Towards Understanding Network Traffic through Whole Packet Analysis

Abdulrahman Hijazi Hajime Inoue
Ashraf Matrawy P. C. van Oorschot Anil Somayaji
Carleton Computer Security Laboratory, Carleton University

Carleton University School of Computer Science TR-07-06 (Febuary 2007)

Abstract

We present ADHIC, an algorithm that hierarchically clusters network traffic without making assumptions about the structure of packets. Packets are judged similar using patterns of n-byte strings at fixed offsets p, or (p, n)-grams. By sampling packets to find high frequency (p, n)-grams and then applying divisive hierarchical clustering (an unsupervised machine learning method), ADHIC can separate traffic along typical divisions such as IP vs. non-IP traffic, TCP and UDP, and standard applications such as web and SSH traffic without using domain-specific knowledge. It can also correctly cluster data transmitted on non-standard ports, and can even appropriately segregate the traffic from applications that do not use standard ports (such as peer-to-peer programs). NetADHICT, our implementation of ADHIC, is available for download and is licensed under the GNU GPL license.

1 Introduction

The behavior of modern computer networks is fundamentally complex: many users, many uses, numerous protocols, massive operating systems, complex applications, and a wide variety of connected devices. As a result, understanding the behavior of even the simplest local area network can be challenging; nevertheless, for network management, debugging, and security purposes, such analysis has become more and more important. Common analysis strategies have been to classify traffic using predetermined classifiers, based on ports and IP addresses or also using protocol dissectors [1]. While such approaches can help one understand a given set of packets, they are much less helpful when the problem is to understand the overall structure of network traffic.

One strategy for obtaining such a higher level view is to cluster packets into "equivalence" classes based upon some measure of similarity. When packets are compared using header information, clustering can reveal many interesting patterns [4]; however, many patterns of traffic, including peer-to-peer networks, self-propagating malware (worms), and flash crowds are often more properly distinguished by patterns in payloads, not headers. An ideal packet clustering technique, then, would be able to extract patterns from anywhere in a packet.

A key challenge for any whole packet clustering technique is to function online at wire speeds. Most machine learning approaches to clustering assume batch-style analysis and involve a quadratic number of comparisons or more [2]. We have developed a technique for clustering packets that finds semantically interesting clusters, requires no explicit a priori information about the structure or content of network packets, and can be run online in sub-linear time (in the length of the packets). Our technique is based upon two innovations: a simple measure of similarity that we refer to as (p, n)-grams [13], and a clustering technique that we call Approximate Divisive HIerarchical Clustering (ADHIC). Both of these exploit two key features of our problem domain: network traffic has large amounts of redundancy, and optimal clustering is not required to capture the high-level structure of network traffic.

To evaluate ADHIC, we have implemented a packet analysis tool called NetADHICT (pronounced "net addict"). NetADHICT can analyze data as it is received by a network interface, or it can analyze libpcap-format files offline [23]. Observed data is used to incrementally generate and update a (p, n)-gram decision tree, such that at any given point in time, NetADHICT has a classifier tree that reflects the high-level structure of current network traffic.

In experiments using data captured from our lab's production network (see section 4), we have found that NetADHICT can quickly capture the overall structure of traffic. The learned classifier tree typically has a complex structure, often containing over a hundred nodes; the depth of the tree, however, remains small (less than 10 (p, n)-grams), allowing new packets to be quickly classified.

While much of the inferred structure corresponds to typical divisions of network traffic (TCP vs. UDP, web vs. non-web traffic), these divisions are arrived at using (p, n)-grams that generally are only meaningful within the context of a given environment. This quality results in some interesting consequences. First, the structure of the tree directly reflects the relative popularity of different uses of network bandwidth, as defined by applications, networks, hosts, or users, depending upon context. Further, the use of non-standard ports for protocols has little effect on how packets are clustered (i.e., they are grouped as if they used the standard port for the application). Applications that do not have standard ports, such as the Bittorrent peer-to-peer file sharing protocol [20, 16], can also be clustered appropriately—all without requiring any protocol-specific information. Similarly, encrypted traffic is also often clustered appropriately because we are able to appropriately segregate all other traffic. Thus, NetADHICT is a powerful tool for investigating network behavior that does not require prior knowledge of a network's topology or configuration.

To summarize, in this paper we present ADHIC, a novel algorithm in network traffic clustering. We also introduce NetADHICT, our implementation of NetADHICT. Finally, we describe our results which show that the clusters produced by NetADHICT are semantically meaningful, even when confronted with p2p traffic and denied header information.

The rest of this paper proceeds as follows. We review related work in network traffic analysis in Section 2. Section 3 explains (p, n)-gram based ADHIC. Section 4 describes our implementation, NetADHICT, and its configuration in experiments. Sections 5, 6, 7, and 8 present more detail on NetADHICT's performance through an explication of a few decision trees, overall classification behavior, and performance in the face of peer-to-peer traffic and traffic containing no headers. Section 9 places our work in context and discusses limitations and future work, and we conclude with Section 10.

2 Related Work

The work we present takes root in two different research areas. Our tool can be used to improve network traffic analysis in general, while the algorithm at the heart of the tool is a new machine learning algorithm specialized at understanding network traffic. This section summarizes related work in both areas.

Substantial research has explored the question of how to characterize and model Internet traffic. Internet traffic exhibits burstiness which drew the attention of researchers. Seminal work by Leland et al.[11] presented the self-similar properties of Ethernet traffic. The failure of the Poisson model as a representation of wide-area traffic was reported by Paxson and Floyd [15].

Another area of attention is the analysis of the Internet address structure. Moving from exact-lookup match to prefix-match in routing protocols was based on the structure of IP addresses. The change enhanced performance for routing lookup processes. Kohler et al.[8] characterized the structure of destination IP addresses on Internet links, and proposed a multifractal model of the observed addresses.

In addition to techniques that examine individual flows, aggregates of flows have been explored and analyzed. Estan et al.[4] looked at multidimensional clustering of Internet traffic, based on 5-tuples of fields in the header of IP packets (source IP address, destination IP address, protocol, source port, and destination port). Clustering based on these 5-tuples showed patterns of resource consumption (in terms of traffic volumes). A tool called AutoFocus [4] was developed to do such clustering and generate traffic reports. Lan and Heidemann [10] studied the correlation between different flow characteristics of Internet traffic - namely size, duration, rate and burstiness. One important result reported is strong correlations between specific combinations of size, rate, and burstiness. They also present a good discussion on flow classification criteria. Mahajan et al.[12] studied aggregates of flows in the context of preventing flash crowds and denial-of-service (DoS) attacks.

Although not considered traffic analysis, a body of work worth mentioning here is in the area of automatic pattern inference for security applications. One application area is the detection of new attacks with systems such as EarlyBird [18, 19], HoneyComb [9], and Autograph [7]; see also [13, 21] and discussions at the end of this section.

Network traffic characterization and analysis based on machine learning (ML) is receiving attention due to the shortcoming of traditional traffic characterization methods. The main problem with the latter is the *dynamic* nature of Internet traffic where the mixture of traffic in a given network changes rapidly and significantly. Such changes often render characterization methods based on previous knowledge of traffic (e.g. specific header information) obsolete. Furthermore, there is the potential of some users deliberately changing their packet contents to hide this traffic from service providers. The most famous example is peer-to-peer (p2p) traffic using non-default ports and encrypting their traffic.

Traffic classification using machine learning strategies was first proposed in the context of anomaly detection for security purposes by Frank [5] . Many others have followed a similar strategy, giving rise to the field of anomaly intrusion detection; we do not attempt to describe this work here. Our algorithm differs in that we do not attempt to classify clusters as normal or malicious, or in any way label the groups. Our algorithm is a clustering algorithm, not a classifier.

Traditional machine learning approaches have been applied to specific packets fields such as the 5-tuple (source and destination addresses, port numbers, protocol), packet length, and inter-arrival times to classify traffic into pre-determined classes. Clustering traffic into application classes using ML was studied by Zander et al.[24] and McGroger et al.[14]. Another example is the clustering of traffic into different QoS classes by Roughan et al.[17]. A comparison between different ML approaches for traffic analysis was presented by Williams et al.[22], with a focus on the computational performance of the different algorithms rather than the accuracy of classification. Erman et al.[3] compare unsupervised learning to supervised learning algorithms.

Our approach differs from these methods in that we do not rely on any previous knowledge of packet contents, nor do we designate any particular fields as fields of interest in the learning process. Moreover, our clustering algorithm is dissimilar to other machine learning algorithms in common use (see Section 3).

