Identification and Prioritization of Multimedia Traffic in Wireless Access Points

A Thesis

submitted in partial fulfillment of the

requirements for the degree of

Master of Technology

by

Vidya Sagar Kushwaha
&
Hiren Patel

under the guidance of

Prof. Mythili Vutukuru

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai 400076 (India)

June 2016

Abstract

Now a days, computer networks are growing at a very large scale and various new applications are also being evolved. As a result, it becomes more and more difficult for the network administrators to manage the network resources. Since link bandwidth is a limited resource, it is always advisable to have some framework to allocate it to the various running applications depending on their need, while maintaining high bandwidth utilization. But most of the time, network administrators don't have complete knowledge of all the applications running on their network. It becomes important to identify what applications are running and take actions accordingly. Some already proposed approaches can classify the network traffic off-line and some can do the same in the real-time scenario.

In the past, there has not been much work done towards improving quality of multimedia traffic in wireless channel. In this report, we illustrate an approach which focuses on identifying multimedia (audio/video) streaming traffic among other traffic flows, in real time and then prioritizes them. The approach can work in wireless access points. The model uses a training set and supervised machine learning algorithms to classify the traffic. We implement this approach using decision tree classifier as well as K-NN algorithm. The results show that this model is able to classify with 90% offline training accuracy and 87% real time classification accuracy for decision tree algorithm. The results also indicate that the latency and jitter of multimedia flows are reduced by a factor of 36 and 39 respectively. Our approach improves multimedia streaming quality and the average increment in MOS comes out to be 0.7 on an average.

Acknowledgements

We are very thankful to our advisor Prof. Mythili Vutukuru for her valuable guidance and kind support throughout the work. Frequent and regular meetings were very helpful and they always kept us energized. There were times when we failed at some point but she was there to motivate and guide us. It would not have been possible to successfully complete this work without her supervision. We would like to express our gratitude towards Mukulika Maity for helping us in the initial experimental setup and related stuff. We are also thankful to Sridhar Pratapa for his valuable time and help during setting up mikrotik board and other things.

Contents

Chapter 1

Introduction

Traffic classification refers to the task of capturing the traffic by placing some traffic monitoring tool on the network link and identifying a network flow as some application for example HTTP, SMTP, FTP etc., or an application class, for example, video streaming, bulk data transfer etc. Traffic classification can be offline or online. In offline traffic classification, a flow is classified after it's completion. Such kind of classification is helpful in analyzing network usage statistics, efficient planning of network resources and forecasting for provisioning in long-terms. If we can classify a flow while it is still in progress, then it is referred as online or real time classification. Real time classification can help in taking immediate actions such as prioritize, de-prioritize or block the flow, just after the flow has been classified.

1.1 Problem Definition

Traffic classification in a wired and wireless medium has some differences. In wireless AP, queue get builds up at AP because the bandwidth of wireless AP is less than that of wired connection and thus it needs to decide which packets it should forward first, or which flow it should give more priority. Here comes traffic classification in the picture. Traffic classification can help in identifying the flows that need higher priority as per user requirement. Multimedia requires proper bandwidth and it also feels annoying when the streaming does not happen smoothly. This work focuses on developing a model that can solve these two issues. Our model can work in wireless AP setup, and identify multimedia (audio/video) traffic flows among other traffic in real time, and then give them higher priority so that the end users may get better quality.

1.2 Motivation

Different type of applications have different requirements. For example, interactive applications have very less bandwidth constraint but have time constraint, whereas file transfer applications need more bandwidth but do not have time constraint. So we can provide better quality of service by allocating different flows different amount of resources, once we are able to identify the application type.

We have seen a significant shift in the Internet usage pattern from earlier text based data to multimedia contents. In our work, we are trying to improve the streaming quality of multimedia. The motivation behind choosing this class is that if heavy download is going on in parallel then streaming flows do not get the required bandwidth, further the quality starts deteriorating and often the streaming stops for a moment in between. Most of the times, this is not acceptable.

1.3 Related Work

There are many approaches proposed in the past. Some commonly used techniques were based on the transport layer ports. Such methods are not reliable today because many applications use the same port for different kind of traffic, for example port 443 can be used for web browsing and also for video streaming. There are some applications which use random port numbers intentionally, so that they do not get caught.

Another approach can be deep packet inspection. It involves looking into content of IP packet and based on the payload, we classify the flow. This has following concerns.

1. Violation of user data privacy
2. More time complexity
3. If the payload is encrypted, this approach fails completely.
4. Classifier must know syntax of the every application's packet payload which is not the case most of the times.

Some approaches believe that the statistical patterns can be effective to classify the traffic. Roughan [1] used packet and flow level statistical features such as packet size, total flow duration etc., to classify the flows. In another approach, Moore [2] calculates the probability (using Bayes rule) of the given test flow belonging to each of the traffic class in consideration, and finally assigns the class label for which the probability is the highest. It assumes that all features used are independent of each other, which is not true.

Bernaille [3] has proposed an approach which uses the size of first 5 packets (in both the direction) to classify the flows. In another paper, Crotti [4] defines a term statistical fingerprint, which is derived from three basic features, i.e., packet sizes, inter-arrival times and arrival order of packets of a flow. It also defines a term *Anomaly Score* for a flow, which tells *"how far"* the given flow is from a particular application (against which the score has been calculated). A flow is assigned the label of the application for which anomaly score is minimum. These two approaches [3], [4] heavily depend on the packets arrival order which is the major drawback, since packet loss/re-ordering is un-avoidable.

Erman [5] proposes a semi-supervised classification approach which takes few labeled and many un-labeled flows to build the classifier. This approach tries to address the problem of getting large number of labeled training flows. Erman uses clustering algorithm to label the flows and also gives the provision to label a flow as *new/unknown* flow, if that flow is *very far* from all the given classes.

AlSabah [6] classifies the traffic online into interactive, streaming and bulk data transfer classes with 77% accuracy in Tor network. They use first 3000 packets and features as circuit lifetime, packet inter-arrival time and number of cells sent recently. Their approach is useful only in the Tor network.

In contrast to statistical features, there is another approach [7] which tries to study and analyze the behavior of the source host, which is generating the traffic flows and then, accordingly label the flows coming out of that host. This approach does not use packet size or the timing of the packets. It observes how a host behaves at transport layer and tries to find a pattern at social, functional and application level. It builds a graphlet for that host and uses graph matching to classify the host and thus indirectly classifies the traffic. This approach can't be applied in the real-time scenario.

Broido [9] has proposed an approach which classifies traffic into P2P or Non-P2P using TCP/UDP protocol usage and IP/port connection characteristics. They have developed some heuristics for traffic identification. The first heuristic they have used is that if the source and destination IP both concurrently use TCP & UDP then the flow is P2P. The second heuristic is that for any flow if the ratio of the number of distinct ports to the number of the distinct IPs it is connected is 1 then flow is P2P. In web, this ratio is >1 because of multiple parallel connection to the server. They reduce false positives using port number of mail server, DNS and using a fixed packet size in the case of gaming applications.

Graption [8] is a graph based detection approach to classify the traffic as P2P or non-P2P. It tries to capture the network wide interaction among the hosts which are dealing with similar kind of flows and creates a Traffic Dispersion Graph (TDG). It analyses this graph using various graph metrics, i.e., average node degree etc., to distinguish between the graphs associated with different applications. This approach is not applicable in the real-time scenario.

None of these papers has focused on multimedia traffic identification in real time by using only packet characteristics.

1.4 Our Contribution

Our approach uses the statistics of the network flows to capture the behavior of the traffic and classify them using supervised machine learning algorithms. We came up with a feature set which is able to classify multimedia and download flows using first 1000 packets. The feature set includes average window size, number of packets with push flag set, average packet size, standard deviation of packet sizes, average inter-arrival time, standard deviation of inter-arrival times, flow duration and total bytes. We design a classification model which can work in wireless APs. The idea is to implement this logic inside wireless APs with some additional work. We use two ML algorithms, i.e., decision tree classier [10] and K-NN algorithm [11]. Our classifier can also learn the rules at regular intervals. The model just needs basic informations that is available in the packet header. The model classifies the traffic and if a particular flow is identified as multimedia then that flow is provided higher priority so that the streaming experience will be better. We have developed an approach which can be used to label the given training set with minimal manual inputs.

We achieved the offline training accuracy of 87% using KNN algorithm and 90 % with decision tree algorithm. The classification module works with the real time accuracy of 87 % using decision tree classifier. Latency and jitter are also reduced by a factor of 36 and 39 respectively. On an average, MOS score [12] for multimedia flows also increases by 0.7 with our solution.

Chapter 2

System Setup and Labeling Training Data

In this chapter, we describe the setup used to collect traces. We also talk about the procedure we used to label the training data using various heuristics.

2.1 Experimental Setup

In this section, we explain our setup which we used to for trace collection, prioritization and also for real time testing. We want to classify the traffic and prioritize the multimedia flows among all the flows. Ideally, both the tasks should be done in a wireless access point (AP) itself. But as of now, we simulate AP by placing another machine (from now onwards we will refer this machine by saying desktop) just before it. This desktop does all the processing on behalf of the AP, i.e., classification and prioritization. Algorithms & heuristics used in desktop can be implemented in the AP, although it might require some extra work.

Figure 2.1 shows our experimental setup. We used desktop, which has two network interface cards (NIC), where eth0 is connected to the Internet and eth1 is connected to the switch. Before reaching the AP, all the traffic goes through desktop where we do classification and prioritization. Desktop forwards traffic through eth1 interface, which is connected to the switch and subsequently to the AP. Following scripts are executed in the desktop.

1. *nat.sh* : It is used for natting [13]. It translates private ip to public ip for all the packets going outside of the private network, i.e., packets going to the Internet. Similarly, it translates public ip to private ip for all the packets coming from Internet. It changes the source IP of the packets going to the Internet, to its own IP address (eth0 IP address).

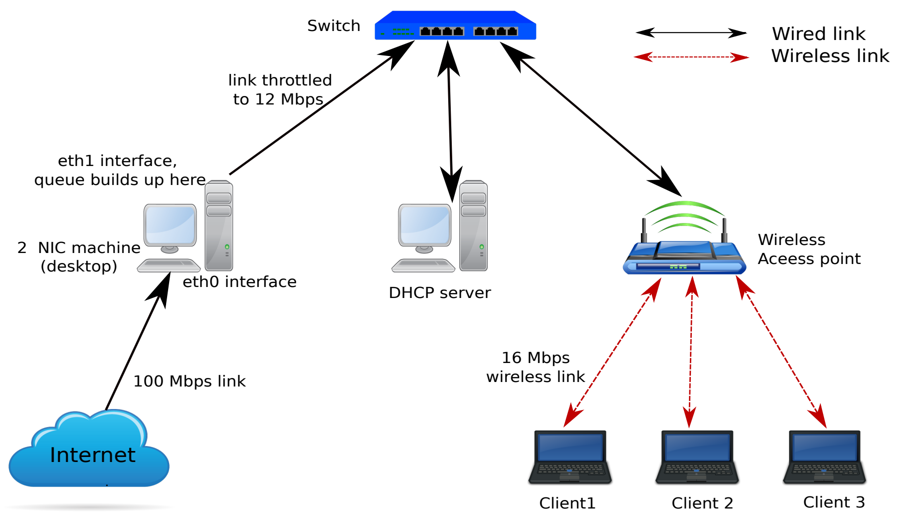

Figure 2.1: Experimental setup

2. *2interface.sh* : It is used to make desktop as a bridge between AP and Internet. It forwards all the packets from eth0 to eth1 interface and vice-versa.

In this setup, we need a DHCP server to provide IP addresses to the AP. We modified two files */etc/dhcp/dhcpd.conf* and */etc/network/interfaces* in DHCP machine as per our requirements. Following changes were made in dhcpd.conf file.

```
range 172.16.1.10 172.16.1.50             // assign a range of IP address
option domain-name-servers 10.200.1.11    // IITB DNS server address
option routers 172.16.1.60                // gateway IP (eth1 interface of desktop)
```

We manually assign an IP addresses to DHCP server (172.16.1.70) and gateway (172.16.1.60). Once DHCP server is started, it assigns IP address to all the devices connected to the switch.

Usually the wireless link (54 Mbps) is slower than the wired link (100 Mbps), so in real scenario (without desktop) when AP is directly connected to high speed Internet (100 Mbps), queue builds up at the AP. We need to prioritize audio/video flows at the node, where the queue builds up. We want the queue to build up at the desktop instead of the AP. This can be done by manually throttling the link between eth1 interface and AP such that its bandwidth is just lower than that of the wireless link.

After throttling, the link between the eth1 interface and AP becomes the bottleneck, so queue builds up at eth1 interface. To measure the bandwidth of wireless link, we used *iperf* [14] bandwidth measurement tool, we found the link bandwidth to be nearly 16 Mbps. We manually throttled the link (between the eth1 interface and AP) to 12 Mbps using Hierarchy Token Bucket (HTB) queuing discipline. HTB is described in detail in section 5.1.

Desktop machine has 2 GB RAM and 2.3 GHz Intel core 2 quad q8400 processor. We use Airtight SS-300-AT access point, which uses 802.11g technology. This AP operates at 2.4 GHz central frequency and uses 20 MHz channel width. We did not put any security (password protection) in the AP because we need packet header information. If the AP is password protected then AP encrypts all the packets and we can not get the header information. In real implementation, classification happens inside the AP itself. In the AP, the packet header is decrypted but clients connected to AP see encrypted packets. So in reality we can use encryption in AP, i.e., we can secure our AP with a password, but at the same time the AP can still read the packet header.

