Graption: Automated Detection of P2P Applications using Traffic Dispersion Graphs (TDGs)

[UCR-CS-2008-06080. June 17, 2008]

Marios Iliofotou
UC Riverside
marios@cs.ucr.edu

Prashanth Pappu
Conviva, Inc.
prashanth@conviva.com

Michalis Faloutsos
UC Riverside
michalis@cs.ucr.edu

Michael Mitzenmacher
Harvard University
michaelm@harvard.edu

George Varghese
UC San Diego
varghese@cs.ucsd.edu

Hyunchul Kim
CAIDA & SNU
hkim@mmlab.snu.ac.kr

ABSTRACT

Monitoring network traffic and detecting emerging P2P applications is an increasingly challenging problem since new applications obfuscate their traffic. Despite recent efforts, the problem is not yet solved and network administrators are still looking for effective and deployable tools. In this paper, we address this problem using Traffic Dispersion Graphs (TDGs), a novel way to analyze traffic. Given a set of flows, a TDG is a graph with an edge between any two IP addresses that communicate. Thus TDGs capture network-wide interactions. We start by exploring the potential of TDGs for traffic monitoring by focusing on graph metrics instead of features of individual flows. We then use TDGs to develop an application classification tool dubbed Graption (*Grap*h-based P2P detec*tion*), which we target specifically for detecting P2P traffic. Graption begins by partitioning traffic flows into clusters based on flow-level features and without the need for application-specific knowledge. It then builds TDGs for these clusters, and uses graph metrics to identify clusters that correspond to P2P applications. Finally, we automatically extract a regular expression for a new P2P application, allowing the use of existing IDS devices and routers to block or rate-limit the detected traffic. We describe trace-driven experiments that show more than 90% precision and recall for P2P detection.

1. INTRODUCTION

Peer-to-peer traffic is here and is planning to stay: what can we do to detect it? We are witnessing a fascinating arms-race between network administrators and developers of applications which try to evade detection. Peer-to-Peer (P2P) protocols have fundamentally affected the business practices of multibillion dollar industries, such as the entertainment and telephone companies. At the same time, worms and viruses continue to be an expensive concern with an estimated cost of $17.5 billion in 2005 [44].

These facts have led to a proliferation of companies that help ISPs detect and potentially delay or block particular applications, especially P2P traffic. As expected, the developers of these new applications have responded by obfuscating their traffic using non-default port numbers.

In this paper we focus on the problem of detecting and classifying P2P applications. While our methods work well for scanning worms and viruses as well, in this paper we focus on the detection of P2P traffic since P2P detection is both a challenging open problem that is not as well studied as malware detection and important to network operators and the industry.

Currently, there still does not exist a reliable method for identifying P2P traffic, despite significant research effort [21, 22, 25] as we explain in Section 5. As a result, there is ongoing debate about how much P2P traffic exists on the network [21, 22]. The challenge is that P2P traffic actively tries to avoid detection by constantly changing aspects of its behavior, and it is sometimes difficult to distinguish P2P traffic from worms [49]. In addition, a practical classification approach should ideally operate with limited a priori knowledge and assumptions, limited manual configuration, and flexibility to adopt to mutating behaviors and emerging P2P protocols.

In addition, identifying P2P traffic is a real concern for content producers, network administrators, and ISP providers. This concern is based on several operational requirements, legal issues, and business practices. Beyond the unauthorized use of system resources, P2P systems have the potential to be used for malicious purposes. For example, P2P networks can be used to launch DDoS attacks [13].

To provide context for our work, we can group most current monitoring tools and application classification methods according to their level of observation as follows: (a) packet level, such as signature-based worm detection, [32, 49] (b) flow level, such as statistical approaches, [27, 55], and (c) host level, such as host-profiling approaches [24, 23]. Each type of approach has its pros and cons without a single method emerging as a clear winner, as we describe in Section 5.

By contrast, our work is based on the concept of network-

wide interaction graphs which form the natural next step in the progression of aggregation at the packet, flow, and host levels. More precisely, we study the "social" interaction of the network as a whole which leads to a directed graph; each node is an IP address, and each edge represents an interaction between two nodes. We use the term *Traffic Dispersion Graph* or *TDG* to refer to such a graph. We argue that there is a wealth of information embedded in a TDG, which the other classification methods do not capture. For example, the TDGs corresponding to P2P applications are connected and have topologies with high average degree.

The main contributions of this paper are:

a. Using TDGs to construct measurement tools: We highlight the power of TDGs for monitoring traffic and detecting new applications. Earlier work [14, 54] used TDGs in the context of security and focused on specific tree-like graphs formed by propagating malware. By contrast, we study TDGs in the context of application identification at large, which produces a richer family of graphs, and opens up new opportunities for visualizing and characterizing network traffic.

b. Identifying P2P traffic: We develop a new methodology based on TDGs for the automatic detection of P2P behavior. Our tool, Graption, first clusters flows, then builds TDGs for each cluster, and finally classifies each TDG using graph metrics. Several advantages of Graption are rooted in the use of TDGs. It is a *plug-and-play* system with a small number of intuitive parameters, and does not require user specified signatures or other hints. Graption makes detection harder to avoid because a misbehaving node can be detected by its connections to the "wrong crowd", even if the node masquerades its behavior at other levels.

c. Differentiating P2P from worms: Past work on detecting malware (e.g., [49]) has often found that P2P traffic is a common false positive when detecting worms. This is because the propagation structure of P2P traffic resembles that of a worm. We show, however, that P2P and worm traffic can be differentiated using TDGs and a few simple new graph measures.

d. Using Graption on backbone traces: Ideally, our method needs multiple points of observation to capture the TDG(s) of a network. We show, however, that TDGs contain sufficient information even when they are generated from a single observation point in the backbone. Backbone traces present several challenges to TDG methodologies: for instance, reverse traffic is often missing. Despite this, we show that Graption (which only requires a few parameters) operates with over 90% precision and recall on CAIDA backbone traces when measured against ground truth provided by signature-based P2P detectors (that require humans to input P2P signatures).

e. Automating P2P signature extraction: Besides detecting a previously-unknown P2P application, Graption also generates a payload signature for the P2P application. This signature can then be used by existing IDSs to drop or rate control the application. Past work in signature extraction ([49, 26]) was in the context of malware detection and used techniques based on finding frequently occurring strings that could occur *anywhere* in a packet. By contrast, Graption uses TDGs to decide when to extract a signature and can generate effective signatures using only the *first 16 bytes* of payload.

In our earlier work [19], we presented TDGs as: (a) a complementary information source for network traffic characterization/modeling and monitoring; and (b) showed that TDGs for a large set of popular network applications have regularity and structure that remain stable over time. In the current technical report, we greatly expand our previous efforts by: (a) using backbone traces with payload from CAIDA, (b) targeting P2P traffic detection, (c) automatic signature extraction for P2P applications, and (d) the development of Graption framework for traffic classification.

The rest of the paper is organized as follows. Section 2 defines TDGs formally and provides some basic examples. Section 3 starts by presenting visualizations of the TDGs of various applications. The section also introduces various TDG graph measures that can be used to differentiate applications. It finally focuses on the use of TDGs for P2P detection. Section 4 uses the observations of the previous sections to construct a system for automatically identifying and extracting a signature for new P2P applications from packet traces. Section 5 surveys related work.

2. TRAFFIC DISPERSION GRAPHS

Throughout this paper, we assume that packets can be grouped into flows using the standard five tuple <srcIP, srcPort, dstIP, dstPort, protocol>. We therefore use the terms "group of flows" and "group of packets" interchangeably. Given a group of flows S, we can define the corresponding TDG to be a directed graph $G(V, E)$, where the set of nodes V correspond to the set of IP addresses in S, and there is a link $(u, v) \in E$ from u to v if and only if some flow in S has a srcIP of u and a dstIP of v.

In this paper, we consider bidirectional flows, and hence direct edges more carefully. We define a TCP flow to start on the first packet with the `SYN` flag set (referred to as the `SYN`-packet), so that the initiator and the recipient of the flow are defined for the purposes of direction. In the case where we only observe the `SYN/ACK`-packet between nodes, the direction of the link is reversed and set to point from $dstIP$ to $srcIP$ (as directly derived from the `SYN/ACK` packet). For UDP flows, direction is decided upon the first packet of the *bidirectional* flow. If in some settings it is unclear which node initiates an interaction, then one could consider TDGs with undirected edges.

Given a set of collected network traffic over a fixed time interval T, there may be a large universe of flows, not all of which are relevant for a given application. An *edge filter* provides selection criteria to choose which flows from the universe belong to a group S defining a TDG. Any monitored

Name	*Date/Time*	*Duration*	*Unique IPs*	*Flows*	*Bytes*	*Packets*	*Mbps*	*Payload*
TR-OC48	2003-04-24/00:00	1 h	2,533,804	15,603,865	95 G	202.5 M	213.2	No
TR-ABIL	2004-06-01/17:30	1 h	6,346,572	23,846,500	714 G	823 M	1,726.2	No
TR-PAY1	2004-04-21/17:59	1 h	10,139,115	38,808,604	435 G	741M	1,068.8	16 bytes
TR-PAY2	2004-04-21/19:00	1 h	9,539,211	37,612,752	374 G	647M	980	16 bytes

Table 1: Set of backbone traces from, CAIDA (TR-OC48), the Abilene Backbone (TR-ABIL), and PAIX (TR-PAY1, TR-PAY2).

network could yield several TDGs, each corresponding to a different edge filter. Each edge filter allows the corresponding TDG to focus on a particular interaction and filter out other flows as noise. For example, we can place each UDP flow f_j in a group S_i, where i denotes the destination port of f_j. Henceforth, we refer to TDGs based on the destination port of a flow as *port-based TDGs*.

