NETWORK TRAFFIC CLUSTERING AND GEOGRAPHIC VISUALIZATION

A Thesis

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Ali Hushyar

August 2009

© 2009

Ali Hushyar

ALL RIGHTS RESERVED

SAN JOSÉ STATE UNIVERSITY

The Undersigned Thesis Committee Approves the Thesis Titled

NETWORK TRAFFIC CLUSTERING AND GEOGRAPHIC VISUALIZATION

by
Ali Hushyar

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

_____

Dr. Mark Stamp, Department of Computer Science Date

_____

Dr. Soon Tee Teoh, Department of Computer Science Date

_____

Robert Hermes, ATT Inc. Date

APPROVED FOR THE UNIVERSITY

_____

Associate Dean Office of Graduate Studies and Research Date

ABSTRACT

NETWORK TRAFFIC CLUSTERING AND GEOGRAPHIC VISUALIZATION

by Ali Hushyar

The exploration and analysis of large databases of information are an ever-challenging task as digital data acquisition continues to progress. The discipline of data mining has often been employed to extract structure and patterns from the underlying dataset. In addition, new research in the field of information visualization is being applied to the same challenge. Visual models engage the invaluable pattern processing abilities of the human brain which leads to new areas of insight otherwise undetected. This research applies the benefits of both data mining and information visualization to the specific problem of traffic analysis on computer networks. This is an important issue as it relates to the ability to understand diverse behavior on the network and provide many fundamental services. For example, distinct traffic classifications and associated traffic volumes facilitate capacity-planning initiatives. Furthermore, accurate categorization of network traffic can be leveraged by quality of service offerings and, at the same time, lend itself to efficient security analysis. In this research, an example of a data processing pipeline is described that incorporates both data mining and visualization techniques to cluster network flows and project the traffic records on a geographic display.

Table of Contents

List of Tables

List of Figures

1 Introduction

The fast and efficient categorization of network communication can help analysts isolate classes of traffic on the network for further investigation. One traditional traffic classification technique is a fixed identification method based on transport-layer fields such as well-known port and protocol numbers. The problem here is that many processes such as Peer-to-Peer (P2P) applications use dynamically negotiated port numbers. Others may not run on well-known port numbers at all such as the practice of running HTTP proxies and rogue mail-relay servers on higher-order ports. More importantly, there are often nested categories of traffic within a traffic class. In [1], the authors describe subclasses of traffic within HTTP depending on the underlying purpose, for example transactional communication, bulk data transfers and tunneled applications. Furthermore, various traffic streams utilizing different port numbers can exhibit the same type of behavior and in fact be operating in the same capacity such as a large file transfers using HTTP or FTP.

Other methods include signature-based classification techniques applied to the packet payload. More and more network applications are rendering this method difficult if not obsolete. As explained in [2], traffic on the wire can be encrypted and employ other masquerading and obfuscation techniques such as fragmentation to make fingerprinting difficult. In addition, signature-based detection methods can only identify those traffic types for which there are signatures available.

To get around these obstacles, one proposal is to characterize network traffic based on features of the transport-layer statistics irrespective of port-based identification or payload content. The idea here is that different applications on the network will exhibit different patterns of behavior which is manifested in the transport-layer statistics. For example, distinguishing between file transfers generated by P2P and FTP processes can be done because P2P connection behavior is typically long-lived and data is bidirectional while FTP is relatively short-lived and data transfers are unidirectional. There have been many clustering algorithms applied to various domains [3]. However, this paper will discuss a relatively new unsupervised clustering technique which at the time of this writing is not known to have been used to address network traffic analysis. This algorithm will be applied to recorded traffic statistics to provide class structure to the data.

To enhance the analysis process, multi-dimensional data sets will be visually projected to expose hidden areas of interest and offer further insight. Many visualization techniques have been developed from 1-Dimensional to N-Dimensional models, and graph and tree based views. In [4], the authors offer a succinct history of information visualization. This paper will take an in-depth look into an existing 2D Treemap model based on an innovative layout algorithm. Treemaps are visualization structures that allow for the projection of large amounts of hierarchical information on a traditional 2D display. This particular Treemap is a projection tool that provides application-specific context to the mined dataset. It is essentially a geographic model of the Internet where IP addresses are organized according to their administrative domain. To illustrate the power

of visualization, IP addresses that engage in communication on the network are visually encoded based on a measure of interest such as number of outgoing connections or number of bytes transferred. This visual encoding is presented on a geographic display that highlights scope and evolution of behavior over time. This paper shows that the combination of clustering and visualization reveals a value-added process of analysis that provides additional insight into the underlying data. It was revealed through personal correspondence with the developers of this geographic Treemap that this tool has not been fully developed nor is it publicly available. As a result, part of this research includes the implementation of the visualization based on the description of the Treemap algorithm.

The rest of this paper is organized into 4 sections. Section 2 and 3 describe the specific statistics used in the characterization of network traffic and explain how the statistics where extracted from packet header traces. Section 4 describes the reasoning behind choosing a new clustering method and the results of the clustering on a sample dataset. Section 5 introduces the geographic Treemap visualization tool, its implementation, and how it is used to project clustered datasets and glean additional knowledge. Screenshots of the Treemap implementation are provided for reference.

2 Determining Dimensions of Network Traffic

To begin, a flow is defined as a channel of communication between two parties on a network. For the purposes of this paper, a flow will be further qualified as a bi-directional communication channel between two endpoints. Flows are uniquely identified by a 5-tuple of fields taken from the TCP/IP headers of the packets. The 5-tuple and its constituent fields are:

(source-address, source-port, destination-address, destination-port, protocol).

In addition to these flow fields, other flow features or dimensions are used to characterize the behavior of a flow. These dimensions are extracted or derived from packet traces. Packet traces usually contain some or all of the packet content for each flow on the wire from which it was collected. Each flow in the trace has been generated by a network process such as a web client or mail client that has engaged a server for a specific transaction. These dimensions are measurements that can describe an aspect of the flow in its entirety or an attribute of the flow in a particular direction. In [5], the authors have identified 249 distinct flow dimensions. These dimensions cover basic statistics such as packet lengths and TCP flag counts to more involved measurements such as variance of packet sizes and flow "modes", the time a flow spends in idle, interactive and bulk transmission states. The objective of characterizing network flows is to find a set of dimensions that are most effective in differentiating and classifying

different types of flows. In this paper, successful classification does not imply a fixed taxonomy but rather a subjective grouping relative to other flows in the dataset.

With so many possible dimensions of traffic, it is expected that some dimensions would contribute more than others to the successful classification of a flow and it is possible that some dimensions actually hinder the classification process. As a result, a great deal of research has been conducted to identify a core set of dimensions that are most effective in classifying network traffic. This is even more significant in that many data mining programs including clustering algorithms scale poorly with high-dimensional data sets. In [6], 19 dimensions of traffic were identified and the authors used the supervised machine-learning Support Vector Machine (SVM) model to classify network traffic into known classes. Table 2.1 shows a high-level taxonomy of traffic classes and examples of applications that would be categorized into those classes.

TABLE 2.1 Traffic Classes and Applications

Class	Representative Applications
BULK	ftp
INTERACTIVE	ssh, telnet, rlogin
MAIL	pop3, smtp, imap
SERVICE	X11, dns
WWW	http, https
P2P	Kazaa,Bittorrent,Gnutella
MULTIMEDIA	voice, video streaming
GAME	Half-life
ATTACK	worm, virus
OTHERS	scan, netbios, ntp, tsp

One caveat in the experiments described in [6] is that the number of available flows in the categories MULTIMEDIA, GAME and ATTACK were two few to properly classify and thus were omitted from the experiments and evaluations. One assumption of this paper is that this set of dimensions will suffice since it has accurately identified flows belonging to the other 7 classes using methods described by other researchers. Adapted from [6], Table 2.2 shows the dimensions used to classify the data into their respective traffic classes.