2.2 Trace Collection

In the previous section we explained the experimental setup. In this section, we show our trace collection process. We use supervised machine learning algorithms for classification. These algorithms use training data to predict the class of the given instance. The training data consists of a large number of labeled instances. To build the training data, we collected traces of audio, video and download flows using Wireshark. We captured traces manually using the setup shown in the figure 2.1. Throughout the experiment, the link between desktop and AP was throttled at 12 Mbps.

All the traces were taken for 4-5 minutes duration and collected in isolation, i.e., performing single streaming or download at a time. We also collected parallel traces. While collecting the traces, we ensured that our traces do not contain the unwanted packets, e.g., after saving a trace file, we generally wait for 1-2 minutes before collecting another trace because otherwise some old packets, retransmitted from the previous connections may get captured in the new trace. For audio traces, we visited 10-12 most popular audio streaming websites. We also took traces of low quality (144p) videos to see if it can create any confusion and reduce the accuracy in classification. We took various types of download traces, e.g. FTP, software download, video download, etc. We also took download traces from various servers located in different countries using Azure servers [15]. We manually collected the traces nearly 400 times, consisting of 20 GB of data containing 12 million packets.

2.2.1 Trace Collection on Smart-phones

Nowadays, mobile phones are commonly used for audio/video streaming and file download in the browsers. Every day new applications are being launched for multimedia streaming. Browsing and streaming behaviour might be different for the mobile phones as compared to the PC. Even different applications have different properties, i.e., some apps buffer the song in advance, some apps load the song as per requirement. So a model which is built using only PC traces might not give much accuracy when we use mobile apps. So we also took traces from various mobile applications like Youtube, Saavn, Gaana, Starsports etc. From each app we took 10-15 traces. We also took download traces from the mobile phone's web browsers, i.e., software download, android apks etc.

2.2.2 Multiple Multimedia/Download Trace Collection

Here "Multiple multimedia trace collection" means that we stream more than one multimedia in different tabs at the same time. Flow features like avg. IAT and its deviation etc., depend on the number of parallel multimedia streaming or downloading files in parallel, e.g., if there are many video streaming together, then average IAT will increase because queue will build up at the AP. In this scenario, real time classification will give incorrect results. To cover this aspect, we also took traces of many multimedia/download flows at the same time. At a time, we downloaded up to 3-5 files together or 3-5 video streamings together.

2.3 Building Training Data Set

In the previous section, we described trace collection process. Once we have all the traces, we need to convert them into proper training set. One pcap (trace file) file has many flows, e.g., one trace file of audio song may consist of 10-20 flows. Out of these flows, only 1 or 2 flows belong to audio streaming, others are very small duration dummy web flows. These dummy flows have very less number of packets and some of these flows have very large inter arrival time, sometimes >3 seconds. We need to remove these unwanted flows to build the correct training set. To filter out the valid flows, we used two different methods 1. Based on packet characteristics (next subsection) and 2. Based on HTTP GET request automation - Auto labeling heuristics (section 2.4).

To get correct classification results, training set must be balanced. If the training set is biased towards one class, then most of the flows will be assigned to that class and accuracy percentage might still be high. Equal number of trace files of each class may not result in a balanced training set, so we need to balance the number of flows for each class. Before merging all the flows from both the classes in the final training data set, we looked for the class having less number of flows and then equalize the flows of other class to that number. Now we have equal number of flows for both the classes.

Based on Packet Characteristics

This approach was used in MTP stage-1. Here heuristic was on the number of packets in a flow and average packet size, i.e., if the number of packets in a flow is >1000 and average packet size is >800, then only the flow will be considered as valid. We found these threshold values empirically.

2.4 Auto Labeling Heuristics

This approach was used in MTP stage-2. In stage-1, we used heuristics to filter out the valid flows based on average packet size and number of packets in the flow. Earlier, we built training set using this rule (based on packet characteristics) and developed the model which achieved 90% training accuracy (offline). But real-time accuracy was very less, nearly 65-70%. Training accuracy can be high due to over-fitting [16]. Later we realized that our heuristic might not be appropriate, so we looked for alternatives to make our training set more accurate. Some research papers use payload based classifier to create labeled training data. We talk about this payload based classifier approach in the next section.

2.4.1 Payload Based Classifier and Issues

Payload based classification refers to searching particular string formats in the entire packet payload (user data). This approach works good for specific applications, e.g., every packet of BitTorrent application has string “0x13Bit” in its payload at some fixed location and by looking at this string, we can say that given packet belongs to Bittorrent. But this method works for specific applications only. For multimedia we have thousands of websites and applications, so it is not feasible to have the knowledge about different formats used by these applications.

We tried some approach similar to this, we checked multimedia file formats in the packet payload, e.g., searching ".mp3" in the entire packet payload. There is no any specific address/location in the payload where we can find this format. This approach worked for some websites, but we had too many false positives. The reason is that there are too many multimedia formats (we considered nearly 30-40 different file formats). Even some of the download packet's payload also contain these formats, e.g., ".aac" (audio file format) text found in download packet's payload. So it seems that even if we find these multimedia file format in the payload then also we can not be very sure that this packet indeed belongs to multimedia.

Pure payload based classifier can not be used to build labeled training data because of non availability of unique format and diversity in the nature of applications. We used another alternative to this payload based classifier which is explained in the next section.

2.4.2 Using HTTP GET Request & Other Heuristics

As we discussed that payload based classifier can not be used to label the training data, we use another alternative to label the training data. Whenever we do multimedia streaming, a HTTP GET request is issued from the client side consisting the file name (e.g., abc.mp3). This GET request is sent in the very first few packets of the multimedia connection. For one connection, we will have one GET request followed by the packets for that audio/video file. We look for multimedia formats in the URI (Uniform Resource Identifier) of HTTP GET request. It is also a kind of payload inspection. In payload based classifier, we look into the entire HTTP payload (user data) while in this approach, we just look into the URI part of HTTP header which is again a part of TCP payload. But our approach is faster because we need to look into only URI, not into the complete payload. If we find any multimedia format (e.g., .mp3, .mp4, .flv etc.) in the URI then we consider that flow as multimedia. During the trace collection, we observed that this HTTP GET request approach is not used in all the websites. Training set labeled using this method will be more accurate, although we can not identify all the multimedia connections in the given trace file.

This approach (left branch in figure 2.2) works in two passes. First we list down all the client's port addresses from which HTTP GET request was sent. In next pass, we fetch all the packets from the trace file, which are incoming to any of the port addresses in the list. For all these packets, we store the required information (arrival time, source IP, destination IP, source

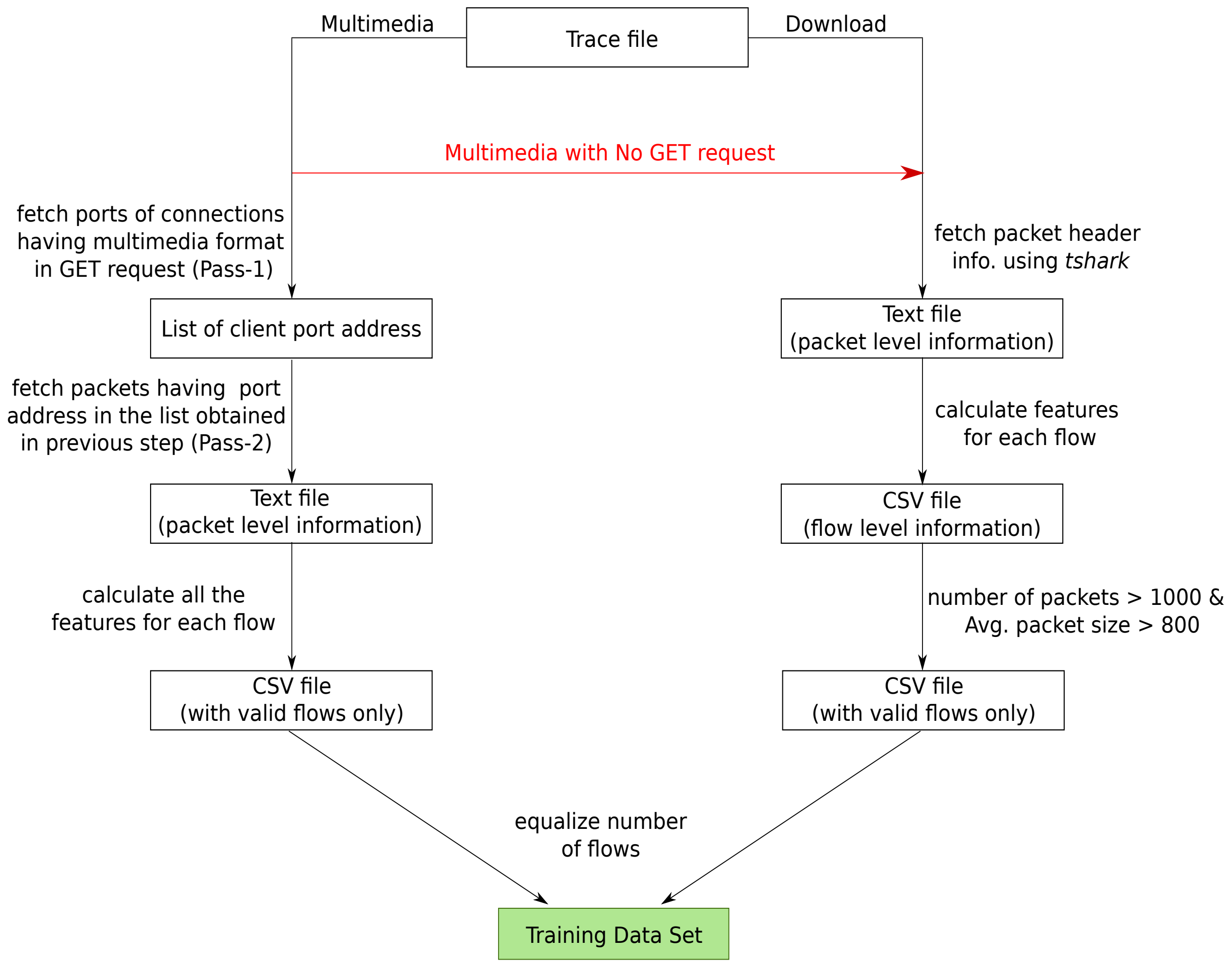
Multimedia
Trace file
Download
Multimedia with No GET request
fetch ports of connections having multimedia format in GET request (Pass-1)
fetch packet header info. using *tshark*
List of client port address
Text file (packet level information)
fetch packets having port address in the list obtained in previous step (Pass-2)
calculate features for each flow
Text file (packet level information)
CSV file (flow level information)
calculate all the features for each flow
number of packets > 1000 & Avg. packet size > 800
CSV file (with valid flows only)
CSV file (with valid flows only)
equalize number of flows
Training Data Set

Figure 2.2: Building Training Set

port, destination port, packet size, window size and push flag) in a text file. We have one entry for each packet in this file. We use python scripts which take this text file as input and generate flow level information in a csv file. This csv file has exactly one entry for every multimedia flow present in the trace file. Each entry contains various feature values, i.e., average packet size, standard deviation of packet size, average inter arrival time (IAT), std. deviation of IAT, flow duration, etc.

We automated the process of labeling the training set from the trace files. While collecting the traces, we follow some naming convention for trace files according to its class. This was necessary because we use different heuristic for download traces.

As mentioned earlier, this HTTP GET request approach is not applicable to all the websites or applications. Multimedia websites for which we do not have HTTP GET request, we use the older approach based on average packet size and number of packets (same approach which was used in MTP stage-1). As we know that given trace file is for multimedia (using naming convention), we extract all the large flows and label them as multimedia. We use this approach for download traces as well because they do not have specific file formats in the GET request. Overall process is shown in figure 2.2.

The process is not completely automated, we need few manual inputs while collecting the traces. We need to follow some kind of naming convention for trace files so that we can identify which trace file belongs to which class and accordingly we can apply the relevant heuristics to label the training set.

2.4.3 Analysis of Connections for Various Websites

Different websites/applications have too much diversity in the GET request behaviour. We list down some of the observations below.

- For some websites, we get only one flow for the HTTP GET request and that one flow is responsible for the complete audio/video streaming.
- For some websites (e.g. Starsports.com), we get nearly 50 HTTP GET requests and out of those, only few flows are responsible for video. Even some connections have less than 10 packets.
- For some websites, we have many parallel connections with HTTP GET requests and all the connections contribute equally.
- For some websites (e.g. Youtube), we do not have any HTTP GET requests and it seems that they use some different mechanism.

2.4.4 Confidence Measurement of HTTP GET Request Approach

GET request heuristics is not so powerful that we can use it for real-time classification inside an AP. We considered many websites and out of which nearly 60% of the websites use this GET request approach. Overall labeling accuracy was 90% for those websites, with 10.6% false negative and 4.4% false positive. False negative is due to the flows which do not have GET request with multimedia formats but still it is actually multimedia. There are some download flows which has some of the multimedia formats in its URI which causes false positives. This happens rarely, so we have very less false positives.

Since we can not cover all the websites using this GET request approach, this method is only useful in labeling the training data for which it has 90% accuracy. Using this training data, we create the model which is generic enough to classify the rest (40% of the websites for which we do not have GET request) of the websites. So this GET request method was used to build the robust training data only, and not to classify the traffic.

Chapter 3

Training Phase

By this time, we have collected the required trace files belonging to different traffic classes and we have automated the process of labeling the training set. In this chapter, we describe how we train our classifier with different feature sets. We also need to decide about the threshold point of the flows, after which we can classify the flows with high accuracy.