Our new algorithm, ADHIC, is substantially different than that described by Matrawy et al.[13]. Although we still employ (p, n)-grams, ADHIC produces a decision tree rather than a linear classifier. This switch to a decision tree has significantly improved runtime performance; it has also resulted in clusters that map more readily onto standard classifications of network traffic. It is likely that ADHIC's clusters are also more suitable for our earlier goal of managing network bandwidth; the suitability of such an application is a subject of future work.

3 Approximate Divisive Hierarchical Clustering

The goal of our research is to devise a technique to group network traffic into semantically meaningful "equivalence" classes using no prior knowledge of packet or protocol structure. Furthermore, the groups should be adjusted over time to adapt to the changing nature of normal traffic patterns.

The strategy we use is to recursively subdivide traffic into binary classes. We stop dividing classes when the resulting traffic is below some configurable threshold in volume or is too similar or dissimilar. This is a common clustering approach called *divisive hierarchical clustering*. We call our approach Approximate Divisive Hierarchical Clustering (ADHIC) because we use a sampled measure of similarity.

This method produces a binary decision tree. It consists of internal decision nodes, and leaf nodes which are the final clusters. Each internal node has two subtree children. Traffic that matches a classification rule within the node is directed to the left, or *true* subtree. The rest of the traffic is directed to the right subtree. Because rightmost subtrees have not matched any classification rules, we sometimes refer to these as *default* clusters. The rightmost cluster of the entire tree is the *global default cluster*.

The tree matches a set of boolean operations. Traffic within each terminal cluster can be viewed as the result of a boolean equation constructed by following the path from the root node to the leaf. Left subtrees are combined with **and** and right subtrees **and not**.

The classification rule we use within ADHIC is based on a feature we call (p, n)-grams. Introduced in an earlier paper [13], (p, n)-grams are n-bytes substrings at a p byte offset. Each decision node within the tree is associated with a (p, n)-gram. If a packet contains the same n bytes at byte offset p as specified by the (p, n)-gram associated with the node, it is said to *match*. We use the following notation to refer to specific (p, n)-grams. A (p, n)-gram substring of length two ($n = 2$), consisting of the bytes 0x00 and 0x08 at offset 43 is denoted by (43, 0x00 0x08).

The tree is generated and adapts through two operators: **split** and **delete**. Splitting occurs when a leaf cluster matches more than some threshold of traffic and that traffic is between a set maximum and minimum difference in similarity (called the **similarity spread**). Similarity in ADHIC is measured by finding a (p, n)-gram such that it is found in roughly half of the packets matched by the cluster. These statistics are measured over a period called the **maturation time** or **maturation window**. Nodes which have been modified within a maturation window of the current time are locked and cannot be split or deleted.

Deletion occurs when a subtree has not matched a minimum threshold of traffic during the most recent maturation window. The subtree's parent node is also deleted. The parent node's other tree, the one not deleted, becomes the direct child of the parent node's parent.

For performance reasons, splitting and deletion do not occur continuously. They are restricted to intervals of several minutes. This is called the **update period**. The similarity spread, maturation window, update period, and the thresholds for splitting and merging are all configurable parameters.

We considered using a slightly more complex set of operators. Internal (p, n)-gram nodes could be merged instead of deleted. Multiple (p, n)-grams within a node would correspond to a logical **or**, to go with existing **and** and **not**. Individual (p, n)-grams within nodes would then be deleted if they did not match a packet within the maturity period. This more closely approximates the operator regime used in related work [13]. We found, however, that the quality of the clusters produced by ADHIC with either operator regime was similar. ADHIC produces much better clusters than the algorithm introduced in the earlier paper, regardless of the set of operators used.

Figure 1, a sample decision tree produced by our AD-

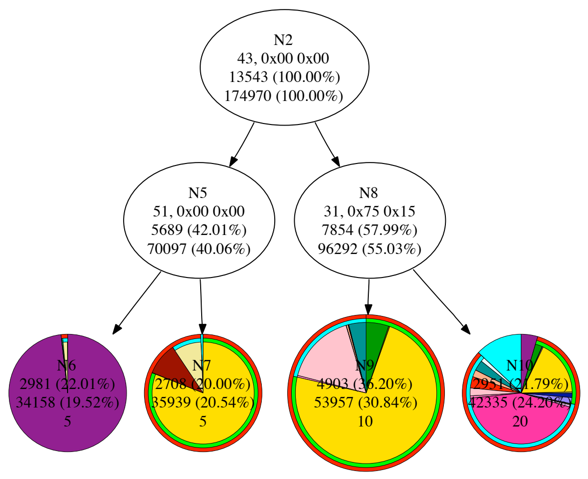

Figure 1: An example decision tree. Internal nodes are (p, n)-gram nodes. A packet is classified by starting at the root node, N2. If it contains the (p, n)-gram (43, 0x00 0x00) as a substring, the packet is compared with the left child. If it does not, it is compared with the right. This is done recursively until a leaf node is reached; these signify clusters. For example, cluster N9 matches packets that do not contain (43, 0x00 0x00) and do contain (31, 0x75 0x15). Actual trees are much larger. See Figure 2 for a complete, generated tree. Each node in the tree has a node identifier (e.g. N8), and statistics of packets count and percentage seen by the node per update period in the upper line and per maturity window in the lower line (see Section 4 for more information about these parameters). While internal nodes contain also the (p, n)-gram value, leaves have the number of different protocols at the bottom line where each protocol is represented by a slice (with a size proportional to its volume) in the pie chart. The circle size of the leaf node (cluster) represents the number of packets seen.

HIC algorithm, shows traffic split into two clusters by the (p, n)-gram (43, 0x00 0x00). These two clusters were split into four terminal, or leaf clusters. The traffic that matched (43, 0x00 0x00) is matched against the (p, n)-gram (51, 0x00 0x00).

The space of (p, n)-grams is very large and the total number of (p, n)-grams that appear in a trace, though much smaller than the total possible, is also large. The number of (p, n)-grams in a single packet is simply the length of the packet minus the length of the (p, n)-gram. More simply, the number of (p, n)-grams for each trace is about the number of bytes in the trace. The space of (p, n)-grams is dependent on the length of each packet and the length of the (p, n)-gram. For example, there are approximately 384,000 (p, n)-grams of length 1 ($256 * 1500$), 100 million of length 2, and 6.5 trillion for length 4. In this paper, for empirical reasons, we set $n = 2$. Because of the frequency distribution of (p, n)-grams in network traffic, the (p, n)-gram space covered by traces is always sparse. For a discussion of the settings of this and other parameters, see Section 4.

By viewing packets as individual (p, n)-grams, the algorithm treats packets as high dimensional vectors (the number of dimensions is the packet's length). However, the (p, n)-grams are not independent. Clearly, the effective information provided by the (p, n)-gram vector is reduced by its overlapping nature, but also because the presence of one (p, n)-gram often affects the probability of other, non-overlapping (p, n)-grams. It is this property we exploit in our splitting function.

This method of splitting is far more efficient than that used in most divisive hierarchical clustering methods. Most choose a split that minimizes entropy in the generated groups [2]. This makes sense because a minimization of entropy ensures that similar items are grouped together. Unfortunately, entropy minimization is a slow calculation as it requires a separate computation for each of the $\binom{n}{2}$ choices for each split.

Our splitting algorithm uses individual (p, n)-grams as a proxy for the more expensive entropy calculation. The (p, n)-gram frequency distribution in network traffic conforms to a power-law analogous to Zipf's law [25]. Because power-law distributions decay very quickly, very few (p, n)-grams are frequent enough to be candidates for splitting. Thus, it becomes feasible to estimate (p, n)-gram frequencies for splitting by sampling from packets, further increasing the efficiency of the algorithm.

ADHIC differs from most clustering algorithms in one other significant way. ADHIC is online. Most clustering algorithms assume the entire set of data is available for the computation. Decision tree generators typically have a fixed division between training, usually supervised, and their use. ADHIC overlaps classification and tree construction. By allowing deletion of subtrees as well as splitting, ADHIC adapts to changing traffic patterns.

4 Implementation and Experimental Setup

NetADHICT is our implementation of the ADHIC algorithm. NetADHICT is licensed under the GNU GPL and available from `http://www.ccsl.carleton.ca/software`.

In our experience, NetADHICT effectively segregates all structured protocols. NetADHICT begins, usually, by separating IP traffic from TCP. Subsequently, in lower nodes of the tree, it sequesters specific protocols. As one would expect, it often segregates packets not only by protocol, but also by other characteristics. For example, TCP control packets with zero-length payload, such as SYN, FIN, RST, or ACK are often clustered separately. Also, common protocols are often grouped together and individual protocols often have the same shaped subtrees.