TABLE 2.2 19 Dimensions of a Flow

0	Total-num-pkt	Total number of packets in the flow
1	Ave-pkt-len	The average packets size of a flow
2	Pkts-sent	The number of packets sent for the flow
3	Send-avelen	The average send packets size of a flow
4	Send-var	The variance of send packets' size
5	Recv-avelen	The average receive packets size of a flow
6	Recv-var	The variance of receive packets' size
7	Var-recv-size	The variance of received packets' size
8	Duration	The duration of the flow
9	Protocol	The protocol (TCP or UDP)
10	Send-port	The source port of a flow
11	Recv-port	The destination port of a flow
12	Pkts-ratio	The number ratio of send and receive packets
13	Byte-ratio	The byte ratio of send and receive packets
14	Num-SYN	The number of SYN packets
15	Num-RST	The number of RST packets (rst)
16	Num-FIN	The number of FIN packets (fin)
17	Window-size	The average window size (window_size)
18	Window-var	The variance of window size

The authors of [6] further applied a "discriminator selection method" based on a modified sequential forward selection algorithm to identify an even more optimized set of dimensions that improved the classification accuracy of their SVM approach. The final sequence consists of 9 dimensions in decreasing order of significance and is listed by

their index values as {11, 13, 4, 8, 14, 7, 9, 1, 6}. The selection method determined that this set of dimensions yielded the highest rate of accuracy at 96.92%.

In this paper, additional modifications were made to this dimension set to better fit the requirements and objectives of this thesis. Dimension 11 (destination port) was omitted to avoid identification influenced by port-based methods and dimension 9 (protocol) was removed from consideration since the scope of this paper is restricted to TCP-based flows. Finally, the assumption is made that both dimensions 6 (receive packet size variance) and 7 (received packet size variance) refer to the same measurement in that no distinction is made between the set of packets sent by the destination host and the set of packets that were actually received by the source host. The final set consists of 6 dimensions {13, 4, 8, 14, 1, 6}. Although a decrease in accuracy is expected from removing the dimensions discussed above, and although further study into the extent of this accuracy degradation is not within the scope of this paper, it will be shown that this dimension set is sufficient to aid in the general categorization and exploration of network traffic.

3 Extracting Dimensions from Packet Traces

The core set of dimensions discussed in the previous section can be readily extracted and/or derived from packet traces such as those stored in the standard PCAP binary format. The decision was made that a laboratory-controlled set of simulated traffic would be generated rather than using publicly available packet traces. This was done to verify the results of this paper and establish proof-of-concept. Streams of flows were generated using the Ixia IxLoad load-testing appliance [7]. This hardware was used to generate Layer7 flows for various application protocols through a Layer2 switch. The TCPDUMP utility [8] was run on a packet capture server that sniffed all the traffic that entered the switch. Flows were generated for HTTP, FTP, SMTP and TELNET. This mixture of traffic offers a variety of network behavior from quick transactional flows to monolithic bulk file transfers, from randomized messaging communication to low-bandwidth persistent transactional sessions.

After the flows were captured in a binary packet capture format, the TCPTRACE utility, as described in [9], was used to generate flow-level statistics for each session in the packet capture. An example of the TCPTRACE output for a single flow is given in Table 3.1. The red-highlighted statistics are organized into two columns, one for each direction of the flow. This output was used to calculate the 6 key dimensions.

TABLE 3.1 Sample TCPTRACE Output

```
TCP connection 1:
      host a:        10.1.1.2:28366
      host b:        10.1.2.2:80
      complete conn: yes
      first packet:  Wed Jun 17 16:42:10.396696 2009
      last packet:   Wed Jun 17 16:42:11.412719 2009
      elapsed time:  0:00:01.016023
      total packets: 44
      filename:      ixiacap
   a->b:                                b->a:
     total packets:            18         total packets:            26
     ack pkts sent:            17         ack pkts sent:            26
     pure acks sent:           15         pure acks sent:            1
     sack pkts sent:            0         sack pkts sent:            0
     dsack pkts sent:           0         dsack pkts sent:           0
     max sack blks/ack:         0         max sack blks/ack:         0
     unique bytes sent:       194         unique bytes sent:     32939
     actual data pkts:          1         actual data pkts:         23
     actual data bytes:       194         actual data bytes:     32939
     rexmt data pkts:           0         rexmt data pkts:           0
     rexmt data bytes:          0         rexmt data bytes:          0
     zwnd probe pkts:           0         zwnd probe pkts:           0
     zwnd probe bytes:          0         zwnd probe bytes:          0
     outoforder pkts:           0         outoforder pkts:           0
     pushed data pkts:          1         pushed data pkts:         12
     SYN/FIN pkts sent:       1/1         SYN/FIN pkts sent:       1/1
     req 1323 ws/ts:          N/Y         req 1323 ws/ts:          N/Y
     req sack:                  Y         req sack:                  Y
     sacks sent:                0         sacks sent:                0
     urgent data pkts:          0 pkts    urgent data pkts:          0 pkts
     urgent data bytes:         0 bytes   urgent data bytes:         0 bytes
     mss requested:          1460 bytes   mss requested:          1460 bytes
     max segm size:           194 bytes   max segm size:          1448 bytes
     min segm size:           194 bytes   min segm size:          1083 bytes
     avg segm size:           193 bytes   avg segm size:          1432 bytes
     max win adv:            2896 bytes   max win adv:            2896 bytes
     min win adv:            1448 bytes   min win adv:            2701 bytes
     zero win adv:              0 times   zero win adv:              0 times
     avg win adv:            2815 bytes   avg win adv:            2709 bytes
     initial window:          194 bytes   initial window:         2896 bytes
     initial window:            1 pkts    initial window:            2 pkts
     ttl stream length:       194 bytes   ttl stream length:     32939 bytes
     missed data:               0 bytes   missed data:               0 bytes
     truncated data:          192 bytes   truncated data:        32893 bytes
     truncated packets:         1 pkts    truncated packets:        23 pkts
     data xmit time:        0.000 secs    data xmit time:        0.009 secs
     idletime max:         1007.1 ms      idletime max:          1006.5 ms
     throughput:             191 Bps      throughput:            32420 Bps
```

To explain how the dimensions were calculated, references will be made to the relevant statistics in Table 3.1. Dimension 13 (byte ratio of send and receive packets) was calculated from the "actual data bytes" count for both directions of the flow. Dimension 4 (send packet size variance) is the statistical variance of the packet sizes sent by the source. An approximation of this variance was calculated from three reference packet sizes, the "max segm size", "min segm size" and "avg segm size" counts in the first column. Dimension 8 (duration of flow) was calculated from the "elapsed time" statistic which was converted to milliseconds for all flows. Dimension 14 (number of SYN packets) is the total number of SYN packets seen in the flow in both directions. This was extracted from the "SIN/FIN pkts sent" count in both directions. Dimension 1 (average packet size of flow) was calculated as a weighted mean by taking the product of the "total packets" and "avg segm size" statistics in each direction, adding those values, and then dividing by two. Finally, Dimension 6 (receive packet size variance) is the statistical variance of the packet sizes sent by the destination. This calculation was the same as Dimension 4 except that the operands are the segment size numbers from the destination to the source found in the second column. All dimensions were min-max normalized for each flow by dividing the dimension value by the range of values recorded for that dimension in the data set. This normalization was done so that no single dimension could over-influence the clustering. These normalized flow records were used as input to the clustering algorithm described in the next section.

4 Clustering Algorithms

Clustering has been applied to many problems in data mining. Given a set of objects, the objective of a clustering algorithm is to identify subsets or clusters of similar objects. Finding the optimal number of clusters for a set of data points is considered an NP-Hard problem. One genre of clustering techniques is known as Partitional clustering and the following algorithms are taken from this category.

4.1 K-means and K-medoids

One of the most popular Partitional approximation algorithms for the clustering problem is known as K-means [10]. Given a dataset *X* and a predetermined number of clusters, *k*, each represented by a cluster "mean" or "center" point m_i, K-means refines the set of cluster groups and associates each data point to one of those clusters. It does this by defining an objective function which includes a pair-wise similarity function such as n-space Euclidean distance *Ø()*. K-means optimizes the objective function between every data point *x* to its associated cluster center (1).

$$\sum_{i=0}^{k} \sum_{x \in \mathcal{X}} \phi(x, m_i) \qquad \textbf{(1)}$$

K-means is an iterative algorithm that first reassigns the data points to the closest center and then recalculates the cluster centers for each round. The new center of a cluster is the mean value of the data points associated with that specific center. Alternatively, a data point itself can be used as the center by simply using the data point closest to the

calculated mean center. The idea is that K-means eventually converges to a locally optimal solution.

