Smart Home Network Management with Dynamic Traffic Distribution

Chenguang Zhu Xiang Ren Tianran Xu

Table of Contents

1. Introduction

In home networks, different applications often compete for limited bandwidth. Under such scenarios, high bandwidth consumption applications can be disruptive to others. For example, a BitTorrent download session can deteriorate the quality of an important Skype conference call. In such case, it would be useful for the home router to be able to identify the different traffic flow types, and allocate more bandwidth to the higher priority applications, such as Skype.

One common solution to achieve some level of traffic control is to install simple QoS rules on the router, but doing so only offers limited flexibility -- it uses a limited number of rules, such as port numbers, to decide the type of flow. A malicious application could, for example, tweak its port number to circumvent bandwidth limitations. As a result, a comprehensive identification scheme is needed to provide more accurate QoS rules, such as one that examines the payload content of the packet in depth. While such flow identification schemes are difficult to adopt in traditional networks due to their complexity, the flexibility of software-defined networking makes them feasible. One can not only easily experiment with novel identification techniques, but also easily adjust traffic rules.

To improve the current state of home network traffic control, we propose to use a software-defined controller to perform smart flow type identification and use traffic adjustment to allocate bandwidth for different types of application flows accordingly.

2. System Design

Our system consists of a simple network emulated using Mininet and a Floodlight controller. We built a module on controller that identifies application types. The switch forwards incoming packets to the controller. The controller then uses our flow identification module to determine the application type, and according to the type, pushes different flow rules to the switch.

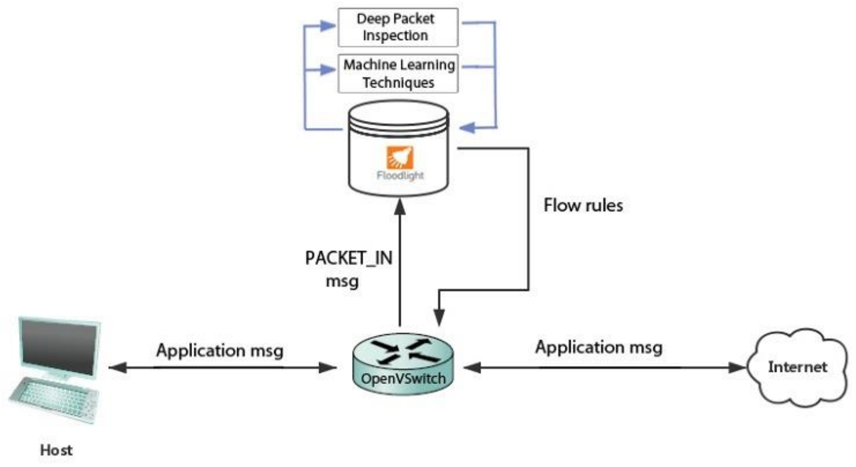


Figure 1. System overview

2.1 Flow Identification

Shallow packet inspection and deep packet inspection are two traditionally popular methods for flow identification. Shallow packet inspection identifies packets by inspecting header fields. While being simple and fast, it suffers from lower accuracy. Deep packet inspection, on the other hand, inspects the data part of packets. Because it uses rules specific to the characteristics of each type of flow, deep packet inspection can be much more accurate. However, one not only need to maintain a huge database of application specific flow features, but also has to update the rules frequently to keep up with application upgrades.

For our system, we created simple rules to implement simple deep packet inspection functionalities. However, we will focus on novel machine learning based flow identification techniques. Since machine learning uses statistical analyses, it can automatically extract the flow characteristics. More specifically, we experimented with both clustering and classification based algorithms for identification.

2.2 Traffic Adjustment

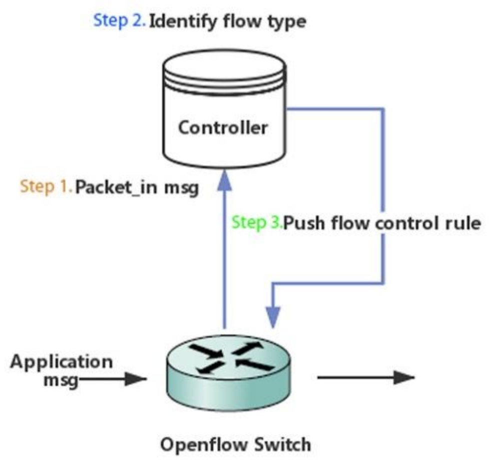


Figure 2. Traffic control work flow.