Much of the structure NetADHICT typically finds would also be found through more traditional header analysis techniques. Because NetADHICT looks at traffic with no pre-existing biases, though, it also classifies using unconventional measures. (p, n)-grams corresponding to zero-value padding, Ethernet frame addresses, checksums, and payload contents are all found in NetADHICT decision trees. This use of unconventional features seems to allow NetADHICT to be consistent in its classification of traffic even if header data is omitted (see Section 8).

We evaluated NetADHICT's performance at network analysis through a series of experiments on our laboratory's network. After every update period NetADHICT produces a snapshot of its decision tree and updates a log of statistics. The network analysis in the experiments described herein was produced using the time-series of the trees and logs produced by NetADHICT.

Parameter	Value
(p, n)-gram length	2
update period	10 minutes
maturation window	3 hours
split threshold	2%
delete threshold	0%
similarity spread	20%
sampling rate	20%

Table 1: NetADHICT parameters used in our experiments

We tested NetADHICT against four network traces from the Carleton Computer Security Laboratory (CCSL). Because ADHIC relies on viewing packets in their entirety (non-anonymized full packets), we have been unable to conduct experiments on larger networks due to privacy considerations. However, these traces are from a graduate student laboratory with over 15 machines, two network printers and about a dozen regular users. The lab also provides web, webmail, IMAPS, DNS, SSH, SMTP, and CUPS services to external hosts.

Each of the four traces consists of incoming network traffic captured throughout a course of seven consecutive days. Bidirectional traffic was captured by our network tap and we then excluded traffic whose MAC source was our lab. The intention was to examine only incoming traffic. ADHIC is capable of clustering bidirectional traffic, but we felt that a decision tree that mixed incoming and outgoing traffic would be less usable by network administrators.

The traces for our experiments are taken from the following dates: August 13-19, 2004, December 10-16, 2005, January 20-26, 2006, and April 3-9, 2006. All Ethernet packets are used for analysis with the exception of the August 2004 trace which was included for comparison with Matrawy et al.[13]; it contains IP traffic only. Table 2 provides the protocol traffic statistics for these traces calculated using a traditional 5-tuple classifier (i.e. using primarily Ethernet type, IP protocol, and port information) that we wrote.

All the NetADHICT experiments referenced herein were run with a fixed set of parameter values (see Table 1), determined by exploring several sets of alternative values using the four traces. In our testing environment, NetADHICT is not highly sensitive to most of these parameter values, and tends to produce qualitatively similar trees under many settings; thus, we have chosen these values as a reasonable trade-off between accuracy and runtime performance. For example, slightly better results were obtained with a 2 hour maturation window; analysis runtimes are much faster, however, with a 3 hour maturation window.

The one significant exception to this, however, is the value of n. In past work with an earlier proposed algorithm for DoS mitigation [13], the value of $n = 4$ was effective; for NetADHICT, the best results were obtained using $n = 2$.

On an Apple Mac Pro with 1GB of main memory and 2.66 GHz "Woodcrest" cores, NetADHICT is able to process 2.65MB per second, or 8405 packets per second. With the extensive logging and visualization framework used to analyze our experiments, NetADHICT is substantially slower. This speed is not sufficient for monitoring a high-speed link in real time; however, it is more than sufficient for a research prototype primarily running on offline traces. The lightweight nature of our algorithm should permit a high-performance implementation. Such work, however, is a topic for future research.

5 NetADHICT Decision Tree

To illustrate the effectiveness of NetADHICT, we next describe in detail an example decision tree and the types of clusters it produces. Figure 2 shows a decision tree produced after four days of execution. We have also added an annotated, symbolic version that is easier to explain in the same figure.

The first (starting from left to right) subcluster at N11 primarily divides ARP from non-ARP traffic. About 95% of traffic in this subtree consists of ARP (Address Resolution Protocol). Six of eight leaf clusters are *singular* (we refer to clusters that contain only one protocol, as classified by a traditional classifier using 5-tuples and other network header fields, as *singular* and denote them with a rounded gray box in the tree graph) with one almost entirely ARP and the other providing a classic "de-

Protocol	August 13-19, 2004	December 10-16, 2005	January 20-26, 2006	April 03-09, 2006
IPv4	9461744 (100.00%)	13014116 (85.73%)	11910975 (83.29%)	8684678 (87.00%)
TCP	4445742 (46.99%)	10141461 (66.81%)	7747114 (54.17%)	6107644 (61.18%)
TCP Unknown	543377 (5.74%)	87450 (0.58%)	861184 (6.02%)	44388 (0.44%)
DNS	0 (0.00%)	3 (0.00%)	0 (0.00%)	0 (0.00%)
MS WBT/MS RDP	18 (0.00%)	15 (0.00%)	9503 (0.07%)	9812 (0.10%)
IPP	481 (0.01%)	400141 (2.64%)	568669 (3.98%)	486936 (4.88%)
IMAPS	51629 (0.55%)	58799 (0.39%)	118224 (0.83%)	166291 (1.67%)
HTTPS	18192 (0.19%)	50889 (0.34%)	17492 (0.12%)	123983 (1.24%)
SSH	755329 (7.98%)	372847 (2.46%)	497408 (3.48%)	245513 (2.46%)
MS Streaming/RTSP	121034 (1.28%)	69326 (0.46%)	69768 (0.49%)	97814 (0.98%)
MYSQL	0 (0.00%)	5 (0.00%)	0 (0.00%)	10 (0.00%)
SMB	3468 (0.04%)	899 (0.01%)	1272 (0.01%)	274 (0.00%)
DCE_RPC	3391 (0.04%)	637 (0.00%)	352 (0.00%)	220 (0.00%)
MSNMS	1527 (0.02%)	5449 (0.04%)	1542 (0.01%)	3775 (0.04%)
XMPP	0 (0.00%)	433 (0.00%)	807 (0.01%)	1221 (0.01%)
TCP Sophos	0 (0.00%)	20515 (0.14%)	8453 (0.06%)	6080 (0.06%)
URD	0 (0.00%)	181 (0.00%)	196 (0.00%)	87 (0.00%)
TCP No Payload	2359999 (24.94%)	7442843 (49.03%)	4891984 (34.21%)	4069094 (40.76%)
RTSP	0 (0.00%)	7684 (0.05%)	17959 (0.13%)	2319 (0.02%)
NBSS	10661 (0.11%)	5017 (0.03%)	3804 (0.03%)	2639 (0.03%)
IRC	51 (0.00%)	20 (0.00%)	1566 (0.01%)	0 (0.00%)
NNTP	0 (0.00%)	2 (0.00%)	49 (0.00%)	258 (0.00%)
TELNET	6072 (0.06%)	45 (0.00%)	21 (0.00%)	14 (0.00%)
FTP	136476 (1.44%)	3292 (0.02%)	26310 (0.18%)	5298 (0.05%)
SMTP	1493 (0.02%)	23038 (0.15%)	17701 (0.12%)	25926 (0.26%)
IMAP	79 (0.00%)	2511 (0.02%)	612 (0.00%)	978 (0.01%)
CVS	1 (0.00%)	16 (0.00%)	2 (0.00%)	11646 (0.12%)
POP	985 (0.01%)	3461 (0.02%)	1362 (0.01%)	6033 (0.06%)
HTTP	431479 (4.56%)	1585943 (10.45%)	630874 (4.41%)	797035 (7.98%)
UDP	4730279 (49.99%)	2526432 (16.64%)	3864788 (27.02%)	2288802 (22.93%)
UDP Unknown	2493908 (26.36%)	5951 (0.04%)	8618 (0.06%)	3346 (0.03%)
DNS	21968 (0.23%)	91051 (0.60%)	60339 (0.42%)	67311 (0.67%)
CUPS	336827 (3.56%)	314578 (2.07%)	296348 (2.07%)	128278 (1.28%)
IPSec	0 (0.00%)	2 (0.00%)	44 (0.00%)	55 (0.00%)
WHO	6719 (0.07%)	6719 (0.04%)	6714 (0.05%)	6650 (0.07%)
XDMCP	75 (0.00%)	1 (0.00%)	124 (0.00%)	14 (0.00%)
RTP	0 (0.00%)	176322 (1.16%)	1587449 (11.10%)	248642 (2.49%)
MS SQL	813 (0.01%)	2 (0.00%)	0 (0.00%)	0 (0.00%)
NBDGM	67167 (0.71%)	80742 (0.53%)	73665 (0.52%)	62802 (0.63%)
DCE_RPC	6059 (0.06%)	26693 (0.18%)	25066 (0.18%)	17826 (0.18%)
MDNS	14542 (0.15%)	3380 (0.02%)	2333 (0.02%)	8843 (0.09%)
Ganglia	0 (0.00%)	402664 (2.65%)	233585 (1.63%)	219859 (2.20%)
NBNS	55779 (0.59%)	54753 (0.36%)	187518 (1.31%)	177278 (1.78%)
RIPv1	410864 (4.34%)	41948 (0.28%)	41940 (0.29%)	41538 (0.42%)
HSRP	1307515 (13.82%)	1306755 (8.61%)	1306607 (9.14%)	1293451 (12.96%)
DHCP	853 (0.01%)	2767 (0.02%)	18639 (0.13%)	1257 (0.01%)
SNMP	405 (0.00%)	187 (0.00%)	193 (0.00%)	288 (0.00%)
NTP	608 (0.01%)	5200 (0.03%)	5050 (0.04%)	5250 (0.05%)
SRVLOC	6177 (0.07%)	6717 (0.04%)	10556 (0.07%)	6114 (0.06%)
ICMP	4620 (0.05%)	82944 (0.55%)	36000 (0.25%)	28430 (0.28%)
EIGRP	280532 (2.96%)	261475 (1.72%)	261672 (1.83%)	258756 (2.59%)
IGMP	571 (0.01%)	1804 (0.01%)	1401 (0.01%)	1045 (0.01%)
ARP	N/A	1779217 (11.72%)	1990922 (13.92%)	877582 (8.79%)
RARP	N/A	0 (0.00%)	0 (0.00%)	18 (0.00%)
IPX	N/A	32 (0.00%)	23 (0.00%)	11 (0.00%)
IPv6	N/A	172 (0.00%)	94 (0.00%)	194 (0.00%)
ETHER (old)	N/A	385789 (2.54%)	399206 (2.79%)	420344 (4.21%)
Total no. of Packets	9461744	15179630	14301220	9982829
Total Size in MB	6421	3415	3510	2089
Average Packet Size in B	711	236	257	219

