CS216 Final Report Decision Tree for Packet Classification

Yifan Zhang, Yifei Xu, Chun-Yen Lee

June 16, 2023

Abstract

Packet classification is a crucial process in packet-switching networks, enabling efficient routing, prioritization, filtering, and policy enforcement. In this project, we leverage machine learning techniques, specifically the decision tree algorithm, to implement a packet classifier using the P4 programming language. By training a decision tree on a dataset of IoT traces, we can classify incoming packets based on their attributes such as packet size and protocol type. The implementation is integrated with the Mininet network emulator for evaluation. Our results demonstrate the effectiveness of decision trees in packet classification, providing a flexible and interpretable solution for efficiently handling network traffic.

1 Introduction

Packet classification is a common process in packet-switching networking that involves analyzing incoming data packets and assigning them to specific categories or classes based on some predefined rules. Packet classification algorithms can determine the destination for each packet based on packet information such as IP address, port numbers, protocol types, and payload. By doing packet classification, switches, and routers can efficiently handle network traffic, which can enhance performance and network security.

Machine Learning (ML) has emerged as a promising technique for determining packet classification algorithms based on historical data. ML algorithms can learn patterns and make predictions based on past observations. One popular ML algorithm for classification is the decision tree. A decision tree is a flowchart-like structure where each internal node represents a feature or attribute, each branch represents a decision rule, and each leaf node represents a class label. Decision trees are widely used in various domains, including packet classification.

In this project, we aim to implement a packet classifier with P4 based on the idea from the paper[1]. We first train our own decision tree using the dataset [2] mentioned in the paper. The dataset consists of IoT traces collected on September 22nd, 2016, and includes information such as device MAC addresses, connection types, and classifications. We preprocess the dataset and classify the devices into different categories, such as smart-home devices, sensors, audio devices, video devices, and miscellaneous.

Using the decision tree algorithm, we build a packet classifier in P4 that can assign incoming packets to the appropriate category based on their characteristics. We integrate the P4 code with the Mininet network emulator to simulate a network environment for testing. We evaluate the correctness of the packet classifier by sending test packets and verifying the classification results. Our code is organized in a Github repository `https://github.com/AlexLee1999/Decision-Tree-in-P4`.

The remainder of this report is organized as follows: Section 2 provides background information on decision trees and packet classification. Section 3 describes the implementation details of our project, including preprocessing the dataset, building the decision tree, and implementing the classification algorithm in P4. Section 4 presents the evaluation results of the packet classifier. Finally, Section 5 concludes the report and discusses potential future work.

2 Background

2.1 Decision Tree

A decision tree is a widely used machine learning algorithm for classification and regression tasks. It is a flowchart-like structure where each internal node represents a feature or attribute, each branch represents a decision rule, and each leaf node represents a class label or a value to predict. Decision trees are particularly useful for their interpretability and ability to handle both categorical and numerical data.

The process of constructing a decision tree involves recursively partitioning the data based on different features and their values. The goal is to find the most informative features that split the data in a way that maximizes the separation between classes or reduces the variance in the case of regression. Various algorithms, such as ID3, C4.5, and CART, are commonly used to build decision trees.

Decision trees have several advantages. They are easy to understand and interpret, allowing human experts to validate the generated rules. Decision trees can handle both categorical and numerical features and can handle missing values by employing different splitting strategies. They also provide feature importance rankings, indicating the relevance of each feature in the classification process.

2.2 Packet Classification

Packet Classification is an essential element in packet-switching networks. The goal of packet classification is to efficiently route, prioritize, filter, or apply policies to network traffic. Several algorithms have been proposed to solve the packet classification problem.

One of the approach to solve packet classification problem is divide and conquer. Bit vector linear search, an algorithm proposed by Lakshman *et al.* [3], exploited the idea of divide and conquer to solve packet classification problem. For each value in the database, the algorithm uses a bitmap to indicate the rule which the value matches. If the value matches rule i, the ith bit in the bitmap is set, otherwise it's 0. By using the bitmap, we can do and operations and find out the rule which the packet matches.

Alternatively, decision trees offer an interpretable and efficient solution. By constructing decision trees based on packet attributes such as packet size, protocol type, and source/destination IP addresses, it is possible to classify packets into different categories or apply specific actions based on the decision tree's rules. This approach provides flexibility and adaptability in handling various network traffic scenarios.

3 Implementation

In this project, we use P4 to define the behavior of our switches. We focus on implementing decision-tree-based packet classifier with P4. To test the result, we use mininet to build the network topology, and send packet from virtual source node to the virtual destination node via the switch. To integrate P4 and mininet efficiently, we use the framework from P4 tutorials[4] to build our project.

