1 INTRODUCTION

All information on the internet is communicated in chunks of data known as packets. The formats of such packets are standardized in the form of protocols. In a commercial setting, there can be a large number of protocols involved in the network traffic, and it is often important from the point of view of security to establish baseline behavior for the network so that anomalous behavior can be detected. It is common to do this by inspecting packets beyond the usual wrappers for standardized network communication. This form of analysis is usually termed as Deep Packet Inspection (DPI).

But there are two challenges to doing DPI effectively. The first is to acquire specifications of the protocols involved. While vendor-neutral communication is often necessarily standardized as open protocols, vendor-centric communication follow proprietary protocols that have no public documentation. The second challenge is simply the problem of scale. It is difficult to incorporate knowledge of the large variety of protocols in a typical network setting, especially when they are not even known prior to deployment.

Currently, the manual process of deriving unknown protocol specifications for purposes such as DPI can be a complex, time-consuming, and challenging process. This is because 1) A protocol message usually contains many unknown fields; 2) Any individual field may be dynamic and have variations in size; 3) It is highly probable that there are complex relationships or dependencies between the fields. As mentioned in [7], it took 12 years for the open-source SAMBA project to manually reverse engineer the Microsoft SMB protocol [4].

In this work, I developed and evaluated Analytics, a tool which can help in the analysis of packet data for deep packet inspection. In its current state, Analytics is effectively a library consisting of functions that supports manual analysis. Given a set of packets, the tool attempts to discover constants, enumeration (enum) fields and strings among them, and provides visualization to aid analysts. If the packets are packed in the same protocol format, such analysis can reveal aspects of the protocol specification. While the tool supports only manual analysis at the moment, I show through experiments conducted on a variety of protocols that Analytics offers a good basis for automation in future. Analytics is written in Python-and uses the free graphing software Plotly for visualization. [11]

I have evaluated Analytics against fourteen open protocols, in packet capture files with at least fifty packets of a given protocol. The data obtained from Analytics was interpreted and analysed to make inferences about the given protocol, and inferences were then cross-checked against the online documentation of the protocol to evaluate the efficacy and utility of Analytics. The experimental results are encouraging. Across the fourteen protocols tested, Analytics has an average accuracy in detecting constants of 76.808% and an average accuracy in detecting enum fields of 88.625%. Although Analytics is limited in the fact that it is unable to tell the meanings or significance of the fields, it allows their boundaries to be identified, which is very important for protocol analysis. Analytics does not require accessing protocol source code and is therefore applicable to analyzing network environments with proprietary or unknown protocols.

2 DESIGN AND FUNCTIONS

Analytics was created with the aim of deriving protocol specification information from network traces. The current implementation of Analytics is concerned with discovering constants and enumeration values (enums) in protocol formats. A separate heuristic for each is implemented. Analytics also has a visualization component built on top of the free graphing software Plotly [11]. From these, an analyst can make several inferences about the protocol's structure and highlight key areas of interest in a protocol format.

One area of interest are constant fields, these fields are unvarying across every packet in the protocol. Another area of interest are enum fields, which encode a limited range of values.

As an example, let us take version 4 of the Internet Protocol (IPv4). The IPv4 header is shown in Figure 1 with what can be considered to be its constant and enums marked. For IPv4 packets, the Version field must be set to 4 and is therefore a constant. The Flags field is an enum type that indicates fragmentation. The Protocol field is an enum type with its values indicating the protocol at the next layer;

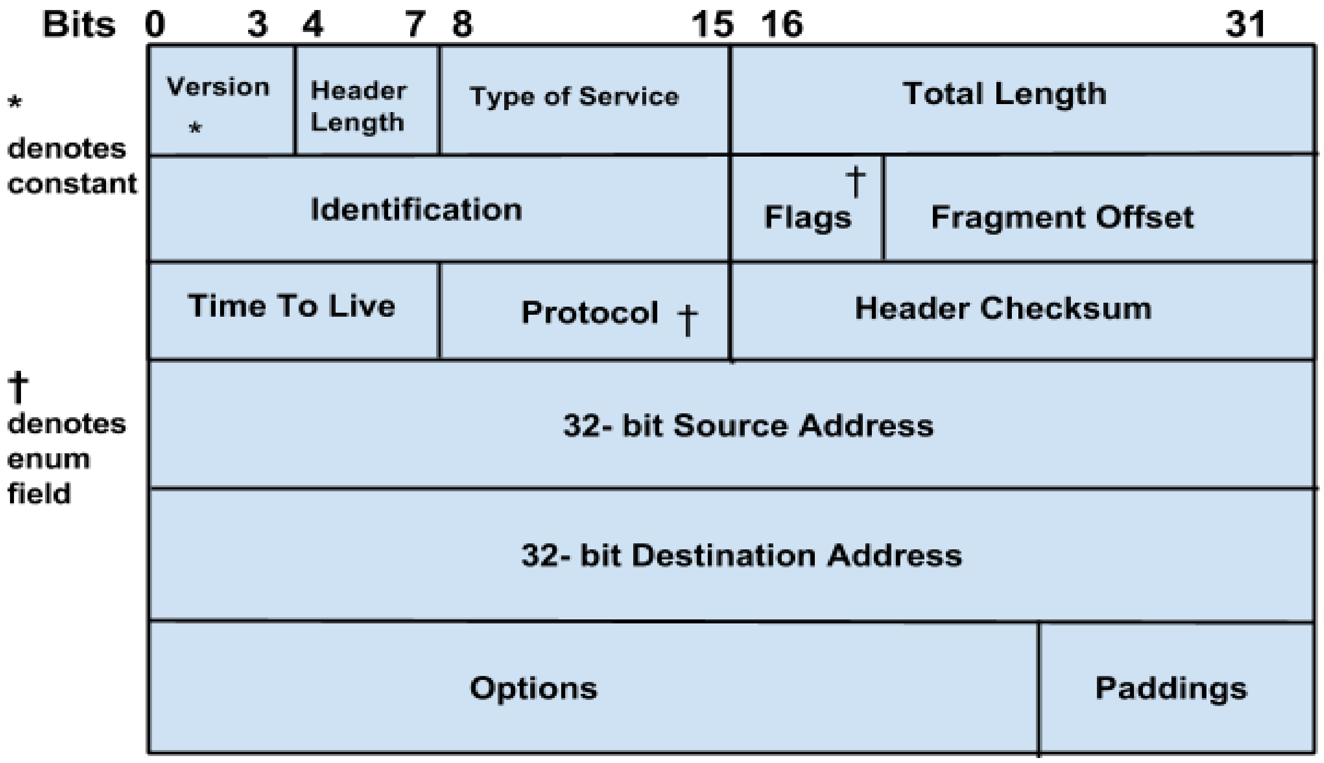


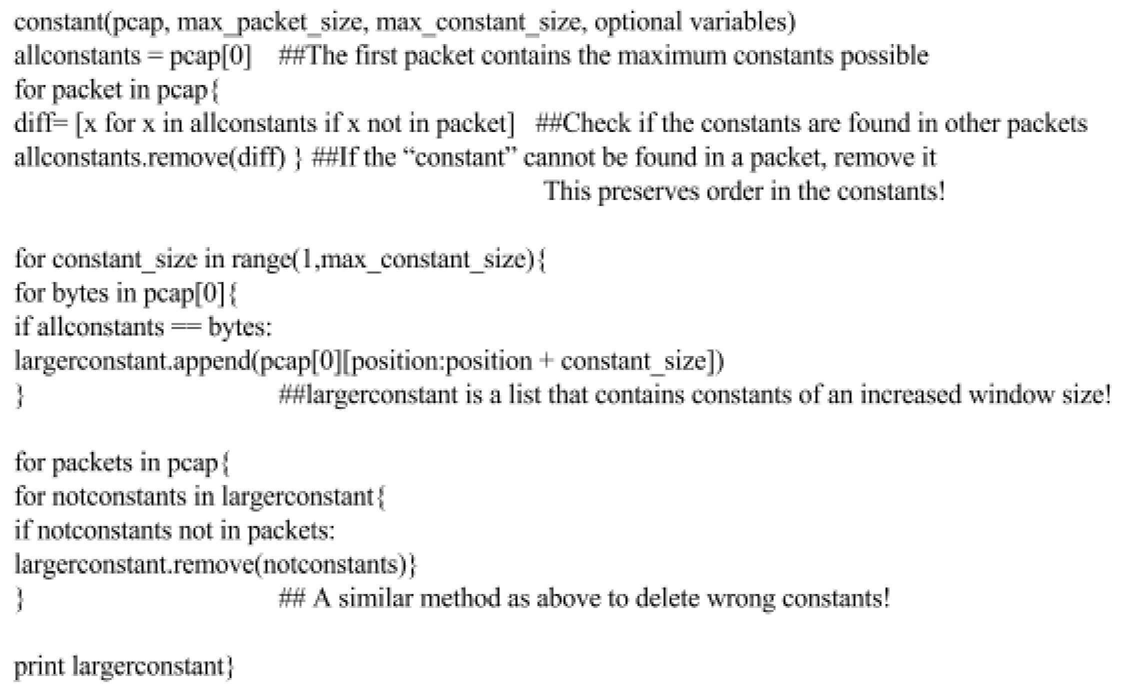
Figure 1: Illustration of a IPv4 header with significant fields highlighted

In the next sections, I discuss the key features of Analytics and their implementation.

