Discovering Packet Structure through Lightweight Hierarchical Clustering

Abdulrahman Hijazi Hajime Inoue Ashraf Matrawy P. C. van Oorschot Anil Somayaji
Carleton Computer Security Laboratory, Carleton University, Ottawa, ON - K1S 5B6, Canada
Email: {ahijazi, hinoue, amatrawy, paulv, soma}@ccsl.carleton.ca

***Abstract*— The complexity of current Internet applications makes the understanding of network traffic a challenging task. By providing larger-scale aggregates for analysis, unsupervised clustering approaches can greatly aid in the identification of new applications, attacks, and other changes in network usage patterns. In this paper we introduce ADHIC, a new algorithm that clusters similar network traffic together without prior knowledge of protocol structures. Packet similarity is determined through comparisons of substrings within packets at distinguishing offsets. ADHIC is notable in that 1) it produces a hierarchical decomposition of network traffic in the form of a cluster-identifying decision tree, 2) it needs only a small fraction of packets (about 3% in our traces) to generate the tree, and 3) it clusters packets at wire speeds. We find that ADHIC appropriately segregates well-known protocols, clusters together traffic of the same protocol running on multiple ports, and segregates traffic from applications, such as p2p, that do not use standard ports. Potential applications include network performance analysis, real-time alerts of flash crowds or worm activity, and dynamic DoS-resistant bandwidth management. NetADHICT, our implementation of ADHIC, is available for download and is licensed under the GNU GPL license.**

I. 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. Such analysis, however, for debugging, network management, and security purposes is increasingly important. Common analysis strategies have been to classify traffic using predetermined classifiers, based on ports and IP addresses or also using protocol dissectors [2]. 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.

The goal and strategy of this work has been to devise a technique to better understand network traffic through grouping them into semantically meaningful equivalence classes, or clusters, using no prior knowledge of packet or protocol structures. When packets are compared using header information, clustering can reveal many interesting patterns [5]; 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. Indeed, some protocols are now designed to make it difficult to classify them through header information. An ideal packet clustering technique, then, would be able to extract patterns from anywhere in a packet.

We have developed an unsupervised clustering algorithm, ADHIC (Approximate Divisive HIerarchical Clustering), that creates semantically interesting clusters without manually labelled training data or complex statistical features. Instead, it represents clusters using fixed-offset, fixed-length strings, a pattern we refer to as (p, n)-grams [16]. Most (p, n)-grams are rare and correspond to fragments of payload data. High frequency (p, n)-grams, however, correspond to the structural features of network packets. By choosing (p, n)-grams of appropriate frequency, we can represent the structure of network traffic without any other assumptions regarding the contents of packet headers or payloads.

More specifically, ADHIC clusters packets using an iteratively derived (p, n)-gram decision tree, where each (p, n)-gram is chosen on the basis of its frequency in observed traffic. Because the presence of (p, n)-grams can be efficiently measured, and because the frequencies of common (p, n)-gram can be estimated using small samples, ADHIC can continually monitor traffic *and* adjust its clustering performance online and at high speed.

To evaluate ADHIC, we implemented a packet analysis tool called NetADHICT [7] (pronounced "net addict"). In experiments using data captured from our lab's production network (see section V), we have found that ADHIC can quickly capture the overall structure of traffic in a way that reflects the relative popularity of different uses of network bandwidth. 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 a given environment. Because classification is often accomplished without direct reference to ports, the use of non-standard ports for protocols has little effect on clustering performance. Applications that do not have standard ports, such as the BitTorrent p2p file sharing protocol [23], [19], can also be clustered appropriately without requiring any protocol-specific information. Similarly, encrypted traffic is also often clustered appropriately because we are able to appropriately segregate all other traffic. These results show that ADHIC holds promise as a powerful yet lightweight method for discovering patterns in the structure of network traffic.

The rest of this paper proceeds as follows. We review related work in network traffic analysis in Section II and explain

the key design decisions underlying ADHIC in Section III. Section IV describes ADHIC, while Section V reviews our implementation, NetADHICT. Sections VI, VII, and VIII present more detail on ADHIC's performance through an explication of a few decision trees, overall classification behaviour, and performance in the face of p2p traffic. Section IX discusses limitations and future work, and we conclude with Section X.

