

Inherent Behaviors for On-line Detection of Peer-to-Peer File Sharing

Genevieve Bartlett[†] John Heidemann[†] Christos Papadopoulos[‡]

[†]USC/ISI [‡]Colorado State University {bartlett,johnh}@isi.edu, christos@cs.colostate.edu

Abstract—Blind techniques to detect network applications—approaches that do not consider packet contents—are increasingly desirable because they have fewer legal and privacy concerns, and they can be robust to application changes and intentional cloaking. In this paper we identify several behaviors that are *inherent* to peer-to-peer (P2P) traffic and demonstrate that they can detect both BitTorrent and Gnutella hosts using only packet header and timing information. We identify three basic behaviors: failed connections, the ratio of incoming and outgoing connections, and the use of unprivileged ports. We quantify the effectiveness of our approach using two day-long traces, achieve up to an 83% true positive rate with only a 2% false positive rate. Our system is suitable for on-line use, with 75% of new P2P peers detected in less than 10 minutes of trace data.

I. INTRODUCTION

Identifying and filtering network traffic is central to firewalls and intrusion-detection systems. The majority of these systems deployed today use ports or packet signatures to classify traffic for filtering. While fast and effective for typical traffic, these approaches are becoming less effective because both ports and packet contents are easy to conceal, either intentionally or accidentally.

We see three reasons for a greater need to identify network applications by packet header information alone, rather than packet payload. First, benign traffic often varies its port usage and packet contents. For example, traffic using remote-procedure calls, multiplexed protocols such as SOAP, or UDP-based protocols such as SIP often varies port usage and communicates ports out-of-band. An increasing use of traffic encryption hides packet-contents, both with network-level approaches like IPsec, and application-level tunnels like ssh or TLS.

Second, malware and protocols that receive mixed acceptance often intentionally hide their identity by varying port usage and packet contents. Protocols such as Skype and P2P file sharing often hide themselves out of concern for restrictive use policies in some networks.

Finally, even when traffic is not accidentally or actively concealing itself, government and ISP policy concerns often prevent analysis of data packet contents. For example, in the United States, laws about student privacy and wiretapping can be interpreted to preclude analysis of packet data contents.

The goal of this paper is to identify network applications based on their *inherent* characteristics without considering packet contents. We therefore distinguish application behaviors that are easily changed or *incidental*, from those behaviors that are *inherent* and would incur a performance penalty or require application restructuring to change. In this sense, we are investigating blind techniques to identify applications [7].

We evaluate our approach by considering two popular P2P file sharing applications: BitTorrent and Gnutella. We evaluate our detection methods with two full day traffic traces taken from a regional ISP in 2005 and 2006, and compare our detection rates to ground truth obtained by manual analysis of the data.

The contribution of this paper is the identification and evaluation of several metrics that are applicable to blind identification of multiple types of P2P file sharing applications. Unlike prior work, our metrics can work when traffic views are incomplete and/or unidirectional, and our protocols are light-weight and able to run in near-real-time. We show that even with an incomplete traffic view (two of five links to an ISP), these metrics can detect hosts running BitTorrent applications with an 83% true positive rate with a 2% false positive rate and detect hosts running Gnutella with a 75% true positive rate with a 4% false positive rate. Our approach is suitable for environments where privacy is a strong concern, and where on-line identification of P2P sharing can trigger secondary checks that could not be done after-the-fact, perhaps to differentiate between sharing of open and restricted content. Of the P2P peers caught by our system, 75% required less than 10 minutes of trace data to determine P2P activity.

II. RELATED WORK

There are three general areas of related work: detection based on port, packet payload, and traffic behavior.

Port- and payload-based signatures are widely used today. Unfortunately, both rely on incidental behaviors: port assignments are easily changed, and payload contents can be hidden by encryption or randomization. We therefore do not consider these approaches further.