3.1 Pre-processing the Dataset

The authors in the paper [2] have collected data of IoT traces and organized them into csv files. We use one of the dataset collected on September 22nd, 2016 as our training set. Also, the authors provide the mapping between the devices and the devices' MAC addresses. Based on the paper [1], we classify these devices into five categories, which are smart-home devices, sensors, audio devices, video devices, and miscellaneous. We implement a Python script to process the csv file and add the categories in the file base on the source MAC addresses. A sample of the devices and their classification is listed in Table 1.

3.2 Implement the Classification Algorithm in P4

Implementation details of Decision Tree:

After data processing is done, we built the decision tree with Gini impurity splitting criterion. When building the decision tree, the max depth we chose is 3, and the only features we used is packet size and IP protocol. Details on our choice is in the evaluation section.

Device Name	MAC Address	Connection Type	Classification
Smart Things	d0:52:a8:00:67:5e	Wired	smart-home devices
Amazon Echo	44:65:0d:56:cc:d3	Wireless	audio devices
Netatmo Welcome	70:ee:50:18:34:43	Wireless	smart-home devices
TP-Link Day Night Cloud camera	f4:f2:6d:93:51:f1	Wireless	video devices
Samsung SmartCam	00:16:6c:ab:6b:88	Wireless	video devices
Dropcam	30:8c:fb:2f:e4:b2	Wireless	video devices
Insteon Camera	00:62:6e:51:27:2e	Wired	video devices
Insteon Camera	e8:ab:fa:19:de:4f	Wireless	video devices
Withings Smart Baby Monitor	00:24:e4:11:18:a8	Wired	video devices
Belkin Wemo switch	ec:1a:59:79:f4:89	Wireless	miscellaneous
TP-Link Smart plug	50:c7:bf:00:56:39	Wireless	smart-home devices
iHome	74:c6:3b:29:d7:1d	Wireless	smart-home devices
Belkin wemo motion sensor	ec:1a:59:83:28:11	Wireless	sensors
NEST Protect smoke alarm	18:b4:30:25:be:e4	Wireless	sensors
Netatmo weather station	70:ee:50:03:b8:ac	Wireless	sensors
MacBook	ac:bc:32:d4:6f:2f	Wireless	Misc.
Android Phone	b4:ce:f6:a7:a3:c2	Wireless	Misc.
IPhone	d0:a6:37:df:a1:e1	Wireless	Misc.
MacBook/Iphone	f4:5c:89:93:cc:85	Wireless	Misc.

Table 1: A sample list of the device and its classification

Automated Parsing and Translation of Decision Tree:

To implement decision trees in P4, we created an automated parser that reads decision tree logic from a text file and generates P4 code to represent this logic. The parser is implemented in Python.

Reading Decision Tree Logic:

The decision tree logic is read from a text file where each line represents a set of conditions followed by an action, for example:

```
when pkt_size<=155 and ip_protocol<=11 then 4;
```

A regular expression is used to extract the conditions and action. The parser iterates through each line, extracts the conditions, and the action. Decimals in the conditions are removed.

Generating P4 Code:

Based on the parsed conditions, the Python script generates P4 code that represents the decision tree within the ingress control block of a P4 program. Each set of conditions is translated into nested if-else statements. An example of the generated P4 code for a node of the decision tree is as follows:

```
if (pkt_size<=155) {
    if (ip_protocol<=11) {
        set_custom_value(4);
    }
}
```

This code is generated dynamically based on the decision tree logic read from the file.

Integration with P4 Program:

The generated P4 code is inserted into a template P4 file which defines the necessary headers, parser, and deparser logic. The P4 program is structured to parse the packet headers and extract relevant fields such as packet size and protocol. The decision tree logic is placed inside the ingress control block. In our example, the action of setting a custom value (TTL in the IPv4 header, in this case) is executed based on the conditions in the decision tree.

Action Execution:

The generated P4 code makes use of a custom action ('set_custom_value') to set the TTL field of the IPv4 header. The action takes an 8-bit value as a parameter. This action is executed whenever the conditions of the decision tree are satisfied.