TDG edge filters can use either packet-level filters or flow-level filters. Packet level edge filters only use information in the current packet such as the port number. Flow-level edge filters require the monitor to maintain state about past packets to know when to add an edge in the TDG. An example is the filter where an edge is added between two nodes only if a flow with more than ten packets exists between them.

Finally, TDG edges can also be annotated to provide further information. For example, a weight could be assigned to each edge corresponding to the total number of flows or bytes sent between the two nodes.

Port-based TDGs offer a simple way to extract the TDG for a well-known application (e.g., DNS). However, port-based TDGs do not work for applications that use ephemeral port numbers. In order to study the TDG properties of P2P applications we used flows verified to be P2P by a signature-based payload classifier, as we describe in Section 3. By using the packet payload, we address the limitation of using port-based edge filters for the study of TDGs.

In the following section, we study TDGs corresponding to a variety of network applications, emphasizing P2P protocols. By profiling P2P traffic, we can extract the right set of features to use in Graption (Section 4) for their detection.

Name	*% in Flows*	*% in Bytes*	*% in Packets*
Gnutella	0.95(6.78)	0.17(1.59)	0.81(6.32)
eDonkey	2.96(21.16)	2.22(21.17)	2.84(22.21)
FastTrack	0.55(3.92)	0.74(7.10)	0.97(7.61)
Soribada	7.76(55.44)	0.07(0.63)	0.97(7.63)
MP2P	0.41(2.93)	0.01(0.14)	0.07(0.53)
BitTorrent	0.60(4.26)	4.59(43.81)	4.37(34.24)
All P2P	13.85	9.19	12.10
P2P Suspects	1.59	6.74	6.63
Web(80/443)	27.45	41.99	35.51
SMTP	1.65	1.10	2.431
DNS	6.65	0.32	1.586
Games	0.71	0.54	2.84
Unknown	0.97	1.98	3.22
No Payload	28.06	0.46	6.04
Rest	19.07	37.68	29.64

Table 2: Application breakdown for TR-PAY1. Values in parenthesis show the percentage of each P2P application over the entire P2P identified traffic.

3. TDGS AS A TOOL

In this section, we show that TDGs can provide powerful visualization, enable the introduction of new metrics and techniques for network traffic analysis, and provide new ways for identifying applications. We then specifically steer the discussion towards TDGs and metrics that can help us identify P2P traffic. Throughout this section (and in the rest of this paper) we show the applicability of TDGs using a set of backbone traces. We start by describing these traces.

3.1 Data Set

To study TDGs, we used a variety of backbone traces from three geographically different locations captured over different months, years, and times of day. Our traces are summarized in Table 1. In more detail, TR-OC48 is collected from an OC48 backbone link of a Tier-1 ISP [7]. TR-ABIL is from an OC-198c link from the Abilene backbone. Both traces are publicly available [7], IP anonymized, and do not contain payload. Traces TR-PAY1 and TR-PAY2, are collected from an OC48 link of a Tier-1 ISP at Palo Alto Internet eXchange (PAIX). These traces contain up to 16 bytes of payload from each packet thereby allowing the labeling of flows using signature matching techniques similar to the ones described in [21, 24, 47]. Our traces with payload information increase the confidence of our findings compared to using only port-based TDGs.

For the sanitized traces (TR-OC48 , TR-ABIL), we used port-based TDGs which work surprisingly well for legacy applications [25] such as DNS, Web, SMTP, NTP, etc. Moreover, using our payload classifier we observed that many P2P applications were still using their default ports during the period of when the traces were collected. P2P applications with default port numbers included eDonkey (on ports 4661,4662, 4665), Soribada (on port 22321), Gnutella (on port 6346), and MP2P (on port 41170). FastTrack (KaZaa) was one of the first P2P protocols to allow the user to change its port number, and thus the majority of its flows do not use the default port of the protocol. This observation also agrees with findings from [21, 47]. We used port-based TDGs (when applicable) to verify our real algorithm, and to support our claim that the behaviors of applications do not vary greatly at different monitoring points.

We note that if one is restricted to a single observation point, then backbones provide a good vantage point because of the aggregation at such a point. We also show that our results are not very sensitive to the choice of backbone point.

Ground Truth Payload Classifier (GTPC). We used a payload classifier to establish the ground truth of flows in our traces. GTPC is similar to the methods described in [21,

24, 47] where the payload of each packet is compared to a predefined set of signatures for P2P applications, DNS, Games, Chat, Web, Email, etc. All traffic with no match to a signature is labeled as unknown. GTPC further classifies a subset of the unknown flows as "P2P Suspects": these are flows whose source or destination node was found to have at least one P2P flow matching a payload signature.

Traffic Mix. The results from running GTPC over the payload traces are summarized in Table 2. While GTPC may miss some P2P traffic if the list of signatures is incomplete, past work has shown that the list is fairly accurate.

The breakdown by P2P application gives statistics for the top six well-known P2P applications, which corresponds to 95% of the P2P flows in our traces. These P2P applications are: Gnutella, FasTrack (also known by its well-known client KaZaa), Soribada, eDonkey, MP2P (Blubster), and BitTorrent. The remaining P2P protocols, including protocols such as Ares, WinMX, Goboogy, SoulSeek, PeerEnable, OpenNAP, Freechal, and others, are either not used currently or contribute a small fraction of the traffic. When we report results for all P2P applications, all the protocols above are considered, but we report specific results only for the top six P2P protocols described above.

3.2 TDGs as a visualization tool

An advantage of TDGs is their ability to provide visual insight into network-wide traffic. By contrast, previous visualization of traffic in monitoring tools has largely been limited to measures of traffic volumes on a per flow basis. The graphs in Figure 1 show visualizations of a sample set of TDGs corresponding to various network protocols. The TDGs from TR-PAY1 are verified using GTPC, and for the other traces we used port-based TDGs.

We highlight the following observations from the figure.
(i) Network protocols have different graph structures, which gives TDGs descriptive power in classifying applications.
(ii) P2P applications (as in Figure 1(c)(f)) appear to have distinctive structures, as compared to client-server activity (Figure 1(b)) and worm scanning activity (Figure 1(d),(e)).
(iii) The TDG structure of a P2P application are consistent in different settings, as in Figures 1(c)(f), which show the same P2P application over two different backbone links. We also observe a similarity in scanning activities from two different exploits at two different links in Figure 1(d)(e). We claim that port-based TDGs are an attractive solution for capturing the behavior of worms and other exploits since these applications often target a specific application operating at a fixed port number.
(iv) Visualization can pinpoint abnormalities. In Figure 1(a) we can see the effect of the appearance of abnormal scanning activity in SMB (Server Message Block/NetBIOS application). The resulting graph shows many disconnected star-like components similar to graphs from worm scanning activity, while also showing islands of legitimate activity with groups of connected nodes. This example demonstrates how the TDG visual profile of a standard application can be used to detect deviant behavior in the presence of anomaly. Similarly, if at some point the TDG for traffic at TCP Port 80 appears significantly different, it could be a new benign or malicious application tunneling its traffic under that port, or a change in the behavior of the traditional application.
(v) The right set of metrics is very important to achieve good discrimination between different behaviors. As previously explained, Figure 1(a) shows suspicious activity on the TDG corresponding to the SMB (NetBIOS) protocol. On the other hand, Figure 1(b) visualizes the behavior of a typical client-server application. Though the two TDGs look alike, we will show later that the two graphs can be distinguished using simple graph metrics, such as the directionality of edges.

Although the main goal in the paper is the design of automatic methods to identify P2P applications, we note that good visualization methods can often be a viable alternative to automated methods for network monitoring. Moreover, visualizations often pinpoint key features (e.g., the connectedness of a TDG), which can then be translated into quantitative measures (e.g., the size of the largest connected component).