3.1 Feature Selection

The very first thing that we need to classify the traffic is a good feature set. A feature or discriminator is an attribute whose value differs for different types of traffic classes. For example, average packet size is a feature, generally it differs for web surfing flows and download flows because in download flows almost all packets are of full size which is not the case with web surfing flows. But having just one feature is not enough to classify the traffic. So we need to dig into more features. Some other features can be standard deviation of packet sizes, number of distinct host that the client host is communicating, average inter-arrival time (IAT), standard deviation of IAT, flow duration, total bytes etc. There are many more features that one can use.

But the next question arises that how many features should we use? Should we use all the features? The answer is *no*. As we know that calculating feature values takes CPU time and memory, and also taking large number of useless features might degrade the classification accuracy. So we should choose minimal set of features that have good discriminating power.

Initially we selected 6 features, i.e., average packet size, standard deviation of packet sizes, average inter-arrival time (IAT), standard deviation of IAT, flow duration and total bytes. We call this set as *feature set-1* in the rest of the report. It is intuitive to have average IAT as a feature because it can differentiate between download flows and multimedia flows. Generally average IAT for download is less as compared to multimedia or web surfing flows, as the packets arrive at a very fast rate. Similarly, the standard deviation of IAT is lower for most of the download flows as compared to the multimedia class. Flow duration also might vary for different type of flows to reach a specific number of packets, for example, to reach 1000 packets a download flow may take just 1-2 seconds whereas multimedia flows might take up to 4-6 seconds.

We have used Weka [17] to train the classifier. Weka is a data mining software in Java. It has the collection of the majority of the ML algorithms for data mining related tasks. We can use Weka and provide the training data in a specific format to train the classifier. We apply 10-fold cross validation tests, as well as percentage split (66%) to see how good the combination of features are, and how much accuracy we get. In 10-fold cross validation, weka divides the training data set into 10 parts, and uses 9 parts to train the classifier and remaining 1 part to test the accuracy. It does 10 iterations for the given data, and in each iteration 1 particular part is chosen as test data and remaining as training data. In percentage split method, the data is divided into 2 independent parts based on the percentage we mention, one is used to train the classifier and other is used for testing.

We build training data set for above mentioned 6 features. We also tried to remove or add some of features in order to classify with better accuracy. We got offline training accuracy of 82 % with K-NN, when we took first 1000 packets, with feature set-1.

After doing some study, followed by few experiments, we came up with two other features i.e. average advertised window size and number of packets with push flag *set*. The packets having push flag *set* indicate that the packet has some kind of urgency. An application sets this flag when it wants that the packet does not get stored/buffered at the intermediate nodes. Mostly push flags are set for communication where we don't want to see much delay, for example live video streaming or some other critical applications.As different operating systems and their versions might have different implementation of TCP rate adaptation algorithm, to maintain the uniformity across all our training process and testing results, we have used ubuntu machines only.

We observed that by adding these two features to the existing *feature set-1*, we get higher accuracy using the same number of packets, i.e., first 1000 packets of the flow. We call this new feature set as *feature set-2*. We also tried to analyze the training data points with respect to just two features, e.g., standard deviation of packet size and average advertised window size. Figure 3.1 shows that using these two features, we can roughly separate these two traffic classes. It can be seen that the advertised window size is higher and standard deviation of packet size is lower for download flows as compared to that of multimedia flows. If we draw a line at advertised window size = 31000, parallel to x axis, most of the download flows will lie above that line. Since a lot of download flows are overlapping on each other around the point (x=100, y=33000), it seems that there are just 40-50 blue dots, but that is not the case. Of course, some of the multimedia flows (red points) also lie above that line but we can use other features to separate them out. Similarly we can draw a vertical line at standard deviation of packet size = 200, parallel to y-axis, then most of the download flow fall on the left side of the line. We can classify all the points more accurately, using rest of the six features.

3.1.1 Presence of UDP Flows

We use receiver's window size and push flags in our feature set-2 but UDP header does not have these two fields, so our model built using feature set-2 can not be used for UDP flows. So we analyzed the proportion of the UDP flows and found that UDP packets constitute less than 0.5% in video, nearly 1% in audio and less than 0.1% in case of download class. So overall UDP packets constitute less than 1% of the total packets in our training data. Since it is a negligible fraction, we have ignored them as of now.

3.2 Algorithms Used

There are many ML algorithms which can be used for classification purpose. We used K-nearest neighbor (K-NN) and decision tree algorithm to train the classifier.

In K-NN method, each time we want to classify a flow, we call weka (more details in chapter 4). Weka calculates K nearest neighbors based on the euclidean distance between the given test instance to all the training instances. After that, it assigns the tag of the majority of those K neighbors. We varied K from 1 to 50 to see at what value of K we get best accuracy. For our training data, K=1 gives the best accuracy (see chapter 7 for more details).

Decision tree method makes use of a decision tree as a predictive model which tries to generalize the value range of features for a particular class. While classifying a test instance it compares the test instance's values along the path from root to the leaf node. Internal nodes

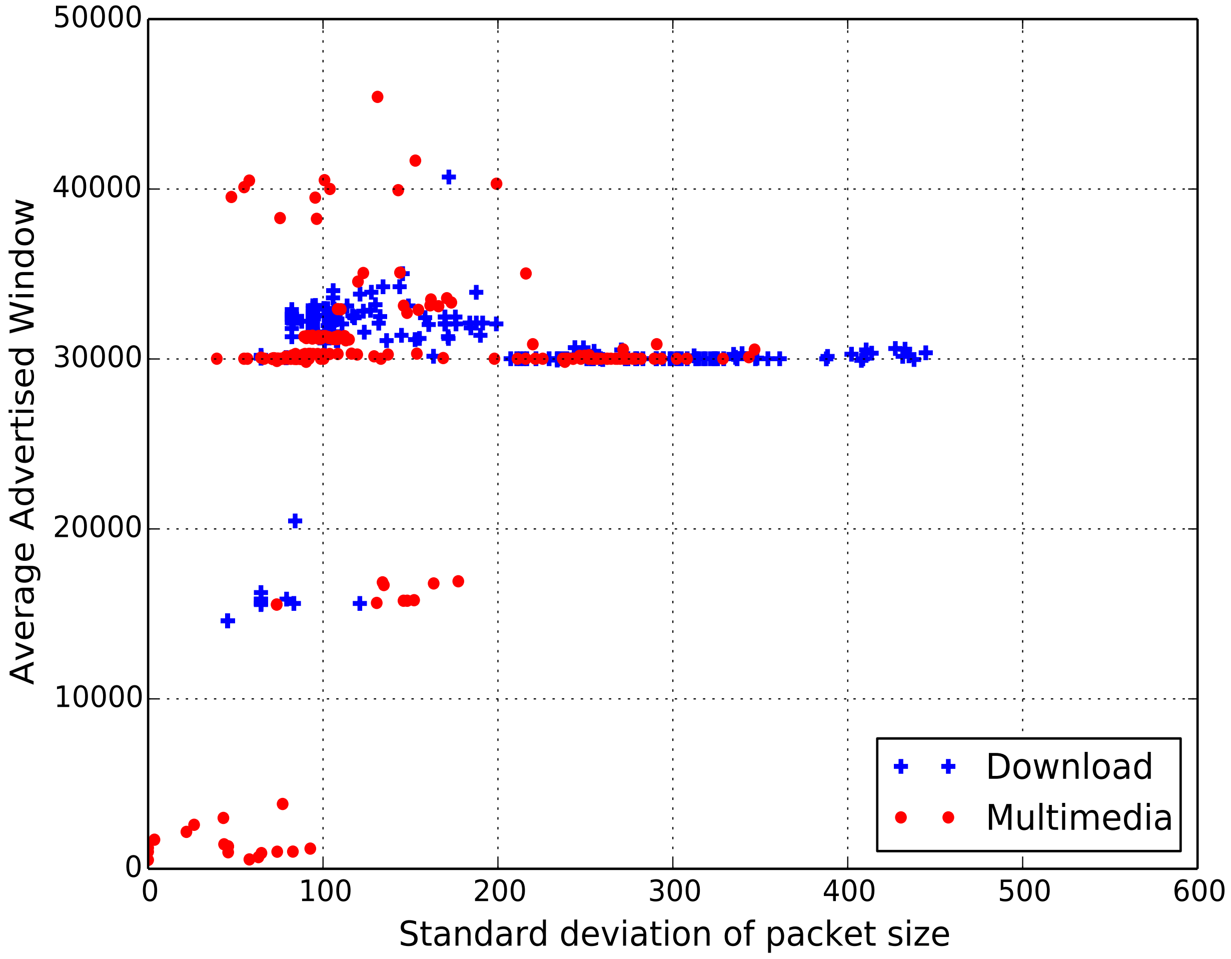

Figure 3.1: Separating multimedia and download flows using two features

represent a particular value of a feature and leaf nodes are class tags. When it reaches to the leaf node of the decision tree then the tag associated with the leaf node is assigned to the test instance. When we train the classifier using decision tree method, it generates *hierarchical if-else rules* using the feature values for each class. An example rule can be like: if (f1 > a and f2 > b and f2 < c) then it belongs to traffic class P, where f1, f2 are features and a, b, c are constant values.

We use this algorithm to generate the required rules and apply those rules in our script. Whenever we want to classify a flow, we check if the flow features pass those rules. The logic behind having these static rules is that the classification becomes very very fast, because all we need to do is to check some *if-else* conditions, and that takes very less time. We can get good accuracy subject to the condition that the rules we put are accurate.

Initially we used K-NN approach for classification, as it was more feasible to update the rules automatically with the setup, whereas in decision tree approach some manual intervention (hard-coding the *if-else* rules) is needed. But on the other hand, K-NN needs high computational power and memory, so we further explored the decision tree approach as it was more suitable in the AP scenario.

We tested the accuracies on both the models (K-NN and decision tree), but implementation was done only for decision tree approach.

3.3 Threshold Point for Classifying a Flow

Which ever algorithm we use, we need to wait till a point when we have sufficient information about a flow, so that we can classify it using our training data. We want to wait for few packets (or few minutes) to make the decision, but at the same time we do not want to wait too long. This is why the threshold point selection comes in the picture. At this point, we will discuss what all ways we tried before choosing a particular threshold point.

3.3.1 Based on First N Packets

In this approach, we classify a flow after first N packets have arrived for this flow. We tried different values of N between 500 to 5000 packets. After N=1000 packets, accuracy does not increase much. Also, there in no point in choosing very high value of N such as 2000 or 3000, because there are many multimedia websites where it's rare to reach these many packets in a single flow, as they generate multiple parallel flows for streaming. Detailed results are explained in chapter 7. We have used this approach in all our experiments to compare the accuracy of different ML algorithms as well as the effectiveness of different feature sets.

3.3.2 Other Approaches

We tried following two approaches as well, though we did not use them later in our final solution, as we got better results using first N packets approach.

Based on First M Minutes

In this approach, we wait for first M minutes for a flow before calculating its features and classifying it. As we observed during trace collection, that some websites do advance buffering of multimedia content initially and after that, packets arrive at a slower rate and their packet sizes are also comparatively smaller than the initial burst. So initially download, audio and video all behave somewhat similar, i.e., audio and video flows try to buffer/download some advance portion of the streaming content, so this behavior is similar to download flows in the very beginning. Download flow's behavior is nearly constant for its entire flow duration, i.e., till the file download is finished. So we thought that if we wait for few minutes to classify a flow, accuracy might increases because we can discriminate audio/video and download flow with more accuracy when multimedia flow becomes stable, e.g. after the advance buffering phase ends.

Based on Particular Time Interval

As we found that first N minutes approach might be useful, initial behavior of some audio/video and download flows are somewhat similar. So we can skip first 1 or 2 minutes of the flow, i.e., we start observing a flow after it has been at least 2 minutes since the first packet for that flow arrived. After that we store flow information for all the packets that arrive in next 1 minute duration. In short, we use packets which arrive between 2-3 minutes interval for a flow. Since we consider a flow as valid only if it has at least 1000 packets, while considering the particular time interval, we could not find enough valid flows between 2-3 minutes because many of the valid flows of audio/video (specially audio and small duration videos) have already buffered most of their content before 2 minutes itself and they do not have enough packets left to arrive. Also, we can not consider all the flows between 2-3 minutes as valid because all the flows are not audio/video flows. The idea behind trying this approach was that, we should not process every packet in the interval, in which most part (feature values) of the flows from the different classes, behave in similar way.

In this chapter, we described the different feature sets and their discriminating powers. We also discussed the training process. We looked into different approaches that can be used to define the threshold point after which a flow gets classified. In the next chapter, we talk about the work-flow of this classification model in the live environment.

Chapter 4

Real-Time Traffic Classification

Till now, we discussed about feature selection and training the classifier. We also decided the threshold point in terms of number of packets (first N packets of a flow). Now, we talk about the work-flow of classification model and describe how we capture packets, associate each packet to a flow and finally classify the flow.

4.1 Live Traffic Capturing

We start with the basic setup as shown in the figure 2.1. Our final goal is to prioritize packets belonging to multimedia class. Prioritization has meaning only if there is some bottleneck point or link and packets gets queued there. We must apply our classification rules at that point only, so that we can decide which live flow belongs to which class and accordingly prioritize or de-prioritize a particular flow.

In case of our setup, bottleneck link is the link between eth1 interface and the wireless AP (figure 2.1). So we run all the classification and prioritization scripts at desktop machine itself. Now, the task is to capture each incoming packet and associate it with a flow (4-tuple). We have python script which uses pcapy module that runs on desktop machine. The script captures all the packets at eth1 interface in real time and fetches all the parameters needed such as source IP, destination IP, source port, destination port, packet size, packet arrival time, advertised window size and status of push flag. This script processes each packet one by one. Since we just need packet headers and not the payload, we only capture first few bytes of a packet.