An alternative is to detect based on network behaviors such as an application's communication pattern, since

this can often be more difficult to conceal. Karagiannis et al., identify P2P traffic from connection patterns and the concurrent use of UDP and TCP [6]. Constantinou and Mavrommatis classify P2P traffic based on connection direction and number of peers in a connected group [4]. In later work, Karagiannis et al. introduce BLINC [7], a general classification mechanism that classifies hosts based on protocol usage, port usage and connection patterns. These methods rely on behavior that is inherent to P2P applications. While our approach is similar in that it uses inherent behavior, our metrics require significantly less state than the methods used in this previous work; we quantify our on-line detection time and show that we can operate in near-real-time. Furthermore, we show that our approach is effective even when presented with less than complete traffic to an ISP and unidirectional traffic.

Closest to our work is that by Collins et al. [3] who distinguish BitTorrent flows from FTP, HTTP and SMTP flows between pairs of hosts. They study three metrics: packet size (looking for small control messages), amount of data exchanged between hosts, and rate of failed connections. We do not consider packet size to be an inherent metric since it is easily spoofable. The later two metrics are inherent, and we have independently determined that failed connections are an important indicator of P2P traffic. Our work differs from theirs through the addition of two other behaviors (ratio of incoming-to-outgoing connections and privileged-to-non-privileged ports); by demonstrating that this approach applies to multiple kinds of P2P traffic, not just BitTorrent; by demonstrating that our approach works even with an incomplete traffic view; through use of an adaptive windowing approach that is more robust to manipulation of connection timing; and by demonstrating that our approach can operate on-line rather than post-facto.

III. BEHAVIORS IN P2P

In this section we investigate three behaviors of P2P applications. The first two of these behaviors are inherent to P2P; the last behavior we include for comparison purposes. In Section IV we map these behaviors to specific metrics for detection.

Our target applications are BitTorrent and Gnutella. Both are popular file sharing protocols described in detail elsewhere. For our purposes, the important characteristic of BitTorrent is that a *peer* typically contacts a *tracker* to find other peers. It then directly communicates with many peers (often 20). Gnutella instead uses a two-tier system of *leaf nodes* and *ultrapeers*. Leaf nodes typically talk only to ultrapeers, while ultrapeers communicate widely with both each other and leaf nodes.

A. Peer Coordination and Failed Connections

Since peers are end-user machines, there is considerable churn as they come and go frequently [2]. For efficiency and scalability, P2P mechanisms which track the presence of peers do so imperfectly, and this information is quickly out of date when given to new peers or peers searching for content sources. As a result, an inherent behavior of P2P sharing is many *failed attempts to contact peers* that have left the network. At the network level, these failed contacts result in TCP RST messages from a busy or no-longer participating peer, or in multiple SYN packets attempting to start a connection and timing out.

This behavior is not only common to P2P traffic, but relatively uncommon among client/server applications. In client/server protocols, the servers are often well known and persistent. Failures are usually due to misconfiguration or hardware failure and there are not usually small clusters of failures.

B. Bidirectional Connections

P2P applications not only start connections with peers, but each peer attempts to maintain this network independently. Since, generally speaking, peers are equivalent, this means each initiates and receives new connections. Client/server hosts instead primarily either initiate connections (clients) or receive them (servers). Thus, unlike client-server applications, we can identify P2P applications by their *inherent balance of incoming and outgoing connections* over time.

While all P2P protocols have bidirectional connections, different protocols vary in the details. All BitTorrent peers have significant bidirectionality, typically starting 5–10 outgoing connections when they join the network, and accepting 5–10 incoming connections over time. By contrast, Gnutella's leaf nodes have a wider range of bidirectionality, maintaining around 5 outgoing connections and accepting incoming connections only when serving content. *Shielded* leaf nodes (often nodes behind NAT boxes) only make outgoing connections.

C. Unprivileged Port Usage

P2P file sharing applications are typically user-level processes operating on a variety of platforms and user environments, using unprivileged ports. Thus, a P2P file-sharing connection will typically have source and destination ports above 1024, unlike server applications such as mail and web servers, which typically use well-known privileged ports.

Unlike the other behaviors, the use of unprivileged ports is not inherent, since applications can often easily

choose either kind of port. We use this method for comparison, but in Section V we demonstrate our methods work well without relying on this behavior.