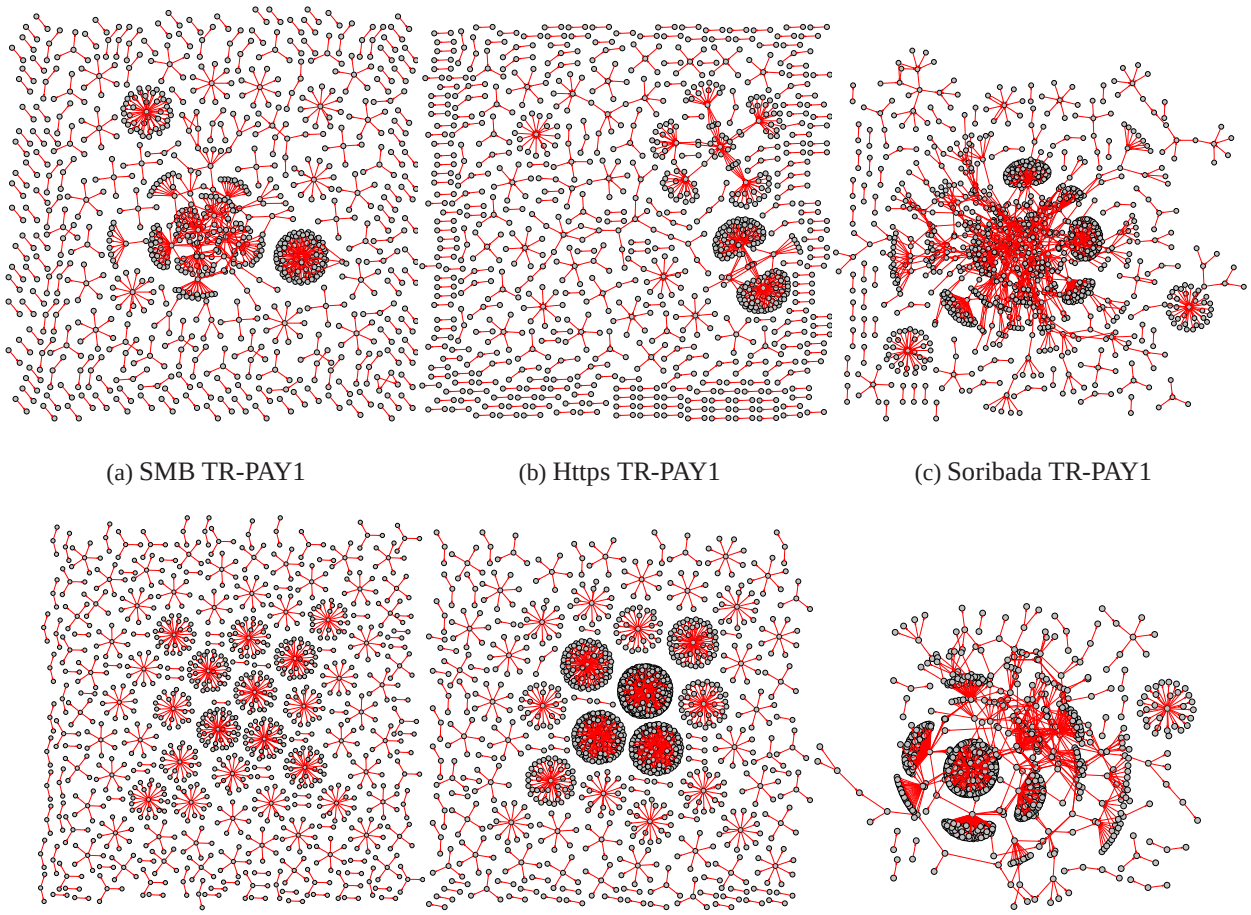
3.3 TDGs and graph metrics

Identifying the right metrics to compare graph structures is a challenging question that arises in many disciplines. The idea is to use several graph metrics, each capturing some graph characteristics, until a set of metrics is found that distinguishes the target graphs.

We now describe several graph metrics we have found useful. The list includes metrics, such as graph diameter and connected components, not previously used in traffic analysis studies. We impose the additional constraint in the final choice of our metrics for Graption that the chosen metrics should be intuitive, so that setting thresholds and interpreting the results is easy for network administrators.

Node degrees. We make use of the dK-series introduced in [34, 33] for modeling Internet topologies. Briefly, the $0K$ term is the average node degree, given by $2|E|/|V|$; the $1K$ term is the degree distribution; the $2K$ term is the joint-degree distribution (or JDD) and captures the probability that an edge exists between nodes of degree d_1, d_2; and so forth. We have found the JDD to be very useful in highlighting differences between various applications. For detecting P2P traffic, we found the (simpler and more intuitive) average degree measure to be sufficiently powerful, an observation supported by the P2P TDGs visualizations of Figure 1.

Directionality. The direction of edges (as defined in Section 2) provides significant information about the role of the node in its interactions with other nodes. For example, we know that pure clients only initiate traffic, pure servers should never initiate traffic, and that some P2P nodes play both roles. Thus we distinguish if a node is an initiator, a recipient, or both. To capture this quantitatively, we define



(a) SMB TR-PAY1 (b) Https TR-PAY1 (c) Soribada TR-PAY1

(d) MyDoom Worm TR-ABIL (e) Slammer worm TR-OC48 (f) Soribada TR-OC48

Figure 1: All TDGs visualizations are from graphs with $1,000$ nodes ($|V| = 1000$).

	Proto.	Avg_{deg}	Avg_{inDeg}	Avg_{outDeg}	V_{InO}(%)	V_{sink}(%)	BiDir(%)	$GWCC$	Dim. (Eff.)
1	Web	2.539	2.275	2.864	0.126	55.672	0.000	44.733	19(9)
2	FTP	1.687	1.823	1.144	35.552	16.721	22.998	30.357	3(2)
3	IRC	1.549	3.132	1.029	0.000	24.727	0.000	11.636	6(4)
4	POP3	1.550	2.543	1.114	0.073	30.404	0.000	8.367	14(10)
5	HTTPs	2.013	5.742	1.219	0.101	17.424	0.000	31.803	20(8)
6	SMB	1.842	1.126	5.041	0.070	81.733	0.000	24.731	14(10)
7	MSNP	2.887	4.501	2.124	0.000	32.065	0.000	93.094	11(6)
8	SpamAs.	3.015	21.778	1.620	0.000	6.923	0.000	98.846	8(6)
9	Stream	1.503	1.795	1.198	7.003	35.831	4.449	6.678	9(4)
10	SMTP	3.118	4.529	2.202	5.291	29.153	0.140	83.783	20(8)
11	DNS	3.719	2.353	8.106	5.364	76.297	6.450	44.554	16(10)
12	Game	7.420	9.349	4.992	16.938	23.769	5.032	96.480	14(7)
13	NTP	2.266	2.289	1.995	11.050	40.662	8.539	70.365	16(10)
14	BitTorrent	1.822	2.111	1.495	5.051	38.496	1.808	54.600	27(14)
15	Gnutella	3.872	4.145	3.259	7.022	40.086	1.708	92.232	20(8)
16	FastTrack	3.390	3.468	3.068	5.095	44.238	1.854	65.644	30(13)
17	MP2P	7.489	7.282	6.931	6.537	45.415	2.039	97.182	16(7)
18	eDonkey	3.704	4.550	2.621	13.207	28.170	3.243	93.953	23(8)
19	Soribada	24.440	54.659	13.285	15.337	7.215	1.730	99.831	10(5)
20	MyDoom	1.908	1.002	19.997	0.027	95.228	0.014	3.936	10(4)
21	Bobax	2.027	1.019	196.723	0.000	99.485	0.000	96.273	15(7)
22	Slammer	1.984	1.002	99.961	0.000	99.008	0.000	73.088	14(5)

Table 3: TDG metrics from TR-PAY1. Row 1-9: client server, 10-13: collaborative protocols, 14-19: P2P file sharing applications, 20-22: port-based TDGs of worm scanning activity (MyDoom TCP/3127, Bobax TCP/5000, Slammer UDP/1434).

InO to be the percentage of nodes that have both incoming and outgoing edges. Similarly, we can measure the percentage of nodes that are sinks (only incoming edges) and sources (only outgoing edges). This process partitions the nodes V in three disjoint sets V_{InO}, V_{snk}, and V_{src}.

We define two new metrics: the average in-degree for sinks ($Avg_{inDeg} = 2|E|/|V_{snk}|$), and average out-degree for sources ($Avg_{outDeg} = 2|E|/|V_{src}|$). In general by using directionality we can enrich information about degree distributions to capture correlations between nodes with different roles. This illustrates the potential of TDGs to suggest new metrics.

Connected Components. Unlike many other real-world complex networks, TDGs can be disconnected. To quantify this graph characteristic, we call the largest weakly connected component of a graph the giant weakly connected component (GWCC), and measure the size of the GWCC as a percentage of the total number of nodes in the graph. Here, we use connectivity in its weak sense, ignoring the direction of the edges. While metrics based on connected components can be captured by high-orders of dK-series [33], we find that treating them separately allows for an easier presentation.

Distance Metric. We find that the shortest distance (shortest-path) between all pair of nodes also provides an important distinguishing feature for BitTorrent. The diameter of a graph, defined as the maximum distance between two nodes, is sometimes sensitive as a metric [33] since the removal of a single link can potentially significantly change it. For a more robust metric, we use the *effective diameter*, which we define as the 95-th percentile of all pairwise distances in the graph. High effective diameters where observed in P2P applications such as BitTorrent and FastTrack.

We applied our collection of graph metrics to a variety of different TDGs, including TDGs based on port-based filtering and TDGs directly derived from the ground truth of flows using our GTPC. In Table 3, we summarize various measurements for TDGs from TR-PAY1 that were verified with our GTPC. Similar measurements where derived from TR-PAY2 but are omitted for brevity. Graphs were generated over 5 minute bins; the choice of the length of this interval is elaborated next in this section.

3.4 TDGs distinguish between applications

The goal in considering a variety of TDGs metrics is to enable application classification, including the detection of P2P traffic. Here we examine which metrics can help us separate applications, and aim to find a small number of intuitive and easy to compute metrics that can allow automation of the detection process.

3.4.1 Using TDGs for Profiling P2P

What are the characteristics we expect to see from a sampled network-wide view of a P2P overlay?

From Table 3, we can see that TDGs for many P2P applications have a relatively high average degree compared to TDGs of other protocols such as Streaming, FTP, SSH, etc. A natural explanation is that P2P applications require a high number of connections to efficiently perform tasks such as answering content queries and sharing files. However, even though average degree is a good metric, by itself it is not enough to clearly distinguish P2P activity. We can observe this from Table 3 where applications such as Spam Assassin (SpamAs.) also have high average degree. Later in the section we will show that by including other features, such as the effective diameter of the graph, we can better isolate P2P activity. Similarity between P2P and SMTP, DNS, and Games will be discussed later in more detail.

	Low InO	**High InO**
Low Avg Deg	IRC, POP3, SMB, MY-PC, HTTPS, HTTP, NetBIOS	BitTorrent (P2P), FTP, NTP, SNMP, News, Streaming,
High Avg Deg	Spam-Assassins, Chat (MSN)	eDonkey, SMTP, MP2P, FastTrack, Soribada, DNS, Gnutella , Half-Life

Table 4: Grouping various applications according to their average degree and InO features. High average degree is > 2.8 and high InO is $> 1\%$.