For each packet captured, we check if this packet belongs to an existing flow. If the answer is *'No'*, then we create a new entry for this new flow and again go back to process next packet. If that packet belongs to some existing flow then we update the packet count and other data structures such as list of packet size, list of inter-arrival times etc., for that flow. After processing each packet, we check if the flow has reached the threshold point (first N packets). If *yes*, then we calculate the feature values and use the classifier to predict the tag for that flow. If the predicted tag is multimedia then we prioritize it otherwise we de-prioritize it. To enforce prioritization, we pass that flow information (4-tuple) to the prioritization module which immediately puts that flow in the high or low priority band, depending on the class tag. Figure 4.1 shows this work flow.

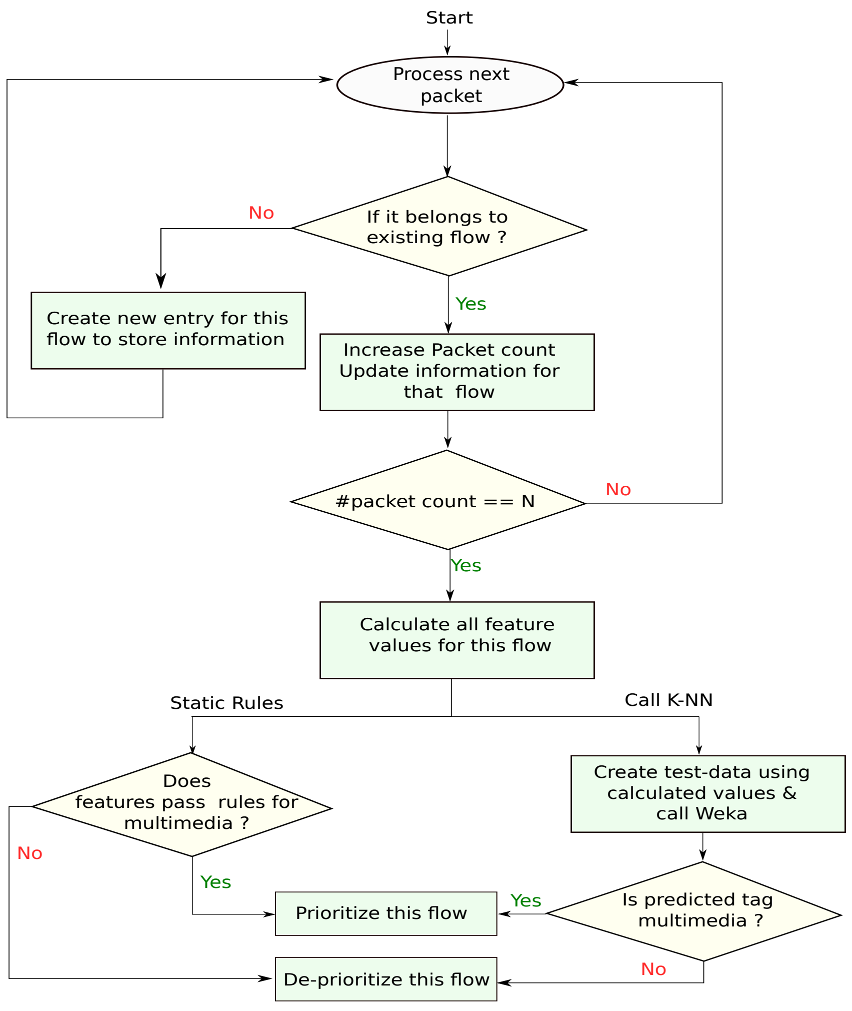
Start
Process next packet
If it belongs to existing flow ?
No
Create new entry for this flow to store information
Yes
Increase Packet count Update information for that flow
#packet count == N
No
Yes
Calculate all feature values for this flow
Static Rules
Call K-NN
Does features pass rules for multimedia ?
Create test-data using calculated values & call Weka
No
Yes
Prioritize this flow
Yes
Is predicted tag multimedia ?
De-prioritize this flow
No

Figure 4.1: Live Classification and Prioritization

4.2 Classification Process

We use two supervised machine learning algorithms for classification, i.e., K-NN and decision tree algorithm. Both use large number of pre-labeled data to build a model and then using this model, it classifies new test instances. We already described the training phase in chapter 3 . In this section, we describe how we use the model to classify the traffic in real time scenario.

We also need to periodically update the training data and re-build the classifier at regular intervals so that the accuracy does not decrease over the period of time. Figure 4.2 shows the overall classification process. After a flow has got it's first N packets, we calculate the flow features and then the flow can be classified using decision tree rules (as described in the section 3.2) or by using K-NN (using Weka) each time.

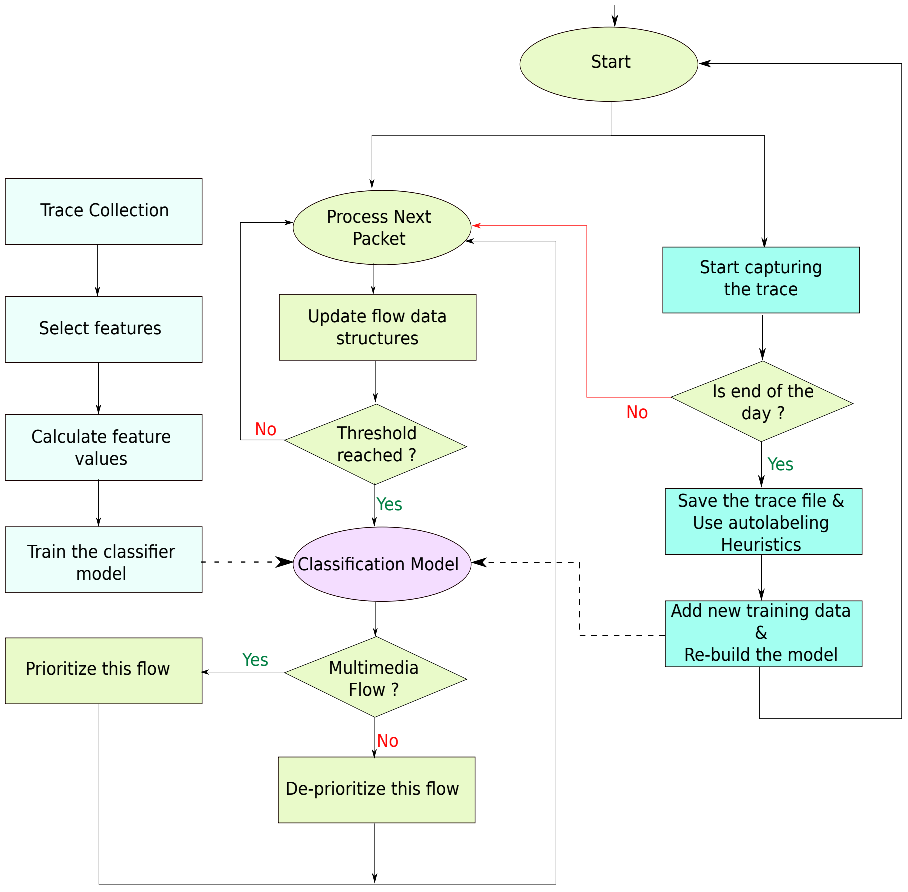

Figure 4.2: Big picture showing the learning process

4.2.1 Using Decision Tree Rules

We have created the static rules with the help of decision tree classifier (BFTree) in Weka tool. We apply the *if-else hierarchical conditions* for the multimedia class that need to be prioritized.

As we already have the calculated feature values at this point, we simply check if these feature values pass the hierarchical if-else conditions, if yes, then prioritize that flow immediately, else de-prioritize that flow. Some times we found that the rules generated by decision tree classifier are not exactly similar to what we expected.

4.2.2 Using K-NN Algorithm

In this approach, there is no such thing as static rules. We use K-NN algorithm and call weka to get the predicted tag for the flow in consideration. Since Weka needs training set (or model) and the test data in a specific format, we put the calculated features in an arff file. We also have the arff file of the training data set that we have already created in chapter 3. We call K-NN algorithm with the test instance and the training data arff files.

Based on K nearest neighbors, Weka returns the predicted tag. If the tag is multimedia, then the flow information (4-tuple) is passed to the prioritization module which puts that in the high priority band immediately. So any further packets belonging to that flow, will start getting prioritized.

In this approach, Weka needs to take the training file (in arff format) and train itself each time we want to classify a flow. We can make it more efficient by re-using the already built classifier model. We build the model offline only once and then put it there in the desktop machine. So that when ever we have to classify a flow, while calling Weka we simply pass the classifier model and the test file (in arff format) and Weka returns the tag as usual. This reduces the time taken to classify the flow.

4.2.3 Feedback Mechanism

Since network traffic characteristics may not be stable for long duration, the flow features for the same class start deviating from the values that we had calculated long time back. So there is a need to keep updating these rules. The solution is to re-train the classifier at regular intervals. There are two ways to update the model.

Using the Model Output Tags

During the real-time classification, after each flow gets classified as either multimedia or download, we keep on appending the feature values and the *predicated tag* as a vector in a temporary file, and after certain number of flows have been classified, say 100 flows, we add these 100 flow's feature vector along with the tag in the existing training data, and re-train the model again. We can also update the classifier by the end of everyday, just add the flows features and tags that have been classified on that day.

But there is a small problem with this approach. What if the model has a low accuracy? In this case, many of the classified flows might get tagged wrongly and we are adding these in our training set. This might further lower the accuracy of the classifier. So before applying this approach, we better be sure that our model is having high accuracy and performing well.

Using the Auto-labeling Heuristic

In order to avoid the above issue, we rely on our auto-labeling algorithm itself. In this approach (figure 4.2), we save the trace of all the connections during the real-time classification in the background and at the end of each day, we give this trace file to auto-labeling algorithm. The auto-labeling algorithm fetches all the multimedia flows and calculates their feature values. Similarly download flows also get fetched using some manual intervention in labeling process. Finally these new flows along with their labels are added in the existing training data and we re-build the model again to reflect those changes.

4.3 Temporal Accuracy

We build our first decision tree model in February 2016 and tested the real-time classification accuracy on various multimedia streaming websites, along with many downloads. We got 89% accuracy. After 3 months, we again used the same model to test on same websites and few others as well. We observed that the results were similar, i.e., overall 87% accuracy this time. So we can infer that the model works properly even after long time as well, that too without re-training it in between.

4.4 Testing on New Websites/Mobile Apps

While testing over the same websites/apps on which the training data was taken, even if the results are good we can't be very sure that the model is indeed a good one. So we tested the model on un-seen/new websites and apps also, for both downloading and multimedia streaming. There was not much difference in the results. So it tells that the model is generic enough to work properly on most of the un-seen websites with good classification accuracy.

In this chapter, we discussed about real-time packet capturing, fetching information, classifying the flows. Next we will talk about prioritization module and how we validate the improvement in the multimedia streaming.

Chapter 5

Prioritization & Quality Validation

Once we identify a flow as multimedia, we need to prioritize that flow to get better quality. Linux has inbuilt mechanism for traffic manipulation, known as Linux traffic control (*tc*) [18]. It is used for traffic shaping, scheduling, policing, dropping, marking etc. We use scheduling mechanism of *tc* for prioritization. It has various queuing disciplines, e.g., first in first out (FIFO), stochastic fair queuing (SFQ), token bucket filter (TBF), priority queue (PRIO), hierarchy token bucket filter (HTB) etc. We analyzed these queuing disciplines (qdisc) and found that PRIO and HTB are useful in our case. All these qdiscs work on outgoing link of an interface. We did prioritization on eth1 interface of desktop (figure 2.1). Whatever traffic comes from Internet through eth0, gets processed on desktop and then it passes through the queue (HTB or PRIO) before reaching AP. We tried both HTB and PRIO.

5.1 Hierarchy Token Bucket (HTB) Queuing Discipline

HTB [19] uses the concept of token bucket filter, i.e., we can limit the maximum bandwidth that a HTB queue can use. We limited it to 12 Mbps. Since HTB is hierarchical, if we limit the root, then all the subclasses will be throttled automatically. In HTB we can add as many children classes as we need. In each class (band), we can define two parameters independently, i.e., *rate* and *ceil*. Value of *rate* parameter is the minimum guaranteed bandwidth given to the band and *ceil* is the maximum allowed bandwidth given to it. All the bands together can use the maximum bandwidth available to HTB, i.e., 12 Mbps.

We need to assign priority to each band. The band having priority index 0 has the highest priority. If the priority index is higher, priority of that band is lower. HTB is work conserving scheduler. If we want a packet to be sent into a particular band, then first we need to mark that packet accordingly and using a filter, that marked packet will be put into the desired band.

To mark the packets, we use iptables [20] rules. We mark all the packets belonging to multimedia flow to a specific number. Iptables rules use 4-tuple information that is provided by the classification module. We apply these iptables rules on the POSTROUTING chain of the mangle table to mark the packets according to the IP and port address. *Tc* filters check each packet, look at the marked number and accordingly put that packet into the respective band. We make a new iptables rule for each multimedia flow and execute it. Earlier we used to flush all the older rules from the interface then we append the new rule to the existing set of rules and re-apply all the rules at once. This takes more time as the number of the iptables rules increases, e.g., single iptables rule takes 200ms to run while 1000 rules take nearly 6 seconds to execute. Again re-running the previous rule does not make any sense, so for scalability purpose we update only new rules and keeping older rules as it is.