II. Related Work

Many researchers have studied the problem of characterizing the structure of network traffic. Some, for example, have modeled the "burstiness" of packet inter-arrival times on Ethernet networks [13] and wide area networks [18]. Others have examined the structure of destination addresses in IP traffic [12]. Rather than manually choosing and fitting models to observed traffic, some have chosen to apply machine learning methods to automatically observe patterns in the structure of network traffic. For example, automated network traffic classification using machine learning strategies was first proposed in the context of anomaly detection for security purposes by Frank [6]. Many others have followed a similar strategy, giving rise to the field of anomaly intrusion detection. Much of the recent work in applying machine learning methods to network security, however, has focused on detecting the characteristic patterns of worm propagation [21], [22], [11].

To classify traffic into predetermined classes, supervised machine learning approaches such as Bayesian classification [24] and expectation maximization [17] have been applied to specific packet fields such as the 5-tuples, packet length, and inter-arrival times. While many network protocols can be identified by patterns in the size, directionality, and rate of packet exchanges [10], others have argued that it is necessary to look at some payload information in order to properly classify network traffic [14].

Rather than distinguishing between "good" and "bad" traffic or identifying members of predefined classes, our approach has been to cluster packets in an unsupervised fashion. Unsupervised clustering approaches have been applied to the problem of application identification using the the size and direction of the first few packets of a TCP connection [1]; it has also been used to aggregate packets into different quality of services classes [20]. While it might seem that unsupervised methods would always be at a disadvantage, Erman et al. [4] showed that this need not be the case in the domain of identifying applications using header data.

Perhaps the approach closest to our own in spirit is that of AutoFocus [5]. In this work hierarchical clusters were first formed on the basis of individual IP header fields; unidimensional clusters exceeding a fixed threshold were then aggregated to form multidimensional clusters. Rather than build up higher-level clusters using header information, ADHIC does top-down clustering using features that can be from anywhere in a packet.

One key motivation for identifying aggregates of network traffic has been to manage packet volumes. In particular, "Pushback"-type approaches have been developed to identify and throttle high volume flows [15] in order to mitigate denial-of-service attacks. Many problematic network conditions, such as distributed denial-of-service attacks (DDoS) and flash crowds, however, are properties of groups of flows, not individual ones. To address this problem, in earlier work [16] we proposed that network traffic could be managed using aggregates defined by (p, n)-grams. We presented a linear classifier that could balance traffic between multiple (arbitrary) queues and mitigate the spread of a simulated high-speed worm.

The key advance of this work is that we present and evaluate an algorithm, ADHIC, that uses (p, n)-grams to hierarchically cluster packets into semantically interesting (and generally easily recognizable) classes, and does so in an efficient manner in terms of both training time and online classification performance. While ADHIC already forms the basis of a promising network monitoring tool, NetADHICT [7], in addition we expect that ADHIC's clusters will be better suited to the problem of bandwidth management than the arbitrary collections of (p, n)-grams generated by the earlier algorithm [16]. Applying ADHIC to the network denial-of-service problem, however, is the subject of future work.

III. Capturing Packet Structure

Our goal with this work has been to devise a technique for clustering network packets into meaningful clusters without using any domain-specific knowledge, and to do so efficiently and online so that changes in network traffic may be tracked as they occur. To achieve this goal, we chose to create a divisive hierarchical clusterer. Divisive clusterers work in a top-down fashion: they assume all data first belong to one cluster, and then this cluster is iteratively divided into smaller clusters to capture finer-grained details. Divisive clustering was a good fit for our goals because we wanted to capture large scale patterns rather than the fine-grained details of network behavior. Further, we chose a hierarchical clustering approach because Internet traffic has an encapsulated structure, with HTTP encapsulated in a TCP session whose packets are encapsulated in IP and, typically, ethernet packets. Such a structure is best represented with a hierarchy rather than with a simple collection of clusters.