IV. IMPLEMENTATION

The previous section outlined three P2P file sharing application behaviors which are identifiable at the network level. In this section we translate these behaviors into testable metrics, with empirically derived values which correspond to P2P. We describe how these metrics are used and combined in Section IV-C.

A. Translating Behaviors to Metrics

a) *Peer coordination and failed connections*: As discussed in section III-A, coordination with other peers often leads to bursts of failed connections. We capture this behavior with the following ratio of *failed connections*:

$$\mathbf{M}_{\text{FC}} = \frac{\text{failed}_{\text{out}}}{\text{successful}_{\text{out}} + \text{failed}_{\text{out}}}$$

where $\text{failed}_{\text{out}}$ is the total number of new outgoing connections that fail and $\text{successful}_{\text{out}}$ is the total number of new outgoing connections that were successfully established. Values of \mathbf{M}_{FC} tend to be low for normal clients and servers, medium (0.1–0.8, our thresholds) for P2P hosts, and high (more than 0.8) for hosts doing port scans.

b) *Bidirectional connections*: As discussed in section III-B, P2P clients both initiate and receive new connections. To capture this behavior we use the following ratio of *bidirectional connections*,

$$\mathbf{M}_{\text{BC}} = \frac{\text{successful}_{\text{in}}}{\text{successful}_{\text{out}} + \text{successful}_{\text{in}}}$$

where $\text{successful}_{\text{in}}$ and $\text{successful}_{\text{out}}$ is the total number of new, successfully established, incoming and outgoing connections. The metric \mathbf{M}_{BC} will be close to 1 for servers, and close to 0 for clients. We consider values between 0.2 and 0.9 indicative of P2P hosts.

c) *Unprivileged Port Usage*: As discussed in section III-C, although the individual port number varies, P2P clients connect to unprivileged ports. Thus we define:

$$\mathbf{M}_{\text{UP}} = \frac{\text{successful}_{\text{user2user}}}{\text{successful}_{\text{in}} + \text{successful}_{\text{out}}}$$

where $\text{successful}_{\text{user2user}}$ is the number of successful connections which have a source and destination port in the unprivileged range and $\text{successful}_{\text{in}} + \text{successful}_{\text{out}}$ is the number of total new connections at that host which were successful. For clients and servers, the expected value for ratio \mathbf{M}_{UP} is near 0. Hosts doing user-level P2P run closer to 1; we consider any value over 0.2 to indicate a potential P2P host.

B. Metrics to Tests

We must now map individual metrics into binary tests that confirm or disclaim P2P traffic on a host. P2P traffic corresponds to medium values of each ratio, so we define high and low thresholds h_X and l_X . We use high values to indicate non-P2P behaviors (such as port-scanning), so exceeding h_X terminates the test as a non-P2P host. Low values often occur when a new host appears, so we consider values below l_X as inconclusive. Values in-between the thresholds after a warm-up number of connections positively indicate a P2P host.

Each metric is evaluated over a set of connections. To do this we use an adaptive windowing process described in section IV-C.

While each metric by itself corresponds to a specific P2P behavior, we found individual metrics to be noisy. We therefore test multiple metrics in parallel. A negative P2P result from any metric disqualifies a host, while a positive result from all metrics is required to flag the host as participating in P2P activities.

Failed connections captured by \mathbf{M}_{FC} primarily occur for a very brief period, whereas our other two metrics rely on behaviors seen over longer periods of time. This causes metric \mathbf{M}_{FC} to often trigger before the other metrics, and then be “washed out” by the time other metrics trigger. To prevent \mathbf{M}_{FC} from being “washed out”, we define a “sticky” equivalent, \mathbf{M}_{sFC} , which always tests positive for P2P for the rest of the current window after metric \mathbf{M}_{sFC} is triggered.

C. System Operation

Our system runs on top of a continuous network tracing infrastructure [5]. We transform the packet-level trace into a flow-level trace by observing only the TCP SYN and SYN-ACK packets.

