructed and tested on a Tofino switch, demonstrating the system's practicality.
- **Reduced resource overheads:** AdaFlow efficiently manages resources, minimising overheads while maintaining telemetry data quality and system detection capabilities.
- **Maintaining accuracy and generality**: Despite the constraints and optimizations, AdaFlow is able to maintain high accuracy in detecting a wide range of network attacks across various networks.

▼ Machine Learning phase

In this phase, network traffic analysis was focused on using machine learning techniques to detect malicious flows. The following steps were performed as part of the study:

1. A covert channel dataset was selected as the source of network traffic data. This dataset can be used to detect and analyse covert attack. Thereby evaluating the effectiveness of the system.
2. A feature space "F" consisting of 11 features, essential for characterizing network traffic and detecting anomalies, was considered. These features are:
    a. **Flow IAT Min**: Flow Inter-Arrival Time (IAT) Min represents the minimum time interval between the arrivals of two consecutive packets in a flow. This feature can help identify rapid packet transmission patterns, which could indicate malicious activities like flooding attacks or network scans.
    b. **Avg Packet Size**: The average packet size is calculated by dividing the total size of all packets in a flow by the number of packets in the flow. Analyzing packet size can help identify abnormal traffic patterns, as some types of attacks may involve large or small packets to exploit vulnerabilities or avoid detection.
    c. **Subflow F.Bytes**: Subflow F.Bytes denotes the total number of bytes transmitted in the forward direction (from the source to the destination) for a subflow. This feature can help detect data exfiltration or other malicious activities involving high-volume data transfers.
    d. **Flow Duration**: Flow duration is the time elapsed between the arrival of the first and the last packet in a flow. Longer flow durations could indicate ongoing malicious activities, such as data exfiltration or command-and-control communication.
    e. **Total Len F.Packets**: Total Len F.Packets refers to the total length (in bytes) of all packets in the forward direction (from source to destination) within a flow. This feature is useful for detecting high-volume data transfers or anomalous traffic patterns.
    f. **Active Min**: Active Min is the minimum time a flow is active, i.e., the time between the arrival of the first packet and the last packet with data payload. This can help identify short-lived malicious connections, such as those used for reconnaissance or command-and-control communication.

g. **Active Mean**: Active Mean is the average time a flow is active. Analyzing the mean activity time can help identify connections that deviate from the normal traffic pattern, potentially indicating malicious activity.

h. **Init Win F.Bytes**: Init Win F.Bytes represents the initial window size in bytes for the forward direction of a flow. Malicious actors may manipulate the window size to impact network performance or evade detection.

i. **PSH Flag Count**: The PSH flag count is the number of packets with the Push (PSH) flag set within a flow. The PSH flag is used to indicate that the sender wants the receiver to process the data immediately. A high count of PSH flags may indicate attempts to disrupt network services or exploit vulnerabilities.

j. **SYN Flag Count**: The SYN flag count refers to the number of packets with the Synchronize (SYN) flag set within a flow. SYN flags are used to initiate a connection. An unusually high count of SYN flags could suggest a SYN flood attack or network scanning activities.

k. **ACK Flag Count**: The ACK flag count is the number of packets with the Acknowledge (ACK) flag set within a flow. While ACK flags are part of standard network communication, an unusually high count could indicate attempts to establish unauthorized connections or manipulate network traffic.

3. A vanilla binary decision tree was trained on the covert dataset using the selected 11 features. Following are the metrics calculated

   a. **Recall**: The capacity of a classification model to accurately detect positive cases (in our study, harmful flows) is measured by recall, also known as sensitivity or true positive rate. It is determined by dividing the number of true positives (TP) by the total number of actual positives.

   b. **Accuracy**: An accuracy metric is used to assess a classification model's overall performance. It is calculated as the percentage of cases in the dataset that were correctly categorised (both true positives and true negatives).

Evaluation metrics for decision tree trained on 11 features	**Scores**
Recall	0.9126974099810486
Accuracy	0.6884138520735357