Existing approaches to divisive hierarchical clustering, however, were not suitable to our task because they typically employ an entropy minimization calculation which is $O(n^2)$ in the number of clustered items [3], something that is too expensive for a high-speed implementation. What we wanted instead was a "sub-linear" algorithm: it should only need to look at a small portion of a packet before deciding in what cluster it belongs to. High-speed routers face a similar problem: they must classify incoming packets quickly in order to decide where to send them. In general, they cannot perform calculations on the entire contents of a packet; instead, they look at a subset of bytes in a packet header (the destination IP address in particular) before making their choice. We thus decided to base our divisive hierarchical clusterer on a generalization of what high-speed routers are optimized to

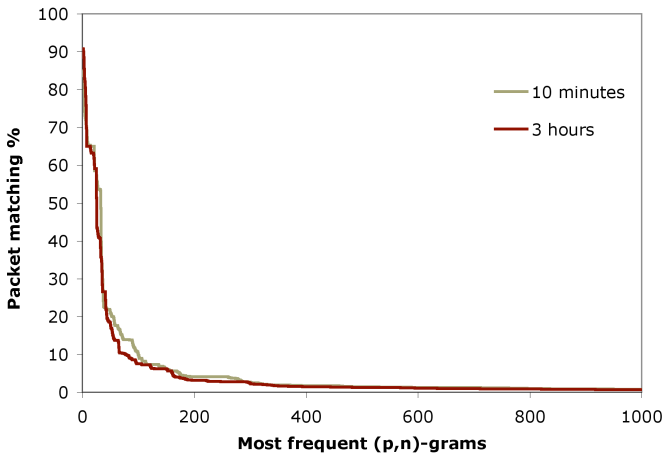

Fig. 1. (p, n)-gram frequency distribution in network traffic: percentage of packet matching of the most frequent (p,n)-grams taken from a sample 3-hour maturation window and a subset sample 10-minute update period in the January trace (See Section V).

observe: (p, n)-grams [16]. A (p, n)-gram is an n-byte string offset p-bytes from the start of a packet. For example, in an IP packet encapsulated inside of an ethernet packet, the destination IP address is 4-byte string offset 30 bytes. A wide variety of similarities in packet headers can be represented using one or more (p, n)-grams; further, payload similarities can also be represented so long as a given pattern is present at a consistent offset. In addition, packets containing identical data will share a variety of (p, n)-grams. Of course many patterns, such as the presence of a string anywhere in a packet, cannot be efficiently represented using (p, n)-grams. Our hypothesis, though, was that there was sufficient richness in the (p, n)-gram representation that we could still capture the high-level structure of network traffic.

A key problem that we faced was how to choose the right (p, n)-grams. If we need to perform complex packet analysis to find them, the performance advantages of (p, n)-grams would be lost. However, in our experiments we found that there are a significant number of high and moderate frequency (p, n)-grams (see Figure 1), the frequency of which appears to follow a power-law analogous to Zipf's law [25]. Further, these (p, n)-grams are most often structural ones: they match protocol or application-level headers (See Section VI). Thus, so long as we use relatively high frequency (p, n)-grams, we will be capturing packet structure—without any assumptions regarding packet structure.

We chose two bytes ($n = 2$) for the length of (p, n)-grams. Long (p, n)-grams provide a large amount of context, providing more semantically meaningful splits, but long (p, n)-grams are not found frequently. Shorter (p, n)-grams are easier to find in large quantities, but may not be as meaningful. We considered (p, n)-gram lengths of 1, 2, and 4. Only $n = 2$ created a proper number of (p, n)-grams—at $n = 1$ there were too many to consider efficiently while $n = 4$ provided too few.

```
for each node:
    if(node.traffic < minimum)
        parent.delete(node)
    else if(node.is_leaf && node.traffic > maximum)
    {
        png = find_png(node, cache)
        if(png) node.split(png)
    }
```

Fig. 2. Pseudocode for the ADHIC adjustment algorithm. `cache` holds a sample of recently observed packets.

IV. Approximate Divisive Hierarchical Clustering

We call our clustering algorithm (Approximate Divisive HIerarchical Clustering). ADHIC produces a binary decision tree that is used to assign packets to specific clusters. Using statistics gathered while classifying packets and small, periodic samples of full packets, ADHIC incrementally adjusts the tree to improve accuracy and track network behavior changes.