Our system operates with both bidirectional and unidirectional traces. If only unidirectional traffic is present, we identify failed connections by four or more duplicate SYNs. If bidirectional traffic is present, we identify failed connections by duplicate SYNs which go unanswered for 20 minutes.

We then compute the ratios required for each of our tests. We process the data sequentially, on-line, evaluating the metrics for each IP address once we have a 10 connections “window”. If at any time the metrics indicate a positive or negative result we classify the host as P2P or non-P2P for that window. If classification is indeterminate, we add connections to our window until we can reach a conclusion. We remove connections from a window when they are more than 20 minutes old. These

parameters are chosen for a timely response, but our results are relatively insensitive to the specific values [1].

V. EVALUATION

In the following sections we evaluate our approach to determine how detection accuracy interacts with false positive rates. Our evaluation uses network packet traces from two (out of five) links at Los Nettos, a regional ISP in the Los Angeles area serving both commercial and academic institutions. One link captures bidirectional traffic, while the other captures only outgoing traffic. For each host, we see only a fraction of the peers it might contact, demonstrating that our methods are effective even with partial data.

We collected two datasets, each 24 hours long on August 31, 2005 and October 3 2006. We see qualitatively similar results for both traces and present only the 2006 data here due to space constraints.

A. Detection Accuracy for BitTorrent

We first look at detection accuracy to verify that our approaches successfully identify BitTorrent traffic.

To establish ground truth, we first identify flows on the default BitTorrent tracker port (6969), and then verify that the destination is a tracker by contacting the host. We classify these confirmed BitTorrent peers as *known BitTorrent hosts*.

The Known BitTorrent section of Table I shows we verified 130 hosts were running BitTorrent. We first observe that each individual metric is successful at detecting the majority of known BitTorrent hosts (85–92%). We observe that our inherent metrics (\mathbf{M}_{FC} and \mathbf{M}_{BC}) detect nearly as many P2P hosts as \mathbf{M}_{UP} .

Finally, we observe that the combined metrics perform almost as well as the stand-alone metrics at detecting true positives (83%–84% compared to 85–92%), and the combined metrics perform much better in reducing false positives (2% instead of 13–25%). In addition, we observe that $\mathbf{M}_{\text{sFC+BC}}$ does nearly as well as \mathbf{M}_{all} , while using only inherent information.

B. Understanding false positive rate

Even if the system performs well at detecting P2P hosts, it will not be useful if it also falsely tags many non-P2P hosts as P2P. We therefore evaluate the false positive rate of individual and combined metrics.

It is easy to confirm the presence of known P2P traffic to a host, it is significantly more difficult to prove absence of P2P traffic. To roughly define non-P2P hosts, we first remove all known P2P hosts from our population (of 9,656 hosts) and select half of the remaining hosts. (We will use the other half in Section V-D.) While these hosts include no known P2P hosts (those running on well

known P2P ports), there may be some hosts using non-standard ports for P2P. We assume these non-standard ports are non-privileged, and discard 608 hosts that have non-privileged-to-non-privileged ports as *potential P2P*. We label the remaining hosts as *likely non-P2P*. This decision is conservative (since it is possible to use privileged ports for P2P), but it also removes hosts which are running a number of non-P2P applications (such as passive FTP and gaming applications) from the likely non-P2P category. We address our ability to distinguish between P2P and other non-privileged-to-non-privileged applications in section V-D.

We use this set of likely non-P2P hosts to look for false positives in our metrics. We expect individual metrics to have some number of false positives: port scanners and misconfigured machines or servers can accidentally trigger \mathbf{M}_{FC} , and some services that have bidirectional connections (such as DNS) and user machines that host some servers can trigger \mathbf{M}_{BC} . Note that we cannot consider \mathbf{M}_{UP} with this methodology because our definition of likely non-P2P distorts this metric.

The likely non-P2P section of Table I shows these results. Individual metrics show moderate-to-high false positive rates (13–25%). Because the number of likely non-P2P hosts is so much larger than the number of known P2P hosts, these false positive rates imply 5–10 errors for every true positive. Such high false positive rates mean that an individual metric is impractical without additional confirmation. Examination of specific traces suggest that many \mathbf{M}_{FC} failures are due to false identification of port scans as P2P. We examined a few cases of \mathbf{M}_{BC} failure; they were typically due to user hosts that also run small server applications.