2.1 IDENTIFYING CONSTANTS

Analytics uses a simple algorithm to identify common chunks of data in a given set of packets regardless of where they occur. If the packets are all in the same protocol format, the common bytes can indicate constants specified in the protocol specification. This is especially so for larger contiguous chunks. The chunks of data may represent paddings, protocol version, or reserved fields, et cetera. As an example, for the HTTP protocol, Analytics is easily able to identify common fields such as “HTTP/1.1”, “Host”, “Connection”, and “Content Length”. However, it is up to the user to draw his own conclusions about the protocol format from the data extracted from Analytics.

The algorithm is as follows: Analytics will read the first packet of the capture file and loop through the rest of the packets. The first packet contains all possible constants in the file, so all its data will be put into a list. The following packets will then be compared against the list, if an item in the list is not found in the current packet, it will be removed from the list. The end product is a list of constants of one byte. The process continues, with Analytics restarting from the first packet, but the window size increases by one, until the window size hits the maximum constant length. Analytics will look at the next byte of the constants found in the first list, and checks if they can be found in the packet capture file. This repeats, with the window size increasing by one every iteration, until the maximum constant length has been reached. Analytics gives the user a list of constants of increasing size that was found in every packet, regardless of position.



```
constant(pcap, max_packet_size, max_constant_size, optional variables)
allconstants = pcap[0]    ##The first packet contains the maximum constants possible
for packet in pcap{
diff= [x for x in allconstants if x not in packet]   ##Check if the constants are found in other packets
allconstants.remove(diff) } ##If the "constant" cannot be found in a packet, remove it
                                                      This preserves order in the constants!

for constant_size in range(1,max_constant_size){
for bytes in pcap[0]{
if allconstants == bytes:
largerconstant.append(pcap[0][position:position + constant_size])
}                              ##largerconstant is a list that contains constants of an increased window size!

for packets in pcap{
for notconstants in largerconstant{
if notconstants not in packets:
largerconstant.remove(notconstants)}
}                              ## A similar method as above to delete wrong constants!

print largerconstant}
```

2.2 IDENTIFYING ENUMERATION VALUES

Analytics also has a `statistics` function that checks every position in every packet and outputs the number of different values that can be found in the file. It will also notify the user if a position in the file has been detected to be a "constant" (with only one value) or an enumeration value (with two to four values). The threshold for determining if a position is an enumerator value is by default four or less. `statistics` prints the number of distinct bytes at every position onto the shell, up to a default window of size 5. The function has the option to show the Shannon Entropy value [12] of every position in the packet. This is one of the main highlights of the graph for enum fields. The Shannon Entropy level of each position is calculated by counting the number of different values that occur in every position and the number of times that they occur; probability of a value occurring = number of times value occurred / total number of packets. Entropy value is calculated by $\sum_i (p_i) \log\left(\frac{1}{p_i}\right)$, where p_i is the probability of the value occurring.

2.3 STRINGS AND OTHER FUNCTIONS

Analytics also contains other functions that we do not focus on in this paper. Analytics can check for and print ASCII strings that it has found in every packet. Analytics can also read binary files and perform the above analyses on them. Finally, Analytics allows the user to map their own function onto the packets. (See Appendix for details)

2.4 PLOTS

Analytics also includes graph plotting functions for the above main functions, called `constant_plot` and `statistics_plot`, for easy visualization and analysis of the network protocol. Analytics uses Plotly [11] for its graphing functions.

`constant_plot` allows the user to input a constant found in `constant` and view a bar graph or heatmap of all the positions where the given constant was found.

`statistics_plot` presents the number of different values that was found in every position of all packets in the network trace, and also displays its entropy level. There are also two line markers for the user to easily identify constant and enum fields in the graph.

3 EXPERIMENTAL DEMONSTRATION

To evaluate the utility, value and ease of use of Analytics, I have used Analytics to analyse and understand the structure of fourteen network protocols headers. These are: ICMP, TCP, IPv4, IPv6, UDP, ARP, DNS, NetBIOS Net Service, LLMNR, IGMP (Version 3), SCTP, OSPF, and SMB. These are all open protocols in order to facilitate the cross checking of my findings with the online protocol specifications. Packet capture files of at least fifty packets of a single network protocol were used during experimentation, as it was deemed to be large enough size for the experiments. Packet capture files were obtained from Contagio [13], or my personal network traces on different networks during usual internet browsing. In order to ensure that only a single protocol was being evaluated by Analytics, packet capture files were initially filtered by Wireshark [1] before experimentation. Packets from different files obtained were merged into a single file in order to simulate more generic network traffic data for Analytics to work on, in order to simulate a "chokepoint" of various network traces, to ensure precision of results. The experiments was conducted by having Analytics analyze the single merged packet capture file of various files of the same protocol. Aggregated results of the experiment are shown in Section 3.4.

As the method of data collection involved only endpoint network traces, certain "pseudo-constants" may be detected by Analytics that are constant within the file or network traces themselves, but are not true constants as defined by the protocol specification. This also applies to enum fields. Despite efforts to reduce this experimental error, more false positives for both fields than the original protocol specification are expected to be obtained.

I am going to demonstrate the use of Analytics to analyse the TCP protocol, focusing on detection of constants and enums. TCP is a very widely-used open protocol and had very easily obtainable packet traces. I was aware of the structure of the TCP protocol [10] prior to experimentation, so this was conducted with the foreknowledge of the TCP protocol.

3.1 EXAMINING THE TCP PROTOCOL

A single packet capture file consisting of 200 TCP packets was used in the examination of the TCP protocol. A TCP packet refers to any packet that contains the TCP protocol header. These packets were filtered by Wireshark [1] to ensure that they contain a TCP header. Packet capture files were analysed with `statistics_plot` and the analysis is detailed below.

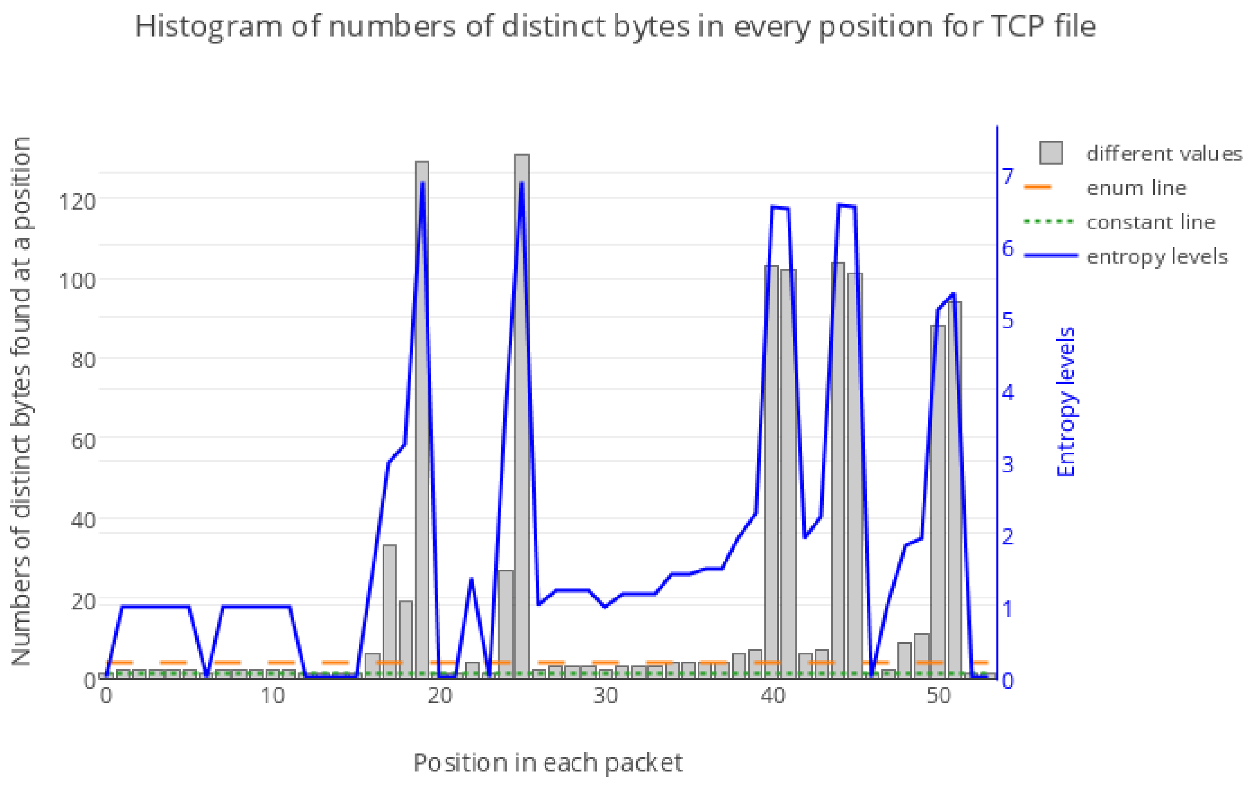


Figure 3: A packet capture file of 200 TCP packets was analysed with statistics_plot()