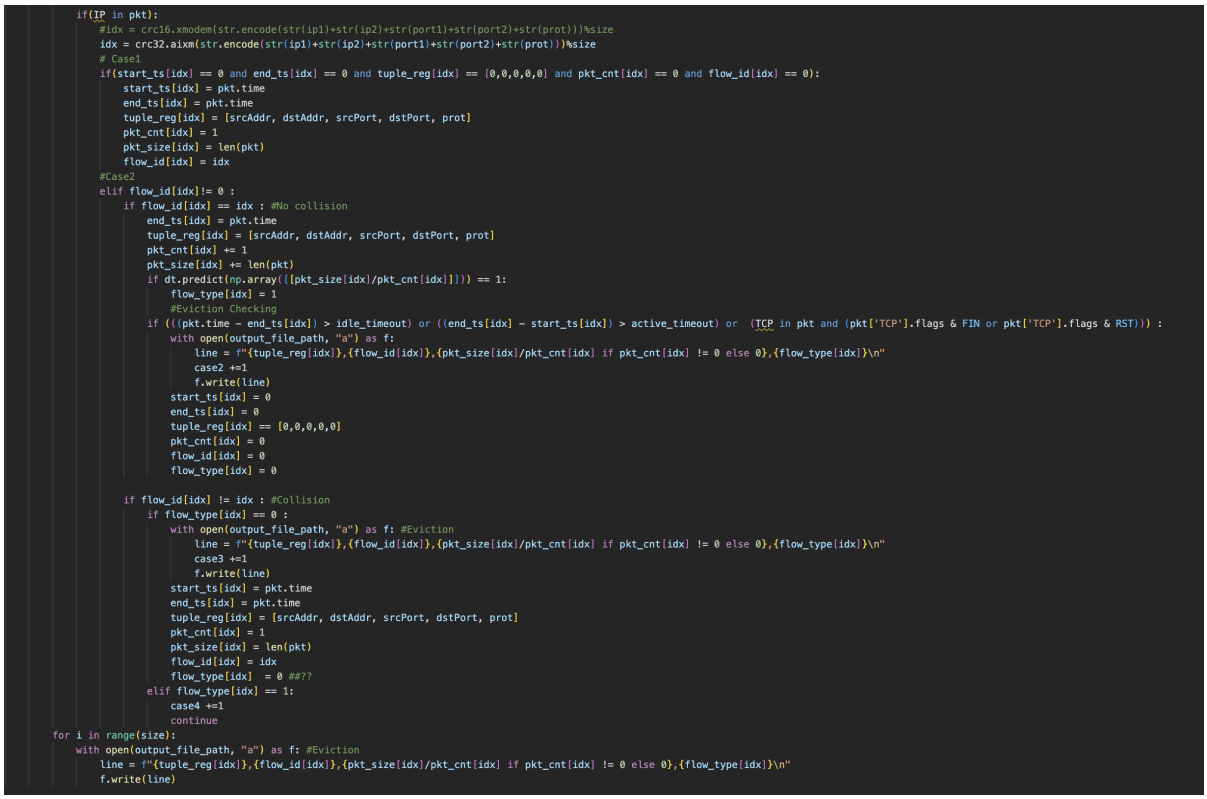
4. A minimal feature subset that maximised the recall and accuracy is **average packet size.** See the metrics of Decision tree when trained on average packet size alone.

Evaluation metrics for decision tree trained on average packet size	**Scores**
Recall	0.9774
Accuracy	0.690

Therefore important feature used to detect Covert attack is average packet size.