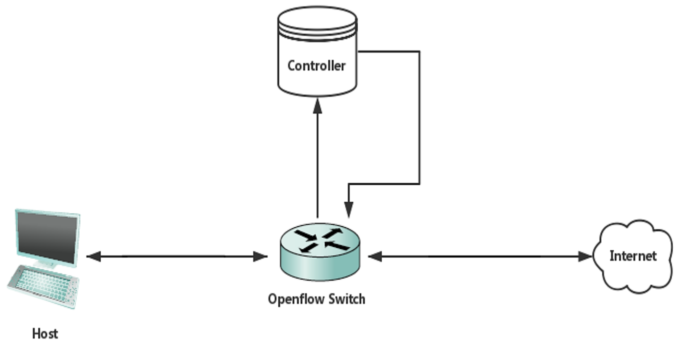
After our controller module identifies the flow type based on the first packet of a flow, we specify for the controller to push a new rule to the switch and set priority for this flow entry. Subsequent packets in the flow will follow the same rule.

3. Implementation

This section explains aspects of our implementation of the system in the following order: first we explain the topology emulation, then packet arrival processing, packet identification, and finally, flow adjustment.

3.1 Topology Emulation

The most basic home network topology that we emulated consists of a host, which connects to the internet through an openVSwitch, and a Floodlight controller that communicates with the switch. Under this topology, all the traffic between the host and the internet goes through the switch and can be analyzed and categorized by the controller.



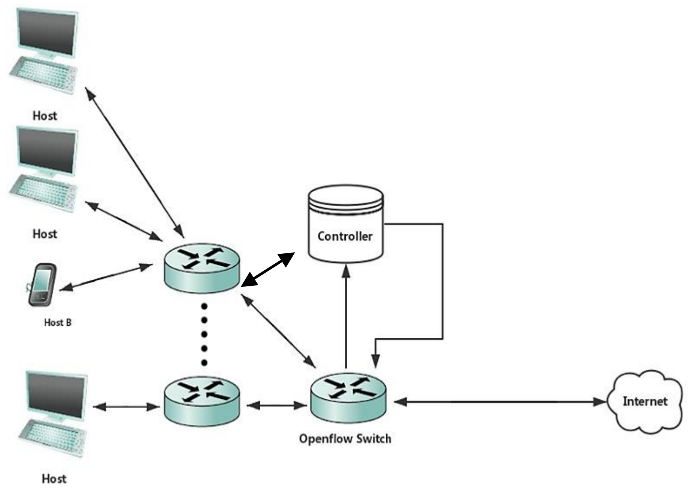


Figure 3. A simple topology used for testing

Figure 4. A more realistic topology

We could implement more complex topologies with Mininet that potentially better resemble realistic home networks. Such topology could include a large number of different types of hosts and more interconnecting switches.

3.2 Packet Arrival

When the openVSwitch receives the first packet of a new flow that does not correspond to a flow entry in the flow table, the switch sends a PACKET_IN message encapsulating the packet to the Floodlight controller. On the Floodlight's side, our custom module keeps listening to PACKET_IN messages. When the message comes in, the module starts to analyze the packet, using machine learning or deep packet inspection methods to predict the flow type of the packet.

3.3 Packet Identification

Machine Learning Algorithms: We used three types of machine learning algorithms. For clustering we used K-Means and mixture of Gaussians (MoG) and for classification we used support vector machine (SVM). Clustering algorithms are unsupervised, and group data points into k clusters, where points belong to the nearest cluster. Whereas k-means only forms circular clusters, MoG makes less assumptions about data distribution than K-Means and generally can be more accurate [1]. Classification algorithms such as SVM assigns data into categories, and learns the assignment rules from the true type – label of each data point in the training set [2]. We used implementations of the aforementioned algorithms from the scikit-learn [3] libraries to build our identification module.

Dataset Selection： We need to select adequate dataset for training our machine learning models. A popular choice is to use publically available research traces, which offer representative workloads, however, such traces are hard for us to determine the true flow type of each packet, and will complicate our experiments. As a result, we chose to generate our own traffic, and capture the packets using Wireshark. This approach allows us to obtain accurate flow types for the packets, and easily label each packet.

Feature Selection： We also need to decide on which features to train on. A variety of flow characteristics have been used as features, as summarized in Table 1. However, these features require an extra step of first identifying flows in the training dataset. To simplify our algorithms, we use a packet level characteristic, where we examine the first N bytes of a packet's data part. We varied N to see how our model respond differently. In subsequent experiments, we also consider the destination port number of the packet as part of our feature.