K-medoids [11] is closely related to K-means in that it is also a Partitional algorithm that creates clusters by optimizing an objective function. However, K-medoids selects cluster center points that are actual data points, also known as "exemplars". Finding representative data points as centers is often called prototyping. The most significant difference between K-means and K-medoids is that after each round of the algorithm, K-medoids compares the cost of swapping a cluster center point with a candidate data point to determine a possibly new center point rather than taking the average of the data points as the new center. In [12], the authors show that in the problem domain of document clustering, K-medoids is more accurate than K-means in determining document clusters.

One of the major downsides of both K-means and K-medoids is that the arbitrary initialization of cluster centers can lead to poor clustering. Also, both techniques require the user to provide the number of clusters to generate a priori and there is a potential that unbalanced cluster sizes are formed due to the initial selection of cluster centers. As a result, this research was focused on finding a clustering method that has the simplicity and efficiency of K-means, the accuracy of K-medoids, and avoids the shortcomings mentioned above.

4.2 Affinity Propagation

This leads to a relatively new unsupervised clustering algorithm called Affinity Propagation (AP) developed by researchers at the University of Toronto [13]. Instead of considering a single candidate exemplar at a time, characteristic of classic K-mediods, Affinity Propagation simultaneously considers all data points as possible exemplars by sending messages between data points to determine the degree of exemplar suitability. The input parameters to AP is a data structure of similarity measures between every two data points in the data set. The similarity value between any two points is given by *s(i,k)* which measures how well-suited data point *k* is to be the exemplar for data point *i*. Similarity calculations for this algorithm are based on negative Euclidean distance. AP does not require the user to provide the number of target clusters beforehand, instead the algorithm is configured with "preferences" where each data point is given a self-similarity value *s(k,k).* This value is a weight that influences the algorithm as to whether or not the data point becomes an exemplar at all. Bigger preference values will result in a higher number of clusters while smaller values will yield a smaller number. The assignment of preference values is a powerful "knob" in the calibration of this algorithm. According to AP developers, the recommendation is to use a range of values from the minimum, the median, to the maximum similarity measure as the preference value depending on the desired number of clusters.

In the AP algorithm, each data point or "node" will take on the perspective of two node types simultaneously, either a simple data point looking for its exemplar or a potential exemplar looking to find its data points. Given the input similarities and

preferences to the algorithm, two message types are exchanged between the data points. The first message is sent when a node is looking for its exemplar. It is called the "responsibility" message which is sent from data point *i* to data point *k* (2).

$$r(i,k) \leftarrow s(i,k) - \max_{k' \text{ s.t. } k' \neq k} \{a(i,k') + s(i,k')\} \qquad (2)$$

Equation 2 expresses how suitable data point *k* is to be the exemplar for data point *i* with respect to other potential candidate exemplars. Responsibility considers the similarity between the two points in the first term and subtracts the maximum similarity that point *i* may have with any other data point *k'* in the second term. *a(i,k)* is the second message type and will be described shortly. Upon initialization though, *a(i,k) = 0* for all data points and is not part of the first round of "responsibility" calculations. If *i=k*, then *r(k,k)* expresses a "self responsibility" which measures the value of *k* as an exemplar itself. This value is calculated from *k*'s preference, *s(k,k),* which is an input parameter to the algorithm minus the second term which expresses whether *k* is better suited to remain a data point assigned to a different exemplar.

The second message is sent when a potential exemplar node is looking for its data points. It is called the "availability" message which is sent from a data point *k* to data point *i* (3).

$$a(i,k) \leftarrow \min\left\{0, r(k,k) + \sum_{i' \text{ s.t. } i' \notin \{i,k\}} \max\{0, r(i',k)\}\right\} \qquad (3)$$

Equation 3 expresses how suitable data point *k* is to be an exemplar for data point *i* with respect to how other data points view it as an exemplar. Availability considers the "self

responsibility" of the data point *k* in the first term and adds any positive feedback from the other data points as to *k*'s suitability as an exemplar in the second term. If *i*=*k*, then *a(k,k)* expresses a "self availability" (4).

$$a(k,k) \leftarrow \sum_{i' \text{ s.t. } i' \neq k} \max\{0, r(i',k)\} \qquad (4)$$

Equation 4 measures how effective data point *k* will be as an exemplar given the positive feedback from other data points.

At each iteration of the algorithm, AP "sends" these two messages between each pair of nodes and combines the values of *a(i,k) + r(i,k)* for each node *i*. This process ultimately converges to a set of suitable exemplars by specifying a certain number of maximum iterations or halting the process when the sum of the calculation above does not change for a configurable number of rounds. At this point, for each *i*, the index of *k* that corresponds to the maximum value of *a(i,k) + r(i,k)* is either an exemplar if *k*=*i*, or otherwise identifies an exemplar *k* for a data point *i*.

4.3 Affinity Propagation Implementation

Source code for Affinity Propagation is not provided by the developers though they do offer MATLAB functions that implement the algorithm along with Windows/Linux C binaries [14]. They also provide an online Web interface to the algorithm where users can upload data to be processed by AP. For this thesis, the decision was made to integrate both the data mining aspect with the visualization component and choose a single implementation platform. As a result, the AP algorithm was coded using Java and the logic was applied to the flow records previously described in Section 3.

The constructor to Class *AffinityPropagation* takes the parameters shown in Table 4.1. It also shows the main routine called *iterate()* which first updates responsibilities, then updates availabilities, and finally combines both responsibilities and availabilities. The main FOR loop continues this process until either the maximum number of iterations has been reached or the stop criteria, which determines change between the values at each round, is met. Independent verification of the correctness of the AP Java implementation was attained by comparing the results of the program with the online AP interface using the example data sets provided.

TABLE 4.1 Affinity Propagation Code Summary

```
50    /*
51     * @param numPoints: initialize two-dimensional array
52     * @param numCandidates: initialize two-dimensional array
53     * @sim: two-dimensional matrix of similarity measures
54     * @maxRounds: maximum number of iterations
55     * @stopCriterion: number of consecutive rounds with no change in cluster assignment
56     * @dampingFactor: number in range (0,1) to prevent oscillation of message values
57     */
58    AffinityPropagation(int numPoints, int numCandidates, BigDecimal[][] sim,
59                        int maxRounds, int stopCriterion, double dampingFactor) {


92    public void iterate() {
93
94        System.out.println("Starting AP....");
95
96        for(int round = 1; ((round <= this.maxRounds) && (this.stopCount <= this.stopCriterion)); round++) {
97
98            calculateResponsibilities();
99            System.out.println("\nFinished calculating Responsibilities for round " + round);
100
101           calculateAvailabilities();
102           System.out.println("Finished calculating Availabilities for round " + round);
103
104           combineCalculations();
105           System.out.println("Finished combining calculations for round " + round);
106
107           this.totalIterations++;
108       }
109
110       resolveReflections();
111       createReport();
112   }
```

4.4 Clustering Network Flow Records Using Affinity Propagation

To show that AP properly clusters network flows, the algorithm was presented with a set of 200 normalized records each consisting of the 6 key flow dimensions. These records reflect a mixture of application protocols including HTTP, SMTP, FTP and TELNET. The output of the algorithm is a summary section describing the algorithm findings and a complete breakdown of cluster assignments. The latter is an array of size

200 with the value at each cell indicating the cluster to which the flow with the corresponding array index belongs to. In Table 4.2, the output has been arranged in a spreadsheet and color-coded to show the cluster assignments. The first number in the cell is the flow record number and the second number is the cluster center which corresponds to another flow record. This second number represents the data point that has been identified as the best prototype/center for that cluster.