ADHIC's trees 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 with **and not**.

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*. A (p, n)-gram substring of length two ($n = 2$), consisting of the bytes 0x00 and 0x08 at offset 43 is denoted (43, 0x00 0x08).

The tree is generated and adapts through two operators: **split** and **delete**. It starts with one leaf cluster (thus one node), and then 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. This (p, n)-gram becomes associated with the internal node from which the new two leaf clusters branch. Operation statistics are measured over a period called the **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. See Figure 2 for a pseudocode representation of the ADHIC adjusting algorithm.

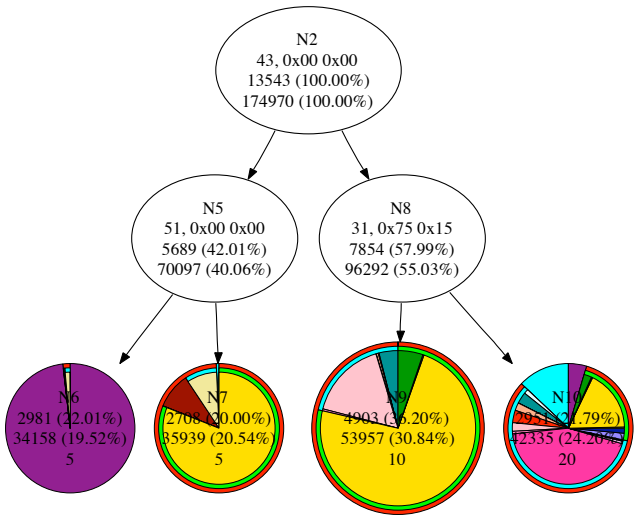

Fig. 3. An example decision tree (actual trees are much larger). Internal nodes are (p, n)-gram nodes. Each node in the tree has a node identifier (e.g. N8), and statistics of packet count and percentage seen by the node per update period in the upper line and per maturity window in the lower line (see Section V for more information about these parameters). While internal nodes contain also the (p, n)-gram value, leaves indicate 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. Slices with one color represent Ethernet protocols. Slices with two colors are IP protocol types including TCP and UDP. Slices with three colors are specific TCP and UDP protocols

For performance reasons, splitting and deletion only occur periodically. The duration between truee updates is called the **update period**. The similarity spread, maturation window, update period, and the thresholds for splitting and merging are all configurable parameters.

In addition, ADHIC does not calculate the frequency probabilities of all packets. If we accept errors, we need only sample a constant number of packets during each update period. For a 5% error rate, using the rule of thumb that the number of observations is $\frac{1}{B^2}$, only 400 packets need to be recorded. This is about 3% of all packets on our network.

Figure 3, a sample decision tree produced by our ADHIC 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. A packet is classified by starting at the root node, N2. If it contains (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).

It is important to note that the (p, n)-gram frequency distribution shows similar behaviour when taken on two samples: a 3-hour maturation window, and a subset 10-minute update period (see Figure 1).

V. Implementation and Experimental Setup

NetADHICT [7] is our implementation of ADHIC. It is licensed under the GNU GPL and available from the Carleton Computer Security Laboratory (CCSL) website [8].

Protocol	January 20-26, 2006	April 03-09, 2006
IPv4	11910975 (83.29%)	8684678 (87.00%)
TCP	7747114 (54.17%)	6107644 (61.18%)
IPP	568669 (3.98%)	486936 (4.88%)
SSH	497408 (3.48%)	245513 (2.46%)
...	...	...
UDP	3864788 (27.02%)	2288802 (22.93%)
CUPS	296348 (2.07%)	128278 (1.28%)
RTP	1587449 (11.10%)	248642 (2.49%)
...	...	...
ICMP	36000 (0.25%)	28430 (0.28%)
EIGRP	261672 (1.83%)	258756 (2.59%)
...	...	...
ARP	1990922 (13.92%)	877582 (8.79%)
ETHER (old)	399206 (2.79%)	420344 (4.21%)
...	...	...

TABLE I

An excerpt of content statistics for two of the analyzed network traces.