Total number of packets per flow	Flow duration	Packet lengths statistic (min, max, mean, std dev.) per flow	Payload lengths	Payload content

Table 1. Popular features for flow identification [4]

Training and Prediction Workflow： With training dataset and features selected, we can now train the machine learning models. Our training and prediction workflow is as follows: We first parse captured packet into byte arrays, then extract first N bytes of a packet's payload, or optionally include the destination port as input feature. We then train the models using K-Means, MoG and SVM algorithms from scikit-learn libraries, and finally we save the trained models and use them for future predictions.

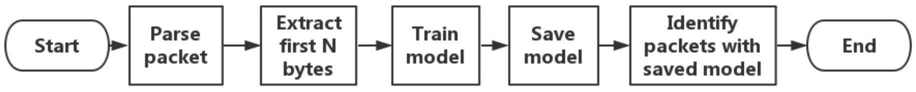
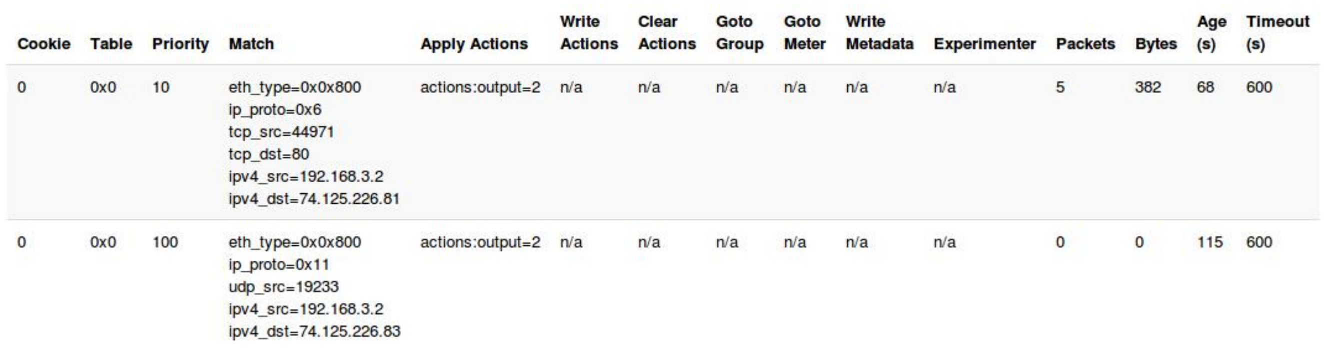


Figure 6. Workflow for model training and prediction

3.4 Traffic Adjustment

Once a flow's application type is identified, the controller sets different priorities for the flow entry based on application type, then pushes the rule to the switch. However, setting flow entry priorities does not actually affect the bandwidth allocated to the flow, it simply helps show our abilities to identify and control the flow. For this project, we do not emphasize the actual bandwidth control functionality, and instead focus on the flow identification feature. Controlling the bandwidth of a flow can be difficult, not only does it require complex rules, the underlying switches may also not support such control features. One possible way to control the rate of a specific flow is to have multiple paths where we direct a flow through paths of different link rates.



Cookie	Table	Priority	Match	Apply Actions	Write Actions	Clear Actions	Goto Group	Goto Meter	Write Metadata	Experimenter	Packets	Bytes	Age (s)	Timeout (s)
0	0x0	10	eth_type=0x0x800 ip_proto=0x6 tcp_src=44971 tcp_dst=80 ipv4_src=192.168.3.2 ipv4_dst=74.125.226.81	actions:output=2	n/a	n/a	n/a	n/a	n/a	n/a	5	382	68	600
0	0x0	100	eth_type=0x0x800 ip_proto=0x11 udp_src=19233 ipv4_src=192.168.3.2 ipv4_dst=74.125.226.83	actions:output=2	n/a	n/a	n/a	n/a	n/a	n/a	0	0	115	600

Figure 7. Example flow rules in our system

4. Evaluation

We tested our system on 4 kinds of flows: HTTP, SSL, Skype [5], and BitTorrent [6], using 3 types of machine learning algorithms: K-Means, Mixture of Gaussians (MoG), and Support Vector Machine (SVM). We evaluated the models' performances by varying input feature lengths, and whether to include port number as feature.

4.1 Using Payload Data as Features

We trained our models based on features taken from the first N bytes of the TCP or UDP payload, where N is set to 2, 3, 4, 8, and 10. As a result, we look at the 54th to 54 + Nth bytes of the payload for a TCP packet, and 42th to 42 + Nth bytes for a UDP packet.

4.1.1 K-Means Clustering

For K-Means, we specified for the algorithm to group input packets into 8 clusters. The number of cluster is greater than the number of flow types, for higher accuracy. Figures 8 - 11 are resultant packet clusters.