TABLE 4.2 AP Results for 200 Record Dataset

Number of identified clusters: 5
Net similarity: -12.021999
Sum of point-to-exemplar similarity: -4.071999
Sum of exemplar preferences: -7.950000
Number of iterations: 51

1|58	21|58	41|58	61|58	81|133	101|133	121|58	141|58	161|58	181|174
2|58	22|58	42|58	62|58	82|174	102|133	122|58	142|133	162|58	182|152
3|58	23|58	43|58	63|58	83|133	103|133	123|58	143|174	163|58	183|174
4|58	24|58	44|133	64|58	84|133	104|133	124|58	144|144	164|58	184|144
5|58	25|133	45|133	65|58	85|133	105|174	125|58	145|133	165|58	185|174
6|58	26|58	46|133	66|58	86|133	106|133	126|58	146|174	166|58	186|174
7|58	27|58	47|58	67|58	87|133	107|58	127|58	147|174	167|133	187|133
8|58	28|58	48|58	68|58	88|58	108|58	128|58	148|133	168|58	188|133
9|58	29|133	49|58	69|58	89|58	109|58	129|58	149|133	169|133	189|133
10|58	30|133	50|133	70|133	90|58	110|174	130|58	150|174	170|133	190|133
11|58	31|133	51|133	71|133	91|58	111|133	131|133	151|174	171|133	191|174
12|58	32|58	52|133	72|133	92|58	112|133	132|133	152|152	172|133	192|144
13|58	33|58	53|133	73|133	93|58	113|174	133|133	153|133	173|133	193|133
14|58	34|58	54|133	74|133	94|58	114|133	134|133	154|133	174|174	194|58
15|58	35|58	55|133	75|58	95|58	115|133	135|133	155|133	175|144	195|58
16|58	36|58	56|133	76|58	96|58	116|133	136|174	156|58	176|58	196|58
17|58	37|58	57|133	77|58	97|58	117|133	137|174	157|58	177|58	197|58
18|58	38|58	58|58	78|174	98|58	118|133	138|133	158|58	178|58	198|58
19|58	39|58	59|58	79|174	99|58	119|58	139|58	159|58	179|133	199|58
20|58	40|58	60|58	80|133	100|133	120|58	140|58	160|58	180|174	200|58

Conn#	Source	Destination	ByteRatio	SendPktVar	Duration	SYN	RecvPktVar	AvgPktSize
58	10.1.1.6:28423	10.1.2.8:80	5.62E-09	1.32E-06	0.013307711	1	0.087865786	1.007132921
133	10.1.1.3:28498	10.1.2.11:25	4.42E-05	0.886710421	0.396945857	1	6.28E-04	0.236830659
144	10.1.2.2:20	10.1.1.2:28509	1.000000006	0.987577311	0.099071201	1	0	0.820861129
152	10.1.2.12:25	10.1.1.4:28514	2.06E-08	3.34E-04	0.251848881	0	0.975358678	0.224730274
174	10.1.1.2:28538	10.1.2.13:23	3.80E-07	3.51E-05	0.004451019	1	8.78E-04	0.00802362

When AP was first run on the data set, the algorithm finished with 41 different clusters for a dataset of 200 records. This ratio of clusters to records did not efficiently summarize the flows. The program was subsequently tuned by choosing preference values for the data points to be the smallest calculated similarity measure as recommended by the developers of AP. Lower preference values result in a smaller number of clusters. As a result, the recalibrated AP algorithm summarized the records into 5 separate classes. Table 4.2 shows that AP identified clusters (58, 133 and 174) for HTTP port 80, SMTP port 25 and Telnet port 23 respectively. In addition AP distinguished the FTP data channel (cluster 144) apart from the control channel. Though not shown, the FTP control channel was assigned to the TELNET group which makes sense since both traffic types have similar persistent, low-bandwidth traffic characteristics. Furthermore, AP created cluster 152 which groups flows initiated by an SMTP server process back to a mail client. This was ideal behavior since this type of traffic is inherently different from the rest of the SMTP flows in the dataset.

In [13], the authors state that Affinity Propagation finds clusters with lower error rates than other methods and that the running-time of AP is a fraction of that of other techniques. That being the case, it is evident that computing the responsibilities and availabilities for an N-by-N similarity matrix is memory and CPU intensive for large N. The developers claim that AP can cluster up to 23K data points in a few hours using a modern single-core computer. The AP implementation for this research could handle 10% of that load over the same time period. I attribute this performance degradation to a couple factors. First, Java was used for speed of development and proof-of-concept

prototyping. Coding techniques that optimize performance for numerical vector computations would give a faster running time. For example, Java objects (multidimensional array of BigDecimal) were employed instead of C++ numerical vectors (valarray), and the AP calculations operated on pair-wise objects instead of entire vectors. The implementation also ran on a mid-range level laptop with limited resources with respect to CPU and RAM. In addition, it is also possible that using the latest compilers would generate more efficient Java byte-code that would rival the speed of native machine code generated from languages such as C/C++.

Assuming we can attain the maximum performance from the given software and hardware, the fact remains that most databases of network flows are exceedingly large, hundreds of thousands if not millions of flows over the course of a day or week. Scaling this solution for large input sizes would require modifications to the algorithm which have been proposed by the original authors. For example, AP can be modified to operate on a smaller subset of similarities. This can be done by randomly selecting pair-wise data points that send messages at each iteration or omitting similarities altogether for selected data point pairs beforehand. In [12], the researchers recommend a scalable hybrid K-means / Affinity Propagation clustering solution that leverages bisecting K-means for top-level clustering which creates balanced groups of data points and then applies AP to each group for the final clustering. Finally, application-specific sampling strategies can also be applied to the original datasets. For example, network flows grouped by flow masks could be treated as duplicate entries and omitted from processing [15]. This type of compression can be applied at many levels at the cost of some precision and accuracy.

In this thesis, manageable data set sizes were used to prove out the AP clustering algorithm. Performance enhancements are left as future work.

5 Information Visualization

After clustering large databases of flow records and exploring the clusters to understand the nature of the different traffic classes on the network, visualization models can be applied to give users a different perspective on the same dataset. For a Network Operations Center (NOC) analyst, there is a need to access and make sense of vast amounts of network log information. A network analyst is often presented with large amounts of data from hundreds if not thousands of managed hosts and network elements such as routers and firewalls describing the types of connections that have established over the network. Fundamentally, a network analyst should be able to project data using different techniques and to discern structure, patterns and achieve a higher-level of understanding from the visual transformations. Furthermore, the analyst would have controls to explore the data and the ability to modulate the resolution of the presented information. Fundamentally though, the focus is on using the right visualization tools and the right times. In [16], the author says "there is a need for tools that augment human ability to draw insight from abundant or complex data, in order to make decisions: faster, more accurately, with less cognitive effort, and with less training."

5.1 Treemaps

One visualization model that makes efficient use of 2-dimensional display spaces and has the ability to project large amounts of information is a Treemap. Treemaps were invented by Ben Shneiderman [17] to project hierarchical data. Conventional tree layouts depict both tree nodes and the edges between the nodes which make inefficient use of the display space. Alternatively, Treemaps are space-filling structures that operate on rectangles and embed child nodes within parent nodes so that the area of each rectangle on the display is related to the size of the node itself. Treemaps have been used to display many types of data from file system hierarchies where the size of the rectangle at each level is the size of the directory on disk to sports statistics and stock-market data. The layout of a Treemap is determined by recursively subdividing rectangles starting from the "root" rectangle using one of many different layout algorithms.

The most basic layout algorithm is called Slice-and-Dice. The idea here is to stack child rectangles either vertically or horizontally within a parent rectangle and then alternate the arrangement at each subsequent level of the tree. This algorithm makes no layout decisions based on the underlying data and has often produced skinny rectangles with poor aspect ratios. Treemap developers followed with the Squarified layout algorithm which creates more visually appealing rectangles with aspect ratios closer to 1. One of the limitations with Squarified is that node order is still not considered when the layout is computed. As a result, the Ordered Treemap algorithm was developed to achieve balanced node shapes and at the same time preserve order. A variant of Ordered Treemap will be discussed in detail shortly. Figure 5.1 shows how data would be

presented using three different Treemap layout algorithms. Shading is employed to show order. Only Ordered Treemap maintains order from top-to-bottom and left-to-right.