Initially all the traffic go to the default band. We set band 2 as a default band for all the packets. When we find a multimedia flow, we mark packets of that flow and then send them to the higher priority band, i.e., band 1. So we move multimedia flows from default band to the prioritized band. For each band, we use the value of *rate* and *ceil* as 400 kbps and 12 Mbps respectively. If band 1 has no traffic, then full bandwidth will be given to the band 2 because the value of *ceil* for band 2 is 12 Mbps. If band 1 has packets to send, then also band 2 will get at least 400 kbps because the *rate* of band 2 is 400 kbps. We use *rate* value of band 2 as 400 kbps because we do not want to stall other non-multimedia flows. Few lines of the script are as follows.

```
sudo tc qdisc add dev eth1 root handle 1: htb default 2
sudo tc class add dev eth1 parent 1: classid 1:1 htb rate 12mbit
sudo tc class add dev eth1 parent 1:1 classid 1:5 htb rate 400kbit ceil 12mbit prio 0
sudo tc filter add dev eth1 parent 1:0 prio 1 protocol ip handle 5 fw flowid 1:5
sudo iptables -A POSTROUTING -t mangle -d 172.16.1.28 -j MARK –set-mark 5
```

Here, first line creates HTB queue on eth1 interface. Next command limits the HTB bandwidth to 12 Mbps. Now, for each band, we have to define *rate* and *ceil* values. Here, we create a band with the highest priority (prio 0) with classid 1:5. In the next line, tc filter checks each packet for its marked number, if it is marked as 5, then this packet will be put in the flowid 1:5

(prioritized band). In the last line, we have an iptables rule that marks a packet to a specific number (here 5) if IP and port address match. We need to have one iptables rule for each multimedia flow to put its packets in the prioritized band.

Heuristic Used in HTB

We need to have minimum 1000 packets in a flow after which we do classification and hence prioritization. Packets belonging to download flows come at faster rate as compared to that of multimedia flows. Generally multimedia flow may take 4-6 seconds to reach to this 1000 packets threshold, this is the case for the websites which buffer the song in advance. Some websites fetch the packets as and when required, these websites take even longer time to reach the threshold. Download flows just take 1-2 seconds to have first 1000 packets.

We want to prioritize the multimedia traffic over download in less time to get better results. Instead of using two bands in HTB, we now use three bands. We assign different priority to each band. Initially, all the packets go to the band 2 (neutral band). We can identify download flows quickly and immediately put them into the band 3 (lowest priority band), i.e., we de-prioritize download traffic. Now, multimedia flows have relatively higher priority as compared to the download flows, even if those flows are not classified and explicitly prioritized yet. After some time, we can identify multimedia flows, at that point we put them into band 1 (highest priority band). All the traffic flows which do not reach to the threshold value will stay in the neutral band, e.g., web. This will have the same effect that we had when we used only two bands. But in this case, we will start getting better quality in multimedia in less time. In addition to that, small web flows will get relatively higher priority as compared to the download flows.

5.2 PRIO Queuing Discipline

PRIO qdisc is used for prioritization of a specific flow of the traffic, based on filters. By default PRIO qdisc has three priority bands. It gives highest priority to band 1 and lowest to band 3. First, it checks whether band 1 has packets to send, if yes, then all the packets of this band are forwarded, and then only band 2 will be considered and so on. If band 1 has no packets, then total bandwidth will be given to band 2 and band 3 because PRIO is a work conserving scheduler. We can also use different queuing disciplines in the different bands. We assign the traffic to one of these three bands using a filter which is based on IP and port address of the packet. Few lines of the script are as follows.

```
sudo tc qdisc del dev eth1 root
sudo tc qdisc add dev eth1 root handle 1: prio
sudo tc qdisc add dev eth1 parent 1:1 handle 10: pfifo
sudo tc filter add dev eth1 protocol ip parent 1: prio 1 u32 match ip dst 10.129.26.77 flowid 1:1
```

Here, In the first line, we delete the existing queue on eth1 interface. In the next line, we create PRIO queuing discipline on top of eth1 interface. Now, we need to assign handle for each of three band. Here we used pfifo (first in first out) queue in each band. We can use any queue e.g. pfifo, HTB, SFQ in any band. In the last line, *tc* filter is used to put a packet, in a particular band, depending on some conditions. Here all the packets having destination IP address 10.129.26.77 will be put into band 1 (highest priority band). So all packets destined to this particular IP will be prioritized over other packets.

Since we simulate the AP in the desktop, we need to throttle the outgoing link of the desktop as the reason mentioned in section 2.1. So we need to do prioritization and throttling together at the same interface, i.e., eth1. We tried to throttle the link between eth1 interface and the AP, using wondershaper traffic shaping tool [21]. Wondershaper itself uses Linux traffic control (*tc*) mechanism . While using PRIO qdisc, we were able to prioritize the traffic but could not throttle the link using wondershaper at the same time. The reason is that, these two rules can not be applied at the same, at the same interface. First we tried PRIO before HTB, but as PRIO can not do both throttling and prioritization at the same time, finally we used HTB in our experiment.

5.3 Quality Checking Using Mean Opinion Score (MOS)

After classification and prioritization of a flow, we need to check the quality of multimedia flow. There are many alternatives available to check the quality of multimedia, i.e., we can measure throughput, average latency, jitter, etc. One standard measurement is MOS (Mean Opinion Score) [22]. MOS score value ranges between 1 to 5, with 5 being excellent quality and 1 being the worst quality. In reality, MOS score does not exceed 4.4 even for the best quality audio/video streaming. MOS score depends on the various parameters, i.e., link speed, codec used, jitter, round trip time (RTT), packet loss percentage etc.

In our setup, we used the same bandwidth for all the experiments, so we measured MOS score using jitter, latency (RTT) and packet loss percentage. Latency between node 1 and node 2 is the time that a packet takes to reach from node 1 to node 2 and to return back from node 2 to node 1. Jitter is the variation in the latency. Jitter shows connection strength. If jitter is less, it means that latency does not vary much, which shows better connection quality. High jitter may cause packets to get queued in the buffer, which may further lead to packet drops and subsequently results in quality deterioration.

There are many licensed softwares available for MOS score calculation. But we found the equations for MOS score, so that we can calculate MOS score on our own. The developers of the PingPlotter software have mentioned the equations [23] that they have used for MOS score calculation. We used the same equations to calculate the MOS score. We have scripts which take a trace file as input and calculate latency, jitter, loss percentage and MOS score. We used *tcp.analysis.ack_rtt* option of tshark command, which gives the latency of every packet by taking the difference of packet's departure time and the same packet's acknowledgment arrival time. We used *tcp.analysis.lost_segment* option of tshark to find the number of lost packets.

We tested our MOS script by manually prioritizing IP address of a particular website. With parallel download and without prioritization of multimedia flow, we got a MOS score nearly 3.6, latency 55 ms and jitter 72 ms, loss percentage 4 for an audio song. For the same website and the same song, with parallel download and prioritization, we got a MOS score nearly 4.35, latency 2 ms, jitter 3 ms and loss percentage <1. We performed many more experiments with and without prioritization and found similar results as above. So we can say that our MOS script can find the difference in the audio/video quality. We have used bandwidth of 12 Mbps in our setup. In the real world, a common man has much lower speed and in that case, we will get more difference in the two MOS scores, which are calculated with and without prioritization.

Initially, our MOS scripts were giving unexpected results for various experiments. The reason was that our prioritization script in wireless setup was not working properly. We had tested prioritization script in LAN setup using scp download and it was working fine in that setup. The reason behind the script not working in wireless setup is that sometimes intermediate hops of Internet network become the bottleneck and in this case the queue builds up at those intermediate hops. The effect of prioritization is visible only if we apply it at the node where the queue builds up. We want the queue to build up at eth1 interface of desktop and to ensure this, we did one scp download in parallel with other flows. So even if the website server or some intermediate hops of the Internet are themselves bottleneck, then also we will have a queue on eth1 interface. After using scp download in parallel, we got expected MOS score improvement.

By this time, we have talked about classification and prioritization part on a setup (figure 2.1) which is equivalent to working inside an AP. Next, we tried to implement the model on mikrotik board and subsequently on a laptop after converting it into an AP.

Chapter 6

Implementation on OpenWRT and Laptop AP

As we discussed in previous chapter, we performed the classification on desktop machine setup (figure 2.1). We collected traces on this set up and also tested on the same set up. Now we tried to implement this model on the mikrotik board [24]. Mikrotik board is an open source version of wireless access point.

6.1 OpenWRT Installation and Issues

This implementation was not successful and we moved to other alternative, i.e., making laptop as AP. Though we put following efforts to make it work. We started with installation of openWRT. Earlier we tried to compile the OpenWRT and during that process we faced many errors which included many git errors which were trying to download files from the links which were not valid. We needed to get the files manually from various sources to get the installation done. But soon, we realized that we did not need to compile it, as we do not require to make any changes in the drivers etc. Finally we downloaded compiled binaries [25] from the Internet. We tried barrier breaker [26] as well as chaos calmer [27] versions. We needed to install python on mikrotik board, as we have used python pcapy to capture the packet live and doing further processing. We somehow managed to install python on the board, but to install pcapy module there were other connected dependencies, i.e., setuptools. We tried to resolve up to few dependencies but every time some other dependencies kept popping out. At this point we decided to move to other alternative implementation.

6.2 Laptop AP : System Set-up

This setup was compatible with our scripts but we also needed to care about the amount of CPU and memory taken by our model and scripts, because our ultimate goal is to build a model which is feasible to work in an AP where limited space and very less computational power is available.

In desktop machine setup (figure 2.1), we classify and prioritize flows at eth1 interface and clients are connected to AirTight AP. Here in this set-up (figure 6.1), eth0 interface of the laptop is connected to the Internet and wlan0 interface is used as access point. All the scripts related to classification and prioritization are applied at the wlan0 interface.

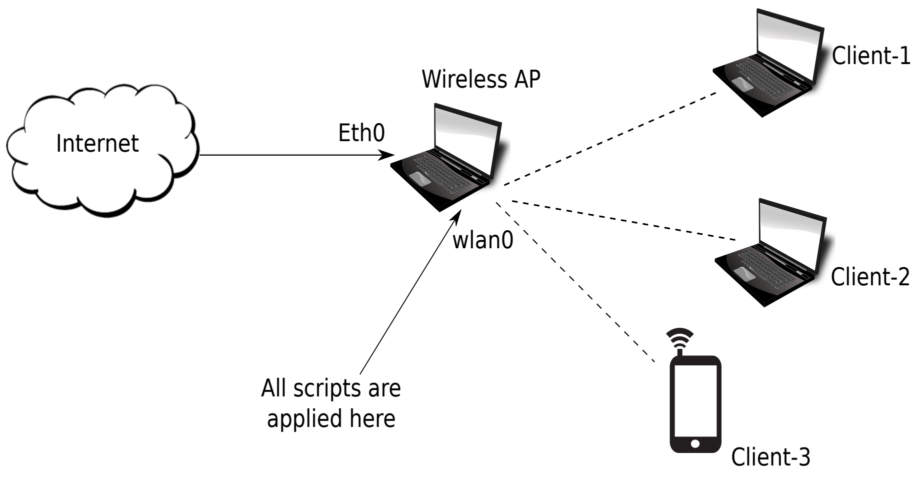

Figure 6.1: Laptop as AP

This setup is similar to the mikrotik implementation in the sense that we do all the things at one unit which was not the case with desktop machine setup (figure 2.1).

In the next chapter, we discuss the different results, observations and shortcomings of our approach.

Chapter 7

Results

In this chapter, we demonstrate the performance of our approach in terms of real time classification accuracy. We also present the different results related to prioritization module and MOS score. Further, we try to compare the CPU and memory overhead for our approach with other approaches. We also talk about various observations during this project. In the end, we discuss about the limitations of our model.

7.1 Different Feature Sets and Accuracy

As we mentioned in section 3.1, we used two feature sets. Figure 7.1 shows the comparison of both the feature sets. It is obvious that the feature set-2 performs better than feature set-1, for the same number of packets taken in a flow. The only difference is that, in feature set-2, we use two additional features, i.e., average advertised window size and number of packets with push flag *set*. These two features have played major role in achieving the high training accuracy of 90% using first 1000 packets with decision tree approach. We have already discussed about the relevance of these two features in section 3.1. We achieved per class training accuracy of 90.4% for multimedia and 89.6% for download class using feature set-2 and first 1000 packets.

Selecting K in K-NN Approach

We tried K-NN algorithm with different values of K between 1 to 50 (figure 7.2). For our training data set, **K=1** gives the best classification accuracy. We had taken nearly 10 traces from each website. When the value of K increases, K-NN searches for more neighbors for the

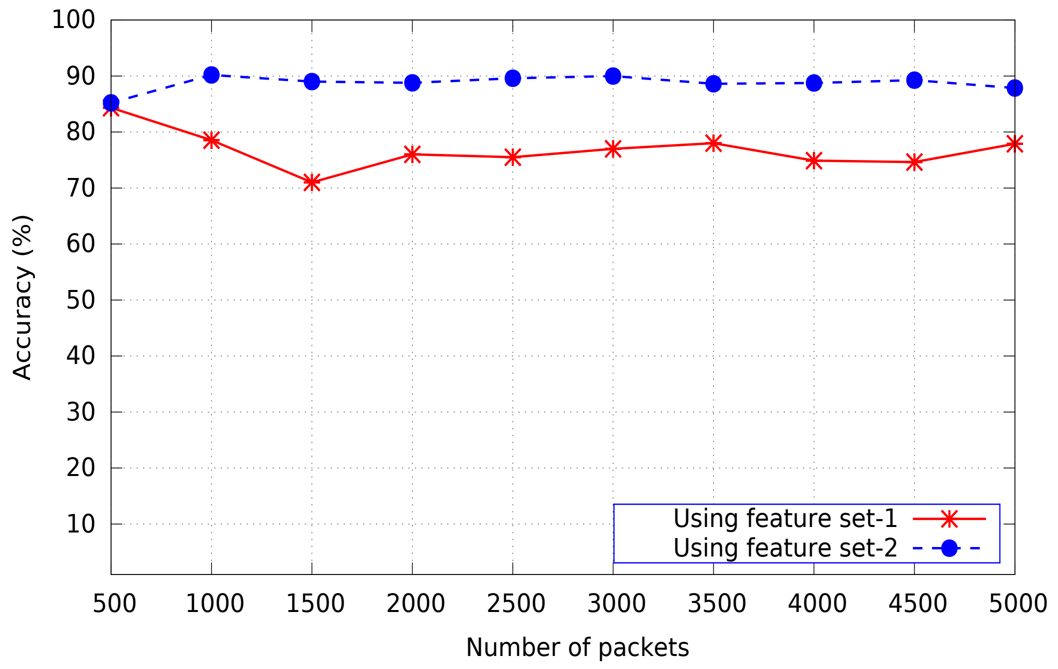

Figure 7.1: Comparing two feature sets

given test flow, but majority of them may not belong to that website. This might be the reason that as the value of K increases, accuracy decreases.