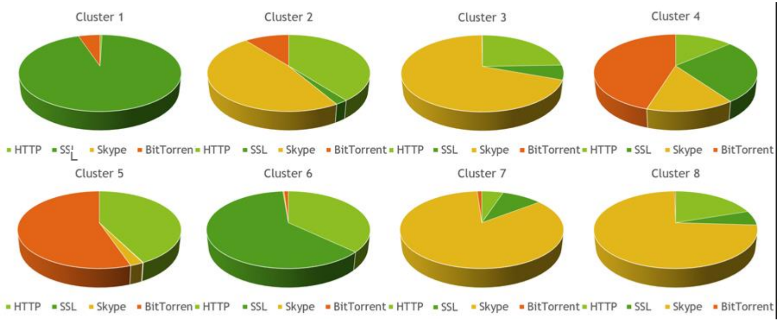


Figure 8. K-Means clusters with 2 bytes

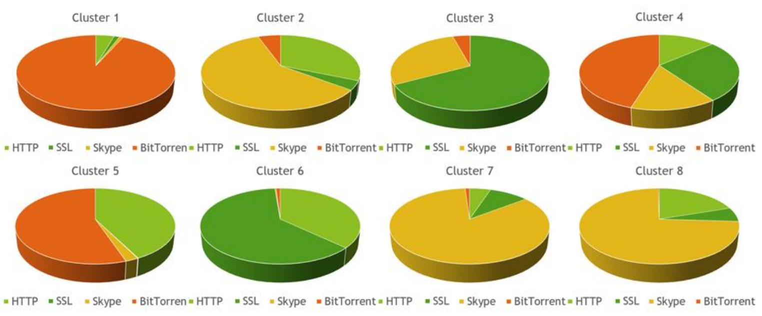


Figure 9. K-Means clusters with 3 bytes

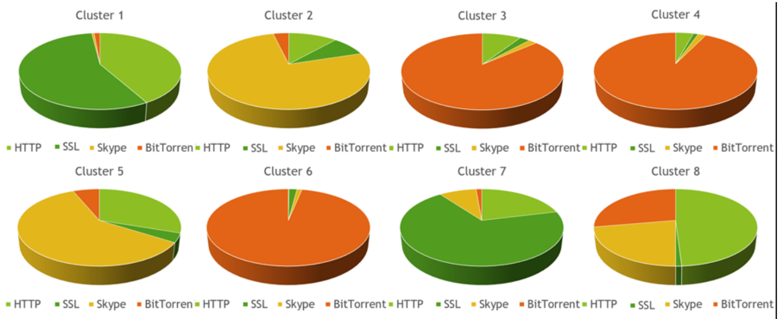


Figure 10. K-Means clusters with 8 bytes

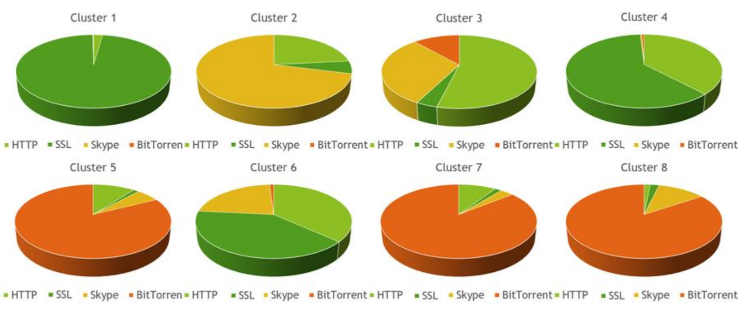
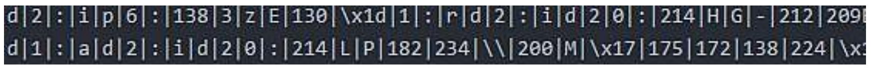


Figure 11. K-Means clusters with 10 bytes

A cluster is more accurate when it primarily consists of one kind of packet. Note that different clusters can have very different sizes. The first cluster mostly contains packets from Skype, and we have high confidence that subsequent packets grouped into the cluster are Skype packets. In the second cluster, SSL packets are the majority, but it also contains a significant number of BitTorrent packets, making it a less accurate cluster.

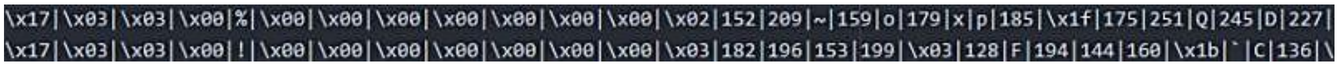
Clusters are also not static and change according to the varying feature lengths. When there are 2 bytes, BitTorrent packets do not have its own majority, and is confused with SSL, HTTP, and Skype. However, when we increase the feature length to 3 bytes, BitTorrent packets can now be clearly grouped into two clusters, but the clusters for HTTP and SSL become less distinguishable. K-Means achieves its best result around 8 bytes, where each cluster now has a clear majority type.