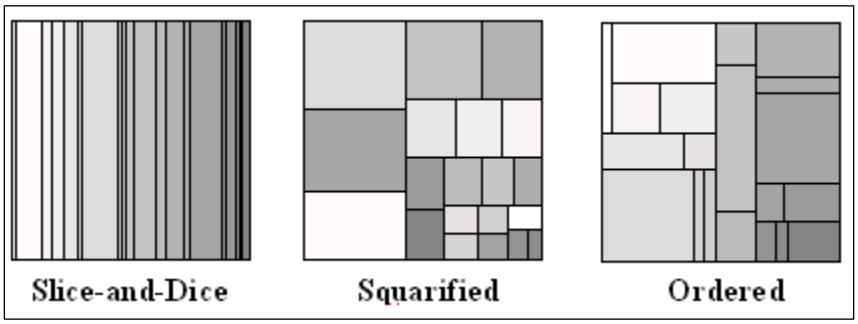

Fig. 5.1. Classical Treemap Layouts.

Although shading is used as an indication of order in Fig. 5.1, typically, shading or coloring of leaf nodes in a Treemap represents the magnitude of a measure of interest. For example, a Treemap depicting the structure of a file system might use color intensity to describe the number of file accesses over a specific time period.

In [18], researchers have extended the concept of the Ordered Treemap and have devised an altogether new model to depict a geographic map of the Internet. Geographic visualization is a powerful method to project data. The result is what is called a Hierarchical Network Map (HNMap) that attempts to display IP prefixes as geo-entities positioned relative to each other. This concept portrays IP prefixes as having an inherent hierarchical structure. They can be grouped according to 4 levels of classification: Continent → Country → Autonomous System (AS) → IP Prefix. Each of these categories corresponds to different levels in the Treemap layout. Similar to traditional Treemaps, the size of each node is directly proportional to the number of leaf items for that branch. In an HNMap, the size of each node is based on the number of IP prefixes.

Depending on the measure of interest, such as number of flows or number of packets, nodes will be assigned a color intensity based on the magnitude of the measured feature.

One important objective of an HNMap is to preserve neighborhoods of similar nodes. Therefore, the layout algorithm must define an ordering for each level. The first two levels are Continent and Country. The attributes used to order these geo-entities relative to one another are average latitude and longitude coordinates. Autonomous System nodes are ordered according to the median IP address in the range of addresses for that AS. The authors recognized that simple ordering based on AS numbers did not yield any interesting patterns when data was projected via HNMap. IP prefixes are ordered according to the median address as well. A summary is given in Table 5.1.

TABLE 5.1 HNMap Ordering

Level	Ordering Data	Dimensionality
Continent	Average coordinate of each continent	2
Country	Average coordinate of each country	2
AS	Median IP address of contained prefixes	1
Prefix	Median IP address	1

Traditional Treemaps use a single layout algorithm for all nodes, however, HNMap uses different types of nodes and thus requires different layout methods for each type. The authors derived a geographic layout algorithm called HistoMap which is based on the Ordered Treemap algorithm. HistoMap is used for the Continent and Country levels. AS entities are rendered using a modified 1-Dimensional version of HistoMap. Finally, IP prefixes are rendered by the popular Strip Treemap [19] layout algorithm. Strip is a linear layout algorithm that divides the display area into strips or rows and assigns nodes

to each strip while at the same time optimizing the average aspect ratio of the rectangles in the strip. This is a simple algorithm that terminates when all the nodes have been processed.

5.2 The HistoMap Layout Algorithm

The logic of HistoMap is explained in [18]. To layout a collection of nodes or elements, HistoMap selects a pivot from the list such as the middle element. The list is split into two sections where elements on one side of the pivot are less than the pivot and elements on the other side are greater than the pivot. The idea is that the display area into which these elements are to be embedded can be split either vertically or horizontally. For a vertical split, a pivot is selected and the elements are ordered based on latitude. Elements smaller than or equal to the pivot latitude would be placed into the lower rectangular area. Elements greater than the pivot latitude are placed into the upper rectangular area. This is done since latitude values get increasingly smaller as you descend from North to South. For a horizontal split, a pivot is selected and the elements are ordered based on the longitude coordinate. Elements smaller than or equal to the pivot longitude would be placed into the left rectangular area. Elements greater than the pivot longitude are placed into the right rectangular area. This is done since longitude values get increasingly bigger as you traverse from left to right. In the HistoMap algorithm, both splits are calculated and the qualities of both are compared to determine the better split. Quality is based on the average aspect ratio of the two constituent rectangles for a given horizontal or vertical split. The aspect ratio of a split is given below (5).

$$aspect\_ratio = \frac{1}{N}\sum_{i}\frac{max(w_i, h_i)}{min(w_i, h_i)} \qquad (5)$$

N is the total number of rectangles in the split and *i* is the index over the rectangles. For HistoMap, *N* is equal to 2. The HistoMap algorithm is called recursively for each rectangle in a split until there is one element remaining at which time the rectangle is placed in its final position in the display area.

This method works nicely for laying out Continent and Country nodes. These entities assume positions on the display that distinguish geographic regions. A slight modification is required for Autonomous Systems in that the same ordering data (median IP address) is used for calculating both the vertical and horizontal splits. The result is AS entities that are ordered within countries such that entities placed in the top left corner of a rectangle have numerically smaller median IP prefixes than entities in the bottom right of the rectangle.

5.3 HNMap Implementation

5.3.1 HNMap Backing Data

At the time of this research, no publicly available code or implementation was found for HNMap. As a result, one product of this research is an implementation of HNMap using the description of the algorithm in [18]. To engineer an HNMap implementation, the backing data for the levels of the hierarchy were needed. The appropriate data was acquired from many different resources. First, both continent and

country names and their associated ISO 3166-1 codes where obtained from the ISO documentation. An excerpt of this data is given in Table 5.2.

TABLE 5.2 Excerpt of ISO 3166 Country Data

CC	A2	A3	N	Name
AS	AF	AFG	004	Afghanistan, Islamic Republic of
EU	AX	ALA	248	Åland Islands
EU	AL	ALB	008	Albania, Republic of
AF	DZ	DZA	012	Algeria, People's Democratic Republic of
OC	AS	ASM	016	American Samoa
EU	AD	AND	020	Andorra, Principality of
AF	AO	AGO	024	Angola, Republic of
NA	AI	AIA	660	Anguilla
AN	AQ	ATA	010	Antarctica (the territory South of 60 deg S)
NA	AG	ATG	028	Antigua and Barbuda

Next, each of the 5 regional registries publishes a record of the AS and IP prefix records under their respective domains and includes the countries to which they are assigned. The 5 registries are ARIN, RIPE, AFRINIC, APNIC and LACNIC and these databases are available via FTP from:

- ftp.arin.net/pub/stats/arin/delegated-arin-latest
- ftp.ripe.net/ripe/stats/delegated-ripencc-latest
- ftp.afrinic.net/pub/stats/afrinic/delegated-afrinic-latest
- ftp.apnic.net/pub/stats/apnic/delegated-apnic-latest
- ftp.lacnic.net/pub/stats/lacnic/delegated-lacnic-latest

To determine the AS name associated with an AS number, bulk queries were issued to WHOIS servers. The result is a file with supplemental information about each AS in the query. An excerpt of the file is shown in Table 5.3.

TABLE 5.3 Excerpt of WHOIS AS Query

AS	A2	Registry	Date	Name
1	US	arin	2001-09-20	LVLT-1 - Level 3 Communications, Inc.
2	US	arin	1991-01-10	DCN-AS - University of Delaware
3	US	arin		MIT-GATEWAYS - Massachusetts Institute of Technology
4	US	arin	1984-02-22	ISI-AS - University of Southern California
5	US	arin	1984-02-02	SYMBOLICS - Symbolics, Inc.

Next, to determine the AS to which each IP prefix is assigned, a similar bulk query, containing each IP prefix in the registry domain, was issued to WHOIS. The result is a file with supplemental information about each prefix. An excerpt is given in Table 5.3.