Table 2: Protocol Classification (best viewed in color). The table shows combined content statistics for the network traces used. It presents a hierarchical breakdown of packet counts of various protocols using a traditional 5-tuple classifier (i.e. using primarily Ethernet type, IP protocol, and port information) that we wrote. Protocols with one color level constitute Ethernet protocols including IPv4. Protocols with two color levels are IP protocol types including TCP and UDP. Protocols with three color levels are specific TCP and UDP protocols.

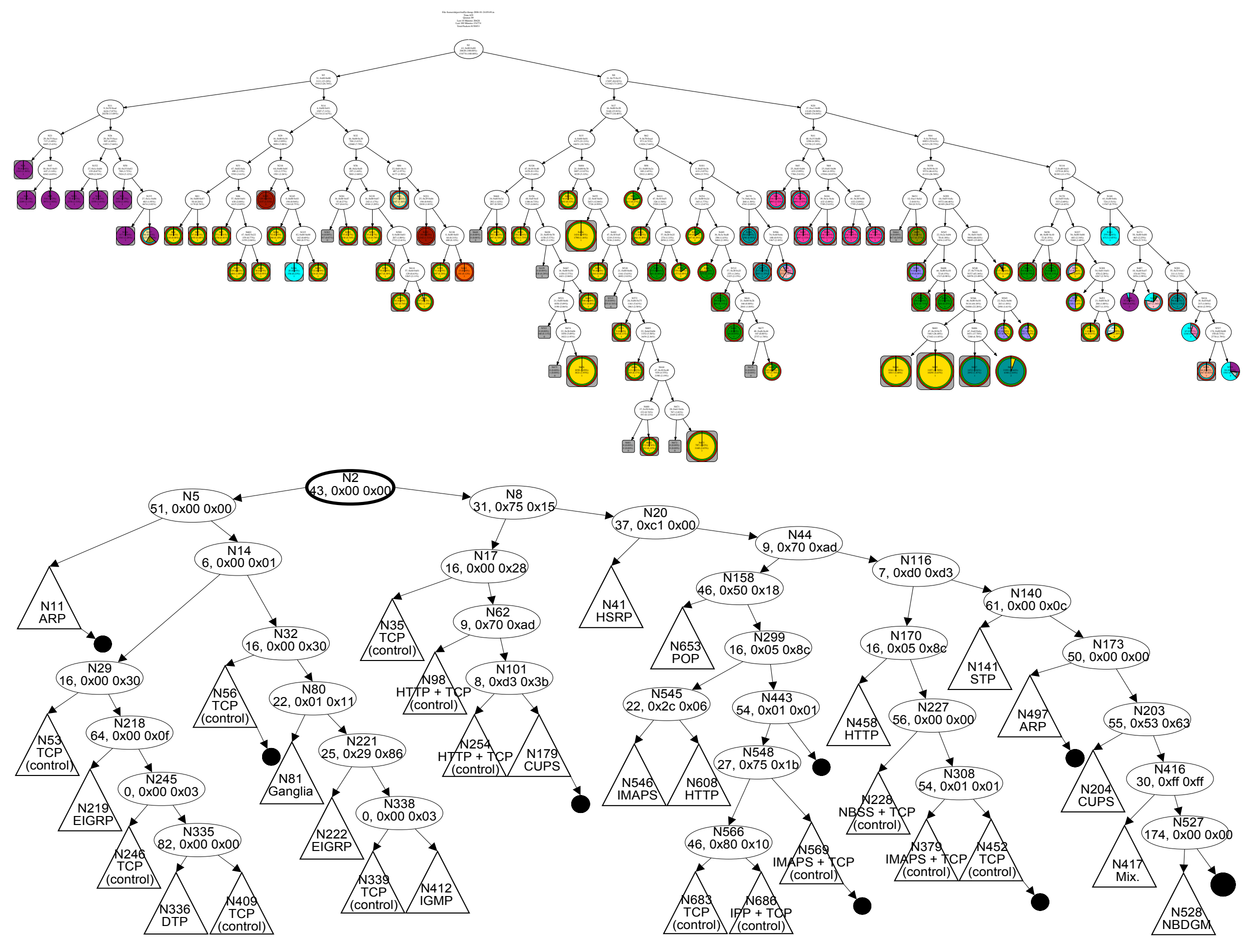

Figure 2: (Best viewed in color and electronically to allow enlargement) A decision tree produced by NetADHICT (top) and a simplified version (bottom), from a snapshot taken 9:49am Jan.24, 2006. The original tree contains 89 terminal clusters and 88 internal nodes. Terminal clusters are represented by pie charts (color key provided in Table 2). Singular clusters are presented on a gray box. In the simplified tree, ovals represent internal nodes, triangles represent subtrees, filled circles represent default clusters.

fault" shape. Default clusters are the rightmost child of a subtree. They are the product of several non-matching rules. Thus, packets in default clusters are "everything that is not something else".

The N14 subtree is a large mix of protocols. The leaf clusters demonstrate the effectiveness of the technique for non-TCP protocols, showing singular clusters for DTP (Dynamic Trunking Protocol—an older Ethernet protocol), EIGRP (Enhanced Interior Gateway Routing Protocol), IGMP (Internet Group Management Protocol), and Ganglia (a cluster monitoring application). All other leaf clusters (except for a default cluster) are singular and contain TCP control packets with zero-length payload pertaining to a web server traffic running over dozens of non-standard ports. Similarly, the subtree with root N17 shows that the algorithm was successful in segregating much of the HTTP traffic. In addition, this subtree perfectly separates CUPS (Common UNIX Printing System) packets in 2 singular clusters. The right most cluster in this subtree functions as a default cluster, and is evenly divided between low bandwidth UDP protocols and HTTP.

The N41 subtree entirely segregates the Cisco HSRP (Hot Standby Router Protocol) traffic. The subtree further divides HSRP evenly in 6 clusters according to their other special attributes. The packets in each HSRP cluster are exact copies, with each type of HSRP packet segregated to a different cluster.

The N158 subtree shows interesting clusters for POP (Post Office Protocol), IMAPS (secure Internet Message Access Protocol), HTTP, and IPP (Internet Printing Protocol). These protocols are what people use the most when they first arrive in the morning: they check email, websites, and print papers. All of the IPP data along with their related zero-length-payload TCP control packets are grouped together at the N566 subtree. On the other hand, IMAPS packets along with their related TCP control packets were automatically grouped together in two clusters, namely: N546, and N569. Moreover, two POP packets resided in one singular cluster at N653. Two default clusters appear in this subtree (i.e. right hand side of N443 and N569) but they are both dominated by TCP control packets and IMAPS and their related TCP control packets respectively.