Moreover, all P2P applications have a relatively high percentage of hosts with dual-roles (high InO), acting both as clients and servers, as compared to most other applications. We note FTP also has a remarkably high InO of 35%. This is due to the nature of the protocol; for the FTP control channel, the client initiates a connection to the server, and for the file transfer itself the server initiates the connection to the client. Hence, besides a high InO, FTP also has a high fraction of bidirectional links (BiDir in Table 3). Moreover, besides FTP, bidirectional links mostly exist only between P2P TDGs and Streaming. The Real Time Stream Control (RTSP) protocol TDG show signs of this behavior[1]. As we can see, using only a single metric is not enough to perfectly isolate P2P behavior, but simple combinations of graph metrics appear to have high descriptive power.

The graph metrics of Table 3 could also be useful for detecting abnormal behavior. For example, if we see a large increase of hosts using port 80 having a high InO, then this might indicate a P2P application tunneling portions of its traffic under port 80.

Threshold Selection. In Table 4, we divide a large set of Internet applications into four different groups based on their InO and Average degree. For this separation we used our empirically derived threshold where an application is set to have high InO if it is larger than 1% and an application is set to have high average degree if it is larger than 2.8. These parameters were also verified using port-based TDGs from

[1] `http://www.ietf.org/rfc/rfc2326.txt`

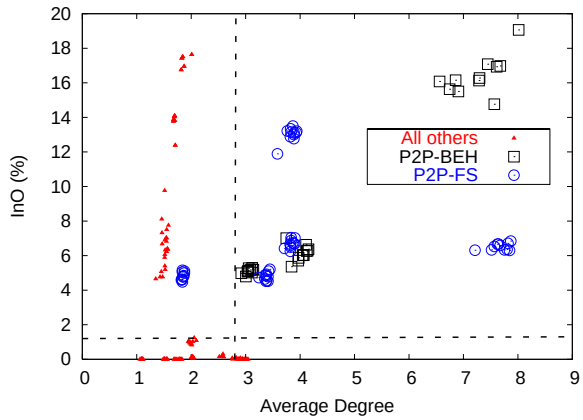


Figure 2: Average Degree and InO scatter plot for various TDGs. *Notation:* P2P-BEH includes DNS, SMTP,and Games; P2P-FS: includes all P2P file sharing applications.

traces TR-OC48 and TR-ABIL, but more detailed results are not shown here because of space limitations. This is exactly where the power of TDGs resides, in that a small number of intuitive parameters can be used to clearly separate different behaviors and applications.

Figure 2 better highlights our selection of metrics and thresholds. We show a scatter plot comparing the InO and average degree for various TDGs with the threshold values indicated using dotted lines. To make the plot easier to read, we removed FTP and Soribada that had much larger InO and average degree respectively.

In Figure 2, the majority of network applications, covering more than 70% of the traffic, are concentrated in areas with either low average degree or low InO, or both. On the other hand, P2P applications stand out from most other applications by their high average degree and InO. These features are, however, also shared by DNS, SMTP, and some on-line game applications [9] (e.g., Half-Life). Note that not all on-line games directly exchange packets between players. The typical game protocol architecture is based on the client-server paradigm, where all the clients directly communicate only with a server. However, since there might be many active servers for the same game, hosts usually contact more than one server when joining a game room. For the cases of DNS and SMTP, hosts in both applications can have dual roles with DNS servers querying each other for name resolution and SMTP servers directly connecting with each other for mail exchanges.

Since legacy applications will continue to use their default port numbers (namely port 53 for DNS and port 25 for SMTP), we can use their ports as a simple a heuristic, also used in prior works [22, 25], to separate these application from P2P traffic. As for game protocols, many network administrators tend to lump P2P file sharing and games into the same set of unwanted applications due to their high traffic requirements. If game applications must be distinguished, this can be done using either well known ports (if there is one) or flow level heuristics. In particular, even though both protocols make use of UDP [22], games have UDP flows with a large number of packets exchanged, in contrast to P2P where UDP is commonly used for messaging and not content sharing. We verified this finding in our measurements. Even though such flow-level heuristics can be used by Graption (Section 4), we only employed simple port-based filtering of legacy applications in order to use as few parameters in our system as possible.

BitTorrent Protocol (BT). The only P2P protocol with low average degree is BT. We attribute this behavior to the unique architecture of BT compared to the other P2P protocols. The main difference is that BT does not use its overlay mechanism for issuing queries directly among peers or for exploring the network, unlike the flooding mechanism that exists for example in Gnutella.

Although BT has a low average degree unlike other P2P protocols, a very distinctive characteristic is BT's high effective diameter. More than 5% of all randomly selected pairs of nodes in a BT TDG have shortest paths longer than 14 hops. This distinctive characteristic separates BT from similar applications which have effective diameters smaller than 5. Thus we set up the threshold for effective diameter at 11. Note that other protocols, such as the Network Time Protocol (NTP) which is not a P2P file sharing protocol, have also relatively high diameter. (This is not surprising as NTP internally uses P2P-like interactions). Since NTP always uses the default port 123, we used this feature to discriminate it from BT.

We now summarize some interesting observations from our measurements. P2P protocols such as Gnutella, eDonkey, and Soribada often have a few nodes with degrees above 500 (within a 5 minute window) which can perhaps be supernodes or eDonkey servers. The same behavior is unlikely in FasTrack which adopts a more distributed architecture [47]. These observations show the power of aggregating across hosts and indicates how we can potentially use TDGs to group the detected P2P protocols into categories according to their architecture. Also, in TDGs where most clients communicate with the same set of servers (e.g., MSN and Spam Assassins from Table 3), we can still have high average degree but we can always use directionality to discriminate this behavior from P2P applications where we have nodes with interchanged roles communicating with each other.

Separating P2P from Worms. From our trace-driven simulations, we found TDGs to be very effective in highlighting IP-range scanning activities without using an explicit per-host degree threshold, but by rather capturing the graph level view of this activity. In Figure 1 we visually show scanning activity by two different exploits captured from two different backbone locations. In the same plot we also show the TDG view of a P2P application (Firgure 1(c)), emphasizing the difference between the two.

Some real-time worm detection efforts observe a high rate

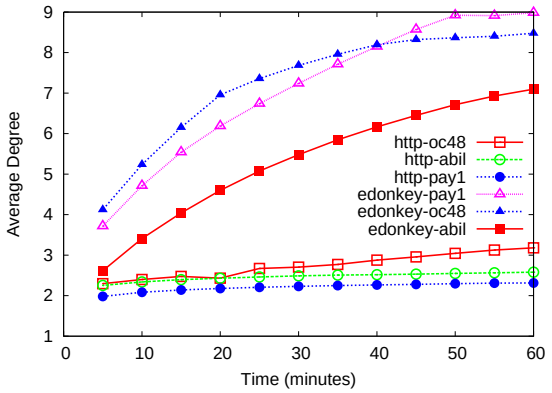


Figure 3: Evolution of average degree over time for the eDonkey P2P and HTTP protocol at three different backbone links (TR-OC48, TR-ABIL, TR-PAY1). The results illustrate the densification of P2P TDGs over time, and the small sensitivity to the choice of Backbone link.

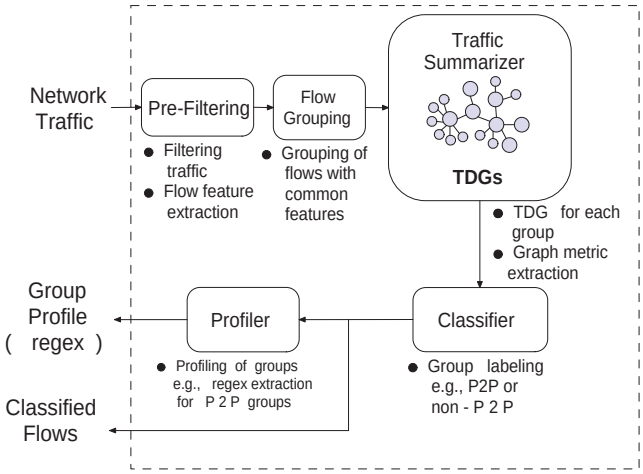


Figure 4: Methodology Overview for Graption.

of false positives from various P2P protocols such as BitTorrent [49]. Scanners over the internet have some distinctive characteristics: hosts with very high degree compared to the rest, and single packet flows, usually at the port of the exploited vulnerability. Similarly, peers in P2P overlays (e.g., Soribada) can potentially show common behavior. For example, in our study we found that for Soribada, most of its flows are single-packet, commonly unidirectional, and with super-peers having very large degrees (two orders of magnitude larger than the average host). Surprisingly, just by simply visually comparing the TDG view of the two we can easily distinguish between scanning and P2P activity (Figure 1).

To translate the visualization into a quantitative measure, from Table 3 we can see that worm activity has low Avg_{inDeg} (< 1.1) and higher $Avg_{outDeg}(> 10)$ within a 5 minute window. We verified this finding in all our traces, and it also agrees with observation in [55].

3.4.2 *Sensitivity in Time and Space*

With 5 minute intervals, we found stability in our suggested TDG measures over disjoint windows in our traces, as well as very good classification results (as we show in Section 4). Experimenting with larger intervals revealed a very interesting observation. Specifically, P2P TDGs grow more dense over time, with a significant increase in their average degree. To demonstrate this behavior, we show the behavior of eDonkey in Figure 3, calculated over 3 different backbone links, with window intervals ranging from 5 to 60 minutes. To contrast this with non P2P TDGs, in the same plot we show the behavior of the average degree of HTTP. Due to its unique architecture, BitTorrent is the only protocol that did not show this significant increase of average degree over time, while other protocols such as Soribada had an average degree of 24 with a 5 minute bin and an average degree of 61 with a 60 minute bin.

Similar observations hold for the InO metric, where for example with eDonkey in TR-PAY1, the InO of a TDG derived after 5 minutes is 13%, and grows to 20% in the TDG over 60 minutes. This indicates that the longer we monitor hosts, the larger a percentage we see acting as clients and servers. On the other hand, the HTTP TDG at the same trace had 0.04% InO with 5 minute bin and ended with 0.07% after 60 minutes of observation. Therefore, the difference between these two TDGs at the InO metric increased even more.