We tested ADHIC against four network traces from the CCSL lab taken during: Aug 13-19, 2004, Dec 10-16, 2005, Jan 20-26, 2006, and Apr 3-9, 2006. CCSL lab is a graduate student laboratory with over 15 machines, two network printers and about a dozen regular users. The lab provides web, webmail, IMAPS, DNS, SSH, SMTP, and CUPS services to external hosts. Because ADHIC relies on viewing packets in their entirety (non-anonymized full packets), the traces are from our own laboratory, where we have proper permission from the lab members. Furthermore, we have sufficient knowledge of our lab to better understand our traces.

In all our experiments, ADHIC effectively segregates all structured protocols. It begins, usually, by separating UDP traffic from TCP. Subsequently, in lower nodes of the tree, it sequesters specific protocols. It is important to note that ADHIC clusters use features that are often semantically meaningful and not protocol-specific. Therefore, 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 or ACK are often clustered separately.

Much of the structure ADHIC typically finds would also be found through traditional header analysis techniques. However, because ADHIC looks at traffic with no pre-existing biases, it also clusters using unconventional measures. (p, n)-grams corresponding to special-value padding, Ethernet frame addresses, checksums, and payload contents are all found in ADHIC decision trees.

Over the course of this research we inspected thousands of complete, individual packets while investigating the behavior of ADHIC and constructing an independent "reference classifier" we used to evaluate the system. This reference classifier is for the most part a traditional tuple-classifier, relying on IP protocol, and port information. Table I provides an excerpt of a sample list of the protocol traffic statistics for two of the traces calculated using the reference classifier.

All the ADHIC experiments were run with fixed parameter values (see Table II), determined by exploring several sets of alternative values using the four traces. In our testing

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

TABLE II
ADHIC PARAMETERS USED IN OUR EXPERIMENTS

environment, ADHIC 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 performance. On an Apple Mac Pro with 1GB of memory and 2.66 GHz "Woodcrest" cores, ADHIC is able to cluster packet data at about 250Mbps. Its current speed is not sufficient to monitor a high-speed link in real time; however it is more than sufficient for a research prototype. The lightweight nature of our algorithm should permit a high-speed implementation; such work, however, is a topic for future research.

VI. ADHIC Decision Tree

To illustrate the effectiveness of ADHIC, we next briefly describe an example decision tree and the types of clusters it produces. Figure 4 shows an annotated, symbolic version of a decision tree produced after four days of execution. The black circles in the graph which constitute the rightmost child of subtrees are called default clusters. They are the product of several non-matching rules. Thus, packets in default clusters are "everything that is not something else".

The first (starting from left to right) cluster (denoted by a triangle) at N11 primarily divides ARP (Address Resolution Protocol) from other traffic. The N14 subtree is a large mix of protocols, where the clusters demonstrate the effectiveness of the technique for non-TCP protocols, showing singular clusters (*singular* clusters are those that the reference classifier reports as exclusively clustering packets of only one protocol type) for DTP (Dynamic Trunking Protocol), EIGRP (Enhanced Interior Gateway Routing Protocol), IGMP (Internet Group Management Protocol), and Ganglia (a cluster monitoring application). All the other clusters 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 and separating CUPS (Common UNIX Printing System) packets in 2 singular clusters.

While the cluster at N41 entirely segregates the Cisco HSRP (Hot Standby Router Protocol) traffic, 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, POP packets resided in one singular cluster at N653.

The remaining right hand clusters contain six singular clusters of HTTP, TCP control packets, STP (Spanning Tree Protocol), IPP, and NBDGM (NetBIOS Datagram Service). The remaining three clusters are a mix of non-IP and UDP protocols such as ARP, and RIPv4 (Routing Information Protocol).

The availability of header information substantially alters ADHIC's clustering behavior. However, payload information is also an extremely useful source for ADHIC to choose (p, n)-grams from. Because ADHIC is not biased in what part of the packet it examines, both header and payload (p, n)-grams can be and are chosen. For example, in Figure 4, EIGRP packets were all perfectly grouped at N219 and N222. The non-IP-header (p, n)-gram (64, 0x00 0x0f) (representing the "hold time" parameter) perfectly segregates one EIGRP set at N219, and the header (p, n)-gram (25, 0x29 0x86) (within the source IP address) does likewise with the other set at N222.