K-Means does not require us to pre-label the type of packets, rather, it groups packets based on their statistical similarities. The fact that we were able to obtain quite distinctive clusters are reasons for us to believe that there are common patterns in the first few bytes of the packets. To validate our hypothesis, we manually examined the payload of the packets converted to a human readable form. Some example payload contents are shown in Figure 12, 13 and 14, where each line is a different segment starting from the beginning of the payload. We added a '|' to separate each byte and escaped each character.



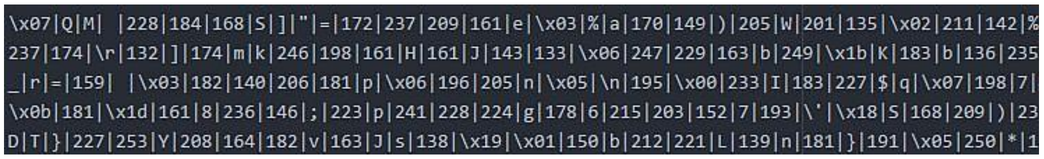
```
d|2|:|i|p|6|:|138|3|z|E|130|\x1d|1|:|r|d|2|:|i|d|2|0|:|214|H|G|-|212|209
d|1|:|a|d|2|:|i|d|2|0|:|214|L|P|182|234|\\|200|M|\x17|175|172|138|224|\x
```

Figure 12. Sample parsed payload bytes for BitTorrent



```
\x17|\x03|\x03|\x00|%|\x00|\x00|\x00|\x00|\x00|\x00|\x00|\x02|152|209|~|159|o|179|x|p|185|\x1f|175|251|Q|245|D|227|
\x17|\x03|\x03|\x00|!|\x00|\x00|\x00|\x00|\x00|\x00|\x00|\x03|182|196|153|199|\x03|128|F|194|144|160|\x1b|`|C|136|\
```

Figure 13. Sample parsed payload bytes for SSL



```
\x07|Q|M| |228|184|168|S|]|"|=|172|237|209|161|e|\x03|%|a|170|149|)|205|W|201|135|\x02|211|142|%
237|174|\r|132|]|174|m|k|246|198|161|H|161|J|143|133|\x06|247|229|163|b|249|\x1b|K|183|b|136|235
_|r|=|159| |\x03|182|140|206|181|p|\x06|196|205|n|\x05|\n|195|\x00|233|I|183|227|$|q|\x07|198|7|
\x0b|181|\x1d|161|8|236|146|;|223|p|241|228|224|g|178|6|215|203|152|7|193|\'|\x18|S|168|209|)|23
D|T|}|227|253|Y|208|164|182|v|163|J|s|138|\x19|\x01|150|b|212|221|L|139|n|181|}|191|\x05|250|*|1
```

Figure 14. Sample parsed payload bytes for Skype

Even though the bytes do not always translate to human readable information, possibly due to encryption or compression, the payloads still exhibit application specific characteristics at the beginning, and showed similarities. We were able to manually detect patterns from HTTP, SSL, and BitTorrent packets. Interestingly, from our rough scan, BitTorrent seems to exhibit roughly 3 kinds of patterns at the beginning of each payload, and here we only showed one of them in Figure 12. We believe these patterns are the reasons K-Means could group the packets into reasonable clusters. We were not able to manually observe any patterns from Skype payloads because they are encrypted, and thus the reason deep packet inspect does not work for them, but the fact that Skype had distinctive clusters suggests latent patterns.

4.1.2 Comparison of Algorithms Based on Feature Length

We compared the correct identification rates for K-Means, SVM, and MoG. The best performance for SVM is 88.975%, 77.65% for K-Means, and 73.55% for MoG. SVM clearly has the best performance, whereas the other two clustering algorithms perform close to each other, as expected due to their algorithmic similarities. The reasonable performances could be explained by the fact that there are indeed a set of application specific bytes at the beginning of the payload, giving rise to clear statistical distributions.