TABLE 5.4 Excerpt of WHOIS IP Query

AS	First IP	IP Prefix	A2	Registry	Date	AS Name
7018	12.0.0.0	12.0.0.0/9	US	arin	1983-08-23	ATT-INTERNET4 - AT&T WorldNet Services
33631	13.0.0.0	13.0.0.0/16	US	arin	1986-04-25	PARC-ASN - Palo Alto Research Center Incorporated
71	15.0.0.0	15.0.0.0/8	US	arin		HP-INTERNET-AS Hewlett-Packard Company
1889	16.0.0.0	16.0.0.0/12	US	arin	1989-05-18	HP-EUROPE-AS Hewlett-Packard Company
714	17.0.0.0	17.0.0.0/9	US	arin	1990-04-16	APPLE-ENGINEERING - Apple Computer, Inc.

Finally, the average latitude and longitude coordinates for each geographic entity were obtained from Maxmind, Ltd [20]. All of this data was organized and stored in a relational database. The necessary code was written to parse the data files and issue update queries to a backend MySQL database. There are 8 tables, 2 for each level of the hierarchy. Each level has its own table describing each of its nodes as well as a table holding the sort or ordering criteria for those nodes. The database tables and their associated fields are listed in Table 5.5.

TABLE 5.5 HNMap Database Tables

```
CREATE TABLE continent_table (
      code VARCHAR(10) PRIMARY KEY,
      name VARCHAR(100),
      latitude VARCHAR(10),
      longitude VARCHAR(10)
      );

CREATE TABLE country_table (
      code VARCHAR(10) PRIMARY KEY,
      code3 VARCHAR(10),
      name VARCHAR(100),
      pcode VARCHAR(10),
      latitude VARCHAR(10),
      longitude VARCHAR(10)
      );

CREATE TABLE as_table (
      asnum VARCHAR(10) PRIMARY KEY,
      registry VARCHAR(10),
      name VARCHAR(100),
      pcode VARCHAR(10),
      process VARCHAR(2)
      );

CREATE TABLE prefix_table (
      ipaddress VARCHAR(20) PRIMARY KEY,
      registry VARCHAR(10),
      prefix VARCHAR(20),
      asnum VARCHAR(10)
      );

CREATE TABLE continent_sort_table (
      code VARCHAR(10) PRIMARY KEY,
      sort VARCHAR(50)
      );

CREATE TABLE country_sort_table (
      code VARCHAR(10) PRIMARY KEY,
      sort VARCHAR(50)
      );

CREATE TABLE as_sort_table (
      code VARCHAR(10) PRIMARY KEY,
      sort VARCHAR(50)
      );
```

5.3.2 HistoMap Logic

With all the data properly stored in a SQL database, coding of the HNMap visualization began. The java implementation of HNMap uses a 3rd party visualization API called Prefuse[21]. Prefuse is a visualization framework for Java based on the Java 2D graphics library and provides the needed data structures for database tables and trees, as well as facilitates the visual encoding and rendering of data.

To get the IP hierarchy into the program, a TreeML XML file of the hierarchy was generated. TreeML is an XML format for representing the nodes and edges of a tree. Prefuse has built-in support for TreeML and automatically converts well-formed TreeML documents into Prefuse Tree data structures for Java. The first step was to query the backend database system for all IP prefixes including the data relevant to its hierarchy. The following SQL join query cross-references the 4 tables for continent, country, AS and prefix and returns a "path" for each prefix in the system:

```
SELECT continent_table.code,continent_table.name,
       country_table.code,country_table.name,
       as_table.asnum,as_table.name,
       prefix_table.ipaddress,prefix_table.prefix
FROM   continent_table,country_table,as_table,prefix_table
WHERE  continent_table.code=country_table.pcode AND
       country_table.code=as_table.pcode AND
       as_table.asnum=prefix_table.asnum
ORDER BY continent_table.code,country_table.code,as_table.asnum
```

The number of prefixes in the system exceeds 60K. An excerpt of the output is presented in Table 5.6 which shows a small number of prefixes in Africa.

TABLE 5.6 Excerpt of SQL Query Results for TreeML Data

CC	Continent Name	A2	Country Name	AS	AS Name	IP Address	IP Prefix
AF	Africa	ZA	South Africa	2018	TENET-1	198.54.66.0	198.54.66.0/24
AF	Africa	ZA	South Africa	2018	TENET-1	198.54.65.0	198.54.65.0/24
AF	Africa	ZA	South Africa	21739	TSOL	196.202.248.0	196.202.248.0/22
AF	Africa	ZA	South Africa	22355	FROGFOOT	196.1.56.0	196.1.56.0/21
AF	Africa	ZA	South Africa	22355	FROGFOOT	41.206.192.0	41.206.192.0/19
AF	Africa	ZA	South Africa	22386	SARB	196.29.240.0	196.29.240.0/20

The result set of this query was converted into an XML file following the TreeML format. A small excerpt of the beginning of such as file is shown in Table 5.7 which shows the root node, “World”, continent “Africa”, country “Angola”, autonomous system “11259” and prefix “41.223.156.0/22”.

TABLE 5.7 Excerpt of TreeML File

```
<?xml version="1.0" encoding="UTF-8" ?>
<tree>
  <declarations>
    <attributeDecl name="code" type="String" />
    <attributeDecl name="name" type="String" />
    <attributeDecl name="sort" type="String" />
    <attributeDecl name="path" type="String" />
  </declarations>
  <branch>
    <attribute name="code" value="World" />
    <attribute name="name" value="World" />
    <attribute name="sort" value="N/A" />
    <attribute name="sort" value="" />
    <branch>
      <attribute name="code" value="AF" />
      <attribute name="name" value="Africa" />
      <attribute name="sort" value="2.1985,17.3971" />
      <attribute name="path" value="Africa" />
      <branch>
        <attribute name="code" value="AGO" />
        <attribute name="name" value="Angola" />
        <attribute name="sort" value="-12.5000,18.5000" />
        <attribute name="path" value="Africa -> Angola" />
        <branch>
          <attribute name="code" value="11259" />
          <attribute name="name" value="ANGOLATELECOM" />
          <attribute name="sort" value="41.223.158.0" />
          <attribute name="path" value="Africa -> Angola -> AS11259 (ANGOLATELECOM)" />
          <leaf>
            <attribute name="code" value="41.223.156.0" />
            <attribute name="name" value="41.223.156.0/22" />
            <attribute name="sort" value="41.223.156.0" />
            <attribute name="path" value="Africa -> Angola -> AS11259 (ANGOLATELECOM) -> 41.223.156.0/22" />
          </leaf>
        </branch>
```

After loading the HNMap data into the Prefuse built-in Tree data structure, the implementation of HistoMap was engineered to operate on this data structure and the rendering was achieved by leveraging the Prefuse visualization API. The code sample below shows the process of splitting the elements in line 466, and making a recursive call for each split in lines 475-476. The recursion ends when the size of the element list is 1 at which time the “else” segment is executed which draws the rectangle on the display.

TABLE 5.8 HistoMap Layout Routine

```
458     /**
459      * Compute the HistoMap layout.
460      */
461     private void layoutHistoMap2D(ArrayList p, Rectangle2D r, boolean is2D) {
462 
463         if(p.size() > 1) {
464 
465             //Split the nodes vertically and horizontally, return best quality
466             ArrayList partitions = splitRect(p, r, is2D);
467 
468             ArrayList p1List = (ArrayList)partitions.get(0);
469             Rectangle2D p1Rec = (Rectangle2D)partitions.get(1);
470 
471             ArrayList p2List = (ArrayList)partitions.get(2);
472             Rectangle2D p2Rec = (Rectangle2D)partitions.get(3);
473 
474             //Recurse on each rectangle
475             layoutHistoMap2D(p1List, p1Rec, is2D);
476             layoutHistoMap2D(p2List, p2Rec, is2D);
477         } else {
478 
479             //Layout node on the display
480             NodeItem node = (NodeItem)p.get(0);
481             setX(node, (VisualItem)node.getParent(), r.getX());
482             setY(node, (VisualItem)node.getParent(), r.getY());
483             node.setBounds(r.getX(), r.getY(), r.getWidth(), r.getHeight());
484         }
485     }
```

5.3.3 HNMap Screenshots

Fig. 5.2 shows the first level of the HNMap, depicting Continent rectangles outlined in yellow, their sizes based on number of IP addresses, and their positions on the map. Six continents are shown including North America (NA), Europe (EU), Africa (AF), Asia (AS), South America (SA), and Oceania (OC).