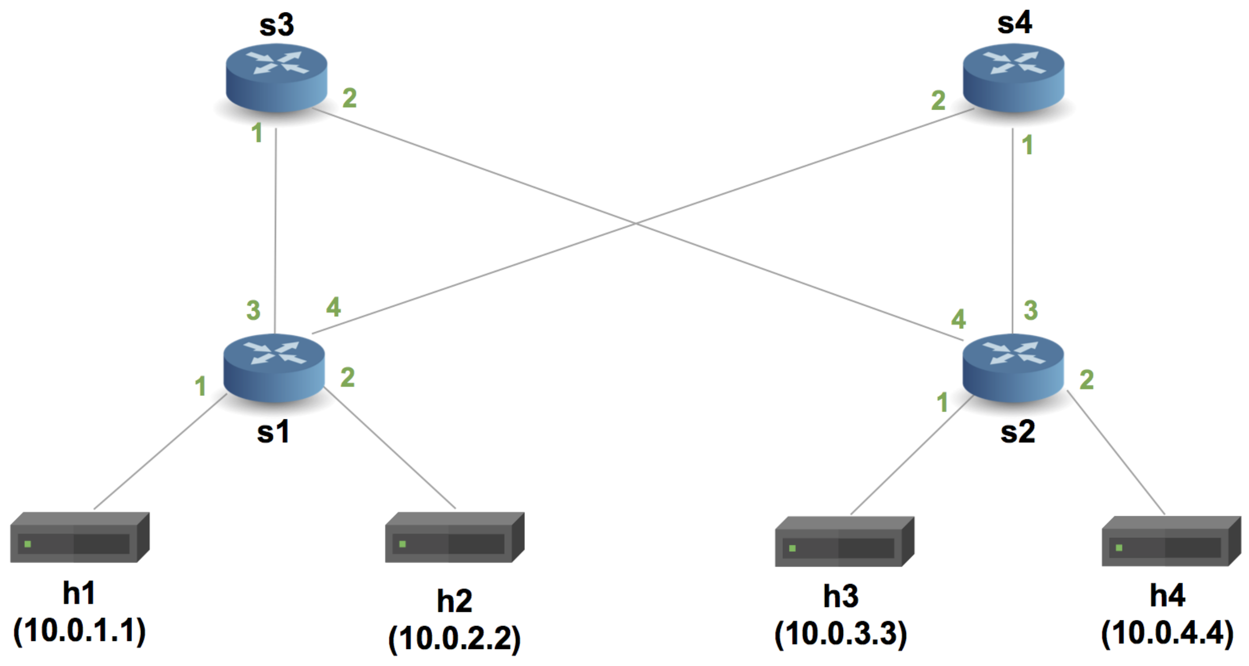

Figure 1: Network topology

3.3 Deploy the Classifier on Mininet

The P4 tutorial [4] provided a complete framework to run our project.

Compiling P4 code and Initialize Mininet with Topology Json File

The framework can help us initialize the Mininet network with compiled p4 code and user-configured topology Json file. To simplify the project, we use existing topology defined in the repository. The network topology is shown in Figure 1.

Sending and Receiving Packets

In addition to building the Mininet, the repository also includes two Python scripts, receive.py and send.py. The receive.py script allows the host in the Mininet network to receive packets via a specific Ethernet interface. The send.py script allows the user to send a packet to a specific IP-address with a user-defined payload. We modify both scripts to support UDP packet sending. Moreover, we modify the send.py script to pass the payload length as an argument, which can allow us to send a packet with a specific length of payload.

4 Evaluation

4.1 Building the Decision Tree

Our consideration on which feature to choose is mainly about generality and practicality. Given that our task is to classify different devices, addresses should not be the feature used in building the tree since there is no strong relation between kind of devices and addresses. In order to choose features, we noticed that one kind of device tend to send only certain kind of packets. For example, the streaming camera on your front door tend to only send UDP packets of some certain sizes. Therefore, the features we chose is packet size and IP protocol.

In terms of maximum tree depth, we set it as 3. The dataset provided has only a few devices and 5 classes. Although the amount of data points is about 800,000, the variations in our dataset is limited since one kind of device send only certain kind of packets. For building decision tree with limited variations, a deep tree would suffer from overfitting. Thus, we decided to use shallow decision tree. We ran k-fold