The remaining right hand clusters contain six singular clusters of HTTP, TCP control packets, STP (Spanning Tree Protocol), IPP, and NBDGM (NetBIOS Datagram Service). Note that NBDGM packets did not match any of the top tree (p, n)-grams, but were finally grouped together at N528 separately from all other packets that went to the tree's global default cluster. The remaining three clusters are a mix of non-IP and UDP protocols such as ARP, and RIPv4 (Routing Information Protocol).

5.1 Special (p, n)-grams

It is worth noting that the tree of Figure 2 features an unusual root node (p, n)-gram because a significant portion of that traffic (about 35% by packet count) is HTTP traffic running on a non-standard port. The root node, (43, 0x00 0x00), was automatically chosen and matches padding on those non-standard HTTP packets that have no payload (mostly TCP control packets). We find that NetADHICT often uses features such as padding to word-lengths to cluster packets.

In addition, NetADHICT often uses zero-byte sequences within packet payloads to distinguish between different protocols. For example, most ARP packets in the N11 subtree were segregated through (51, 0x00 0x00), which is part of the Ethernet trailer. The other ARP packets end up in two far right clusters (nodes N497 and global default cluster). The logical link control Ethernet protocol DTP is partially segregated using zero-byte sequences. The (p, n)-gram (82, 0x00 0x00) groups all DTP packets with padding in their trailers at N336. Shorter DTP packets do not have this padding and are less well separated in other nodes.

Non-zero values are also discovered. STP packets, another Ethernet logical link control protocol (STP and DTP are both labeled as "Ethernet (old)" in the tree and in Table 2), are clustered in N141 by (61, 0x00 0x0c) which appears in their frame check sequence. Other STP packets, with a different frame check sequence are gathered in the tree's global default cluster.

5.2 Header vs. Payload (p, n)-grams

In some instances (p, n)-grams are discovered deep within the payload. An example of this is (174, 0x00 0x00) which uniquely identifies all NetBIOS-DGM packets and segregates them at N528 just prior to the global default cluster. This (p, n)-gram refers to the "reserved" and "parameter count" fields within the packet structure. A further example of this is the (301,0x00 0x00) (p, n)-gram, which effectively segregates 75% of RIPv1 traffic in the August tree (see Figure 5). This (p, n)-gram is part of the padding within the RIPv1 "IP address" field.

Indeed, it is worth mentioning the availability of header information substantially alters NetADHICT's clustering ability. EIGRP packets provide an example of how common patterns within both the header and payload can be used to distinguish. Which pattern is chosen is dependent on the (p, n)-gram frequencies of a particular trace. In Figure 2, EIGRP packets were all perfectly grouped at N219 and N222. At N14, the two sets get departed. The non-IP-header (p, n)-gram (64, 0x00 0x0f) perfectly segregates one EIGRP set at N219, and (25, 0x29 0x86) does likewise with the other set at N222. Interestingly, both sets share the (64, 0x00 0x0f) (p, n)-gram which represents the "hold time" parameter. They both also can be uniquely classified using the checksum and source IP address as is the case with the (25, 0x29 0x86) (p, n)-gram. However because NetADHICT is not biased in what part of the packet it examines, both header and payload (p, n)-grams can and are chosen.

HSRP packets demonstrate this as well. In the January tree, all HSRP packets are segregated by (37, 0xc1 0x00) which uniquely identifies the last byte of the destination port and first byte of the UDP length. In other trees, however, both (48, 0x35 0x00) from the payload and (5, 0x02 0x00) from the header are used. The payload (p, n)-gram represents the "hold time" and "priority" field while the header (p, n)-gram indicates the last and first bytes of the destination and source MAC addresses, respectively.

There are many instances similar to those of HSRP or EIGRP. From our experiments we are led to believe that NetADHICT can effectively segregate structured protocols like most non-IP, and UDP protocols. In some instances, however, NetADHICT does not choose any payload (p, n)-grams that are useful in segregating protocols, but uses only header (p, n)-grams. For example, all IPP packets are segregated by (27,0x75 0x1b), part of the source IP address.

Multimedia traffic like MS-Streaming and encrypted TCP traffic such as HTTPS, SSH, and IMAPS follow this pattern as well. They are either segregated by header (p, n)-grams or are shunted away toward the default subtree because they do not match (p, n)-grams near subtree roots. In the January tree (Figure 2), IMAPS packets were neatly separated from others through header (p, n)-grams like: (22, 0x2c 0x06) in N546 ("time-to-live" and "protocol ID"), and (54, 0x01 0x01) in N569

(NOP, NOP in "options").

It is interesting that NetADHICT is able not only to segregate encrypted packets from non-encrypted packets, but also to differentiate between these protocols in context. For example, in the August trace, HTTPS packets are separated from other encrypted traffic using a portion of the source MAC address. Moreover, SSH packets are separated by identifying a commonality in the "differentiated service" and "total length" fields. Finally, although the remaining SSH and IMAPS packets share a common path, they are separated by (54, 0x01 0x01) which is in the "options" field in SSH and the "content type" and "version" fields in IMAPS.

6 ADHIC versus the Traditional Classifier

The related work on traffic analysis (Section 2) and classification relies on prior knowledge of network protocols. In Section 5 we described a typical decision tree produced by NetADHICT and explained how our terminal clusters are represented by pie charts produced by such a classifier. Singular clusters are those clusters that the traditional classifier reports as containing packets of only one protocol. Those clusters are semantically meaningful. However, that does not mean that clusters that are not singular are not semantically meaningful. Protocol classification is simply one aspect of meaning that NetADHICT can discover.

Nevertheless, such a classification is useful. In this section we give quantitative results comparing NetADHICT's clusters with the traditional 5-tuple classifier. Such a comparison is the closest metric available translating to "semantically meaningful". Although it is not a perfect proxy, our results show that NetADHICT regularly finds and clusters together semantically meaningful packets.

Table 3 summarizes NetADHICT performance and reports the classification rate for application-layer protocols as well as four lower-level groups: TCP, UDP, non-TCP or UDP IP protocols, and non-IP protocols. For the lower-level classes, we define a packet as well segregated if all packets in cluster belong to one of those four lower-level classes. TCP and UDP packets are well segregated from other traffic. The non TCP and UDP packets are poorly segregated in two of the traces; they constitute only 2 to 3% of traffic volume and so are sometimes grouped with other protocols. The structural similarities of packets within each of the four lower-level groups are strong enough that they are easily recognized and separated.

The averages are lower for application-layer protocols. For the most part, the average number of packets clustered in singular clusters varies between 65% and 85% compared to the 80% and above for the lower-level classes. There are several reasons for this.

First, the traditional classifier is, on occasion, simply wrong. In several instances, particularly in the August trace, oddly configured application-layer protocols mixed with flows of the same protocol. For example, we found that one of our University's web servers served pages on port 8080 instead of 80. While NetADHICT clustered packets with this web data along with normal traffic, these clusters were not singular. This is a problem not just with our headers-based classifier, but with any headers-based classifier. Therefore, the application protocols column in Table 3 is a lower bound.

Sometimes the traditional classifier is simply not flexible enough. Our traditional classifier differs from other network header-based classifiers like WireShark [1] in that it treats TCP control packets as a separate category. We added that as a category because NetADHICT often segregates control packets, and control packets are a semantically meaningful group. However, sometimes NetADHICT will simply group all packets of a TCP flow together, lumping control packets in with data packets. The classifier does not classify these as singular clusters. This is another reason Table 3 under-reports the effectiveness of NetADHICT.

Finally, the adaptive nature of ADHIC is also partly responsible for the difference. ADHIC tries to contain network bursts of short duration as they appear. It does not segregate them unless the behavior persists for several update periods. Therefore, at the beginning of a spike, NetADHICT clusters these packets together in with existing clusters. This change in behavior is very clear in the decision-tree visualization. These clusters are often in use segregating other application-layer protocols. The spike is then classified as non-singular and lowers the "Overall %" because it is high volume. The median is more representative of effectiveness. Consider, for an example, Figure 3, which shows NetADHICT clustering performance on all four traces. The y-axis represents the percentage of packets that were clustered in singular clusters at each 10 minute update period. A spike occurs in the August trace at update period 550, which dramatically reduces the singular cluster packet average. When the traffic spike subsides, NetADHICT recovers. Therefore, this proper function of NetADHICT decreases the apparent performance when compared with the traditional classifier.

These three reasons explain the misleadingly low statistics in Table 3. While Table 3 and Figure 3 are presently the best metrics for evaluating ADHIC, they under represent its effectiveness because of the defects inherent in all traditional classifiers.