Even though increasing the time interval of observation increases the differences between P2P applications and other applications under our metrics, we chose to use a 5 minute bin in our experiments since it gave good classification results and stability, while keeping the computational cost lower and responsiveness faster.

It is also very interesting to observe that, in our experience, the TDGs of various applications do not appear to vary substantially across different observation points at the Internet Backbone. Edonkey is a typical example from our set of P2P applications.

Given that we have a collection of features for identifying TDGs corresponding to P2P traffic, the next goal is to find a way to group related flows together.

4. GRAPTION: GRAPH BASED P2P TRAFFIC DETECTION

Some important limitations of current application classification techniques include the need for extensive training, the need to tune various system parameters, and the difficulty of characterizing emerging applications. Currently, when a new application is discovered, it still requires manual payload analysis in order to identify application layer signatures that characterize the application. This procedure is even more difficult with P2P applications due to the proprietary nature of their protocols and the lack of sufficient documentation. Our methodology, Graption, can automate or otherwise simplify this process.

4.1 Methodology Overview

At a high level, Graption groups flows based on packet and flow features, builds TDGs for each group, and finally classifies each group as P2P or non-P2P TDG using graph metrics. Several advantages of Graption are rooted in the use of TDGs. It is a *plug-n-play* system with a small number of intuitive parameters, and does not require user specified signatures or other hints.

In more detail, Graption consists of the following 5 steps, as shown in Figure 4. Although this paper focuses on P2P detection, Graption can be used for general application classification by choosing metrics and parameters appropriately. Thus we start by describing the 5 steps as they would be used for general application classification, following which we specialize the description to detecting P2P traffic.

Step 1. Pre-filtering. The input is network traffic in the form of flows as defined in Section 2. The goal of this first optional step is to utilize external information to exclude any flows that can already be classified. This knowledge could be based on packet signatures, port numbers, or the IP address. In the last step (Step 5), we extract signatures from our P2P classified groups. Thus we can iterate on the set of input flows, and inspect all the traffic that was filtered at this stage.

Step 2. Flow Grouping. We use similarity at the flow and packet level to group flows. The definition of similarity is flexible in our proposed methodology. We can use flow statistics (duration, packet sizes, etc.) or payload if this is available. However, the output of this stage will be a set of groups that ideally contain flows that belong to a single application.

Step 3. Traffic Summarizer. For each group, we construct a TDG as defined in Section 2. Next we quantify each TDG using the metrics described in Section 3. Optionally, additional flow level statistics can be extracted to help in the next step (e.g., most common packet size, or dominant port number). The output is a set of TDGs with their metrics and features.

Step 4. Group Classifier. We use the TDG metrics to identify the application for each group of flows. For the classification, we use a set of rules which in general depends on the focus of the study. In this paper, we set Graption to classify TDGs as P2P or not-P2P.

Step 5. Application Profiler. In this optional step, the system uses the classification from the previous step to create profiles for each type of classified application. A profile for an application could be flow statistics, packet information, payload signatures or any other quantities that can be used to *directly* identify application flows in a packet trace.

We evaluate the efficacy of the Graption framework using standard techniques.

Classification Metrics: As with any classification method, evaluation starts by computing the True Positives, False Positives, and False Negatives. The True Positives (TP) measure how many instances of a given class are correctly classified; the False Positives (FP) measure how many instances of other classes are confused with a given class; finally, the False Negatives (FN) measure the number of misclassified instances of a class.

In our comparisons, we used the following standard metrics: **Precision (P)**, defined as $P = TP/(TP+FP)$; **Recall (R)**, defined as $R = TP/(TP + FN)$; and the **F-Measure**, defined as $F = 2P \cdot R/(P + R)$, combining Precision and Recall.

These 5 steps represent a general framework for application detection that can be specialized based on the target problem and the availability of information. We now describe the specialization to detecting P2P traffic assuming the availability of a few bytes of initial payload in the packet traces.

4.2 Applying Graption to P2P Detection

In this section, we describe how we specialized Graption with the goal of separating P2P from non-P2P traffic.

Step 1: Pre-filtering. Recent work [25] suggests that port-based classification works very well for legacy applications, as legacy applications use their default ports, and tunneling of P2P at such ports is not very common. Thus, we eliminate flows with ports 80 and 443 for Web, port 53 for DNS, and port 25 for SMTP. These applications turn out to be about 65% of the total number of flows. From our measurements, the portion of P2P actually using one of our excluded ports is as low as 0.1%. We note that we can use the signatures extracted at the last step to revisit Step 1 in a second iteration. This can help in capturing any P2P flows that hide under legacy ports, if this phenomenon becomes prevalent in the future.

Step 2: Flow Grouping. This step consists of three functions: (a) feature selection, (b) clustering, and (c) cluster merging.

a. Feature Selection. For efficient flow grouping, identifying features is critical, and especially challenging when applied to backbone traces, where flows do not always appear in both directions [17] given that Internet routing can be asymmetric.

For each flow, we extracted 60 flow features ranging from packet size information (size of first 10 packets, max/min packet size, etc.), timing information (flow duration, min and max inter-packet gap, etc.), TCP flags, total volumes in bytes, number of packets, etc. We also included the first 16 payload bytes of each flow using a methodology similar to [18, 32] for their extraction. All attributes were considered as numerical values with exception of the payload where each byte was considered as a categorical value from the set $\{0, 1, ..., 255\}$.

We used two techniques for finding the most relevant features: Information Gain Feature Selection and Correlation Based Feature Selection [53]. Both algorithms identified the first 16 bytes of the payload as the most important feature by a substantial margin. Intuitively, this is because we expect

the first 16 bytes to contain application protocol headers that repeat. Other prominent features include the size of the first 6 packets, and the maximum and minimum packet sizes.

Thus, in the experiments below we use the first 16 bytes as our feature for flow grouping. As we show, they are sufficient to give very good classification results. Note that even though we use payload, our grouping is agnostic to application semantics, as each byte is considered as a single independent categorical feature. In other words, we do not need a priori information about application signatures. Graption only relies on the fact that application headers are likely to recur; this similarity of payload can be used to group flows into clusters that belong to the same application.

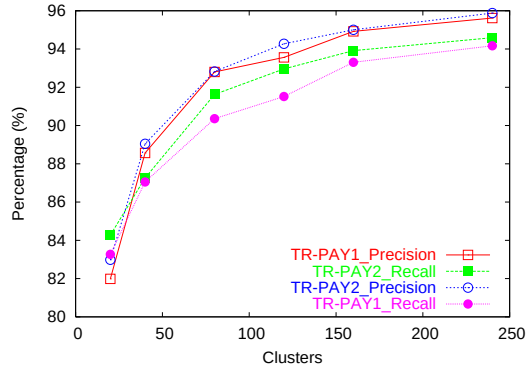
b. Forming Clusters. Given the set of discriminating features, the next step is to cluster "similar" flows. We use the term **cluster** to describe the outcome of an initial grouping using the selected features. Clusters may be merged in the next function of this step into **groups**, which produces the final output.

Feature-based clustering is a well-defined statistical data-mining problem. Graption uses the popular **k-means** algorithm [53]. This algorithm has been recently used for unsupervised clustering of network flows [15, 29], with very good results and low computational cost. The similarity between two flows is measured by the Hamming distance calculated over the 16 categorical features (i.e., the payload bytes). Even though more involved similarity measures exist such as edit-distance or TF-IDF, Hamming distance has been used successfully before [18] and performs very well in our application.

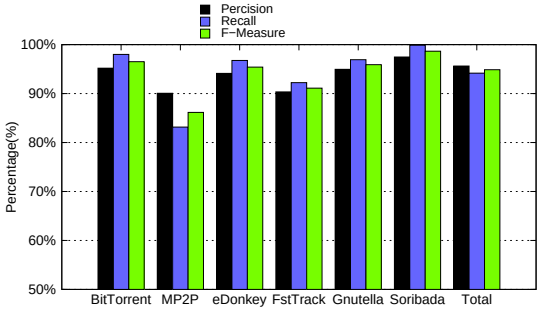
The k-means algorithm generates k clusters. However, the parameter k needs to be tuned to produce an appropriate number of clusters. Graption is not sensitive to the exact value for k, since we do not require that each application maps to exactly one cluster. Instead, we only need enough clusters so that flows from different applications do not share a cluster. Hence, it is reasonable to slightly overestimate the number of clusters.

For evaluating the quality of our generated clusters we used labeled flow instances extracted by the GTPC (used as an oracle). Since we have an a priory knowledge of the class of each instance, we will be labeling each cluster with its *dominant* application. For example, if in a cluster the majority of the flows are Gnutella, then the entire cluster will labeled as such and all the non Gnutella instances will be counted as false positives. Similarly, any instance of Gnutella that does not belong in one of its clusters is considered as false negative.

The effect of varying the number of clusters for k-means is shown at Figure 5(a). To find a reasonable range for k, we vary k from 20 to 240 over each 5 minute interval for both traces with payload (TR-PAY1, TR-PAY2). Figure 5(b) shows the recall and precision for six main P2P applications at $k = 160$. Our results suggest that with $k = 160$ we have very good results with above 90% total precision and



(a) Effect of changing k.



(b) Results per P2P protocol for $k = 160$.