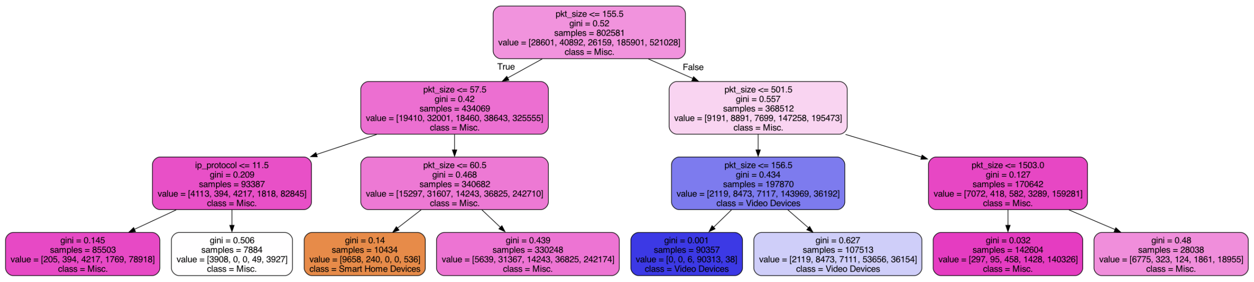

Figure 2: The decision tree diagram

Rule	Description	TTL Value
1	$pkt\_size \leq 155.5$ && $pkt\_size \leq 57.5$ && $ip\_protocol \leq 11.5$	5
2	$pkt\_size \leq 155.5$ && $pkt\_size \leq 57.5$ && $ip\_protocol > 11.5$	5
3	$pkt\_size \leq 155.5$ && $pkt\_size > 57.5$ && $pkt\_size \leq 60.5$	1
4	$pkt\_size \leq 155.5$ && $pkt\_size > 57.5$ && $pkt\_size > 60.5$	5
5	$pkt\_size > 155.5$ && $pkt\_size \leq 501.5$ && $pkt\_size \leq 156.5$	4
6	$pkt\_size > 155.5$ && $pkt\_size \leq 501.5$ && $pkt\_size > 156.5$	4
7	$pkt\_size > 155.5$ && $pkt\_size > 501.5$ && $pkt\_size \leq 1503.0$	5
8	$pkt\_size > 155.5$ && $pkt\_size > 501.5$ && $pkt\_size > 1503.0$	5

Table 2: List of rules in the packet classifier

cross validations with different maximum tree depth and found out that our decision tree achieved highest validation accuracy at tree depth of 3.

4.2 Evaluate the Correctness of Packet Classification

The decision tree diagram is shown in Figure 2. Since the depth of the decision tree is 3, the total number of rules is 8, which are listed in Table 2. To test the correctness of all the rules, we use the send and receive Python script to send test packets. In this test, we use two nodes, h1 as sending node and h2 as receiving node, in the Mininet topology to send and receive packets.

Table 3 shows our testing results. From Cases 1 to 8, we design specific test cases for each rule and use the modified Python script to send the special packet. For each rule, except rule 8 performs as excepted. For the test case for rule 8, the Maximum transmission unit(MTU) for the Ethernet interface in Linux OS is 1500, and the transmitted packet size exceeds the MTU. From Cases 9 to 14, we also use UDP as the protocol to check if the packet classifier worked as expected. Except for Case 14, every case worked as expected.

5 Conclusion

In this project, we successfully implemented a packet classifier using the decision tree algorithm in the P4 programming language. By training a decision tree on a dataset of IoT traces, we were able to classify incoming packets based on their attributes such as packet size and protocol type. Our implementation demonstrated the effectiveness of decision trees in packet classification, providing a flexible and interpretable solution for efficiently handling network traffic. The integration of P4 and Mininet allowed us to evaluate the correctness of the packet classifier in a simulated network environment. Future work could involve further optimization of the decision tree model, exploration of other machine learning algorithms for packet classification, and the integration of additional features for more accurate classification. Overall, this project highlights the potential of machine learning techniques in enhancing packet classification in packet-switching networks.

Case	Payload Size	Protocol	Packet Size	Received TTL	Excepted TTL	Rule
1	3	TCP	43	5	5	1
2	3	UDP	31	5	5	2
3	20	TCP	60	1	1	3
4	40	TCP	100	5	5	4
5	116	TCP	156	4	4	5
6	200	TCP	240	4	4	6
7	1000	TCP	1040	5	5	7
8	2000	TCP	2040	Packet exceed MTU	5	8
9	32	UDP	60	1	1	3
10	72	UDP	100	5	5	4
11	128	UDP	156	4	4	5
12	212	UDP	240	4	4	6
13	1000	UDP	1028	5	5	7
14	2000	UDP	2028	Packet exceed MTU	5	8

Table 3: Results of testing the packet classifier

References

[1] Z. Xiong and N. Zilberman, "Do switches dream of machine learning? toward in-network classification," in *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, HotNets '19, (New York, NY, USA), p. 25–33, Association for Computing Machinery, 2019.

[2] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Classifying iot devices in smart environments using network traffic characteristics," *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1745–1759, 2019.

[3] T. V. Lakshman and D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," *SIGCOMM Comput. Commun. Rev.*, vol. 28, p. 203–214, oct 1998.

[4] p4lang, "P4 tutorials." `https://github.com/p4lang/tutorials`.