▼ Simulation phase

In this phase, I simulated the P4 program in python. Reimplemented priority sketch algorithm for my understanding



```
        if(IP in pkt):
            #idx = crc16.xmodem(str.encode(str(ip1)+str(ip2)+str(port1)+str(port2)+str(prot)))%size
            idx = crc32.aixm(str.encode(str(ip1)+str(ip2)+str(port1)+str(port2)+str(prot)))%size
            # Case1
            if(start_ts[idx] == 0 and end_ts[idx] == 0 and tuple_reg[idx] == [0,0,0,0,0] and pkt_cnt[idx] == 0 and flow_id[idx] == 0):
                start_ts[idx] = pkt.time
                end_ts[idx] = pkt.time
                tuple_reg[idx] = [srcAddr, dstAddr, srcPort, dstPort, prot]
                pkt_cnt[idx] = 1
                pkt_size[idx] = len(pkt)
                flow_id[idx] = idx
            #Case2
            elif flow_id[idx]!= 0 :
                if flow_id[idx] == idx : #No collision
                    end_ts[idx] = pkt.time
                    tuple_reg[idx] = [srcAddr, dstAddr, srcPort, dstPort, prot]
                    pkt_cnt[idx] += 1
                    pkt_size[idx] += len(pkt)
                    if dt.predict(np.array([[pkt_size[idx]/pkt_cnt[idx]]])) == 1:
                        flow_type[idx] = 1
                        #Eviction Checking
                    if (((pkt.time - end_ts[idx]) > idle_timeout) or ((end_ts[idx] - start_ts[idx]) > active_timeout) or  (TCP in pkt and (pkt['TCP'].flags & FIN or pkt['TCP'].flags & RST))) :
                        with open(output_file_path, "a") as f:
                            line = f"{tuple_reg[idx]},{flow_id[idx]},{pkt_size[idx]/pkt_cnt[idx] if pkt_cnt[idx] != 0 else 0},{flow_type[idx]}\n"
                            case2 +=1
                            f.write(line)
                        start_ts[idx] = 0
                        end_ts[idx] = 0
                        tuple_reg[idx] == [0,0,0,0,0]
                        pkt_cnt[idx] = 0
                        flow_id[idx] = 0
                        flow_type[idx] = 0

                if flow_id[idx] != idx : #Collision
                    if flow_type[idx] == 0 :
                        with open(output_file_path, "a") as f: #Eviction
                            line = f"{tuple_reg[idx]},{flow_id[idx]},{pkt_size[idx]/pkt_cnt[idx] if pkt_cnt[idx] != 0 else 0},{flow_type[idx]}\n"
                            case3 +=1
                            f.write(line)
                        start_ts[idx] = pkt.time
                        end_ts[idx] = pkt.time
                        tuple_reg[idx] = [srcAddr, dstAddr, srcPort, dstPort, prot]
                        pkt_cnt[idx] = 1
                        pkt_size[idx] = len(pkt)
                        flow_id[idx] = idx
                        flow_type[idx]  = 0 ##??
                    elif flow_type[idx] == 1:
                        case4 +=1
                        continue
    for i in range(size):
        with open(output_file_path, "a") as f: #Eviction
            line = f"{tuple_reg[idx]},{flow_id[idx]},{pkt_size[idx]/pkt_cnt[idx] if pkt_cnt[idx] != 0 else 0},{flow_type[idx]}\n"
            f.write(line)
```

Detailed algorithm is given below:

Require: A packet pkt with timestamp ti of flow fi
Ensure: Evicted or updated flow

```
for each packet pkt in pcap flow do
    Extract src_ip, dst_ip, sport, dport, prot from pkt
    Compute index idx using CRC32/16 hash of five tuple
    if cache entry at idx is empty then
        Initialize cache entry with flow information
    else
        if flow ID at idx matches the current flow then
            Update flow information
            if eviction conditions are met (idle and active timeouts, TCP flags)
            then
                Evict flow and write to output
            end if
        else
            flow ID at idx doesnt match the current flow (Handle collision)
            if existing flow is not malicious then
                Evict and update with new flow
            else
                Ignore the current flow (do not update)
            end if
        end if
    end if
end for
Write remaining flow information to the output file
```

This Algorithm is designed to efficiently process network traffic, evicting or updating flow information as needed. For each packet in the pcap flow, the algorithm extracts source and destination IP addresses, source destination ports, the protocol along with the flow features. Using the CRC32/16 hash function, an index is calculated based on these extracted information.

When the cache entry at the computed index is empty, the algorithm initializes the entry with flow information. If the flow ID at the index matches the current flow, the flow information is updated, and eviction conditions are checked. If flow gets old or idle, the flow is evicted and written to the output.

In case of a collision, the algorithm handles it by determining if the existing flow is malicious. If the existing flow is not malicious, the algorithm evicts it and updates the entry

with the new flow. Otherwise, the current flow is ignored, and the entry remains unchanged. The algorithm iterates through all packets, and in the end, the remaining flow information is written to the output file.

The algorithm's main focus is to maintain a balance between efficiency and accuracy when processing network traffic. By efficiently handling collisions and evictions, it minimizes the impact of false alarms while preserving the ability to detect malicious flows. The design of algorithm is aimed at maximising the recall rate.