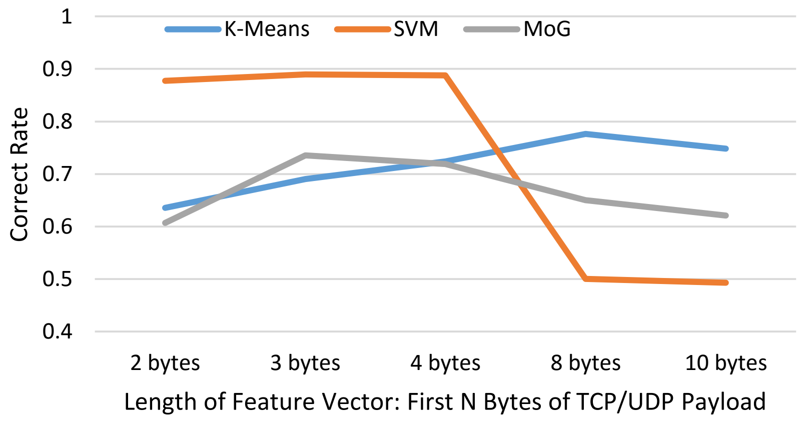
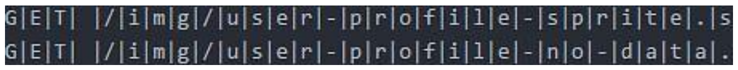


Figure 15. Correct rates under 3 models for different feature length

Interestingly, the best feature length is different for each algorithm. Whereas SVM yields best results for 2 to 4 bytes, K-Means performs best around 8 bytes and MoG 3 bytes. The varying responses to feature length is not unexpected, because each algorithm makes different assumptions about the data distribution. Whereas K-Means assumes circular Gaussian distributions, MoG assumes elliptical Gaussian distributions. Even though K-Means makes more generalization about data, it might be more robust against overfitting outliers, and as a result outperforms MoG. SVM on the other hand, is heavily dependent on points close to the decision boundary, and too much noise around the boundary can cause unclear distinction between different categories. As a result of these characteristics, there seems to be evidence that the first 2 to 4 bytes of the payload contain application specific information, whereas data that follows might be more random.

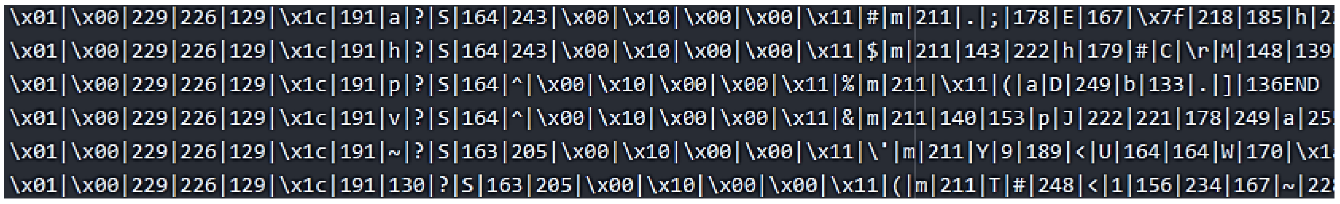
We tried to reason about the varying trends by manually examining the packet payloads. We observed that both SVM and MoG perform best at 3 bytes and worsen at 4 bytes. We infer that this could be influenced by HTTP packets. As seen in Figure 16, 'GET' is a popular HTTP command, and since it has 3 characters, the fourth character is not fixed. So we infer that large numbers of 'GET' commands in our dataset could introduce uncertainty for the fourth bytes. This could have strong effect on clustering since when we use relatively few number of bytes, each byte can have significant influence on grouping and classification decisions.



```
G|E|T| |/|i|m|g|/|u|s|e|r|-|p|r|o|f|i|l|e|-|s|p|r|i|t|e|.|s
G|E|T| |/|i|m|g|/|u|s|e|r|-|p|r|o|f|i|l|e|-|n|o|-|d|a|t|a|.
```

Figure 16. Sample parsed payload bytes for HTTP

We also observed that most algorithms perform badly at or past 8 bytes. This could be caused by random information introduced beyond certain number of bytes. For example, BitTorrent payloads shown in Figure 17 exhibit the same pattern for the first 7 bytes, but bytes beyond 7 start to show variations.



```
\x01|\x00|229|226|129|\x1c|191|a|?|S|164|243|\x00|\x10|\x00|\x00|\x11|#|m|211|.|;|178|E|167|\x7f|218|185|h|2
\x01|\x00|229|226|129|\x1c|191|h|?|S|164|243|\x00|\x10|\x00|\x00|\x11|$|m|211|143|222|h|179|#|C|\r|M|148|139
\x01|\x00|229|226|129|\x1c|191|p|?|S|164|^|\x00|\x10|\x00|\x00|\x11|%|m|211|\x11|(|a|D|249|b|133|.|]|136END
\x01|\x00|229|226|129|\x1c|191|v|?|S|164|^|\x00|\x10|\x00|\x00|\x11|&|m|211|140|153|p|J|222|221|178|249|a|25
\x01|\x00|229|226|129|\x1c|191|~|?|S|163|205|\x00|\x10|\x00|\x00|\x11|\'|m|211|Y|9|189|<|U|164|164|W|170|\x1
\x01|\x00|229|226|129|\x1c|191|130|?|S|163|205|\x00|\x10|\x00|\x00|\x11|(|m|211|T|#|248|<|1|156|234|167|~|22
```