Using Analytics, several patterns in the TCP protocol can be seen. There are two plateaus in entropy levels from the start of the graph to the 12th byte of the graph. These imply that the data is in chunks of 6 bytes. The entropy level of the graph is zero from the 12th to 15th byte. This represents a constant value at these constant positions. There is a sharp peak from the 16th to 19th byte, indicating some form of data value which would cause higher entropy levels. Data values that cause higher entropy values include checksums, header lengths, sequence numbers, and more. There is a sudden drop to zero entropy level at positions 20 and 21 which indicates more constants, followed by an enum at position 22 and another constant at position 23. Another peak was seen at positions 24 and 25, two-byte positions have higher entropy values. The graph then drops low and entropy fluctuates around a value of 1 from positions 26 to 37, and these are all enum fields. Finally, the graph increases to several peaks beginning from position 38, with three peaks occurring at 40-41, 44-45, and 50-51. Constants are found at positions 46, 52, and 53, while a possible enum was found at position 47.

3.2 ANALYSIS

The five constants found independently by `constant` can be matched onto the graph by `statistics_plot`, the constant from the 12th to 15th position is [0x08, 0x00, 0x45, 0x00], the constant from the 52th and 53th position is [0x00, 0x00], and the constant at the 23th is 0x06. Meanwhile, the other constants [0xc4, 0x54, 0x44, 0xe1, 0x36, 0x83] and [0xc0, 0xa8, 0x01, 0x0b] are both found in enumerator positions on `statistics_plot`. Upon further analysis with other TCP files, it was found that [0xc4, 0x54, 0x44, 0xe1, 0x36, 0x83] and [0xc0, 0xa8, 0x01, 0x0b] could not be detected in the other TCP packet files. This implies that these constants are pseudo-constants, and are not true constants as stated in the protocol specification.

4 AGGREGATED RESULTS

These are the aggregated results obtained from the evaluation of Analytics to detect constants and enum fields. Manual analysis of the protocol header was compared against the ground truth obtained from the relevant online documentation. I mark every byte in the protocol header as {Constant, Not Constant} and {Enum, Not Enum} according to the protocol specification in the following way: A byte is constant if and only if it is part of a constant field, and similarly for an enum. Analytics attempts to discover the constants and enums bytewise in the protocol header. The accuracy of each field is calculated as follows: **Accuracy = (True Positives + True Negatives) / Total header length.**

Protocol	**Accuracy (Constants)**	**Accuracy (Enums)**
TCP	100%	95%
IPv4	90%	95%
Ethernet	100%	100%
IPv6	97.5%	92.5%
DNS / NetBIOS-NS / LLMNR	41.7%	91.6%
ARP	50.0%	62.5%
ICMP	100%	100%

UDP	100%	100%
OSPF	30%	80%
IGMP Version 3	50%	62.5%
SCTP	100%	100%
SMB	62.5%	84.4%
Average Accuracy	76.808%	88.625%

The experimental results show that Analytics has a higher average accuracy rate in detecting enums than constants. This is because the aforementioned pseudo-constants (i.e. values that are constant due to the trace collection at endpoints) are more common than pseudo enum fields. Pseudo constants and enums still appear in the merged packet capture despite efforts to diversify its sources, resulting in false positives and thus a lowered accuracy level. This is notable in some of the protocols tested, such as OSPF, DNS, ARP, SMB, and IGMP. For example, in the SMB protocol header as detailed by Microsoft [15], the 8-byte field "Security Features" field is not a constant in two scenarios: If security signatures are negotiated, or if SMB signing has been negotiated. Otherwise, the 8 bytes are treated as a constant reserved field. Due to the method of data collection, I have only obtained SMB packets of the latter scenario, and the "Security Features" field is a pseudo constant. Analytics detects this field as a constant, resulting in a false positive and significantly reduced accuracy.

The protocols tested which have been significantly affected by this phenomenon is OSPF and DNS/LLMNR/NetBIOS which share a similar protocol header format, with an accuracy rate for constants at 30% and 41.7% respectively. OSPF is a 24-byte header with a 2-byte Authentication Type field and an 8-byte Authentication field, among others. It also contains a 4-byte Area ID if one is being used. If the Authentication Type is set to 0 (No Authentication), the following 8 bytes will be reserved. Analytics detects these total of 14 bytes as pseudo constants, resulting in massive accuracy drops. In a similar vein, the 12-byte DNS header contains 2-byte fields of QD, AN, AS and AR Counts, with the first byte or entire field usually reserved. As I was unable to obtain packets where all of these fields demonstrate activity, they are pseudo constants and have to be marked False during evaluation of Analytics.

5 RELATED WORK

This section will describe related work and compare it to Analytics.

Some recent work in automatic protocol reverse engineering such as REWARDS [14], Dispatcher [9] and Polyglot [6] require binary analysis or to monitor binary execution, which is completely different from the methods used by Analytics. Analytics only requires network traces to perform analysis, and does not monitor binary execution.

Two automatic protocol reverse engineering projects that are similar to Analytics in using network traces are Discoverer [7] and Protocol Informatics (PI) [5]. PI uses sequence alignment techniques to infer protocol formats with similar byte sequences. Discoverer takes a step further, and uses recursive clustering techniques with a type-based sequence alignment algorithm to infer message formats. In fact, Discoverer's method of sorting protocols by format is complementary to the use of Analytics, in order to have cleaner network traces and perform more meaningful analysis.

6 LIMITATIONS AND FUTURE WORK

Analytics is limited in the fact that it can only identify a single protocol from a packet capture file. Therefore, packet capture files have to be filtered beforehand to separate packets of different formats before use by Analytics. Tools like Discoverer [7] which filter network protocols by recursive clustering are great to use in combination with Analytics.

Analytics is restricted for now to detecting enum fields at fixed positions. Handling this problem in general is difficult.

Some fields that involve dependencies between packets, like sequence numbers, may be present in network protocols. But Analytics currently affords no support for these. Future work would include incorporating algorithms and heuristics for these.

Currently, Analytics is only a toolkit for researchers for manual analysis. Analytics has great potential to perform automated packet analysis for applications in deep packet inspection, and we plan to do this by implementing machine learning in future work.

7 CONCLUSION

In conclusion, I have presented Analytics, a library of functions which can aid manual analysis of protocol specifications. Given a set of packets, the tool attempts to discover constants, enumeration (enum) fields and strings among them, and provides visualization to aid analysts.

Understanding and deriving unknown protocol specifications has many useful security applications. Analytics can serve as a basis for deep packet inspection, to scrutinize the packets in network traffic in order to monitor and manage the network. The knowledge gained about protocol specifications is also useful in malware inspection and analysis; Analytics can help understand the unknown protocols that malware use to communicate with the main control server. Finally, Analytics can be used as a tool in software testing to check an implementation for conformance to a protocol specification.

REFERENCES

[1] Wireshark https://www.wireshark.org/

[2] The SNORT network intrusion detection system. https://www.snort.org/

[3] Z. Lin, X. Jiang, D. Xu, and X. Zhang. Automatic protocol format reverse engineering through context-aware monitored execution. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08),* San Diego, CA, February 2008.

[4] How Samba Was Written. https://www.samba.org/ftp/tridge/misc/french_cafe.txt

[5] The Protocol Informatics Project. http://www.4tphi.net/~awalters/PI/PI.html

[6] J. Caballero and D. Song. Polyglot: Automatic extraction of protocol format using dynamic binary analysis. In *Proceedings pf the 14th ACM Conference on Computer and Communications Security (CCS'07),* 2007.