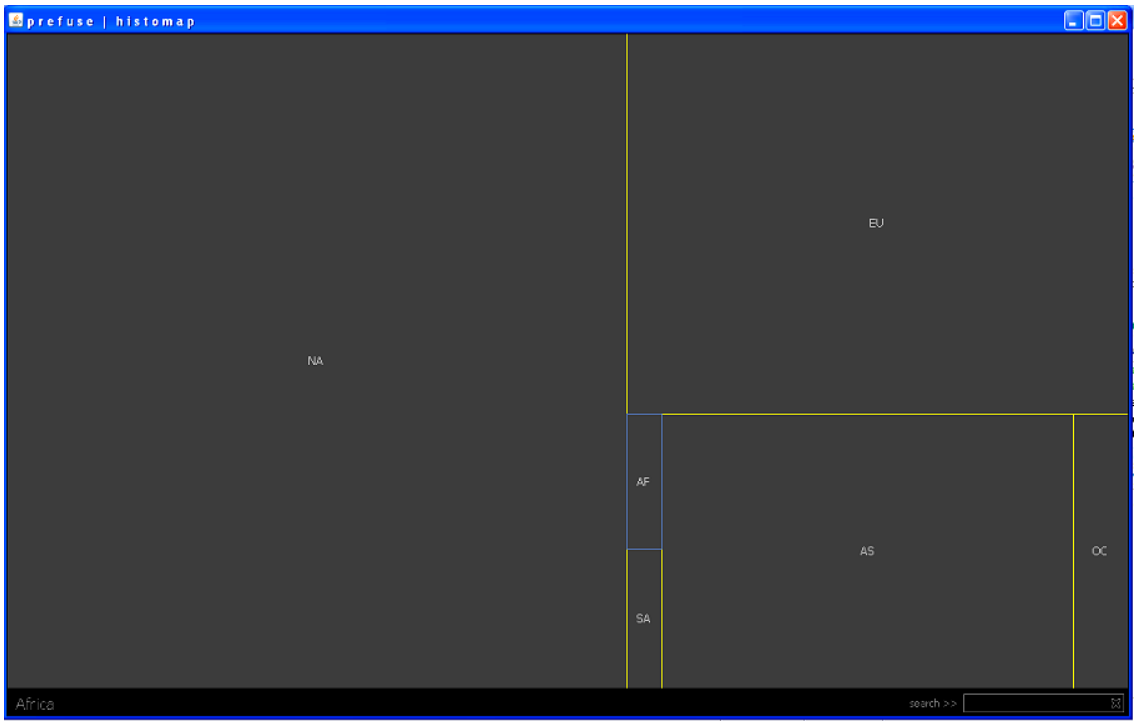

Fig. 5.2. HNMap Continents.

Fig. 5.3 shows the first 2 levels of the map down to the constituent countries for each continent. Again, the layout of both continents and countries depends on both their ordering and the type of split made (horizontal or vertical). The left half of the figure shows North America which is comprised of three countries, Canada, USA and Mexico, stacked from North to South in that order. This implies that a vertical split was calculated for that iteration of the algorithm. USA is highlighted in blue and the path to this node is displayed in the lower right-hand corner as "North America → United States of America".

Fig. 5.3. HNMap Countries.

Fig. 5.3 shows the top 3 levels of the map down to the individual autonomous systems for each country. Easily noticeable and highlighted in blue is the biggest AS in the United States. The path for that AS is shown in the lower right-hand corner as "North America → United States of America → AS27064 (DNIC-ASBLK-27032-27159 – D)".

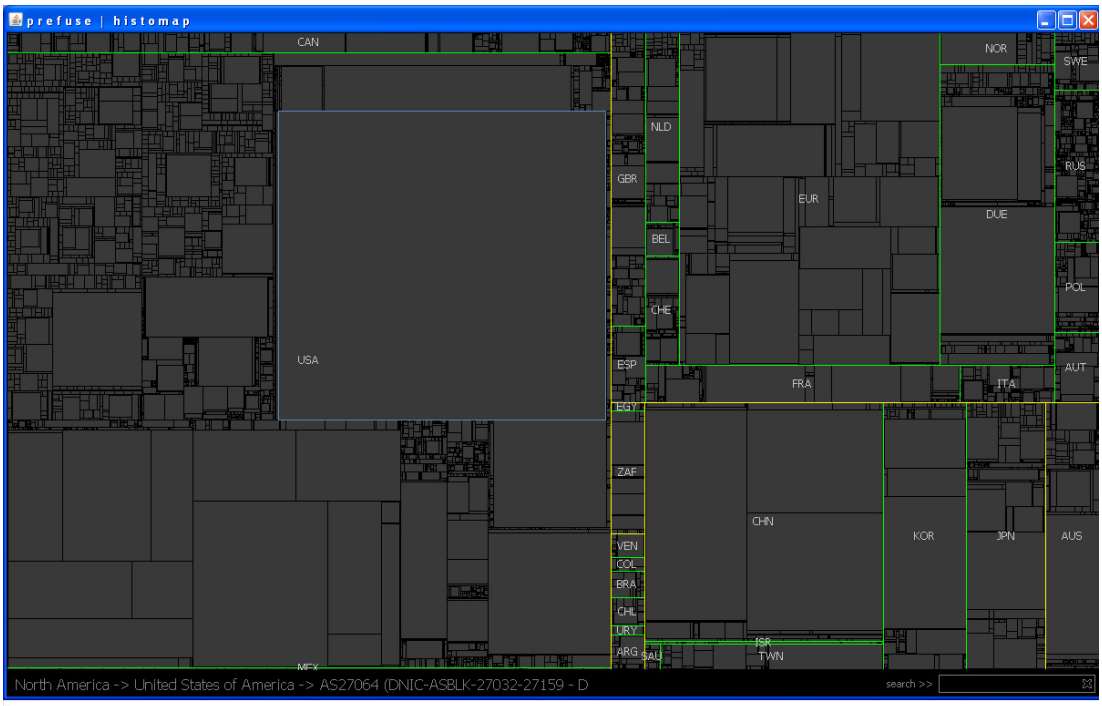

Fig. 5.4. HNMap Autonomous Systems.

Fig. 5.5 shows the IP prefixes for AS 22047 in the country of Chile. Prefixes are ordered linearly and laid out in rows or strips. The highlighted node is prefix "200.86.32.0." Although it is difficult to make out in the figure below, the path in the lower right hand corner is "South America → Chile → AS22047 (VTR BANDA ANCHA S.A.) → 200.86.32.0/20."

Fig. 5.5. HNMap prefixes.

5.4 HNMap Projection

After traffic records have been clustered and investigated, the flows can be projected onto an HNMap with respect to a measure of interest to get a better understanding of the distribution of the behavior. For example, a cluster identified as potential Distributed Denial of Service (DDoS) traffic can be visualized by coloring the HNMap prefix nodes based on the number of outgoing flows found in the cluster. In addition, animating HNMap or taking snapshots of HNMap over time allows the user to observe sharp increases in flow counts over the time period. This provides additional confirmation of the DDoS behavior. Furthermore, the mapping of these data will identify the origins of the BotNet perpetrating the malicious behavior such as the infected prefixes and autonomous systems. This information can be used by service providers to deploy measures at network access points to mitigate and possibly thwart the attack. Fig. 5.6 shows such a scenario of a simulated DDoS attack centered in Asia, specifically from the countries of China, Korea and Japan. Three chronological snapshots are taken showing the increase in intensity of the cluster behavior. Prefix nodes are colored according to the magnitude of originating flows from blue, white, pink, and red depending on the intensity of the measure.