7 Clustering P2P traffic

In the past few years, high bandwidth p2p applications have started to disguise their traffic to avoid traffic shap-

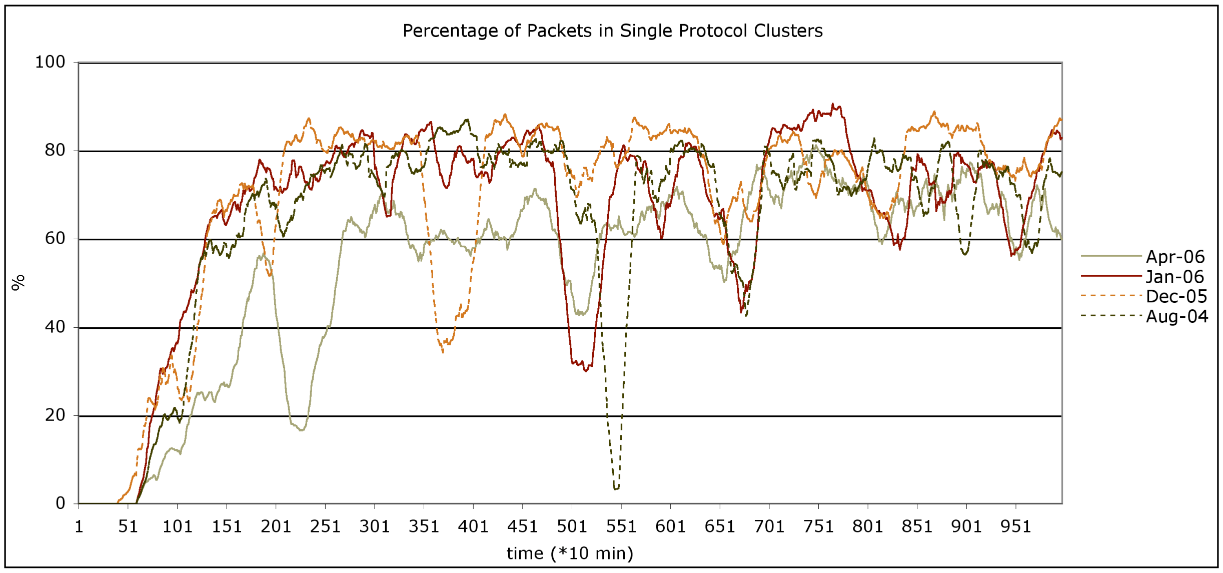

Figure 3: The percentage of packets in singular clusters at each update period for the four network traces.

ing mechanisms at the Internet service provider (ISP) level. Karagiannis et al.[6] showed that these applications may even deliberately use other protocols' port numbers to disguise their traffic. As a result, traffic shaping tools that use specific packet header fields to characterize network traffic will fail to distinguish, for example, between HTTP and p2p traffic that uses port 80. The popularity of these high bandwidth applications may have a great impact on the overall network performance if they cannot be discriminated from others.

We have performed several experiments with NetADHICT against p2p traffic and observed promising results. These experiments used the conventional Bittorrent [20] client software as well as some other p2p technologies, like Octoshape, to download relatively large files (over 500MB) to several of our lab machines. Linux binaries, free public compressed movies, and live video streaming are just examples of what the experiments included. While some p2p captured traffic featured unique source port numbers, many others were running on constantly changing port numbers. Traffic pertaining to these experiments was then individually merged with some of the four datasets tested earlier (see Section 4). In all the experiments, NetADHICT was able to segregate p2p traffic from all others, and cluster it in a small number of leaves.

Figure 4 shows an example of how Bittorrent traffic was clustered together using NetADHICT after it was merged with the January trace. In particular, one cluster (N499) managed to segregate most of the UDP tracker related data packets through (50, 0x00 0x00)—a (p, n)-gram that is not in the IP-header portion; all the other related TCP packets (whether data or control packets) got routed to the tree's global default cluster at N1033 and its adjacent cluster at N1032, as they did not match any of the (p, n)-grams higher in the tree. Due to the huge amount of p2p traffic, further splitting of the default cluster occurs later in the trace, but the Bittorrent traffic was always segregated on its own or in the global default cluster along with a few other unusual packets.

One question we had was whether Bittorrent would be clustered differently if it were run over a standard port. To test this, we changed the port number of all Bittorrent packets to 80 and re-ran our experiment. We found that each packet was clustered exactly as before. Such performance can be explained by two observations. One is that NetADHICT rarely uses ports to cluster traffic. But more significantly, NetADHICT was able to segregate the bulk of the Bittorrent traffic not by characterizing it directly, but by characterizing other network traffic as having patterns that were absent in the Bittorrent traffic. Thus, so long as most well-behaved traffic can be appropriately clustered, evasive protocols can be identified simply by their lack of structural resemblance to other traffic.

8 Clustering without Header Information

All previous network clustering algorithms that we are aware of (see Section 2) require header attributes to cluster traffic. Here we examine the question of whether it is possible to cluster traffic in semantically meaningful ways without relying on header information.

We configured NetADHICT to ignore the first 38 bytes of each packet. This excludes the 14 bytes of Eth-

Dataset	Clustering using **default** NetADHICT settings					Clustering using **no header** information				
Aug 13-19	Application Protocols	TCP	UDP	Other-IP	Non-IP	Application Protocols	TCP	UDP	Other-IP	Non-IP
Median	78.03%	89.76%	94.11%	46.10%	N/A	91.31%	97.49%	95.90%	92.20%	N/A
Std-Dev	18.74%	16.70%	13.71%	12.41%	N/A	20.77%	8.18%	9.41%	15.15%	N/A
Overall %	42.58%	70.03%	50.01%	42.18%	N/A	48.35%	96.29%	76.95%	89.02%	N/A
Dec 10-16	Application Protocols	TCP	UDP	Other-IP	Non-IP	Application Protocols	TCP	UDP	Other-IP	Non-IP
Median	81.17%	89.26%	91.88%	47.79%	79.87%	70.46%	91.72%	91.81%	97.38%	68.19%
Std-Dev	14.87%	13.61%	11.16%	15.37%	9.38%	17.05%	14.99%	11.63%	17.70%	10.82%
Overall %	74.34%	85.12%	85.36%	38.92%	80.55%	61.14%	87.19%	84.43%	83.83%	66.14%
Jan 20-26	Application Protocols	TCP	UDP	Other-IP	Non-IP	Application Protocols	TCP	UDP	Other-IP	Non-IP
Median	77.57%	93.48%	88.17%	95.94%	79.80%	67.40%	97.92%	89.06%	95.98%	68.08%
Std-Dev	16.43%	18.02%	10.84%	27.10%	8.52%	17.65%	15.58%	13.16%	11.76%	12.50%
Overall %	68.57%	87.10%	85.59%	78.94%	79.89%	61.91%	94.50%	85.75%	88.56%	68.31%
Apr 3-9	Application Protocols	TCP	UDP	Other-IP	Non-IP	Application Protocols	TCP	UDP	Other-IP	Non-IP
Median	64.08%	98.41%	94.53%	98.12%	69.95%	73.24%	97.49%	91.72%	98.85%	56.85%
Std-Dev	17.49%	13.15%	13.59%	24.09%	15.52%	18.40%	10.87%	11.67%	7.49%	21.65%
Overall %	55.08%	90.63%	84.36%	79.85%	65.91%	60.60%	93.80%	83.75%	95.51%	61.04%

Table 3: Percentage of packets statistics showing the correlation between NetADHICT clusters and the conventional view of the 5-tuple traditional classifier. The table's left side shows the experiments' results when all portions of the packet are considered during (p, n)-gram generation. The right hand side, however, shows the results when Ethernet header, IP header, and both source and destination port numbers are skipped during the (p, n)-grams generation. Overall % refers to the percentage of packets assigned to single protocol clusters over the entire trace. Median and standard deviation are computed using update period statistics. We exclude the first day of the trace as training.

ernet header, the IP header, and the port information. As a side effect, it also excludes payload information for packets that are not UDP or TCP. We then tested NetADHICT against the four traces again, collected statistics, and examined the decision tree at the same snapshot in time.

The right side of Table 3 shows the new statistics for the four network traces. Performance, as stated above, is mixed as compared to the whole packet experiments.

We expected NetADHICT would perform badly when not allowed to use most of the header information because it usually relies both on header and payload in building the decision tree. However, it pleasantly surprised us by generating trees that were qualitatively similar to the trees produced using all the packet information (see Figure 5). NetADHICT's ability to cluster is occasionally degraded, especially with TCP-based streams, but it still finds a large amount of structure within many protocols.