Figure 17. Sample parsed payload bytes for BitTorrent

4.1.3 Comparison of Algorithms Based on Sample Size

We compared the performance of each model under different train dataset sizes. For K-Means, we fix the number of byes to 8, and for SVM and MoG 3. The performance of SVM and MoG improves as the sample size increases, as expected. However for K-Means there is no clear trend. This could be that the best feature length changes for K-Means as the sample size changes.

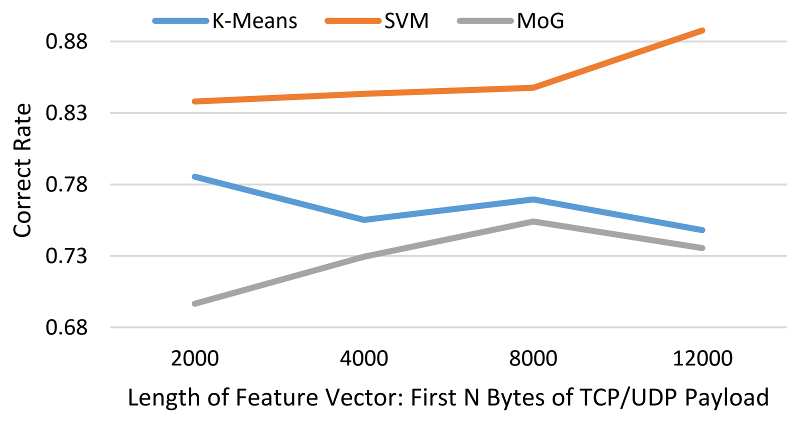


Figure 18. Correct rates under 3 models for different sample sizes

4.2 Using Data and Port Number as Features

Certain header fields can also be useful indication of the application type. Combining header info with payload data could yield stronger features. We decided to use destination port number as an example of header fields, and repeated some of the previous experiments with the new features. The destination port number is presented by the 36th and 37th bytes in the packet.

4.2.1 K-Means Clustering

Adding the destination port number to the payload data results in correct rates almost opposite in trend from the data-only one. Note that in Figure 19, the number of bytes on the x-axis only represents the length of payload data, rather than the total lengths of feature in the case the destination port number is added to the feature.

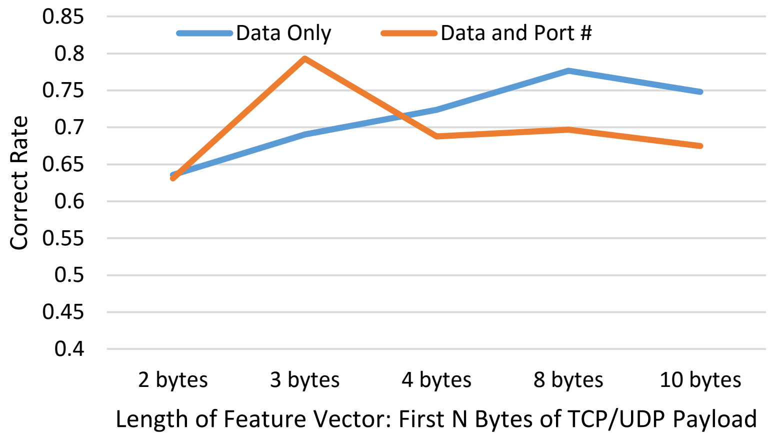


Figure 19. Correct rates for K-Means using data only or data and port number as feature

With the destination port number added, the algorithm now performs best for 3 bytes of payload data instead of 8. The algorithms also achieves a better optimal performance of 79.32% instead of 77.65%.

4.2.2 Comparison of SVM Performance for Different Features

For SVM, the trend in correct rate is very similar for both types of features. Additionally, with destination port number added, the algorithm consistently yields worse performances. It seems that the addition of destination port number is not beneficial to SVM.