[7] W. Cui, J. Kannan, and H. J. Wang. Discoverer: Automatic protocol reverse engineering from network traces. In *Proceedings of the 16th USENIX Security Symposium (Security'07),* Boston, MA, August 2007

[8] B. Guha, and B. Mukherjee. Network Security Via Reverse Engineering of TCP Code: Vulnerability Analysis and Proposed Solutions. In *Proceedings of the IEEE Infocom '96 Conference*, 1996

[9] J. Caballero, P. Poosankam, C. Kreibich, and D. Song. Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS'09),* pages 621-634, Chicago, Illinois, USA. 2009.

[10] J. Postel, *Transmission Control Protocol,* RFC 793, 1981

[11] Plotly https://plot.ly/

[12] D. Ueltschi Introduction to Statistical Mechanics www.ueltschi.org/teaching

[13] Contagio http://contagiodump.blogspot.sg/

[14] Z. Lin, X. Zhang, D. Xu Automatic Reverse Engineering of Data Structures from Binary Execution. In *Network and Distributed System Security Symposium (NDSS),* San Diego, CA, 2010.

[15] http://msdn.microsoft.com/en-us/library/ee441774.aspx

APPENDIX

How to use the constant function:

The user runs the `constant` function and specifies the number of packets in the given file to be checked, along with optional variables to read binary files, view the positions where constants were found in every packet, and to increase or decrease the maximum constant length (default 9).

How to use the enum function:

Optional variables in `statistics` includes the removal of some protocol headers that have been implemented into Analytics (such as IPv4, IPv6, UDP, TCP, and ICMP), the option to read binary files instead of packet capture files, the option to show the user all the different values that was found in each position with their probability of appearing and entropy value, and the starting and ending positions of the packet to be checked.

Strings and other functions:

###Analytics can check for and print ASCII strings that it has found in every packet. The algorithm for finding strings is to loop through the packet and detect if a byte in the packet is a printable ASCII character. If so, Analytics will check if the next few bytes are also printable ASCII characters. If this is true, Analytics will follow the text and search for the first non ASCII character, and check if it is the ASCII null character. If the ASCII null character is found, Analytics will accept the string and print it; otherwise the data is determined not to be a ASCII string and discarded.

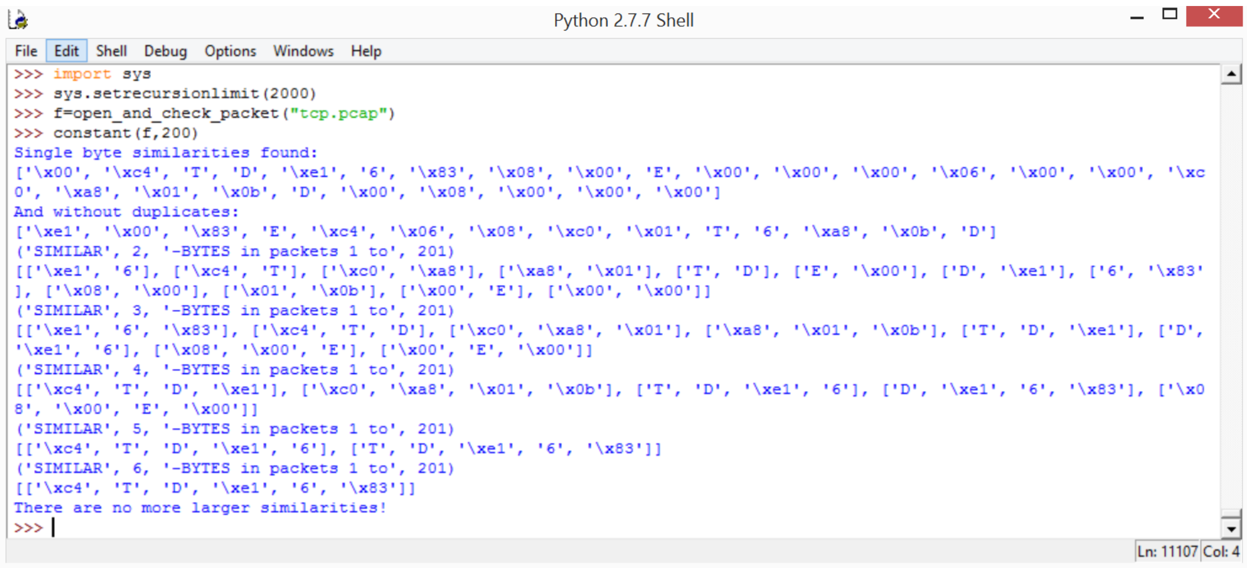
Additionally, Analytics can also read binary files and perform analyses on them, however due to memory limitations there is a cap on the number of bytes that are read in from each binary file. Finally, Analytics also has a function for the user to input commands and manipulate every byte of a given range of packets. ###

Why plot is important:

This is helpful to the analyst, for example, if a constant is always found at the same position, it suggests that the constant is very significant. If a constant is found around the same position but not always exactly at the same position, it may imply that there is a field of variable length in front of the constant. If a constant is found at different positions but at equal intervals from each other, it may also be significant to the user.

omes with a beautiful “constant” line marker and “enumerator” line marker, that marks if the position has one value only, or four values or less, respectively. This helps the user to identify constants and enumerator values at any position immediately. Analytics also identifies the Shannon Entropy level [12] of every position in the file and presents it as a striking line graph along the main graph. This information has been extremely useful to see patterns in a single protocol; random data such as checksums can be identified and filtered out easily, easing the process of qualitative analysis of protocol specifications. `statistics_plot` generates a graph of a given chunk of datasize, thus the size of enumerator, and also the name of the generated graph must be specified.

Constant plot analysis stuff:



```
>>> import sys
>>> sys.setrecursionlimit(2000)
>>> f=open_and_check_packet("tcp.pcap")
>>> constant(f,200)
Single byte similarities found:
['\x00', '\xc4', 'T', 'D', '\xe1', '6', '\x83', '\x08', '\x00', 'E', '\x00', '\x00', '\x00', '\x06', '\x00', '\x00', '\xc
0', '\xa8', '\x01', '\x0b', 'D', '\x00', '\x08', '\x00', '\x00', '\x00']
And without duplicates:
['\xe1', '\x00', '\x83', 'E', '\xc4', '\x06', '\x08', '\xc0', '\x01', 'T', '6', '\xa8', '\x0b', 'D']
('SIMILAR', 2, '-BYTES in packets 1 to', 201)
[['\xe1', '6'], ['\xc4', 'T'], ['\xc0', '\xa8'], ['\xa8', '\x01'], ['T', 'D'], ['E', '\x00'], ['D', '\xe1'], ['6', '\x83'
], ['\x08', '\x00'], ['\x01', '\x0b'], ['\x00', 'E'], ['\x00', '\x00']]
('SIMILAR', 3, '-BYTES in packets 1 to', 201)
[['\xe1', '6', '\x83'], ['\xc4', 'T', 'D'], ['\xc0', '\xa8', '\x01'], ['\xa8', '\x01', '\x0b'], ['T', 'D', '\xe1'], ['D',
'\xe1', '6'], ['\x08', '\x00', 'E'], ['\x00', 'E', '\x00']]
('SIMILAR', 4, '-BYTES in packets 1 to', 201)
[['\xc4', 'T', 'D', '\xe1'], ['\xc0', '\xa8', '\x01', '\x0b'], ['T', 'D', '\xe1', '6'], ['D', '\xe1', '6', '\x83'], ['\x0
8', '\x00', 'E', '\x00']]
('SIMILAR', 5, '-BYTES in packets 1 to', 201)
[['\xc4', 'T', 'D', '\xe1', '6'], ['T', 'D', '\xe1', '6', '\x83']]
('SIMILAR', 6, '-BYTES in packets 1 to', 201)
[['\xc4', 'T', 'D', '\xe1', '6', '\x83']]
There are no more larger similarities!
>>> |
```

Figure 4: The use of constant() to find constants in 200 TCP packets

The largest constant that was found throughout all 200 packets was ['\xc4', 'T', 'D', '\xe1', '6', '\x83'], a six byte chunk of data. The five-byte chunks of data appear to be parts of the above data. Two more distinct constants were found that were four bytes each: ['\xc0', '\xa8', '\x01', '\x0b'] and ['\x08', '\x00', 'E', '\x00']. The three-byte chunks of data are parts of the above data. The only new two-byte chunk of data that appeared was ['\x00', '\x00'], the rest were parts of the above three constants found. Therefore, four distinct constants with a size larger than one have been found in a TCP file: ['\xc4', 'T', 'D', '\xe1', '6', '\x83'], ['\xc0', '\xa8', '\x01', '\x0b'], ['\x08', '\x00', 'E', '\x00'] and ['\x00', '\x00']. Additionally, a distinct constant of size 1 was found, '\x06'.