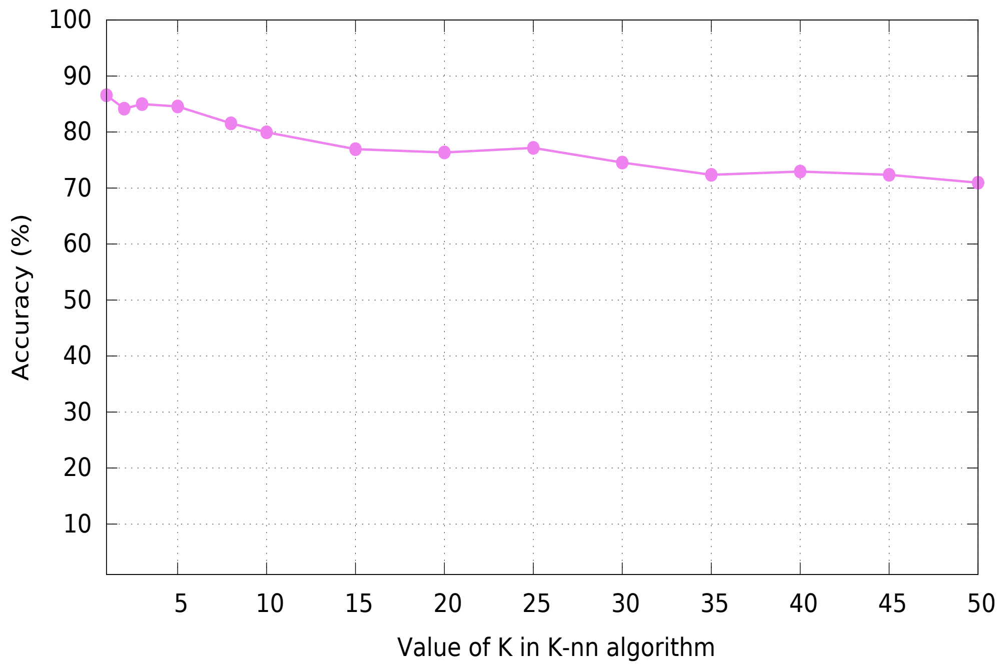

Figure 7.2: Values of K vs Accuracy (For first 1000 packets)

7.1.1 Accuracy in Real-time Classification

As stated earlier, we have implemented the model with *feature set-2* using decision tree rules on the laptop-AP. We evaluated our solution by applying the scripts at 2-interface machine and accessing the Internet on 2 client machines connected to AP. We did streaming of many audio and video from different websites. We also downloaded many files. We manually checked if the flows are getting tagged correctly.

We performed 10 multimedia streaming on multimedia websites and downloaded 10 files on download websites. We did this on different days of week and different times of a day, as we wanted to take into account the different server load conditions at different times. For each website, we did it four times, i.e., total 10*4 = 40 traces. Then we scaled down the correct results to the scale of 10 (figure 7.3). We followed the same procedure with laptop-AP setup also. Y-axis represents how many flows were classified correctly out of 10 (on an average). This figure shows the accuracy on various websites using both the setups.

For example, on starsports.com (website number 37 in figure 7.3) while streaming one video it generates 5-6 connections. Each of which are very similar to each other in terms of all the parameter values, deviation is very very less. So even if we had taken very few training traces for this website, during real-time testing it still achieves high accuracy. Saregama.com (number 35) creates mainly one connection for streaming but all the video streamings have very less deviation as we found in starsports.com and this website also has very high accuracy.

For androidapksfree.com (number 3), we downloaded many files, but almost all the time flows get tagged as multimedia. We checked its feature vectors and found that average window size was lower for this website as compared to other download websites. According to decision tree rules, once a website has lower average window size then there is high probability that it will get tagged as multimedia. For soundcloud.com (number 5), we found that its average window size is on higher side which is why majority of its streaming flows get wrongly classified as download.

We observed that overall accuracy for laptop-AP set up was comparatively low, i.e., 76%. The might be happening because of the fact that the training data was collected on the 2-interface machine setup (section 2.1), not on the laptop-AP. The real-time accuracy also depends on whether the training data was collected under same network conditions/set-up or not. We tested the bandwidth on the clients connected to laptop AP using iperf, and it came around 12 Mbps. This bandwidth difference might cause the change in feature values such as average IAT etc, and this is also a possible reason behind the 11% drop in real-time accuracy as compared to the setup shown in figure 2.1.

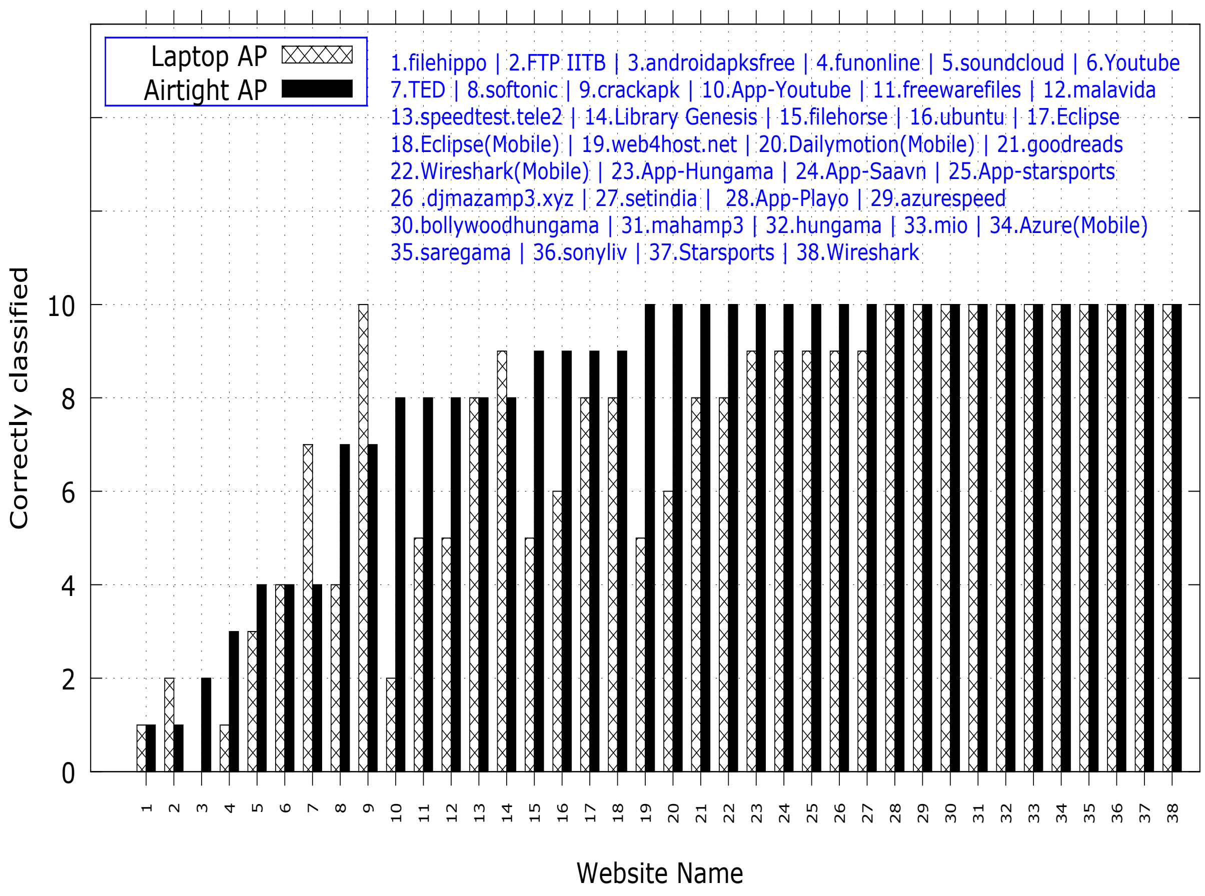

Figure 7.3: Accuracy on various websites

7.2 Multimedia Quality Improvement

We measure the streaming quality improvement using MOS. MOS score is based on parameters such as latency, jitter, packet loss percentage, etc.

7.2.1 Latency Improvement

Latency includes propagation delay, transmission delay, queuing delay and processing delay. In our case, when we have parallel download with multimedia flows, queuing delay is the major factor in latency. When we prioritize multimedia flows, they wait for less time in the AP's buffer as compared to the time it take in buffer in normal settings. The reason behind is that now we have separate queues for multimedia and download flows and packets of multimedia flows will be dequeued first. So our setup decreases latency by decreasing queuing delay in the AP.

Figure 7.4 shows a CDF (cumulative distribution function) graph for average latency reduction when the link between desktop and AP is throttled at 12 Mbps. For one instance of the flow we have one point in the graph. Every point on the graph shows the factor by which the latency is reduced in our setup. For example, value of factor 50 means, without our setup and with parallel download, if we get average latency 50 ms then for the same flow with our setup and with parallel download, we get average latency 1 ms. On an average, we are able to reduce the average latency by a factor of 36. In the best case, our setup can decrease average latency upto a factor of 100.

Figure 7.5 shows a CDF graph for average latency reduction when the link is throttled to 4 Mbps. In this case, we have very high latency (500-800 ms) for every packet. Here, we have a bigger queue in the AP's buffer. Prioritization shows a significant improvement in the latency in this case. We are able to reduce average latency by a factor of 130. As the bandwidth decreases, we get more improvement in the factor by which we decrease the latency.

7.2.2 Jitter Improvement

Our setup decreases jitter as similar to the average latency. We have not shown the graph for jitter in the report since its graph is similar to the graph of average latency. When we have a small queue in AP, latency will not vary too much, so jitter will be less. In a bigger queue, we will have more variation in latency, so more jitter. Similar to the average latency, we are able to reduce the jitter by a factor of 39 on an average when we throttled the link at 12 Mbps speed. When the link is throttled at 4 Mbps, we reduce the jitter by a factor of 164 on an average.

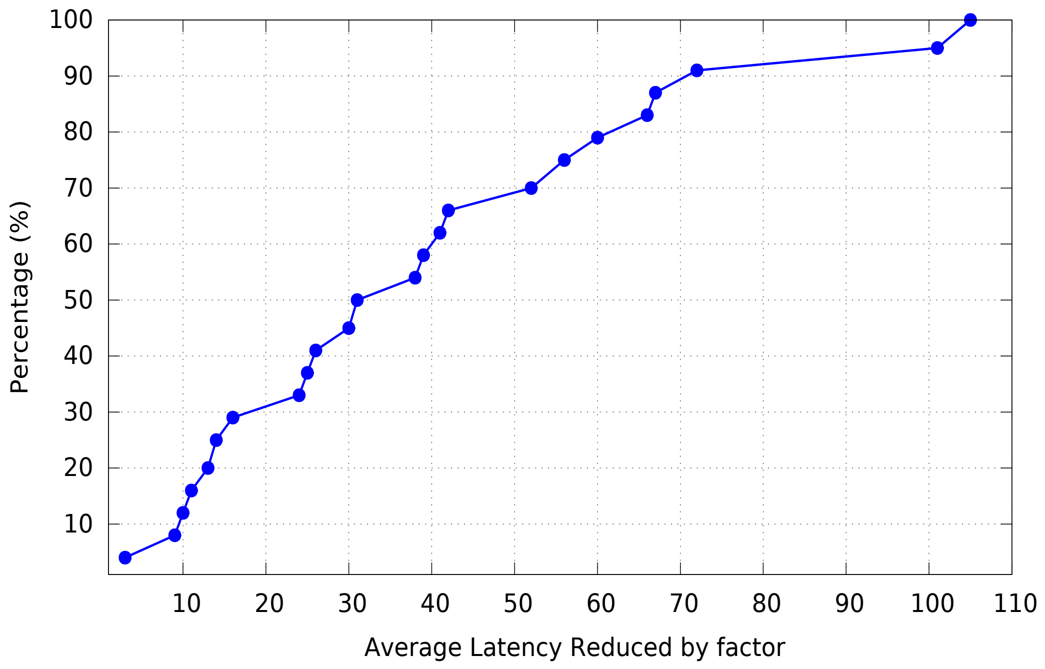

Figure 7.4: Latency Reduction Graph when link is throttled at 12 Mbps.