Figure 5: Classification Precision and Recall for various k in k-means (TR-PAY1, TR-PAY2). In (a) we see the effect of increasing k from 20 to 240 for both traces. In (b) we show the precision, recall, and F-measure per P2P protocol from trace TR-PAY1 with $k = 160$. Similar results are derived from TR-PAY2.

recall over both traces. Larger values of k do not improve the cluster quality significantly. We also do not want to create too many clusters, as this would make further steps more difficult. (In the extreme case, each flow would be its own cluster!)

c. Cluster merging. Given the intuition that clusters correspond to common or similar application level headers, it is likely that the same application generates multiple clusters. For example, many P2P protocols exhibit a variety of interaction patterns, such as queries (often small UDP packets), control packets (small TCP or UDP packets), and file transfers (often large TCP packets), each with significantly different flow and packet characteristics [47, 21].

This motivates the further step of cluster merging, where we merge clusters into groups that, hopefully, belong to the same application. Merging clusters into groups provides a more complete view of the application, and gives more evidence to help understand the structure of the P2P protocol. Further, it is intuitive that the TDG formed by the merged clusters has more information than each TDG taken separately. For example, measures such as the node degree are

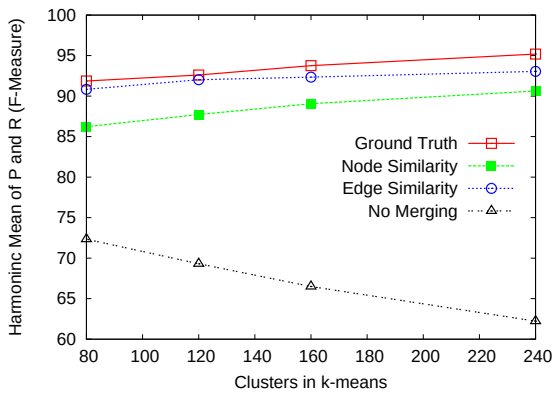


Figure 6: Effect of not using any cluster merging technique compared to using merging based on (a) hosts similarity and (b) edge similarity. The ground truth results are after labeling each cluster according to the application with the majority of flows in the cluster.

likely to be more accurate in the final group than in the individual clusters.

However, cluster merging cannot be based on the chosen set of flow-level features because these were already used to create the clusters in the first place. Instead, in the case of a P2P protocol, it is natural to assume that the TDGs corresponding to each cluster that the protocol generates would share a large number of nodes (i.e., IP addresses).

Based on these observations, we used an Agglomerative Clustering Algorithm[2] that merges clusters with significant overlap in either nodes or edges. The similarity is defined as the number of common nodes (or edges) between two clusters divided by the number of nodes (edges) of the smaller of the two clusters respectively. Two edges are said to be common if they share a common end host. Intuitively, edge similarity gives higher weight to high degree nodes, since if two clusters have a common high degree node then this will result in higher edge similarity.

The cluster merging process starts by hierarchically merging clusters with high similarity and stops when the similarity between all new cluster pairs is below a threshold. We experimented with various similarity threshold and observed that any value within the range 40 to 60% resulted in merging the majority of clusters belonging to the same P2P applications, without merging clusters that belong in different applications, using ground truth as a reference. For the rest of our experiments we fixed a similarity threshold of 50%. We found the best results using edge-based merging as we show below.

Step 3: Traffic Summarizer. The outcome of the previous step is a set of groups of flows, with each group consisting (hopefully) of flows of a single application. In order to classify each group, we generate a TDG on the group in the same way as described in Section 2. Each group yields a TDG that can be summarized using graph metrics. We find that using the the InO, the average degree, and the effective diameter provides sufficient information to classify the graph successfully as P2P with high precision and recall.

Step 4: Group Classifier Based on our previously extracted thresholds (Section 3), Graption classifies a TDG as P2P, with two rules. A TDG is classified as P2P, if either:

Rule 1: the average degree is greater than 2.8, and InO is larger than 1%; *Or*

Rule 2: InO is larger than 1% and the effective diameter is greater than 11.

Rule 1 covers most of P2P protocols except BitTorrent, which is covered by Rule 2.

Our results show that using these simple metrics we can classify P2P flows with 91% percision and 89% recall for TR-PAY1 and 90% recall and 95% accuracy for TR-PAY2.

To evaluate the sensitivity to our parameters, we show in Figure 6 the F-Measure over different values of k for the k-means algorithm using Rule 1 and Rule 2. The plot compares using no cluster merging (△) against merging based on node similarity (■) or edge similarity (○) with a similarity threshold of 50%.

We also show the results derived by labeling each cluster using the ground truth (□) without any cluster merging and by labeling each cluster based on the dominant application it contained (as given by the GTPC). All values are averaged over all the disjoint 5 minute intervals of TR-PAY1, TR-PAY2. As we can see from Figure 6, by using the thresholds from Section 3 and edge similarity to merge clusters, we can achieve both high precision (>90%) and recall (>90%) over a range of values of k. Thus cluster merging appears crucial to providing good classification.

Step 5: Signature Extraction. In this final step, we take the classification from the previous step as a given, and use it to create profiles for each group labeled as P2P. Each profile will contain a set of payload signatures in the general form of a regular expression. The goal of the extraction process is to generate a signature that will minimize false negatives (high recall), while at the same time minimizing false positives (high precision).

This problem was previously addressed in the context of automatically extracting signatures for network worms [49, 26]. Most current pattern based generating systems are based on capturing repeated substrings across all payload instances of a worm. However, in the case of P2P application, a single application can have a variety of different protocol level operations that result in different payload signatures. In addition, the labeled clusters might contain false positives from other applications and therefore there might not be a single signature to describe the entire set. This is similar to the problem of detecting signatures for polymorphic worms which was addressed by the Polygraph [42] system. We modified the mechanisms of Polygraph to extract P2P signatures.

Polygraph operates on two sets of flows: a set of innocu-

[2]Agglomerative clustering algorithms are also referred to as Hierarchical Clustering algorithms.

Application	Precision	Recall	Signatures Extracted by Graption
KaZaa	95.1%	78.1%	$\xc0.*\x28, $?xml version=”1, $\x00\x00\x00.*KaZaA, $\x00\x00\x00\xA9/\x00-, $GET /.hash=, $\xC1.*\x18
Gnutella	98.6%	97.3%	$GND, $GNUTELLA.*\x00, $GET \x2Furi-res\x2FN2R, $0\x82.*\x01\x01, $\x02\x01\x01\x04\x08cMm4-b4
eDonkey	97.2%	92.0%	$\xe3\x0f, $\xe3\x0e, $\xe3.*\x00\x00\x00\x01\x10, $\xe3.*\x14, $\xe3!
Soribada	98.1%	99.9%	$Q:.*\x2B, $\x10\x0b, $\x10\x0c, $\x10\x16
MP2P	89.7%	78.7%	$\x00\x00\x00.*\x89\x84, $\x00\x00\x00.*\x86y, $\x00\x00\x00\x01.*\x18, $\x00\x00.*\x18\xf4, $\x00\x00\x00.*\x81\xad
BitTorrent	99.9%	96.6%	$\x74BitTorrent\x00prot
Total	96%	90.5%	

Table 5: Signature matching results for both traces TR-PAY1 and TR-PAY2. Results are shown for each of the top six P2P applications and the Total P2P precision/recall over both traces. The "\x" notation indicates hexadecimal character. Each signature starts with the character $. Due to space limitations only a subset of the produces signatures are shown for some of the protocols.

ous flows and a set of flows for which a signature is to be extracted. For the so-called innocuous set, we chose to use legitimate flows from HTTP, DNS, and SMTP that are used as a benchmark to keep the false positive rate low. For the second set, we used the flows in any group labeled as P2P from Step 4. We will refer to this second set as the training set of flows. The algorithms in Polygraph use tokens of byte sequence found to be common in many flows of the training pool. The token extraction is based on the longest common substring and string alignment problems as described in [42].

Graption generates signatures in the form of regular expressions suitable for existing IDSes. To do so, we adopted the token-subsequence algorithm in [42]. The algorithm generates ordered token-subsequences that match the majority of flows in the training set while keeping the false positive rate low when applied to the innocuous set. The final output from polygraph is a set of signatures matching the flows from the input training set.

The differences of our work from that of Polygraph are as follows. First, Polygraph aims to extract signatures for polymorphic worms while we focus on P2P detection. Second, Polygraph assumes that the innocuous set and the training set are provided to the system by an oracle. In our case, this is done automatically by Step 4.

Experimental Setup: For training the algorithms we use a set of innocuous flows which comprise of a large set of 300,000 HTTP, 100,000 SMTP, and 200,000 DNS flows. We assert that it will always be possible to extract such training data since these applications are well documented, and innocuous flows can be extracted by logging the legitimate traffic to and from well known Web, DNS, and SMTP servers.

In our experiments, we used port numbers of HTTP, SMTP, and DNS to select our set of innocuous flows from the first 5 minute of the TR-PAY1 trace. For the training set we used signatures of a single cluster at a time. The size of these clusters ranged from a few thousands samples to tens of thousands of samples. Depending on the desired aggressiveness in higher recall or higher precision, Polygraph can be set to extract signatures that match at least $x\%$ of flows in the training set while keeping the FP from the innocuous set lower than $y\%$. For our experiments we used Polygraph with $x = 1\%$ and $y = 0.05\%$.

For evaluating the extracted signatures, we first used Polygraph on the first 5 minute interval of the trace. Next, we evaluated the extracted Regular Expression over the entire TR-PAY1 and TR-PAY2 traces. The set of extracted signatures for the six most popular P2P protocols matched over the traces are summarized in Table 5. Using these traces, our signatures were tested on more than 30 million flows by only inspecting the first 16-bytes of each flow. The overall P2P results on traces TR-PAY1 and TR-PAY2 exhibit 96% accuracy and 90% recall. The number of extracted signatures ranges from one in BitTorrent, to 30 for MP2P. The large number of signatures is to be expected since some P2P applications exhibit a high diversity of application level interaction. This diversity, especially at the UDP level (used extensively by MP2P), can result in a large number of distinct signatures.