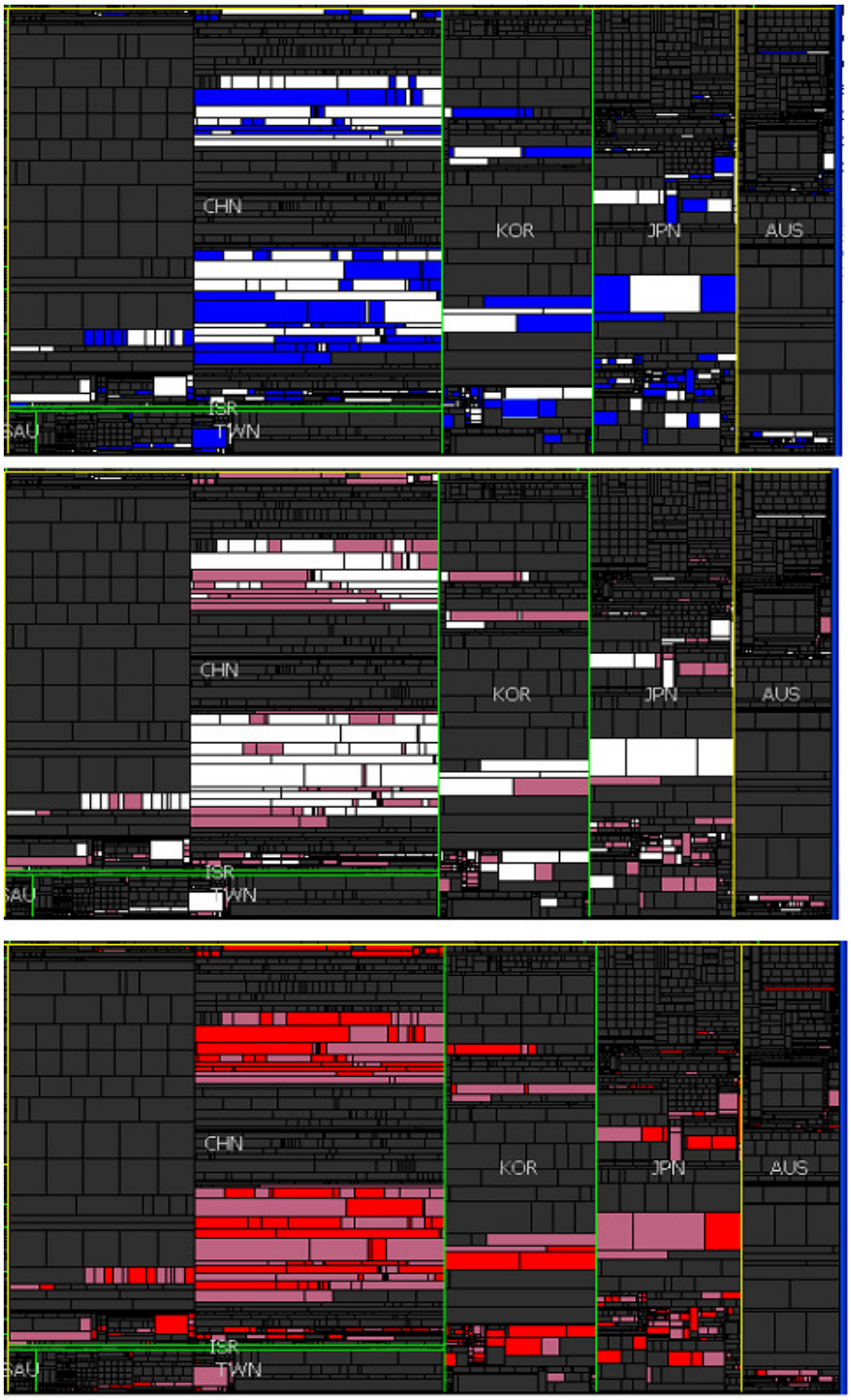

Fig. 5.6. DDoS cluster projection.

Conclusion

Both data mining and information visualization are beneficial to identifying structure and extracting meaning from databases of information. Progress is continual in each discipline, and this research focused on new developments in the areas of clustering and Treemap layout algorithms when applied to the problem of network traffic classification. Clustering algorithms will improve in their accuracy, runtime and scalability, while visualization models can be adapted to project application-specific structure in innovative ways. Taken separately, however, neither approach alone will offer the full range of analysis discussed in this paper. While clustering provides general groupings of network flows based on abstract numerical dimensions, HNMap allows the user to visualize these structures in an application-specific context. Conversely, HNMap provides a visual encoding for network communication attributes; however only the accurate grouping of network flows provides a context in which to understand and explore the projection. It is clear that both data mining and information visualization should be used together as investigative tools for network traffic analysis.

References

[1] A. McGregor, M. Hall, P. Lorier, and J. Brunskill, "Flow Clustering Using Machine Learning Techniques," *Passive and Active Network Measurement*, vol. 3015, no. 5, pp. 205-214, April 2004.

[2] J. Erman, M. Arlitt, A. Mahanti, "Traffic Classification Using Clustering Algorithms," *Proc. SIGCOMM Workshop on Mining Nework Data*, pp. 281-186, Sept 2006.

[3] D. Hand, H. Mannila, P. Smyth, *Principles of Data Mining*, Cambridge, MA: The Massachusetts Institute of Technology, 2001.

[4] S. Petrik, V. Skala. (2009, April 16). *Introduction to Information Visualization Supplementary Material* [Online]. Available: http://herakles.zcu.cz/seminars/docs%5Cinfovis%5Cinfovis.pdf

[5] A. W. Moore, D. Zuev, "Discriminators for Use in Flow-based Classification," Department of Computer Science, Queen Mary, University of London, Technical Report RR-05-13, 2005.

[6] Z. Li, R. Yuan, X. Guan, "Accurate Classification of the Internet Traffic Based on the SVM Method," *ICC*, pp. 1373-1378, June 2007.

[7] Ixia Technical Writers. (2009). *Ixia IxLoad 4.0 EA SP1 User Guide* [Online]. Available: http://www.ixiacom.com

[8] Tcpdump software. (2009). *Tcpdump/Libpcap* (Version 4.0.0) [Online]. Available: http://www.tcpdump.org/

[9] R. Bejtlich, *The Tao of Network Security Monitoring,* Boston, MA: Addison-Wesley, 2005.

[10] N. Alldrin, A. Smith, D. Turnbull. (2003). *Clustering with EM and K-Means* [Online]. Available: http://cseweb.ucsd.edu/~atsmith/project1_253.pdf

[11] A. Reynolds, G. Richards, V. Rayward-Smith, "The Application of K-medoids and PAM to the Clustering of Rules," *Lecture Notes in Computer Science*, vol. 3177, pp.173-178, Oct 2004.

[12] N. Andrews, E. Fox, "Clustering for Data Reduction: A Divide and Conquer Aproach," Computer Science, Virginia Tech, Technical Report TR-07-36, 2007.

[13] B. Frey, D. Dueck, "Clustering By Passing Messages Between Data Point," *Science*, vol. 315, pp. 972-976, Feb 2007. Available: http://www.psi.toronto.edu/affinitypropagation/FreyDueckScience07.pdf

[14] Affinity Propagation Software. (2009). *Affinity Propagation* [Online]. Available: http://www.psi.toronto.edu/affinitypropagation/

[15] F. Dressler, G. Munz, "Flexible Flow Aggregation for Adaptive Network Monitoring," *Proc. 2006 31st IEEE Conference on Volume*, pp. 702-709, 2006.

[16] J. Agutter, J. Bermudez, "Information visualization design: The growing challenges of a data saturated world," *AIA Report on University Research,* pp. 61-75, 2005.

[17] B. Shneiderman. (2008, June 18). *Treemaps for space-constrained visualization of hierarchies* [Online]. Available: http://www.cs.umd.edu/hcil/treemap-history/

[18] F. Mansmann, D.A. Keim, S.C North, B. Rexroad, D. Sheleheda, "Visual Analysis of Network Traffic for Resource Planning, Interactive Monitoring, and Interpretation of Security Threats," *IEEE Trans. On Visualization and Computer Graphics,* pp. 1105-1112, 2007.

[19] B. Bederson, B. Shneiderman, M. Wattenberg, "Ordered and Quantum Treemaps: Making Effective Use of 2D Space to Display Hierarchies, " *ATM Trans. On Graphics,* pp. 833-854, 2002.

[20] Maxmind, Ltd. (2009). *Average Latitude and Longitude for Countries* [Online]. Available: http://www.maxmind.com/app/country_latlon

[21] Prefuse Information Visualization Toolkit. (2007). *Prefuse Beta Release* (Version 2007.10.21) [Online]. Available: http://prefuse.org/