It is important to realize that (p, n)-grams are sometimes discovered deep within the payload. An example of this is (174, 0x00 0x00) (which represents the "parameter count" field within NBDGM packets). This (p, n)-gram uniquely identifies and catches NBDGM packets at N528 just before they mix with the global default cluster.

Multimedia traffic like MS-Streaming and encrypted TCP traffic such as HTTPS, SSH, and IMAPS 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 Figure 4, 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 ADHIC 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, the (54, 0x01 0x01) (p, n)-gram ("options" field in SSH and "content type" and "version" fields in IMAPS) was chosen to separate the SSH traffic from the IMAPS one.

VII. ADHIC vs. the Reference Classifier

Figure 5 shows how the port-based reference classifier evaluates ADHIC clustering performance on all four traces. Our classifier classifies about 80 different protocols encompassing TCP, UDP, as well as other IP and Ethernet specific protocols. The y-axis represents the percentage of packets that were clustered in singular clusters at each 10-minute update period. Sometimes, the reference classifier is less accurate than ADHIC (consider for example traffic running on nonstandard ports), yet ADHIC clusters 60% to 80% of traffic in singular clusters.

Traffic in singular clusters is almost guaranteed to be related, and thus we use this as a performance metric. However, this

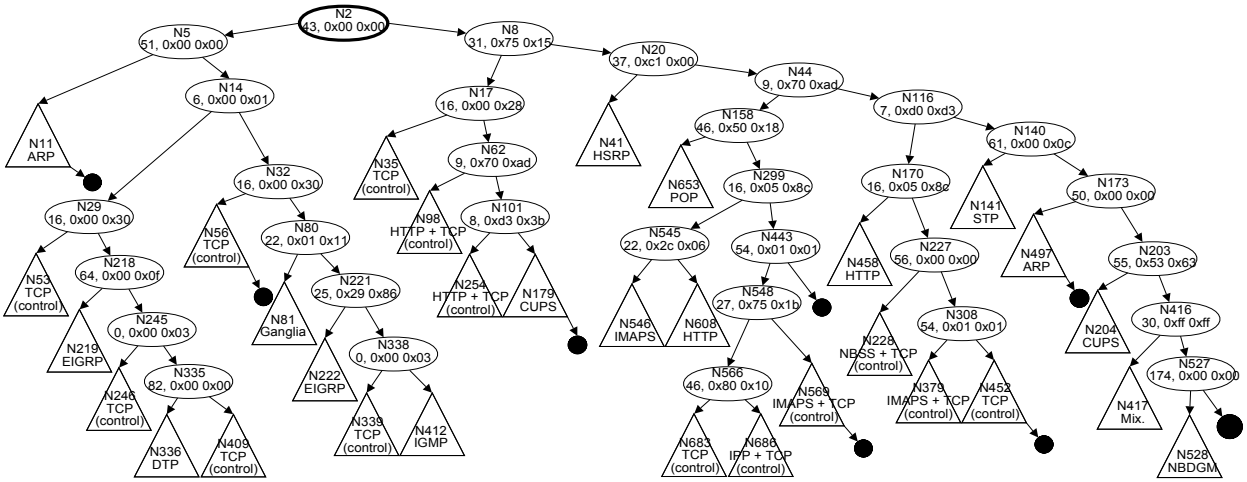

Fig. 4. Sample of an ADHIC annotated tree. This simplified decision tree was produced from a snapshot taken in January 24, 2006. The original tree contains 89 terminal clusters and 88 internal nodes. Here, ovals represent internal nodes, triangles represent subtrees, filled circles represent default clusters.