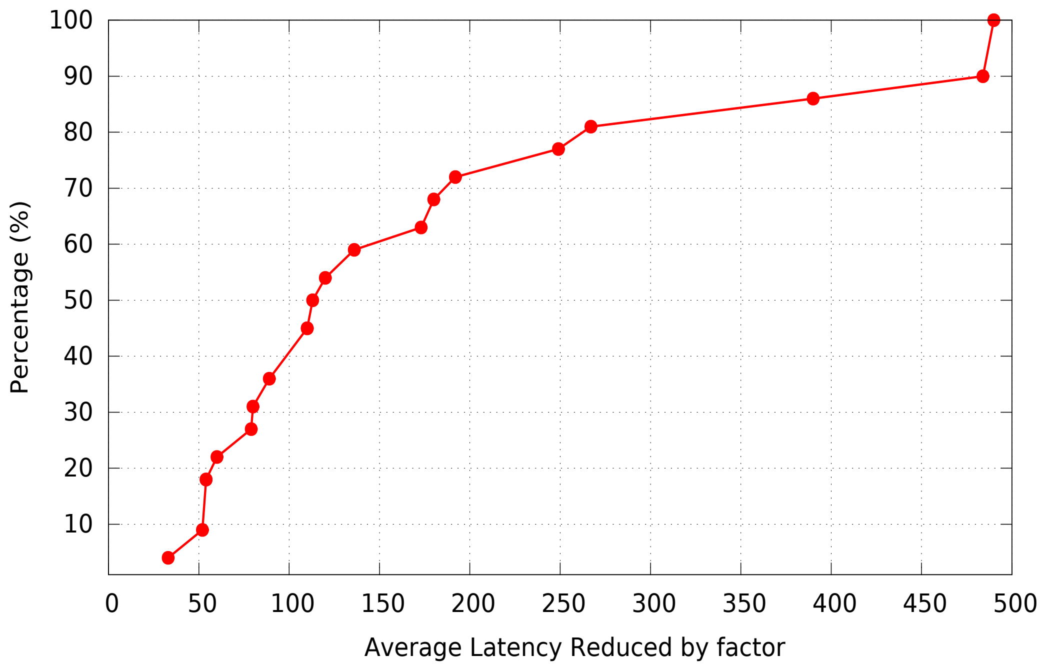

Figure 7.5: Latency Reduction Graph when link is throttled at 4 Mbps.

7.2.3 Loss Percentage Improvement

Packet loss happens due to the congestion in the network. When the buffer of AP overflows, it starts dropping the packets. HTB has one queue for each band. Band 1 (prioritized band) will have only packets from multimedia flows. Band 1 will have a smaller queue as compared to the queue of band 3 (the band which has download packets). So we will have less chance of queue overflow in band 1. On an average, we have 5% packet loss in multimedia flows with parallel download and without our setup. Our setup reduces loss percentage from 5% to 1.5% in case of when the link is throttled at 12 Mbps. Out of 22 experiments we did, one time we got an increment in loss percentage using our setup and one time we had no change in loss. In all the other 20 cases, we were able to reduce the loss percentage. When the link is throttled at 4 Mbps, we have significant improvement in the packet loss. We are able to reduce loss percentage from 20% to 5%. So reductions in the loss percentage are 3.5% and 15% (absolute difference in loss percentage) for 12 Mbps and 4 Mbps speed respectively. These results are shown figure 7.6.

7.2.4 MOS Score Improvement

Improvement in average latency, jitter and loss percentage lead to the improvement in the MOS score. Figure 7.7 shows the CDF graph of MOS score improvement. On an average, we increase MOS score by 0.7. Using our setup, we are able to get the highest possible MOS score, i.e., 4.4, even with the parallel download.

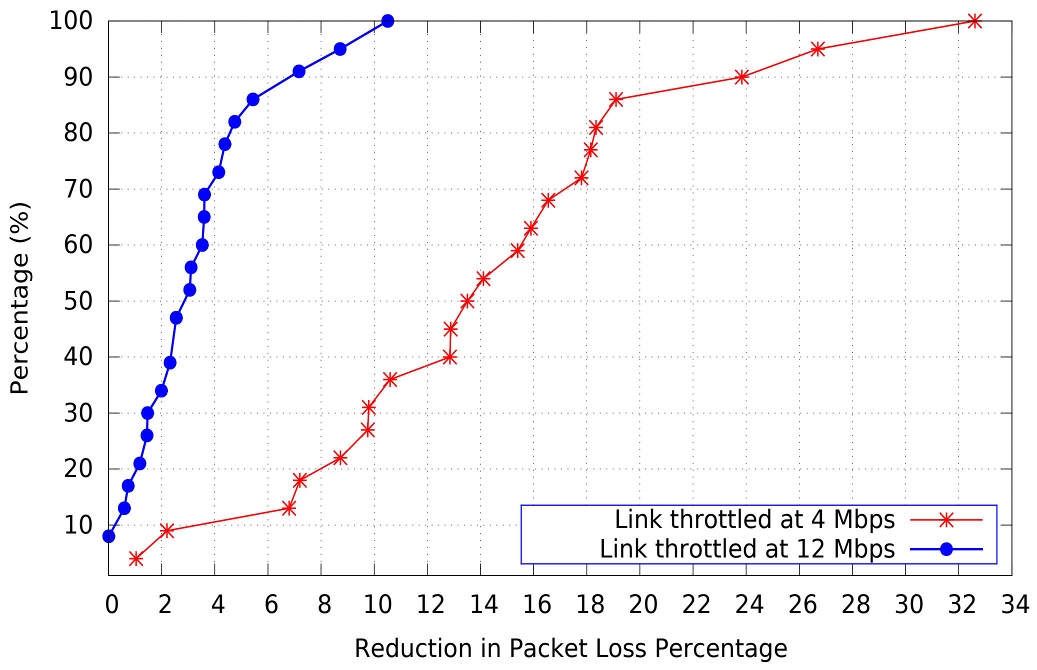

Figure 7.6: Loss Percentage Reduction for 12 Mbps and 4 Mbps speed

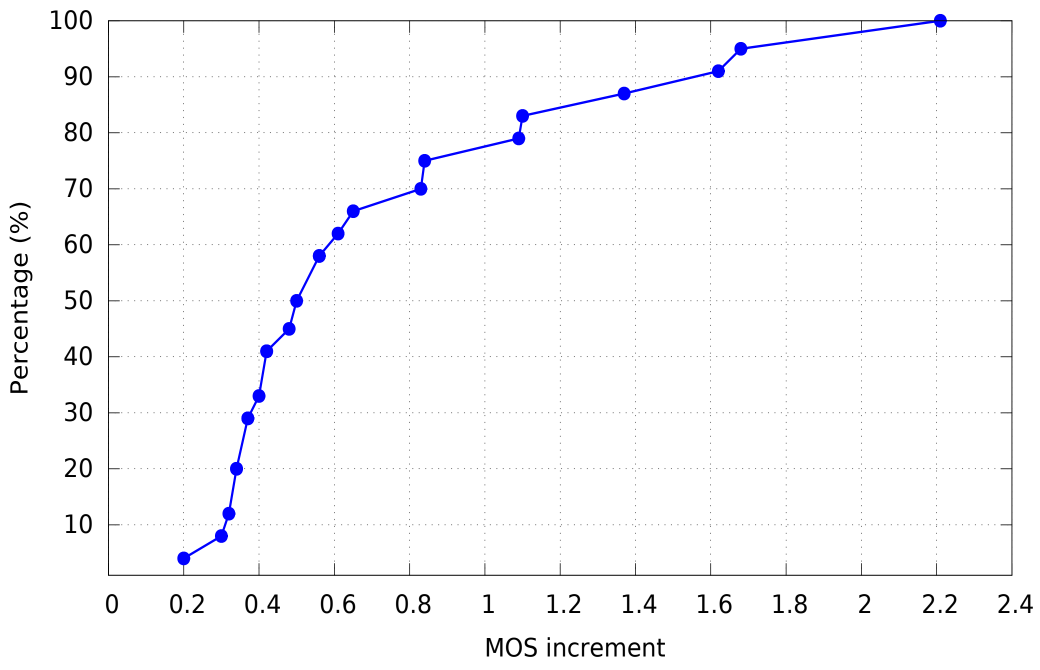

Figure 7.7: MOS score improvement

7.3 CPU/Memory Utilization

As the real AP does not have as much computation power as a PC, so we needed to ensure that the scripts or approach we are using, does not take much CPU and memory. We compare the overhead of our approach with payload based approach and URI (Uniform Resource Identifier) based approach, i.e., GET request approach (figure 7.8 & 7.9). In fact, here we did not used an actual payload based classifier but rather we used a script which searches for some particular strings in the payload. Our purpose was to see how much CPU and memory such a approach might take. In URI based approach, we just check for multimedia file extensions in the HTTP GET request string.

SETUP : While doing this measurement, we used laptop as AP setup. Laptop was Lenovo Thinkpad, with Intel Core i3 CPU @ 2.4 GHz x 4, with 4 GB RAM. We connected client machines to this laptop AP and accessed downloading and multimedia websites and for that duration, we collected the statistics about how much CPU and memory was utilized on the laptop. We also varied the bandwidth because when packets come at a very high speed, the payload based approach may not able to match the speed and thus memory overhead increases due to buffered pending packets. Similar is the case with URI based approach. In URI approach, we fetch the HTTP header from packet and search for specific strings in that.

In decision tree classification, we process each packet one by one and once a flow reaches 1000 packets, we simply calculate all the feature values and then check few simple *if* - *else* conditions to decide the traffic class and thus the CPU utilization never reached more than 5% in this case. Also, it is fast enough that there are not much packets pending in the queues and the memory overhead is always below 0.5% (figure 7.8). If we convert this 0.5% of 4 GB RAM to absolute value, it will be equivalent to 20 MB. Typically a mikrotik board like 433AH has 128 MB RAM and 64 MB storage space, so our model is feasible in terms of memory overhead and thus it can work in APs.

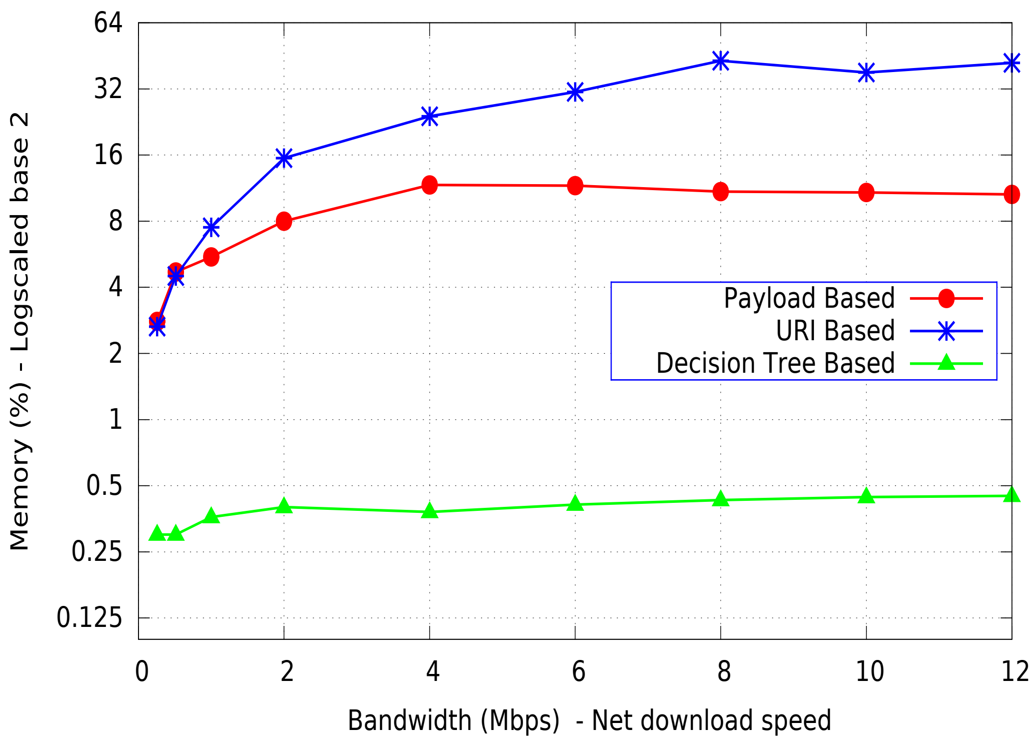

Figure 7.8: Comparing Memory overhead

From figure 7.9, we see that the payload based approach utilizes high amount of CPU as it has to perform string matching in the entire packet payload and that is why we observed nearly 90 to 95% CPU utilization during the experiment. URI approach also searches file extension strings, but only in the http URI in the header and it has comparatively low CPU overhead than payload based approach.

For decision tree rules, it requires very less CPU (less than 5%) and varying the bandwidth does not effect much here because of the low computational overhead. Again, if we convert this 5% of CPU usage for a 2.4 GHz processor then it comes out to be 120 MHz. Generally mikrotik board has 600 MHz computational power unit. So the model is feasible enough in terms of required computational power as well.