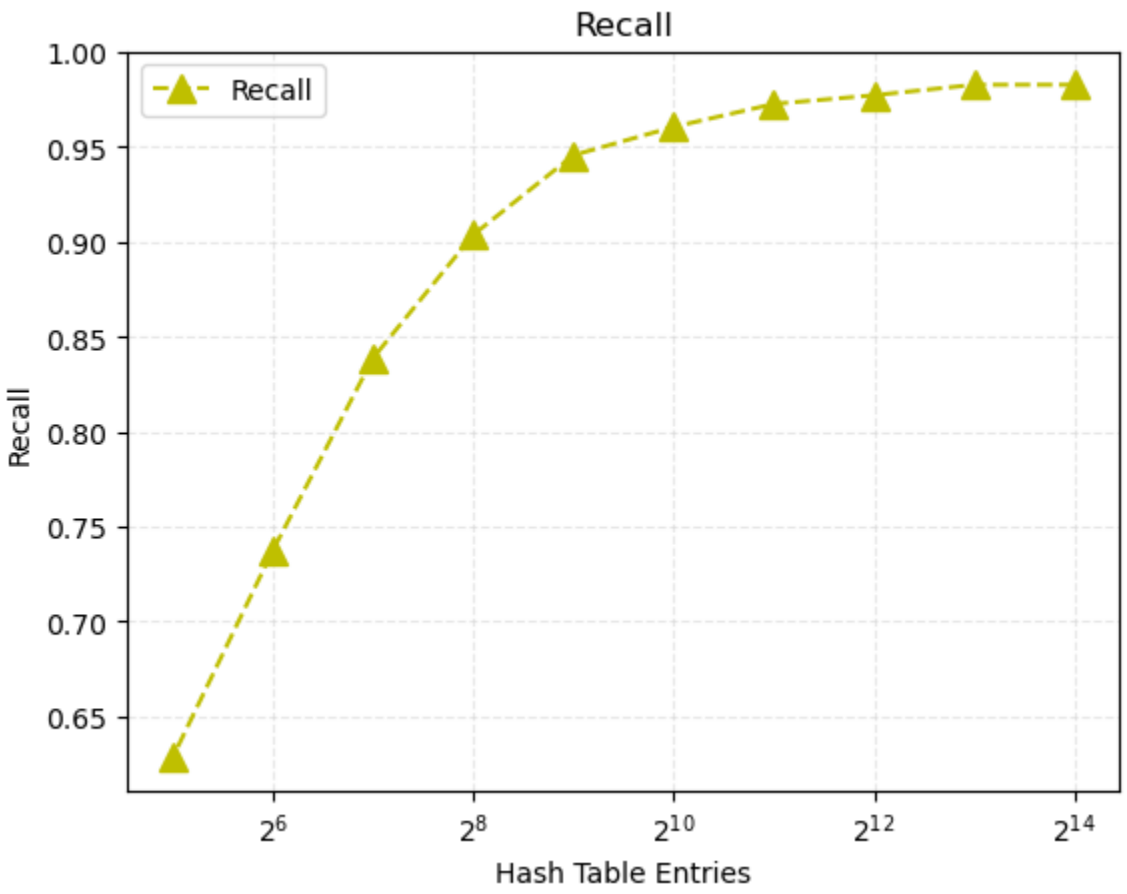
Results with observations

▼ System trained and simulated on the feature average packet size alone.

Recall of the above implemented algorithm over **average packet size** as the only feature is shown in the below table

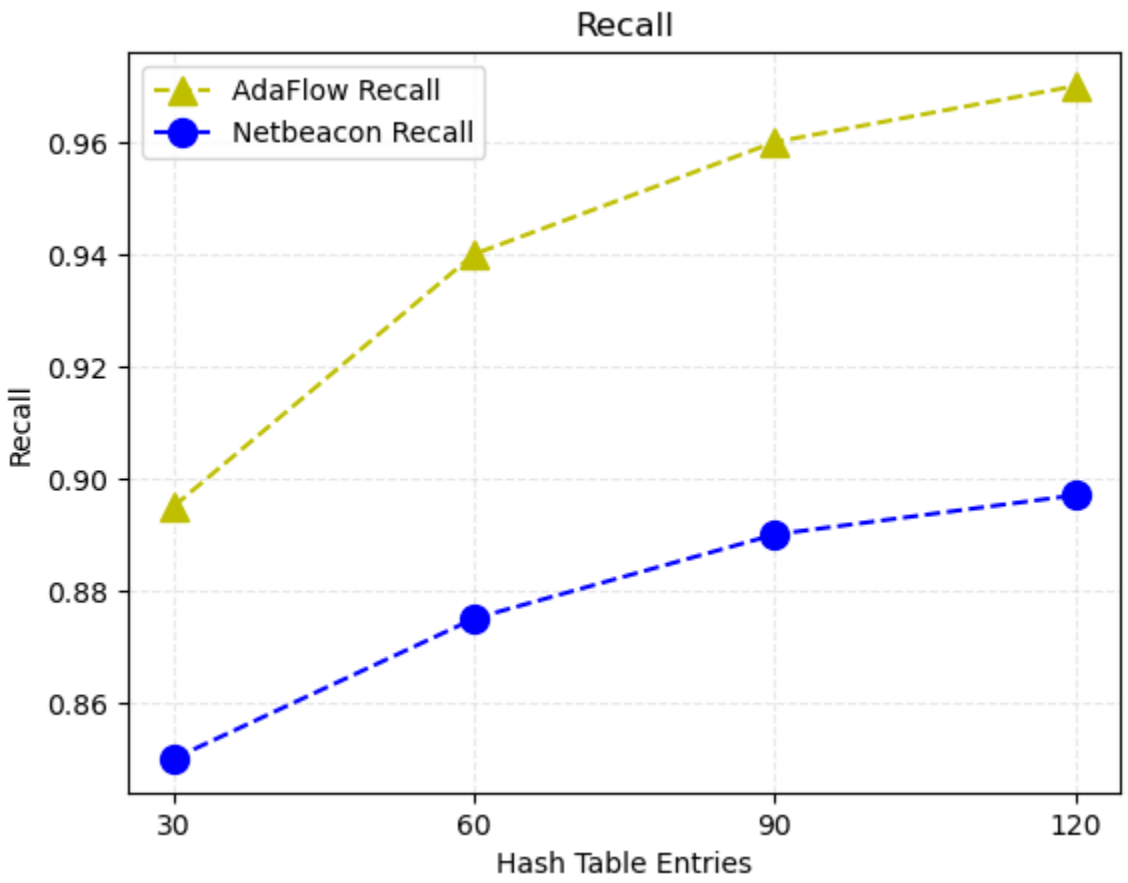
Hash table size	**Recall**
16k	0.9827623233523652
8k	0.9827576113351233
4k	0.9772380857215817,
2k	0.9726598239461909
1k	0.960562238716833
512	0.945637082754838