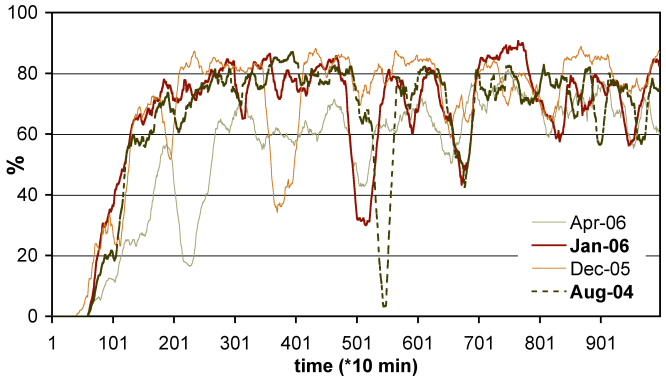

Fig. 5. The percentage of packets in singular clusters at each update period.

measure significantly understates the performance of ADHIC because it omits the cases when related packets have different port numbers. For example, consider the spike that occured in the August trace at update period 550. This spike was produced by a port scanning attack. Virtually all of the scanning packets were placed in the global default cluster; however, because they had widely varying port numbers, the scanning packets dramatically reduced the singular cluster packet average.

VIII. Clustering P2P traffic

In the past few years, p2p applications have started to disguise their traffic to avoid traffic shaping mechanisms at the Internet service provider (ISP) level. Karagiannis et al. [9] showed that these applications may even deliberately use other protocols' port numbers to disguise their traffic. As a result, port-based traffic shaping tools 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.

Our experiments with ADHIC against p2p traffic have demonstrated promising results. These experiments used the conventional BitTorrent [23] client software to download relatively large files (over 500MB). Examples of these files include: Linux binaries, free public compressed movies, and live video streaming. 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 V). Interestingly, in all the experiments, ADHIC was able to segregate p2p traffic from all others, and cluster it in a small number of leaves.

In particular, we saw that one cluster managed to segregate most of the UDP tracker related data packets through a non-IP-header (p, n)-gram; all the other related TCP packets got routed to the tree's global default cluster and its adjacent cluster, 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 occured later, but the BitTorrent traffic was always segregated on its own or in the global default cluster along with a few other unusual packets.

We also changed the port number of all BitTorrent packets to 80 and re-ran our experiment, and found that all packets were clustered exactly as before. This supports the observation that ADHIC rarely uses ports to cluster traffic. More significantly, it shows ADHIC 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 traffic can be appropriately clustered, evasive protocols can be identified simply by their lack of structural resemblance to other traffic.

IX. Discussion

A primary goal in developing ADHIC was to develop an algorithm for capturing the high-level semantic structure of network traffic without using domain-specific information. Our results suggest that the clusters discovered by ADHIC have a close correlation with semantic classes of interest to network

administrators, researchers, and security officers. We find these results both promising and remarkable: ADHIC is both very simple yet very effective. They also suggest many important avenues for future work.

For example, ADHIC inherentely requires structure within packets to operate well. We have shown that ADHIC can often segregate encrypted and obfuscated packets, but this is done by recognizing other structured protocols and then assigning the remaining traffic to default clusters. Will this behavior persist in larger, more complex environments? We suspect it will, so long as the distribution of (p, n)-grams continues to follow a similar power law as traffic in our lab: in this case, there will be plenty of high-frequency, structural (p, n)-grams for ADHIC to extract. While likely, such characteristics need to be verified through further experiments.

We would also like to develop a better measure of "semantically meaningful" clusters. To this point, we have verified the quality of ADHIC's clusters through the use of our reference classifier and standard network analysis tools such as WireShark [2]. ADHIC, however, finds significant patterns that these tools miss. We hope to develop additional measures, ones potentially based upon entropy minimization or other standard machine learning measures [3], that will "upper bound" the structure extraction ability of ADHIC.

We also plan to improve the runtime performance of ADHIC by testing various parameter settings and algorithmic variants. Further, we can accelerate ADHIC by sampling fewer packets. In the experiments reported here we sampled 20% of all packets because we were interested in minimizing error and maximizing repeatability. As we described in Section IV, on our network only 3% of packets are needed to achieve an error rate of less than 5% for (p, n)-gram frequencies.

Ultimately, ADHIC is an analyis technique that complements other analysis strategies. There are fundamental limitations to any approach to understanding network behavior that does not incorporate protocol-level knowledge. Knowledge-based approaches, however, will always lag the latest applications or malicious software. A generic approach such as ADHIC holds the promise of revealing new patterns of behavior before they become significant problems, as well as mitigating those problems when they do occur [16]. Thus, we believe further work on lightweight approaches to extracting patterns in network behavior is a rich area for future research.