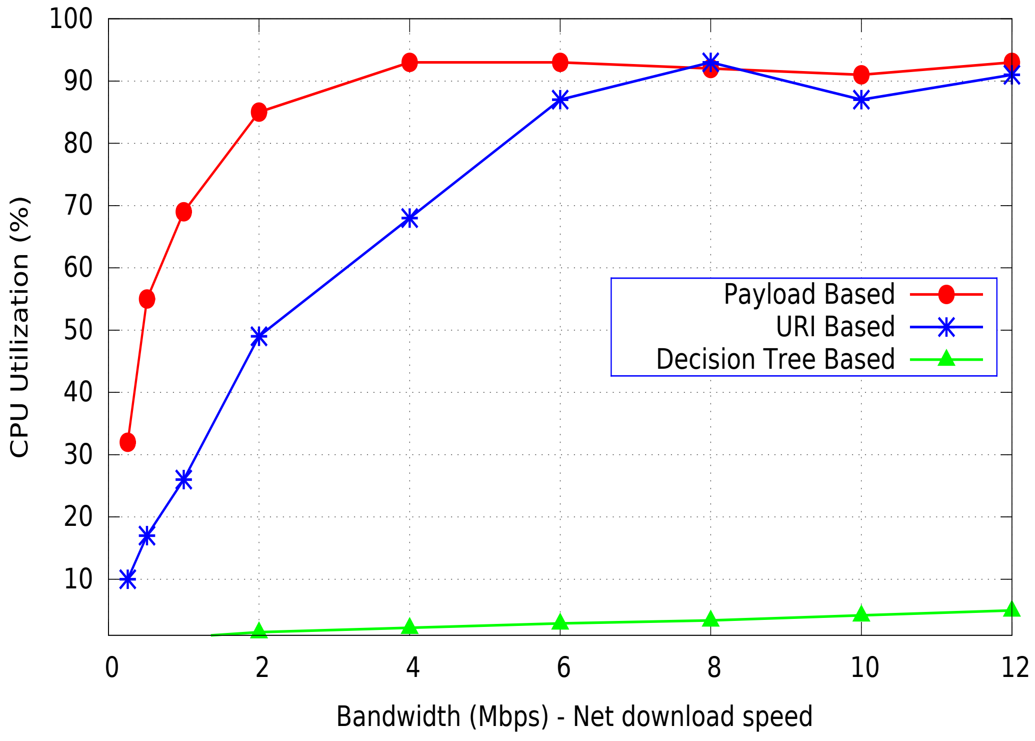

Figure 7.9: Comparing CPU overhead

Just to mention the fact that this CPU or memory overhead of payload or URI approach might change a bit depending on its implementation. But in our case, as we see very very large difference between URI or Payload based approach and our decision tree approach, we are in a position to say that our approach takes negligible CPU and memory as compared to them.

7.4 A Complete Experiment

In order to test and demonstrate the complete process and impact of our model, we performed few experiments. The purpose of these experiments was to show how our script improves multimedia streaming even when 4-5 simultaneous downloads are going on. We used laptop-AP setup (figure 6.1) for this. We executed our scripts and then started 4-5 downloads on websites such as *eclipse.org* and *wireshark.org*. These flows got classified correctly as download and then they got de-prioritized. Then we opened a browser and played a video from *starsports.com*. We noticed that the video quality was very clear and streaming was very smooth. This streaming was also classified as multimedia and got prioritized. We used iptables command to see that appropriate iptables rules were applied.

Now, at this point, we stopped our scripts and flushed all the iptables rules. This situation was similar to a normal scenario without our scripts in place. Just after few seconds, we noticed that the video became blur (earlier it was HD) because of downloads going on in parallel. Now to show impact of our approach, we again executed our scripts. Within 1-2 seconds all the download flows got classified correctly and again got de-prioritized. After nearly 5 seconds the video quality again resumed to HD quality. This experiment shows the effect of our model and how it can improve the quality.

7.5 Heuristics and Observations

During this project, we had to use many heuristics to deal with few exceptions. We also observed many things throughout the work, here we try to put these things together.

- **Defining a flow :** We observed that the websites create many parallel connections between the client and server using different port addresses. If we take 4-tuple (source IP, destination IP, source port, destination port) then we might not get enough packets in each of the flow and thus those flows do not get classified, as we have put a threshold of 1000 packet in a flow. On the other hand, if we define a flow as 2-tuple (source IP, destination IP) then all those parallel flows between the same pair of hosts get merged in to a single 2-tuple connection and the number of packets in that single flow also increases. Now most of the flows can easily cross the 1000 packet threshold. The logic behind thinking about the 2-tuple was that generally a website hosts same class of content, for example, video streaming websites mostly have videos and negligible amount of text or other supporting images etc. So if we prioritize that server IP itself, then all the connections to that server, using different ports get prioritized and that is what we want.

  But it has some drawbacks also. If there is a website that hosts both multimedia and download contents and we performed streaming first, now at this point the website's IP gets prioritized. After this, if we download something from this websites, those download flows also get prioritized, which is against our objective. After analyzing the pros and cons of each of these two approaches, we have finally used 4-tuple to define a flow so that we don't end up prioritizing a non-multimedia flow.

- **Different websites having different behavior :** We tried to cover top 20 websites in audio and video streaming and we also used 20 websites to take traces for download. While real-time classification, we observed that when we download files from some websites, some times the flows get classified as multimedia. Also, while streaming HD videos from some websites, we see that some of the flows get classified as download. It happens because some multimedia websites behave similar to some download websites and vice-versa, and thus the characteristics from both the classes get mixed up.

- **Multiple downloads and streaming multimedia simultaneously :** Though we collected training traces for many multimedia content streaming simultaneously, i.e., opening multiple tabs to stream different video/audio content at the same time. Same thing was done for download also. During real-time testing we found that if we test single streaming/download one at a time then model gives correct tag to it most of the times. But when we open 3 or more tabs to stream multimedia content simultaneously, then the model gives wrong tags for few of them. The possible reason might be that when there are many flows in parallel, feature values start deviating and thus they do not match with the training set. This mostly happens in the case of download class.

- **Consistency on a particular website :** We also observed that there are some websites on which all flows nearly have similar feature values and the values do not deviate much. Generally we get either very high accuracy or very low accuracy on these websites, because most of the flows get the same tag. One possible reason for low accuracy could be the fact mentioned in the next point.

  For the websites who have very random behavior and whose feature values deviate a lot, flows get classified correctly sometimes, but this behavior is not predictable.

- **Removing extra flows in training set :** While preparing the training data, we needed to have a balanced training set for both the classes. So we had to remove many flows from the class for which we had extra flows. But while doing that it was not feasible to identify the websites to which these flows belonged. This might lead to low accuracy on those websites if majority of the flows from that websites got removed before training itself. As most of the flows from a website got removed, we might get high training accuracy as none of the flows from that website is present in the training set, but when we perform real-time testing on that website, we might get low accuracy as the behavior was not captured before.

- **Streaming behaviour of different websites :** Some audio streaming websites buffer the full song in the very first 30-60 seconds, whereas on other websites, songs get buffered in a periodic fashion, i.e., packets get buffered in advance for first 2 minutes of the song, then very few packets arrive in next 2 minutes and after that again packets start buffering. In buffering phase, the multimedia flows look somewhat similar to the download flows, i.e., large packets arrive with less inter-arrival time. This sometimes causes the model to predict an actual multimedia flow as download (false negative). In download flows the same pattern continue till the flows get completed. Most of the websites utilize full available bandwidth (12 Mbps). We also took download traces from the websites, which are not able to utilize full available bandwidth, e.g., some websites can afford maximum download speed of 2 Mbps. Because of this, some slow download flows may get predicted as multimedia (false positive).

7.6 Limitations

We have tried to cover many aspects of traffic classification, but our classification model have following limitations.

- **Specific setup and bandwidth :** We collected traces after throttling the speed to 12 Mbps. Also, while doing the live experiment, we have used the same bandwidth. Since, flow features might differ with the change in the bandwidth, if our solution is used directly on some different network bandwidth configuration, then it might not work with high accuracy and it might need to be re-trained fully on the new traces collected on that network.

- **Short duration flows :** Having the high threshold point (min number of packets after which classification takes place), may cause some short flows (flows with lesser number of packets) to finish up even before getting classified into any class. But as we are dealing with multimedia and download flows which are generally long flows, it is fine to ignore shorter flows for now.

- **False Positive :** As we have described earlier that once we classify some flow as multimedia, we immediately prioritize that flow. But what if a download flow gets classified as multimedia (false positive for multimedia class)? In that case the download flow gets prioritized and this is not what we intend to do. The resulting scenario would be that both the download and the multimedia flows will be in the same high priority band.

- **Considering two traffic classes only :** We have only dealt with two traffic classes and thus any flow gets classified in to these two classes only, depending on the statistical behavior of the flow. So even if a flow is neither multimedia nor download, then also it gets tagged as one of these two classes. But generally flows belonging to other classes apart from multimedia and download are not large enough to reach our defined threshold i.e. 1000 packets, so this is not a major limitation.

Chapter 8

Conclusion

In this report, we have presented an approach which is primarily based on the statistics of the network flows and thus it tries to capture the actual behavior of the flow irrespective of the port or protocol being used. We have also developed some heuristics which can label the given trace files, and thus it is helpful in creating large size pre-labeled training data, though it needs some manual inputs. We have used K-NN and decision tree classifier algorithms to perform traffic classification using eight statistical features.

While most of the past work in this domain is applicable to wired networks and no emphasis in the scenario of wireless APs. Also there has not been much attention to identify multimedia and improve the streaming experience. Our work focuses on real time classification techniques in wireless access points for these classes and provides better quality of service, by prioritizing multimedia flows. Our model achieves 90% offline training accuracy and 87% real-time classification accuracy using decision tree algorithm with first 1000 packets. Our model shows significant improvement in latency and jitter. We are able to reduce average latency by a factor of 36 and average jitter by a factor of 39. As a result, MOS score increases by 0.7 on an average. The overall results we have achieved are encouraging to us but there are still many areas where things can further be improved.

Chapter 9

Future Work

- This work can be further extended to make it more applicable in any general network setting rather than some specific settings used here. We can also make a classifier which can classify almost every websites with high accuracy. Other traffic classes such as browsing etc., can also be taken care of and thus making the model more useful.
- Our auto labeling approach can further be enhanced to work precisely without any manual inputs.
- This whole logic can further be implemented inside the wireless access points with some additional work. Necessary changes in the code of access points can be made, so that it can do the classification followed by prioritization of the target traffic flow.

References

[1] Roughan, M., Sen S., Spatscheck, O., And Duffield N. Class-of-service Mapping for r QoS: A Statistical Signature-based Approach to IP Traffic Classification. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement* (October 2004), pp. 135-148.

[2] Andrew W. Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. *In Proceedings of the International Conference on Measurements and Modeling of Computer Systems*, SIGMET- RICS 2005, June 6-10, 2005, Banff, Alberta, Canada, pages 5060, 2005.

[3] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kav e Salamatian. Traffic classification on the fly. Computer Communication Review, 2006.

[4] Manuel Crotti, Maurizio Dusi, Francesco Gringoli, and Luca Salgarelli. Traffic classification through simple statistical fingerprinting. Computer Communication Review, 2007.

[5] Jeffrey Erman, Anirban Mahanti, Martin F. Arlitt, Ira Cohen, and Carey L. Williamson. Offline/realtime traffic classification using semi-supervised learning. Perform. Eval., 2007.

[6] M. AlSabah, K. Bauer, and I. Goldberg. Enhancing Tor's performance using real-time traffic identification. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 73-84. ACM, 2012.

[7] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. Blinc: multilevel traffic classification in the dark. *Association for Computing Machinery, Inc.*, 2005.

[8] Iliofotou, Marios, Kim, Hyun-chul, Faloutsos, Michalis, Mitzenmacher, Michael, Pappu, Prashanth, Varghese, and George. Graption: A graph-based p2p traffic classification framework for the internet backbone. Comput. Netw., 55:19091920, 2011.

[9] T. Karagiannis, A.Broido, M. Faloutsos, and kc claffy. Transport layer identification of P2P traffic. In *ACM/SIGCOMM IMC*, 2004.

[10] Decision Tree Classifier.
`http://mines.humanoriented.com/classes/2010/fall/csci568/portfolio_exports/lguo/decisionTree.html`.

[11] K-Nearest Neighbour Algorithm.
`http://www.scholarpedia.org/article/K-nearest_neighbor`.

[12] Mean Opinion Score (MOS).
`https://en.wikipedia.org/wiki/Mean_opinion_score`.

[13] Network Address Translation (NAT).
`http://www.vicomsoft.com/learning-center/network-address-translation/`.

[14] Iperf Tool.
`http://openmaniak.com/iperf.php`.

[15] Azure Speed Test server.
`http://www.azurespeed.com/Azure/Download`.

[16] Over-fitting.
`https://en.wikipedia.org/wiki/Overfitting`.

[17] Weka : A Data Mining Software in Java.
`http://www.cs.waikato.ac.nz/ml/weka/`.

[18] Linux traffic control mechanism.
`http://tldp.org/HOWTO/Traffic-Control-HOWTO/intro.html`.

[19] Hierarchy Token Bucket queuing discipline.
`http://www.tldp.org/HOWTO/html_single/Traffic-Control-tcng-HTB-HOWTO/`.

[20] Linux Firewall Tutorial: Iptables Tables, Chains, Rules Fundamentals.
`http://www.thegeekstuff.com/2011/01/iptables-fundamentals/`.

[21] Wondershaper traffic shaping tool.
`http://www.ubuntugeek.com/use-bandwidth-shapers-wondershaper-or-trickle-to-limit-internet-connection-speed.html`.

[22] Mean Opinion Score.
`http://www.voipmechanic.com/mos-mean-opinion-score.htm`.

[23] Mean Opinion Score calculation in Pingplotter.
`https://www.pingman.com/kb/article/how-is-mos-calculated-in-pingplotter-pro-50.html`.

[24] Mikrotik Board RB433AH.
`http://routerboard.com/RB433AH`.

[25] OpenWRT Binaries.
`https://downloads.openwrt.org/`.

[26] OpenWRT Barrier Breaker version.
`https://wiki.openwrt.org/doc/barrier.breaker`.

[27] OpenWRT binaries for Chaos Calmer version.
`https://downloads.openwrt.org/chaos_calmer/15.05/ar71xx/mikrotik/`.