Accuracies for all the hash table entries were around 0.5 ie 50%. Active and Idle timeouts are additional hyperparameters used in the algorithm. The Active timeout is the maximum duration a flow can be active before being evicted, while the Idle timeout is the maximum duration a flow can be inactive before being evicted, they are set to .5 seconds and 0.00001 seconds respectively



▼ Comparison with existing system NetBeacon

Our system is getting better recall result on covert channel detection dataset compared to existing system Netbeacon



Analysis:

Adaflow could achieve better recall than Netbeacon because of differences in their design and focus.

1. Adaflow emphasizes efficient handling of cache management, evictions, and collisions. This focus allows it to better adapt to situations where the network traffic consists of a large number of malicious flows. It can identify and evict malicious flows more accurately, thus increasing the recall (the proportion of actual malicious flows that were correctly identified).
2. On the other hand, Netbeacon mainly focuses on classifying packets and flows in the data plane. While it can still classify flows, its primary purpose is not to handle cache management, evictions, and collisions. As a result, it may not be as effective in identifying and evicting malicious flows in a dataset with many malicious flows, leading to lower recall compared to Adaflow.

▼ System trained and tested on all the 11 features

When the system is simulated on 11 features (as mentioned in the Adaflow paper) with active_timeout = 5s and idle_timeout = 0.01s for all hash table sizes (30, 60, 90, and 120), the **recall was around 0.60.** This finding indicates there is a significant opportunity for improvement in the system's performance.

To improve the system's recall, we must identify the optimal values for **active_timeout** and **idle_timeout**. This involves testing out different values and assessing their impact on the system's performance. Once we have identified the optimal values, we can integrate them into the system to improve its performance.

Optimizing the active_timeout and idle_timeout values is a challenging task that requires careful consideration of multiple factors. We must ensure that the values we select are appropriate for the specific use case and do not negatively impact the system's other performance metrics.

I'm currently working on this optimization process and will continue to provide updates on my progress. By identifying the optimal values for active_timeout and idle_timeout, we can significantly improve the system's recall and overall performance.

Conclusion

In conclusion, this report provides a detailed analysis of the Adaflow system for detecting covert channel attacks.

The AdaFlow system is designed to efficiently provide high-quality telemetry data crucial for machine learning-based intrusion detection, while enabling the detection of various network attacks across diverse networks.

By leveraging the capabilities of high-speed programmable switches and properly designing the AdaFlow cache primitive, the system overcomes the challenges posed by limited stateful memory and strict constraints on per-packet operations in the switch data plane. The results indicate that the system exhibits impressive recall rates, even when trained on a single feature. However, further analysis reveals that the optimization of active_timeout and idle_timeout values can significantly improve the system's recall and overall performance.

To summarize, the observations made in this study showcase the potential of AdaFlow in enhancing network security by detecting covert attack and paved the way for further optimization and development of this promising system.