X. Conclusion

ADHIC, a clustering algorithm based on divisive hierarchical clustering using (p, n)-grams, effectively and efficiently clusters network traffic without specific knowledge of protocol structures. ADHIC segregates most protocols into distinct clusters, even with encrypted traffic or traffic that purposely disguises itself, as with the BitTorrent p2p protocol. While further testing is needed, we believe the approach demonstrated by ADHIC is a promising one for analyzing and managing network behavior.

References

[1] L. Bernaille, R. Teixeira, and K. Salamatian. Early application identification. In *Proceedings of CONEXT*, 2006.

[2] Gerald Combs et al. Wireshark. http://www.wireshark.org, 2007.

[3] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*, chapter Unsupervised Learning and Clustering, pages 517–599. Wiley, 2 edition, 2001.

[4] J. Erman, A. Mahanti, and M. Arlitt. Internet traffic identification using machine learning. In *Proceedings of IEEE GlobeCom*, 2006.

[5] C. Estan, S. Savage, and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. In *Proceedings of ACM SIGCOMM*, 2003.

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

[7] H. Inoue, D. Jansens, A. Hijazi, and A. Somayaji. Netadhict: A tool for understanding network traffic. In *Proceedings of the 21st Large Installation System Administration Conference (LISA'07)*, Nov 2007.

[8] Hajime Inoue, Abdulrahman Hijazi, and Dana Jansens. Netadhict. `http://www.ccsl.carleton.ca/software`.

[9] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos. Is p2p dying or just hiding? In IEEE Computer Society Press, editor, *Proceedings of IEEE GLOBECOM*, Dallas, Texas, November 2004.

[10] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. Blinc: Multilevel traffic classification in the dark. In *Proceedings of ACM SIGMETRICS*, 2005.

[11] H. Kim and B. Karp. Autograph: Toward Automated, Distributed Worm Signature Detection. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[12] E. Kohler, J. Li, V. Paxson, and S. Shenker. On the structure of addresses in ip traffic. In *Proceedings of ACM Internet Measurement Workshop*, 2002.

[13] W. E. Leland, M. S. Taqq, W. Willinger, and D. V. Wilson. On the self-similar nature of Ethernet traffic. In Deepinder P. Sidhu, editor, *Proceddings of ACM SIGCOMM*, pages 183–193, San Francisco, California, 1993.

[14] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M. Voelker. Unexpected means of protocol inference. In *Proceedings of ACM IMC*, 2006.

[15] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling High Bandwidth Aggregates in the Network. *In ACM SIGCOMM Computer Communications Review*, July 2002.

[16] A. Matrawy, P.C. van Oorschot, and A. Somayaji. Mitigating network denial-of-service through diversity-based traffic management. In *Applied Cryptography and Network Security (ACNS'05)*, pages 104–121. Springer Science+Business Media, 2005.

[17] A. McGregor, M. Hall, P. Lorier, and J. Brunskill. Flow clustering using machine learning techniques. In *Proceedings of Passive and Active Measurement Workshop*, 2004.

[18] V. Paxson and S. Floyd. Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995.

[19] J. A. Pouwelse, P. Garbacki, D. H. J. Epema, and H. J. Sips. The Bittorrent P2P file-sharing system: Measurements and analysis. In *Proceedings of 4th Int'l Workshop on Peer-to-Peer Systems (IPTPS)*, Feb 2005.

[20] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield. 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*, pages 135 – 148, 2004.

[21] S. Singh, C. Estan, G. Varghese, and S. Savage. The EarlyBird System for Real-time Detection of Unknown Worms. Technical report - cs2003-0761, UCSD, 2003.

[22] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated Worm Fingerprinting. In *Proceedings of 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI'04)*, December 2004.

[23] `BitTorrent.org`. Bittorrent protocol specification. `http://www.bittorrent.org/protocol.html`.

[24] S. Zander, T.T.T. Nguyen, and G. Armitage. Automated traffic classification and application identification using machine learning. In *Proceedings of IEEE LCN*, 2005.

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