Analysis of the above constants with `constant_plot` revealed several patterns about them. The graph of ['\x08', '\x00', 'E', '\x00'] (Figure 5) was the most significant, revealing that the constant was at a constant position 12, directly telling us that it is a true constant. ['\xc4', 'T', 'D', '\xe1', '6', '\x83'] was found at a ratio of 102:98 in two positions that were exactly 6 bytes away from each other; Position 0 and 6 of the file respectively. ['\xc0', '\xa8', '\x01', '\x0b'] had a similar pattern, at a ratio 102:98 in two positions 4 bytes away: 26 and 30, implying correlation.

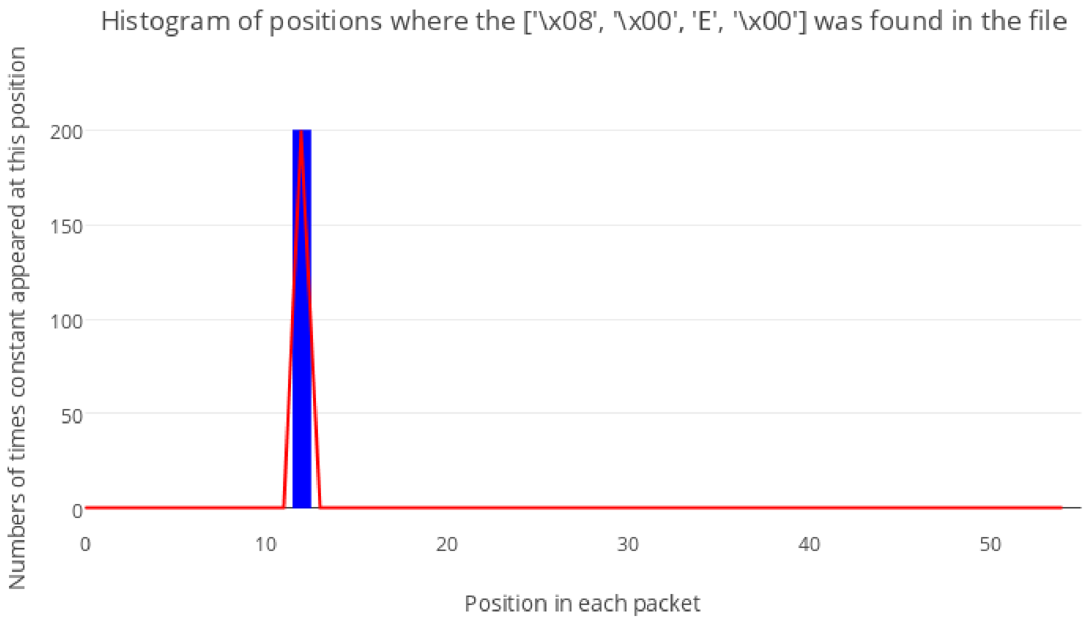


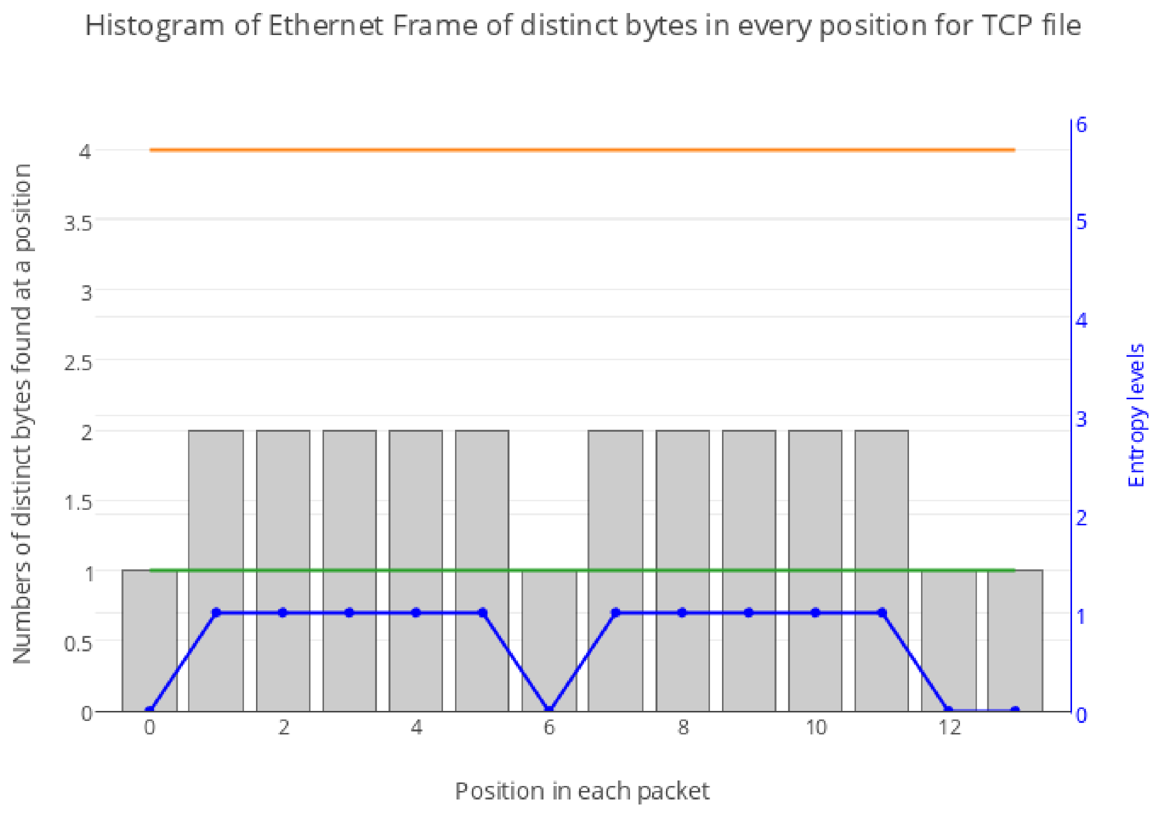
Figure 5: The constant ['\x08', '\x00', 'E', '\x00'] appears at only position 12 in all packets

The results for the data ['\x00', '\x00'] and '\x06' had more complicated patterns. ['\x00', '\x00'] was detected 200 times at position 52, thus it is a true constant. However, it was also detected 122 times at position 15, and 57 times for the positions 53-58. '\x06' was detected 200 times at position 23, thus it is a true constant; however, small clusters of 5 or less were found in other parts of the packet.