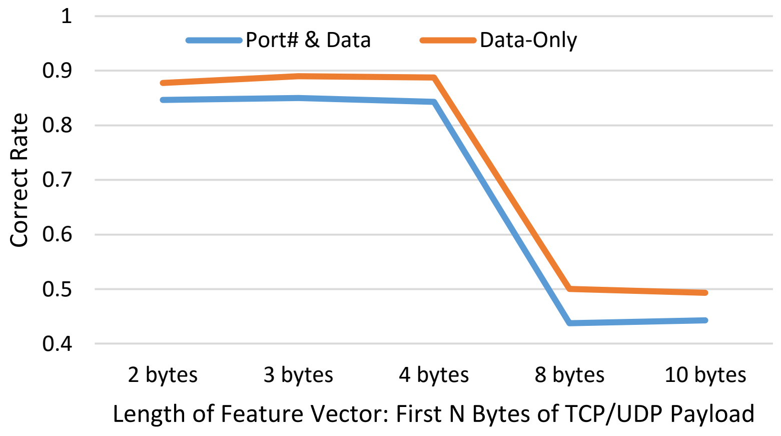


Figure 20. Correct rates for SVM using data only or data and port number as feature

4.2.3 Comparison of Mixture of Gaussians Performance for Different Features

For MoG, the trend in correct rate is again very similar for both types of features. However, now with destination port number added, the algorithm consistently yields significantly better performances. In this case, the addition of destination port number is beneficial to MoG.

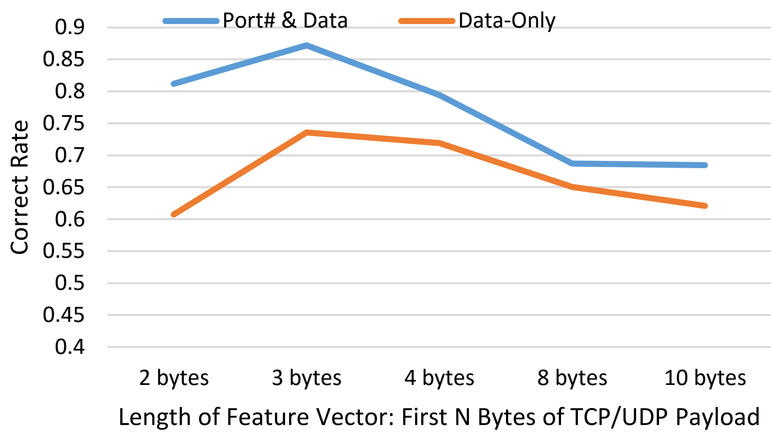


Figure 21. Correct rates for MoG using data only or data and port number as feature

In conclusion, while adding destination port number is beneficial to some algorithms, it can be harmful to others, making it difficult to categorically arrive at a conclusion for its effect.

5. Limitations and Future Work

Due to time constraints, our project has limited scope. We did not use representative and realistic traces, and only used 4 separate kinds of flows for training. In real life, there could be hundreds of different flows mixed together. We also only sampled 12000 packets for training, and sampled packets from contiguous time durations, making our training set less representative.

For future work we would like to build models for more application types, for example, VPN traffic, and media application traffic. We would also like to try additional machine learning algorithms, more sophisticated pipelines of algorithms, as well as better tuning the hyper-parameters for the existing algorithms. Finally, we hope to test our framework on realistic topologies with diverse hosts and more interconnecting switches.

6. Conclusion

In this project, we used machine learning and deep packet inspection techniques to identify application types for different flows based on simple packet payload and header features and pushed rules with different priorities based on the application type. We achieved high accuracy for identification, with a best correct rate of 87.5% using SVM, 79% using K-Means, and 87.2% using Mixture of Gaussians. Furthermore, in our limited training and test sets, we could differentiate traffics from Skype and BitTorrent quite well, a task that is difficult to achieve in Wireshark. Based on this result we conclude that there must be some application specific information in the beginning of the packets payload and this information could be used for effective application type identification in practice.

References

[1] Wikipedia, 'K-means clustering', 2015. [Online]. Available: https://en.wikipedia.org/wiki/K-means_clustering. [Accessed: 09- Dec- 2015].

[2] Wikipedia, 'Support vector machine', 2015. [Online]. Available: https://en.wikipedia.org/wiki/Support_vector_machine. [Accessed: 09- Dec- 2015].

[3] Scikit-learn.org, 'scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation', 2015. [Online]. Available: http://scikit-learn.org. [Accessed: 09- Dec- 2015].

[4]T. Nguyen and G. Armitage, 'A survey of techniques for internet traffic classification using machine learning', IEEE Communications Surveys & Tutorials, vol. 10, no. 4, pp. 56-76, 2008.

[5] Skype.com, 'Skype | Free calls to friends and family', 2015. [Online]. Available: http://www.skype.com/en/. [Accessed: 09- Dec- 2015].

[6]I. BitTorrent, 'BitTorrent', Bittorrent.com, 2015. [Online]. Available: http://www.bittorrent.com/. [Accessed: 09- Dec- 2015].