By comparing the results on the two sides of Table 3, one can see how NetADHICT sometimes performs better when no header information is given during (p, n)-gram generation. This is mainly due to the sequence in which NetADHICT chooses its (p, n)-grams. From a pool of many (p, n)-gram candidates, NetADHICT may pick a payload (p, n)-gram earlier in the tree that works better with packets seen later in the traffic, resulting in better trees and improved segregation results.

The largest difference is in NetADHICT's inability to segregate encrypted traffic when headers are restricted. In the August trace, all encrypted traffic is routed to a single cluster. This contrasts with the earlier experiments that allowed NetADHICT to examine header information, in which each encrypted protocol was routed to a distinct cluster. This is because, without header information, it becomes impossible to distinguish between types of encrypted packets—there is no structural information available.

Finally, we anticipate that the overall clustering percentage of NetADHICT will improve further as we run it against larger and more diverse network traffic. This is because when the network gets larger in terms of number of users and usages, spikes will have smaller impact on the total size of the traffic; and the different protocol volume percentages will have a better chance to stabilize over time.

Table 3 supports this argument. NetADHICT performs best on the trace with the largest number of packets (the December trace, with a median of 81.17%). The higher median reflecting NetADHICT's performance when we reduce the amount of header information does not contradict this result—the best performance in these experiments comes from the trace with the largest average payload size (August, with a median of 91.31%).

9 Discussion

Our goal in developing NetADHICT was to construct a tool for capturing the high-level semantic structure of network traffic without the use of domain specific information. Our results, particularly those in Section 6, suggest that the clusters discovered by NetADHICT have a close correlation with semantic classes of interest to network administrators, researchers, and security officers. Nevertheless, a significant amount of work still remains

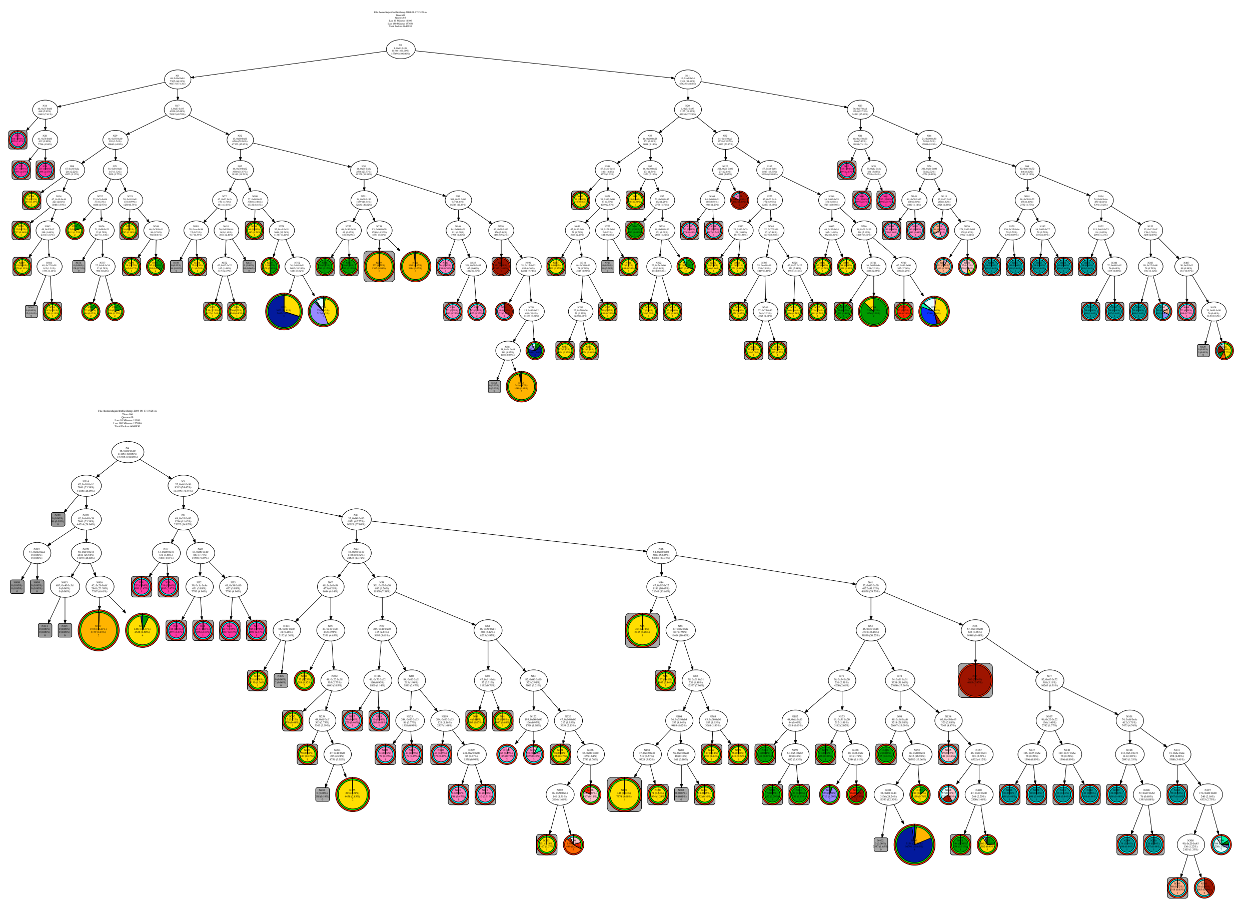

Figure 5: August tree snapshot (best viewed electronically) produced by NetADHICT using default parameters (top) and without looking at the packets' header portion (bottom).

before NetADHICT becomes a tool suitable for production use.

As a tool, NetADHICT requires significant development before it is robust and fast enough to run online on large networks. There also remain several open research questions pertaining to the ADHIC algorithm. ADHIC requires structure within packets to operate well. This is not the case in streaming protocols, payloads of encrypted traffic, or in protocols like Bittorrent that obfuscate their packets. We have shown that NetADHICT can often segregate these protocols, but this is done by recognizing all other protocols and assigning the remaining traffic, the encrypted, streamed, or obfuscated packets, to default clusters.

We do not know whether there is enough structure in these types of protocols for NetADHICT to cluster them appropriately in networks in which they represent a significant volume of traffic. We are not without hope; although a theoretical description of these protocols might preclude proper clustering, NetADHICT has shown that implementation details, such as padding, allows it to cluster traffic that would be impossible to cluster from payload alone.

Even though these three types of traffic represent a growing proportion of network traffic, much network traffic remains packet oriented and unencrypted, and will may well remain so. In our lab network, 40% of traffic consists of UDP and other structured protocols. Of the remaining 60%, much of it consists of HTTP and TCP control packets, which NetADHICT handles well.

Finally, throughout this paper we have discussed NetADHICT's ability to create "semantically meaningful" clusters. We know that NetADHICT does this because we have examined the packets in each cluster with our own classifier and tools such as WireShark [1]. We do not, however, have a metric that expresses how well NetADHICT does compared to other systems, or how well it does under different operator and parameter configurations. Section 6 compared the results of our traditional classifier with NetADHICT. Those statistics are correlated with NetADHICT's performance, but the traditional classifier also suffers from this lack of metrics—

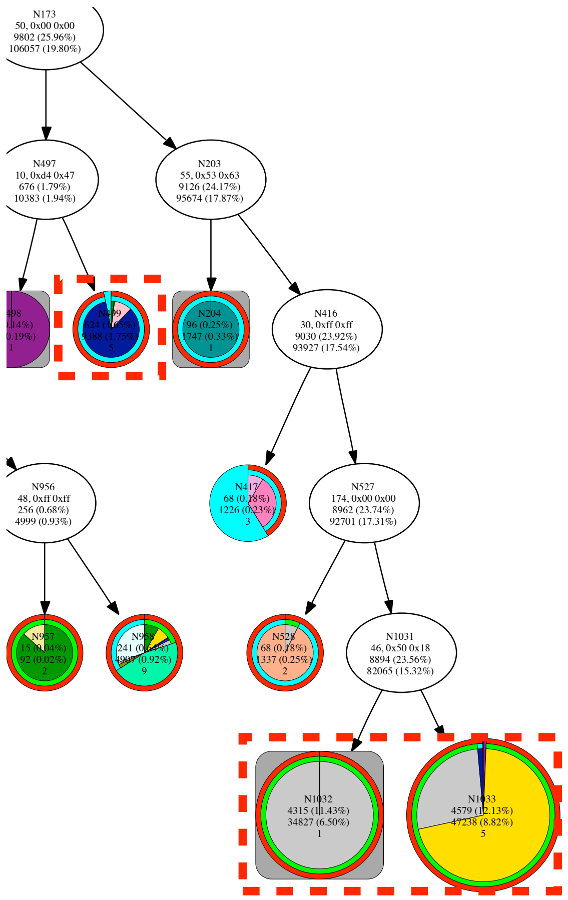

Figure 4: January tree snapshot with the presence of p2p traffic (best viewed electronically). All Bittorrent traffic went to the highlighted clusters: N499, N1032, and N1033. The dark blue cluster (N499) represents the Bittorrent's UDP packets, while the gray and yellow clusters (i.e. N1032 and N1033) represent the TCP data and TCP control packets respectively.