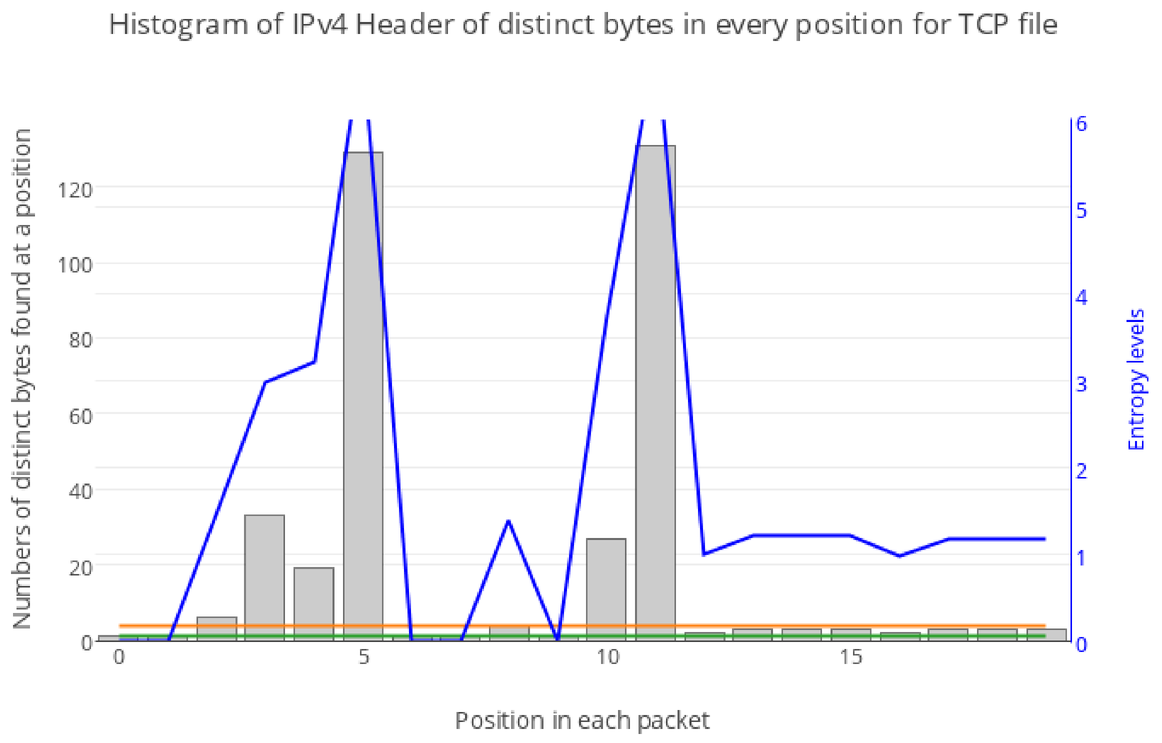
3.3 VERIFICATION OF ANALYSIS

Trend found in Figure 2	**Explanation**
There are two plateaus in entropy levels from the start of the graph to the 12th byte of the graph.	This is the pattern of the Ethernet header, which also contains the source address: ['\xc4', 'T', 'D', '\xe1', '6', '\x83']. There is low variation as the files used did not record many different network addresses, resulting in Analytics delivering false positive enum field.
The entropy level of the graph is zero from the 12th to 15th byte.	This position is for the constant ['\x08', '\x00', 'E', '\x00']. This is a very significant marker that indicates an IPv4 packet. (see Figure 1)
There is a sharp peak from the 16th to 19th byte	Part of the IPv4 header that indicates “Total Length”, followed by an “Identification” that results in a sharp spike at the 19th byte
There is a sudden drop to zero entropy level at positions 20 and 21	This is the “Flags” and “Fragment Offset” of the IPv4 header, which is a constant for certain files, this is shown in Figure 1
followed by an enumerator value at position 22	This is “Time to Live” in the IPv4 header, which does not experience much variation
another constant at position 23.	This is “Protocol” in the IPv4 header, which is a constant of '\x06' that marks that the following protocol will be TCP.
Another peak was seen at positions 24 and 25,	This is “Header Checksum” in the IPv4 header which will definitely experience high variation
The graph then drops low and entropy fluctuates around a value of 1 from positions 26 to 37	Positions 26 to 33 indicate the Source IP and Destination IP of the IPv4 header, which also includes the constant ['\xc0', '\xa8', '\x01', '\x0b']. Positions 34-37 are part of the TCP header, which also denote Source and Destination ports. There is low variation because the files used were not random enough, resulting in Analytics delivering a false positive as an enum field.
Finally, the graph increases to several peaks beginning from position 38, with three peaks occurring at 40-41	This is part of the “Sequence Number” for the TCP header and has high variations

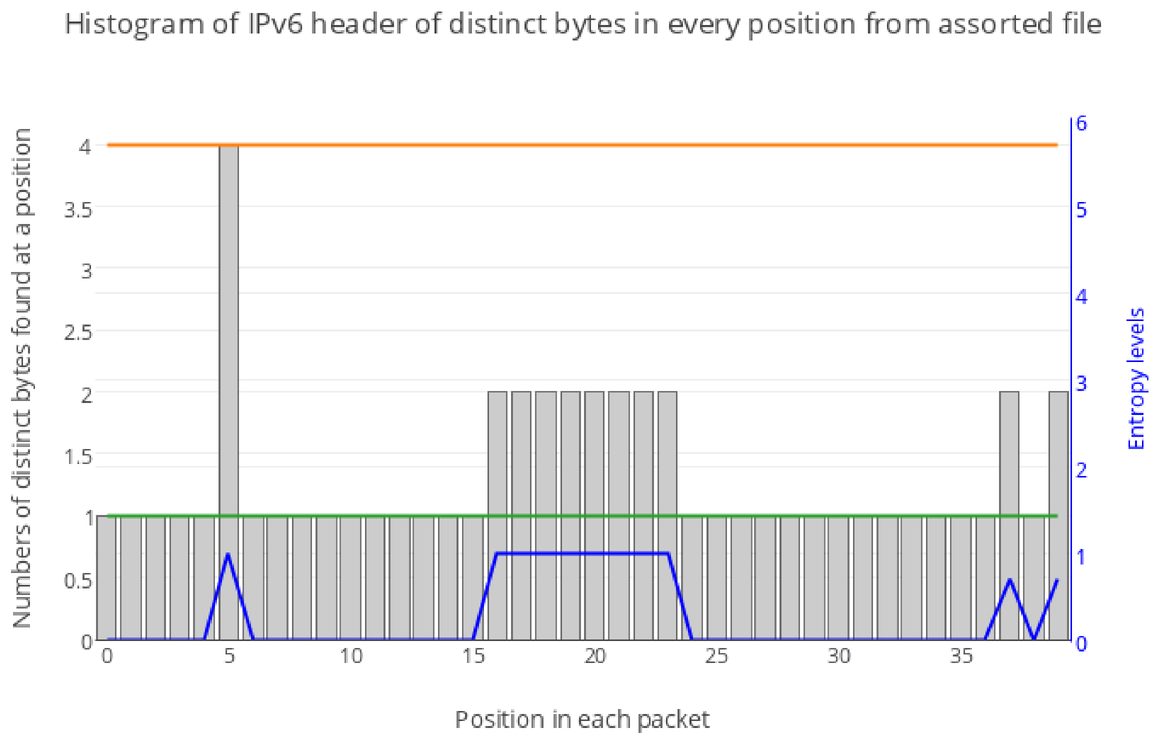
Peak at 44-45	This is part of the “Acknowledgement Number” for the TCP header, which has high variations.
Constants are found at positions 46	This byte is part of the “Header Length” and “Reserved” portion of the TCP header, and although it is a constant in Figure 2, it is not a true constant for all TCP protocols.
Peak at 50-51.	This is the “Checksum” area of the TCP header, which will experience high variation
Constants found at 52 and 53	This is the “Urgent Pointer” of the TCP header which is a constant of ['\x00', '\x00']; usually because there is nothing “Urgent” to report.



Appendix 1: Pattern of the Ethernet header from a TCP file. Appears to be two plateaus

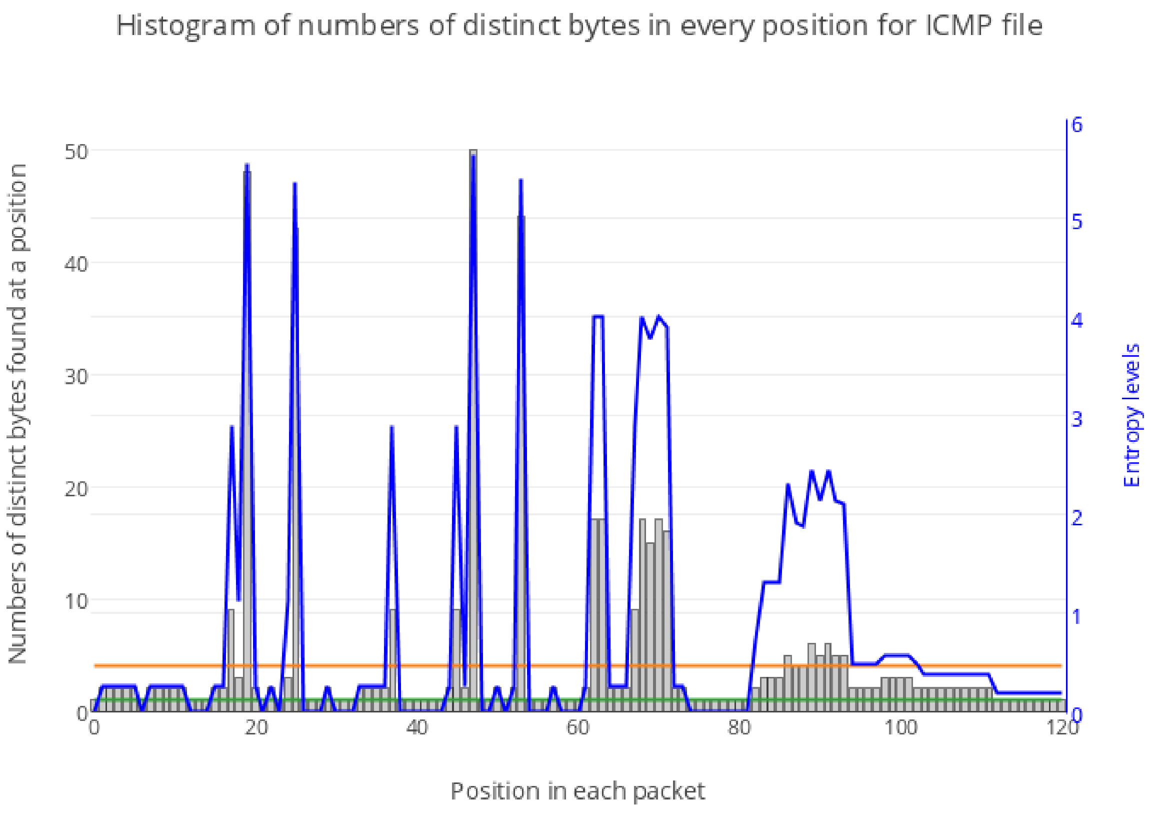


Appendix 2: Pattern of IPv4 header from only TCP files. "Protocol" (position 9) is constant