We found the 78% recall for some applications (Fastrack and MP2P) to be due to flows corresponding to few rarely used signatures that did not form a large enough cluster on their own, and hence did not result in groups that were detected as P2P in Step 4. In addition, some signatures could not be derived while keeping the false positives at a reasonable rate.

In one of our applications, namely MP2P, the signature extraction technique could not extract a regular expression that could match an MP2P signature without creating a large number of false positives from DNS [3]. To improve recall we modified our signature extraction algorithm to also use the exact position of the signatures in the payload. We used the following heuristic. In the case where 95% of the flows

[3]From signature \x00\x00\x00.*\x00\x00\x00\x00

of the P2P application has the signature at a fixed location and if this signature is in a different location from colliding samples in the training set, then the regular expression is modified to match the signatures at fixed locations. Using this simple heuristic we achieved 88% recall for MP2P.

Other Experience: Going back and examining all the False Positive flows for protocols such as eDonkey and KaZaa, we observed that close to 50% of the FP flows are flows that have at least one of its hosts identified (from our ground truth payload classifier) as having an eDonkey flow and KaZaa flow respectively. This means that our signatures managed to capture flows that potentially belong to these P2P protocols, but we were unable to capture them with our GTPC.

5. RELATED WORK

Traffic classification and signature extraction are well-studied problems with significant previous work. Moreover, graphs have naturally been frequently been used as a model in various papers studying network traffic. We describe this previous work, and suggest how our work differs from it.

Traffic Classification. We group classification methods according to their level of observation: (a) packet-level, using well-known port numbers [8], (b) flow-level, using supervised [41, 57, 17, 25, 4] and unsupervised [3, 36, 29, 15, 16], Machine Learning (ML) techniques, (c) host-level [24, 23, 55], and (d) payload-based [21, 40, 47, 18, 32]. For an exhaustive list and comparison of ML methods we refer the reader to [43].

Flow-Level. These methods use flow features to train a ML classifier using previously labeled instances for each class. In the unsupervised case, clustering algorithms are used in order to group flows with similar characteristics together. All methods [3, 36, 29, 15, 16] require manual labeling of clusters. Our work bridges this gap by providing a method to automatically label clusters of flows based on their network-wide behavior.

Host-Based. These methods associate a host with its flows. The work that is most closely related to ours is BLINC [24], which operates by characterizing the connection patterns (e.g., if it behaves like a P2P application) of a single host at the Transport Layer, and use these patterns to label flows. BLINC uses graph models called graphlets to model a host's connection patterns using port and IP cardinalities. Unlike TDGs, graphlets do not represent *network-wide* host interaction. In some sense, TDGs represent a further level of aggregation, by aggregating across hosts as well. Thus it is perhaps fair to say that while BLINC hints at the benefit of analyzing the node's interaction at the "social" level, it ultimately follows a different path that focuses on the behavior of individual nodes.

Other host-based approaches include [22, 23]. Recent papers [2, 20], exploit additional information such as the high number of failed connections in P2P applications compare to Web traffic. These papers [2, 20] differ significantly from our work in that they do not use network wide interaction and they do not extract payload signatures.

Payload-based. Payload-based techniques were introduced in [21, 40, 47] to detect P2P applications. Using available documentation and manual reverse engineering, these approaches extract signatures for various P2P applications. These early methods also support the observation that the first few bytes (packets) are sufficient to classify flows. More recent efforts [18, 32] use the first 64 bytes of each flow's payload as a feature for traffic classification. Their findings also confirm that the first few payload bytes are sufficient. In [18], payload data are used to train classifiers, but the approach did not use TDGs and did extract regular expression. Ma et al. [32] used payload similarity to group similar flows, thereby simplifying the process of flow labeling by a network administrator. Both papers [18, 32] focus on the detection of conventional applications and not P2P applications. With Graption, we automate the process of labeling flows using TDGs, and automate the extraction of regular expression signatures using statistical methods from [42].

Worm Detection. Graphs have been used for detecting worm activities within enterprise networks [50, 14]. A key contribution of these works is the detection of the tree-like communication structure of worm propagation. This characteristic of worms was also used for post-mortem trace analysis (for the identification of the source of a worm outbreak, the so-called patient zero) using backbone traces [54]. These papers did not introduce the variety of graph metrics and techniques we use here, instead focusing on worm propagation characteristics such as high depth in the tree and an increased number of hosts found at each level of the tree. More recent studies use graph techniques to detect hit-list worms within an enterprise network, based on the observation that an attacker will alter the connected components in the network [10].

In the context of network security, anomalies were detected using correlation among features of a set of flows. Statistical methods were used in [55] for automating the profiling of network hosts and ports numbers, but they did not use network-wide interaction graphs as we do here. Statistical techniques are also used for anomaly detection [27] and for detecting scanning activity by monitoring the feature distribution of flows [46]. More efficient methods using sketches are introduced in [30], targeting on-line detection. Non of the above papers focused on P2P detection.

Communities of Interest and Host Similarity. The connectivity behavior and habits of users within enterprise networks is the focus of many papers, including [1, 51, 35, 52]. In [51], graphs are used as a means of modeling connections and grouping similar hosts within corporate networks. Again, these papers differ from ours in terms of how graphs are used. We use graphs to model network-wide behavior of Internet protocols and applications.

Trust propagation networks and other social network are often expressed as graphs (e.g.,[56], [39]), but the problems differ substantially from ours.

Behavioral-based Passive Measurements. Passive monitoring of P2P protocols is studied by Set et al. [48], targeting mainly the profiling of P2P hosts, including the measurement of bandwidth usage and persistence in the overlay. The goal of the measurement is to support traffic engineering and not for P2P detection. Topological aspects of the overlay were highlighted as a way to improve the overlay's sensitivity to failures of highly connected super peers. A similar study for large DNS traces [12] uses graphs in the context of classifying DNS servers according to their role in the DNS-hierarchy and for generating a space-efficient DNS traffic summary. Neither work targets the classification of applications.

A recent study by Latapy et al. [28], measured the evolution of TDG-like graphs between all the hosts exchanging a single packet of any type. Their goal was to show if such huge graphs evolve over time at point where their basic graph properties remain stable. The high aggregation of this graph is very different from our separate view of the traffic generated by different applications. Moreover, in contrast to our work, they did not target application classification.

For the study of World Wide Web, Meiss et al. [37] used sampled Web flows to extract statistics for the behavior of clients and servers regarding their cardinalities and the level of traffic exchanged between them. Their work is extended in [38] comparing their methods with P2P applications and providing a way to group ports based on user similarity. This allows the grouping of all the dominant ports of a single application together.

Preliminary work [11] attempts to detect P2P applications using port-based methods based on the order of temporal appearance of hosts in the trace and the fact that P2P hosts use the same port for incoming and outgoing connections. The work has not had any recent follow-up that we know of.

Signature Extraction. Automating the extraction of payload signatures has been studied for worm detection [49, 26] and polymorphic worms [42, 31]. We are the first to utilize such techniques for automating signature extraction for P2P applications. In addition, Polygraph originally utilized manual insertion of worm flows (perhaps from a honey-pot, or a scanning detection process). With Graption , we automate the process of generating examples via TDGs and use Polygraph at the back end of the system. In contrast to all other techniques, we target only the first bytes of the payload.

Recently Park et al. [45] proposed heuristics for generating application signatures from a set of flows that belong to a particular application (e.g, MSN). Such methods can be used in future versions of Graption in case more complicated signature are needed. Other methods [5, 6] use application binaries and are thus different from our approach that operates using packets seen at the network core.

6. CONCLUSIONS

The underlying theme of our work is a shift away from monitoring packets, flows, or hosts, to monitoring a group of nodes. We classify applications based on the behavior of a community of nodes modeled using what we call Traffic Dispersion Graphs. Simple graph measures allow TDGs of P2P traffic to be distinguished from other kinds of traffic.

However, TDGs are not sufficient by themselves. Building a TDG requires a key that isolates flows of one application from another. This is easy if applications use a fixed port but is infeasible when applications use many ephemeral ports, as new P2P applications do. Other features are needed to group flows before constructing TDGs for each group. Bacause applications use a few common headers, we found that the first 16 payload bytes was a useful feature. The grouping uses similarity measures without advance knowledge of signatures. However, each application can fragment into several clusters, and each cluster by itself has insufficient evidence of application behavior. To remedy this, we merged clusters based on node similarity.

TDGs based on the merged clusters identified P2P applications with high accuracy. A useful side-effect of the methodology is signature extraction. Our offline analysis (in minutes) can yield signatures which can be used by a router for P2P accounting and rate-limiting in real-time. While TDGs are only one component of our system, we believe that TDGs introduce a space of new metrics and techniques for visualizing, summarizing, and analyzing network traffic.

We also developed Graption, an agnostic P2P identification tool, which exhibits good performance in our data traces, and has several key properties. First, the cluster merging step allows detection through association. Nodes and flows can be classified by their connections to an identified group in the TDG even if the information about that node or flow is insufficient for detection by itself. Second, because Graption makes use of information at multiple levels (packet, flow, group) detection avoidance becomes harder: applications must obfuscate their behavior at the packet, flow, host, and network-wide levels.

Third, Graption performs agnostic traffic classification with a small number of intuitive parameters. Graption can detect P2P protocols without a priori knowledge, and also generates effective packet signatures using just the first 16 first payload bytes. Monitoring 16 payload bytes per flow is possible in the current generation of NetFlow devices, and increases the memory requirement only by a small factor.