NetADHICT's classification statistics suffer *because* it does a better job of grouping than the 5-tuple classifier. Finding better quantitative measures for our results remains current and future work.

With these problems in mind, we plan to improve both ADHIC and NetADHICT. First, we want to examine several algorithmic variations. We may consider different features beyond fixed length (p, n)-grams. Permitting NetADHICT to consider longer (p, n)-grams will allow greater analysis of context in splitting. In addition, segregation in protocols that are less packet oriented may be increased by allowing a more fuzzy offset, allowing substrings to match in regions instead at a specific byte.

We may also introduce new operators to ADHIC, including the ability to rebalance the tree. Currently the beginning of training can create nodes with (p, n)-grams that are problematic later in the trace. We saw this in the comparisons of traces with and without headers in Section 8. In addition, we may revisit the conditions under which we split, including more representative entropy calculations for the potential subtrees.

We are also improving the tool. First, we are examining ways to improve its speed. Beyond optimizing the implementation, we can accelerate NetADHICT by sampling fewer packets. We know that NetADHICT can correctly determine (p, n)-grams for splitting using one half to one fifth the number of packets it currently examines (compared to the 20% it currently examines). We believe we can modify ADHICT to adapt its packet sampling measurements to optimize its throughput. That will allow us run with datasets from larger networks.

Beyond speed, we also want to improve NetADHICT's interface. It is currently difficult to use with online traces and impossible to reconfigure while in use. Although it contains a flexible configuration language, including the ability to manually specify decision trees, it is difficult for non-programmers to use. Further integration with other tools is also a priority. Beyond our own traditional classifier, we want to integrate NetADHICT with other tools. Currently NetADHICT forces us to save clusters as pcap files [23] and read them into other tools in order to examine them. The eventual goal of if for NetADHICT to allow one to segregate interesting parts of traffic and then examine them without this complicated work flow.

Ultimately, NetADHICT should be a tool that complements others in our network analysis toolbox. Inherently, there are limitations in any blind approach to understanding network behavior. However, we believe that such a generic approach is essential in order to understand the Internet as it continuously evolves. Knowledge-based approaches will always lag the latest applications or malicious software. A tool such as NetADHICT holds the promise of revealing new patterns of behavior before they become significant problems. We believe this is a rich area for future research.

10 Conclusion

In this paper we presented a technique, based on divisive hierarchical clustering, that can efficiently cluster network traffic into semantically meaningful groups without specific information of packet structure. We also described our implementation of this algorithm, a tool called NetADHICT. In experiments NetADHICT is able to segregate protocols into distinct clusters. It does this even with encrypted traffic or traffic that purposely disguises itself, as with the Bittorrent p2p protocol. It is even effective when forced to rely only on packet payloads. Although not yet robust or fast enough for production use, the approach demonstrated by NetADHICT is a promising technique for network traffic analysis.

11 Acknowledgments

This work was supported by the Discovery grant (AM,PVO,AS) and Canada Research Chair (PVO) programs of Canada's National Sciences and Engineering Research Council (NSERC) and MITACS.

References

[1] COMBS, G., ET AL. Wireshark. http://www.wireshark.org, 2007.

[2] DUDA, R. O., HART, P. E., AND STORK, D. G. *Pattern Classification*, 2 ed. Wiley, 2001, ch. Unsupervised Learning and Clustering, pp. 517–599.

[3] ERMAN, J., MAHANTI, A., AND ARLITT, M. Internet traffic identification using machine. In *Proceedings of IEEE GlobeCom* (2006).

[4] ESTAN, C., SAVAGE, S., AND VARGHESE, G. Automatically inferring patterns of resource consumption in network traffic. In *Proceedings of ACM SIGCOMM* (2003).

[5] FRANK, J. Artificial intelligence and intrusion detection: Current and future directions. In *Proceedings of the 17th National Computer Security Conference* (oct 1994).

[6] KARAGIANNIS, T., BROIDO, A., BROWNLEE, N., CLAFFY, K., AND FALOUTSOS, M. Is p2p dying or just hiding? In *Proceedings of IEEE GLOBECOM* (Dallas, Texas, November 2004), I. C. S. Press, Ed.

[7] KIM, H., AND KARP, B. Autograph: Toward Automated, Distributed Worm Signature Detection. In *Proceedings of the 13th USENIX Security Symposium* (August 2004).

[8] KOHLER, E., LI, J., PAXSON, V., AND SHENKER, S. On the structure of addresses in ip traffic. In *Proceedings of ACM Internet Measurement Workshop* (2002).

[9] KREIBICH, C., AND CROWCROFT, J. Honeycomb - Creating Intrusion Detection Signatures Using Honeypots. In *Proceedings of HOTNETS-II* (2003).

[10] LAN, K. C., AND HEIDEMANN, J. On the correlation of internet flow characteristics. Tech. Rep. ISI-TR-574, USC/Information Sciences Institute, July 2003.

[11] LELAND, W. E., TAQQ, M. S., WILLINGER, W., AND WILSON, D. V. On the self-similar nature of Ethernet traffic. In *Proceddings of ACM SIGCOMM* (San Francisco, California, 1993), D. P. Sidhu, Ed., pp. 183–193.

[12] MAHAJAN, R., BELLOVIN, S., FLOYD, S., IOANNIDIS, J., PAXSON, V., AND SHENKER, S. Controlling High Bandwidth Aggregates in the Network. *In ACM SIGCOMM Computer Communications Review* (July 2002).

[13] MATRAWY, A., VAN OORSCHOT, P., AND SOMAYAJI, A. Mitigating network denial-of-service through diversity-based traffic management. In *Applied Cryptography and Network Security (ACNS'05)* (2005), Springer Science+Business Media, pp. 104–121.

[14] MCGREGOR, A., HALL, M., LORIER, P., AND BRUNSKILL, J. Flow clustering using machine learning techniques. In *Proceedings of Passive and Active Measurement Workshop* (2004).

[15] PAXSON, V., AND FLOYD, S. Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Transactions on Networking 3*, 3 (1995), 226–244.

[16] POUWELSE, J. A., GARBACKI, P., EPEMA, D. H. J., AND SIPS, H. J. The Bittorrent P2P file-sharing system: Measurements and analysis. In *Proceedings of 4th Int'l Workshop on Peer-to-Peer Systems (IPTPS)* (Feb 2005).

[17] ROUGHAN, M., SEN, S., SPATSCHECK, O., AND DUFFIELD, N. Class-of-service mapping for qos: a statistical signature-based approach to ip traffic classification. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement* (2004), pp. 135 – 148.

[18] SINGH, S., ESTAN, C., VARGHESE, G., AND SAVAGE, S. The EarlyBird System for Real-time Detection of Unknown Worms. Technical report - cs2003-0761, UCSD, 2003.

[19] SINGH, S., ESTAN, C., VARGHESE, G., AND SAVAGE, S. Automated Worm Fingerprinting. In *Proceedings of 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI'04)* (December 2004).

[20] BITTORRENT.ORG. Bittorrent protocol specification. `http://www.bittorrent.org/protocol.html`.

[21] VAN OORSCHOT, P., ROBERT, J., AND MARTIN"', M. V. A Monitoring system for detecting repeated packets with applications to computer worms. *International Journal of Information Security* (February 2006), 186–199.

[22] WILLIAMS, N., ZANDER, S., AND ARMITAGE, G. A Preliminary Performance Comparison of

Five Machine Learning Algorithms for Practical IP Traffic Flow Classification. *ACM SIGCOMM Computer Communications Review* (October 2006).

[23] WORKERS, T. Tcpdump public repository. `http://www.tcpdump.org`.

[24] ZANDER, S., NGUYEN, T., AND ARMITAGE, G. Automated traffic classification and application identification using machine learning. In *Proceedings of IEEE LCN* (2005).

[25] ZIPF, G. K. *The Psychobiology of Language*. Houghton-Mifflin, 1935.