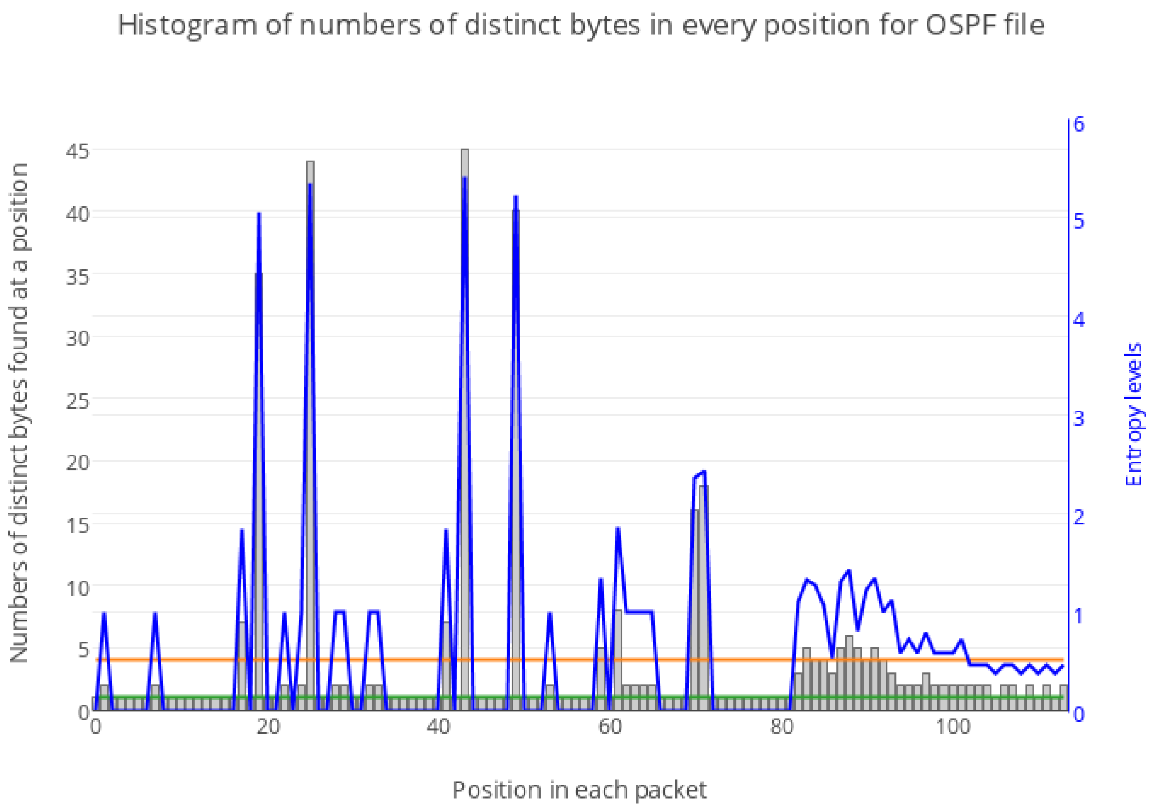


Appendix 4: Pattern of IPv6 header from 130 assorted SSDP, LLMNR and mDNS packets

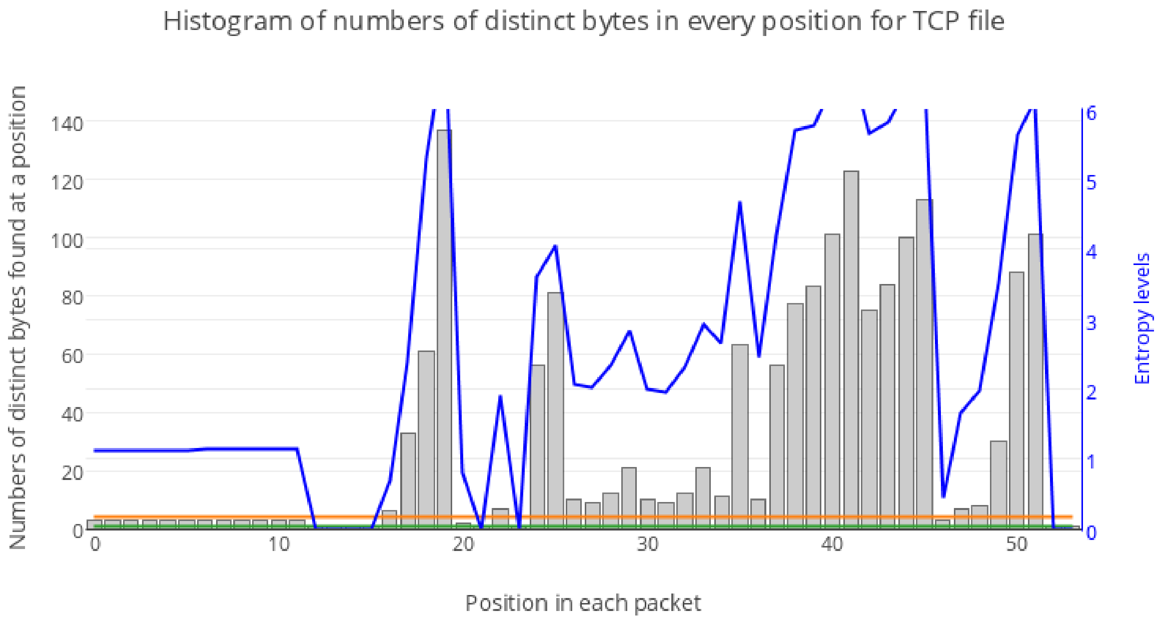
<TCP strip>



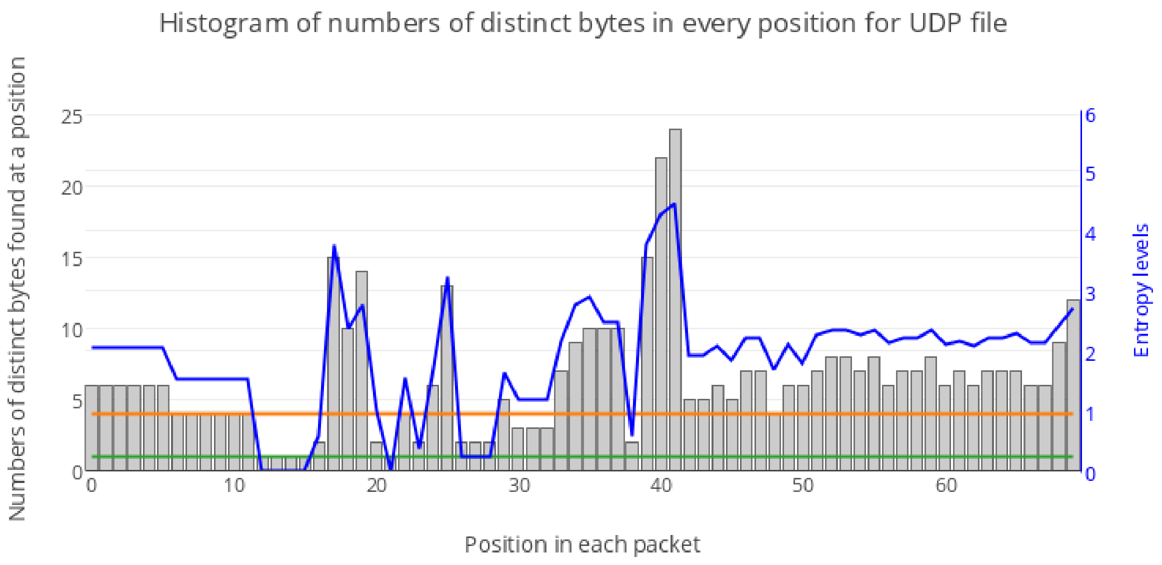
Appendix 6: Graph of 50 ICMP packets. Ethernet and IPv4 header can be clearly seen.