Our hope is that combining multiple metrics can reduce the false positive rate, and $\mathbf{M}_{\text{sFC+BC}}$ shows a false positive rate of only 2% rather than 13–25%. This success is because the false positives are triggered by different circumstances. With all three metrics, \mathbf{M}_{all} eliminates all false positives, but as described above this is an anomaly due to our definition of likely-non-P2P.

From our evaluation of true and false positives we conclude that the combination of at least metric \mathbf{M}_{sFC} with metric \mathbf{M}_{BC} is essential for accuracy and few false positives. Metric $\mathbf{M}_{\text{sFC+BC}}$ shows only a few percent reduction (2–5%) in detection accuracy for BitTorrent, while the percent of false positives is cut in four.

C. Effectiveness for Gnutella

Since our detection methods are based on P2P behaviors in general, and not specific to the BitTorrent protocol, we expect that our system is capable of detecting

metric:	Individual metrics			Combined Metrics		
	M_{FC}	M_{BC}	M_{UP}	M_{sFC+BC}	M_{sFC+UP}	M_{all}
Total unique hosts: 9,656						
P2P hosts : 290						
Known BitTorrent hosts: 130						
True Positives	110 (85%)	114 (88%)	120(92%)	108 (83%)	109 (84%)	108 (83%)
False Negatives	20 (15%)	16 (12%)	10 (8%)	22 (17%)	21 (16%)	22 (17%)
Known Gnutella hosts: 160						
True Positives	123 (77%)	109 (68%)	155 (97%)	93 (58%)	120 (75%)	91 (57%)
False Negatives	37 (23%)	51 (32%)	5 (3%)	67 (42%)	40 (25%)	69 (43%)
Other Hosts : 9,366						
Likely non-P2P: 4,075						
False Positives	530(13%)	1,018(25%)	n/a	81(2%)	n/a	n/a
True Negatives	3,545(87%)	3,057(75%)	n/a	3,994(98%)	n/a	n/a
Discarded as potential P2P: 608						
Unclassified hosts: 4,683						
Flagged as P2P	702 (15%)	1,639(35%)	1,592(34%)	140(3%)	187(4%)	70(1%)
Not flagged as P2P	3,981(85%)	3,044(65%)	3,091(66%)	4,543(97%)	4,496(96%)	4,613(99%)

TABLE I
SUMMARY OF RESULTS OF BITTORRENT DETECTION FOR 2006 DATA SET.

hosts running other P2P applications. To test this claim we next evaluate our approach on Gnutella hosts.

We establish Gnutella ground truth as all hosts that contact known Gnutella ultrapeers. We track Gnutella ultrapeers by joining the Gnutella network repeatedly and recording lists of the suggested ultrapeers.

Some protocol differences between BitTorrent and Gnutella affect our metrics, however. We expect that metrics M_{FC} and M_{UP} will perform well at detecting Gnutella, but because of Gnutella’s two-tiered architecture, M_{BC} will not perform nearly as well, since it is unable to detect shielded leaf nodes and leaf nodes which choose not to share P2P content. Additionally, M_{BC} is more sensitive to incomplete traffic views when detecting Gnutella because Gnutella leaf nodes maintain few outgoing connections and often accept few connections (if any), as discussed in section III-B.

The *known Gnutella* section in Table I shows that M_{FC} alone detects 77% of the Gnutella hosts. However, as discussed in Section V-B, this metric alone has a high false positive rate and so we need combined metrics to reduce false positives. Though the limitations of metric M_{BC} hinder the combined metrics, the combined metrics still detect 58% of the known Gnutella hosts.

Metric M_{sFC+UP} detects nearly as many true positives as M_{FC} alone (75% vs. 77%), and significantly increases the distinguishing ability. For networks where Gnutella is popular, this metric may be preferable to M_{all} .