Finally, we designed Graption as a modular framework where each component can be specialized to the detection problem at hand. For example, we briefly showed that TDG measures could be used to detect scanning worms (Subsection 3.4.1). History teaches us that new (and possibly unwanted) applications keep appearing. In the future, other graph metrics besides those we examine here or other payload features besides the first sixteen bytes may prove more important for classifying these applications, but our framework should still prove valuable. We hope that Graption and TDGs will find uses that go beyond the detection of P2P traffic.

Acknowledgments

The authors thank kc claffy, Marina Fomenkov, and Colleen Shannon from CAIDA for their help and support in providing traces TR-PAY1 and TR-PAY2. Support for CAIDA's Internet Traces is provided by the National Science Foundation, the US Department of Homeland Security, and CAIDA Members. Support for this work was provided by a Cisco URP grant.

7. REFERENCES

[1] W. Aiello, C. Kalmanek, P. McDaniel, S. Sen, O. Spatscheck, and J. Merwe. Analysis of communities of interest in data networks. In *PAM*, 2005.

[2] G. Barlett, J. Heidemann, and C. Papadopoulos. Inherent Behaviors of On-line Detection of Peer-to-Peer File Sharing. In *IEEE Global Internet (GI)*, 2007.

[3] L. Bernaille, R. Teixeira, I. Akodjenou, A. Soule, and K. Salamatian. Traffic classification on the fly. *ACM SIGCOMM Computer Communication Review*, 36(2):23–26, April 2006.

[4] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli. Revealing skype traffic: when randomness plays with you. In *ACM SIGCOMM*, 2007.

[5] D. Brumley, H. Wang, S. Jha, and D. Song. Creating vulnerability signatures using weakest pre-conditions. In *IEEE Computer Security Foundations Symposium (CSF)*, 2007.

[6] J. Caballero, H. Yin, Z. Liang, and D. Song. Polyglot: automatic extraction of protocol message format using dynamic binary analysis. In *ACM CCS*, 2007.

[7] The CAIDA OC48 Traces Dataset. Colleen Shannon, Emile Aben, kc claffy, Dan Andersen, and Nevil Brownlee. http://www.caida.org/data/passive/passive_oc48_dataset.xml.

[8] CAIDA Org. The coralreef project, http://www.caida.org/tools/measurement/coralreef/.

[9] W. Chang Feng, F. Chang, W. chi Feng, and J. Walpole. A traffic characterization of popular on-line games. *IEEE/ACM Transactions on Networking*, 13(3):488–500, 2005.

[10] M. P. Collins and M. K. Reiter. Hit-list worm detection and bot identification in large networks using protocol graphs. In *RAID*, 2007.

[11] F. Constantinou and P. Mavrommatis. Identifying known and unknown peer-to-peer traffic. In *IEEE NCA*, 2006.

[12] C. Cranor, E. Gansner, B. Krishnamurthy, and O. Spatscheck. Characterizing large dns traces using graphs. In *ACM SIGCOMM Internet Measurement Workshop*, 2001.

[13] K. E. Defrawy, M. Gjoka, and A. Markopoulou. Bottorrent: Misusing bittorrent to launch ddos attacks. In *USENIX SRUTI*, 2007.

[14] D. Ellis, J. Aiken, K. Attwood, and S. Tenarglia. A behavioral approach to worm detection. In *ACM CCS WORM*, 2004.

[15] J. Erman, M. Arlitt, and A. Mahanti. Traffic classification using clustering algorithms. In *ACM SIGCOMM MineNet*, 2006.

[16] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson. Offline/realtime traffic classification using semi-supervised learning. *Perform. Eval.*, 64(9-12):1194–1213, 2007.

[17] J. Erman, A. Mahanti, M. Arlitt, and C. Williamson. Identifying and discriminating between web and peer-to-peer traffic in the network core. In *WWW*, 2007.

[18] P. Haffner, S. Sen, O. Spatscheck, and D. Wang. Acas: automated construction of application signatures. In *ACM SIGCOMM MineNet*, 2005.

[19] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, and G. Varghese. Network monitoring using Traffic Dispersion Graphs (TDGs). In *ACM SIGCOMM IMC*, 2007.

[20] W. John and S. Tafvelin. Heuristics to classify internet backbone traffic based on connection patterns. In *International Conference on Information Networking (ICOIN)*, 2008.

[21] T. Karagiannis, A. Broido, N. Brownlee, kc claffy, and M. Faloutsos. Is p2p dying or just hiding? In *IEEE GLOBECOM*, 2004.

[22] T. Karagiannis, A. Broido, M. Faloutsos, and kc claffy. Transport layer identification of p2p traffic. In *ACM SIGCOMM/USENIX IMC*, 2004.

[23] T. Karagiannis, D. Papagiannaki, N. Taft, and M. Faloutsos. Profiling the end host. In *PAM*, 2007.

[24] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: Multi-level Traffic Classification in the Dark. In *ACM SIGCOMM*, 2005.

[25] H. Kim, K. Claffy, M. Fomenkova, N. Browlee, D. Barman, and M. Faloutsos. Comparison of internet traffic classification tools. In *IMRG WACI*, 2007.

[26] H.-A. Kim and B. Karp. Autograph: toward automated, distributed worm signature detection. In *USENIX SSYM*, 2004.

[27] A. Lakhina, M. Crovella, and C. Diot. Mining Anomalies Using Traffic Feature Distributions. In *ACM SIGCOMM*, 2005.

[28] M. Latapy and C. Magnien. Complex network measurements: Estimating the relevance of observed properties. In *IEEE INFOCOM*, Sep 2008.

[29] R. T. Laurent Bernaille and K. Salamatian. Early application identification. In *ACM CoNEXT*, 2006.

[30] X. Li, F. Bian, M. Crovella, C. Diot, R. Govindan, G. Iannaccone, and A. Lakhina. Detection and identification of network anomalies using sketch subspaces. In *ACM SIGCOMM IMC*, 2006.

[31] Z. Li, M. Sanghi, Y. Chen, M.-Y. Kao, and B. Chavez.

Hamsa: Fast signature generation for zero-day polymorphicworms with provable attack resilience. In *IEEE SP*, 2006.

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

[33] P. Mahadevan, C. Hubble, D. Krioukov, B. Huffaker, and A. Vahdat. Orbis: rescaling degree correlations to generate annotated internet topologies. In *ACM SIGCOMM*, 2007.

[34] P. Mahadevan, D. Krioukov, K. Fall, and A. Vahdat. Systematic topology analysis and generation using degree correlations. In *ACM SIGCOMM*, 2006.

[35] P. McDaniel, S. Sen, O. Spatscheck, J. V. der Merwe, B. Aiello, and C. Kalmanek. Enterprise Security: A Community of Interest Based Approach. In *NDSS*, 2006.

[36] A. McGregor, M. Hall, P. Lorier, and J. Brunskill. Flow clustering using machine learning techniques. In *PAM*, 2004.

[37] M. Meiss, F. Menczer, and A. Vespignani. On the lack of typical behavior in the global web traffic network. In *WWW*, May 2005.

[38] M. Meiss, F. Menczer, and A. Vespignani. A framework for analysis of anonymized network flow data. In *NGDM*, 2007.

[39] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *ACM SIGCOMM IMC*, 2007.

[40] A. Moore and K. Papagiannaki. Toward the accurate identification of network applications. In *PAM*, 2005.

[41] A. Moore and D. Zuev. Internet traffic classification using bayesian analysis techniques. In *ACM SIGMETRICS*, 2005.

[42] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically generating signatures for polymorphic worms. In *IEEE SP*, 2005.

[43] T. T. Nguyen and G. Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys and Tutorials*, March 2008.

[44] S. Orr. DDoS Threatens Financial Institutions –Get Prepared, ReymannGroup Inc Report, 2005. www.cisco.com/web/strategy/docs/finance/fin_DOSS.pdf.

[45] B.-C. Park, Y. J. Won, M.-S. Kim, and J. W. Hong. Towards automated application signature generation for traffic identification. In *NOMS*, 2008.

[46] N. Patwari, A. Hero, and A. Pacholski. Manifold learning visualization of network traffic data. In *ACM SIGCOMM MineNet*, 2005.

[47] S. Sen, O. Spatscheck, and D. Wang. Accurate, scalable in-network identification of p2p traffic using application signatures. In *WWW*, 2004.

[48] S. Sen and J. Wang. Analyzing peer-to-peer traffic across large networks. *IEEE/ACM Transactions on Networking*, 12(2):219–232, 2004.

[49] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *OSDI*, 2004.

[50] Steven Cheung and Rick Crawford and Mark Dilger and Jeremy Frank and Jim Hoagland and Karl Levitt and Jeff Rowe and Stuart Staniford-Chen and Raymond Yip and Dan Zerkle. The Design of GrIDS: A Graph-Based Intrusion Detection System. *UCD Technical Report CSE-99-2*, 1999.

[51] G. Tan, M. Poletto, J. Guttag, and F. Kaashoek. Role classification of hosts within enterprise networks based on connection patterns. In *USENIX Annual Technical Conference*, 2003.

[52] J. Tolle and O. Niggenmann. Supporting intrusion detection by graph clustering and graph drawing. In *RAID*, 2000.

[53] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2nd edition, 2005.

[54] Y. Xie, V. Sekar, D. Maltz, M. Reiter, and H. Zhan. Forensic analysis of epidemic attacks in federated networks. In *IEEE International Conference on Network Protocols (ICNP)*, 2006.

[55] K. Xu, Z. Zhang, and S. Bhattacharyya. Profiling internet backbone traffic: Behavior models and applications. In *ACM SIGCOMM*, 2005.

[56] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. Sybilguard: defending against sybil attacks via social networks. In *ACM SIGCOMM*, 2006.

[57] S. Zander, T. Nguyen, and G. Armitage. Self-learning ip traffic classification based on statistical flow characteristics. In *PAM*, 2005.