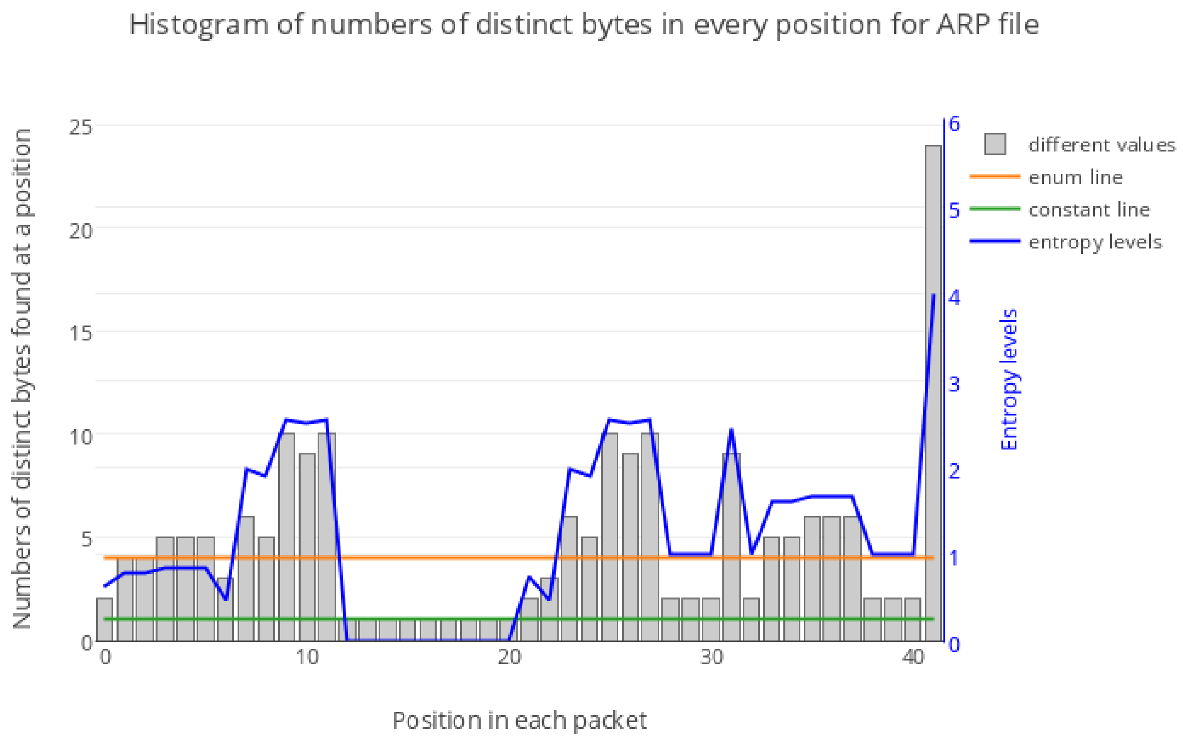
Appendix 7: Graph of 55 OSPF packets. IPv4 header can be clearly seen.



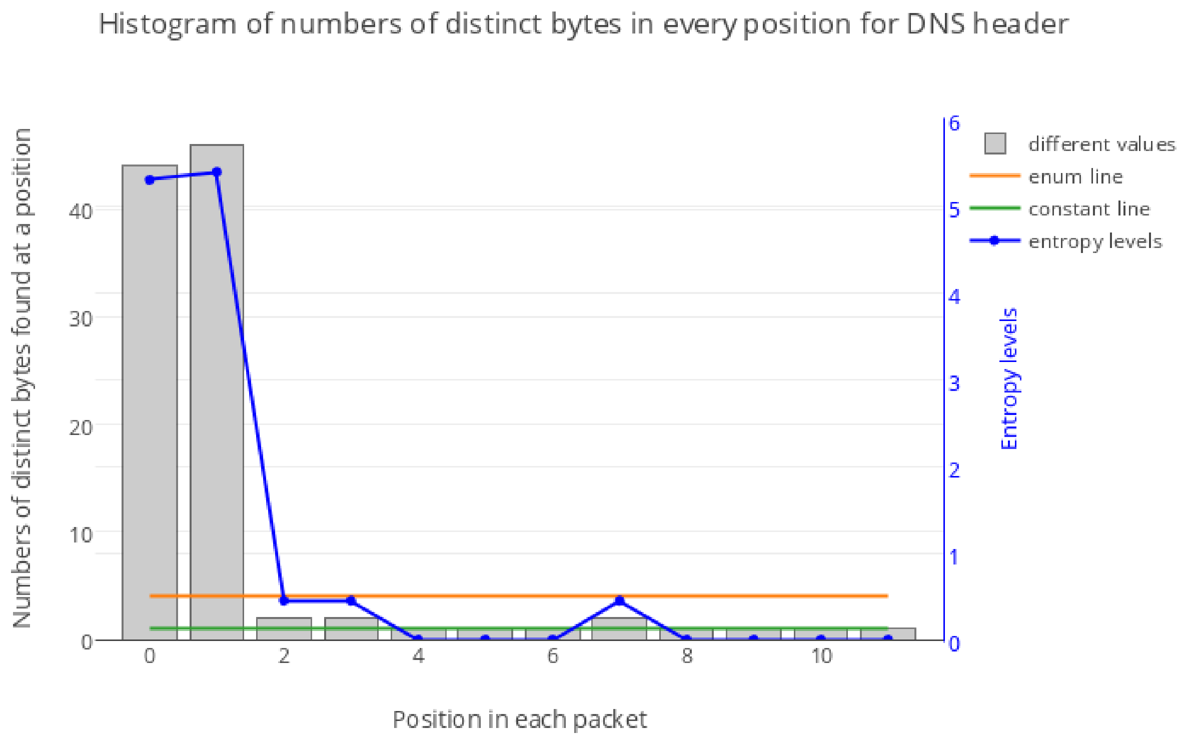
Appendix 8: More random TCP file. Ethernet appears as single plateau, IPv4 can be seen



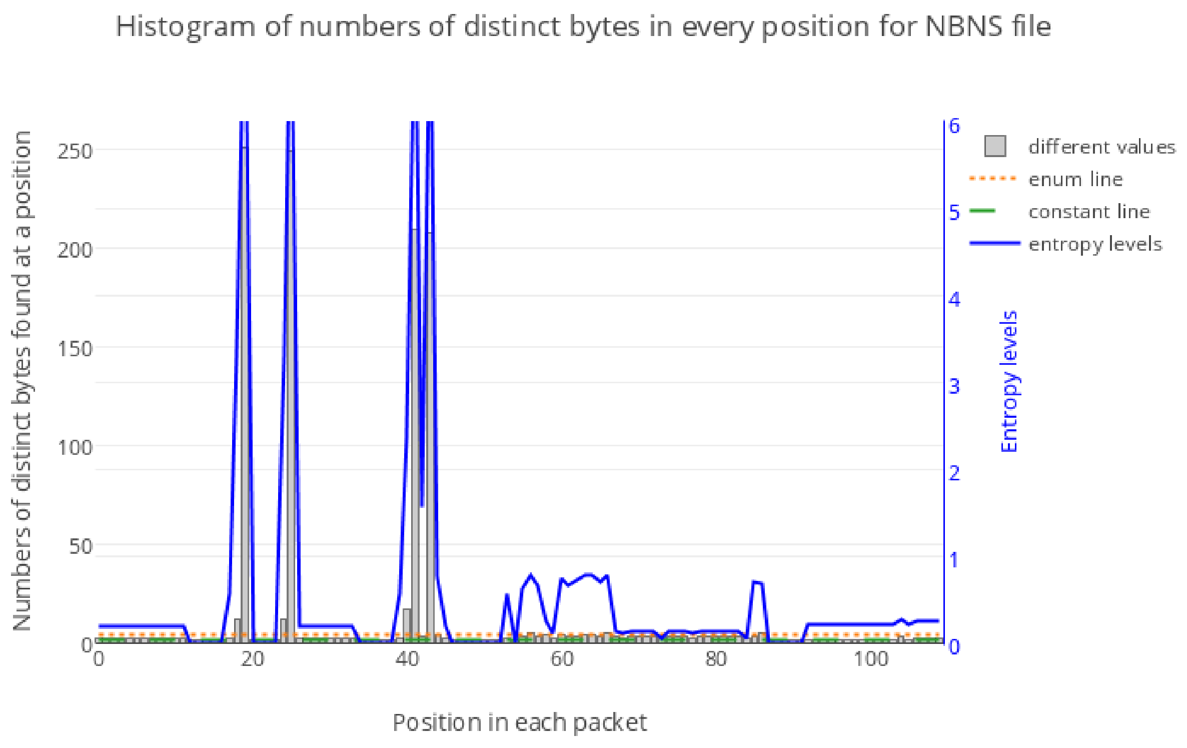
Appendix 9: 30 assorted ICMP, DNS, SSDP and UDP packets, appears to use IPv4



Appendix 10: 274 ARP packets in a merged file, the long string of constants is notable.



Appendix 11: 150 DNS packets with the DNS header filtered out



Appendix 12: 700 NBNS packets total in a merged file, discovered similarities to DNS