D. Estimating previously undetected P2P hosts

Our above analysis isolated traffic into known-P2P and likely-non-P2P categories to study the accuracy of our approaches. We next look at unclassified traffic to

estimate how many hosts appear to be P2P file sharing but were not included in our known-P2P.

To estimate P2P traffic in unclassified traffic, we start with the half of hosts not considered above. These hosts exclude all traffic to known trackers, so they all would be unclassified by detection schemes using known sites. We then run our detection algorithms on these hosts and examine the hosts flagged as P2P.

The *unclassified* section of Table I shows our estimate of P2P traffic in this sample of hosts. Our combined metrics flagged 1.5% (70 hosts) of the unclassified hosts as running P2P applications. Our analysis of the false positive rate of M_{sFC+BC} suggests that at least half of these are true positives. Although we do not know the likely true positive rate for M_{all} , it should be greater since M_{all} reduces further the number of flagged hosts by including metric M_{UP} .

To confirm that some of these 70 hosts have P2P traffic we looked at what ports they use. Of these 70 hosts, 17 made connections to remote hosts on default BitTorrent ports (6969, 6881–6888) and 15 made connections to remote hosts on the default Gnutella port (6346), strongly suggesting that we successfully found true P2P traffic. (If the host had contacted a known tracker or ultrapeer we would have already classified it as known-P2P.) We conjecture that some of the other hosts were doing P2P sharing on non-standard ports, although we could not confirm this. Finally, our analysis of these unclassified hosts indicates the benefit of adding M_{UP} to form M_{all} —this addition reduces the number of hosts identified as potential P2P in half (70 vs. 187). Of the 17 hosts just described none are eliminated, suggesting (not proving) that M_{all} does not reduce the true positive rate.

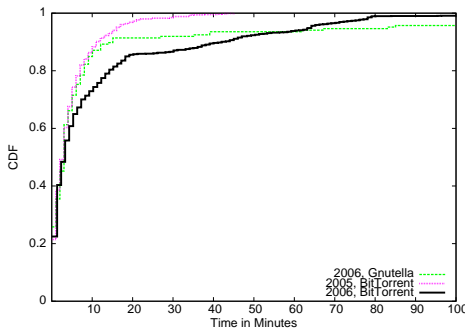


Fig. 1. Time until detection

E. Detection speed

As well as being accurate, we wish to detect P2P hosts quickly. To estimate detection time we consider known P2P hosts. We identify the first contact with a known tracker or ultrapeer as the “start” time and then determine how much later we classify that host as P2P. This start time corresponds to an unrealizable, idealized detection system based on a perfectly known P2P network, therefore, our results represent a conservative estimate of our detection time. Figure 1 shows CDFs of detection time for both BitTorrent and Gnutella. As is shown, about 75% of the time we can identify a P2P client in less than ten minutes, and about one-fifth of the time we can decide within a minute. Given that P2P applications often run for tens of minutes, we believe these detection times are more than sufficient for on-line identification.

To understand the cause of the delay we looked at how the combined metrics operate. For BitTorrent, M_{FC} triggers quickly, but M_{BC} is much slower. These timings are consistent with the behaviors they track, since failed connections occur when a new peer starts up and actively probes other peers, while bidirectional communication happens only later as other peers learn about the target host and connect to it. In the case of Gnutella, M_{FC} often does not trigger till several minutes after contacting ultrapeers. We believe this slower trigger is because ultrapeers are more reliable than typical BitTorrent peers, and with Gnutella M_{FC} is triggered only when a peer attempts to contact remote resources and download files.

In addition to the delay described above, our current implementation batches packet traces into 2–6 minute segments. Thus actual delay in our current implementation is up to 16 minutes 75% of the time. This batching is due to our data collection system [5] and could be eliminated by integrating trace collection with metric evaluation.

F. Parameter sensitivity

Our system has several parameters that affect operation, including the time-out of connections, the minimum

size of the connection “window” and metric thresholds. Due to space limitations we provide a detailed evaluation of these factors in a technical report [1].

VI. FUTURE WORK AND CONCLUSIONS

We have shown that one can map inherent P2P behaviors into metrics that allow on-line detection of P2P hosts, even with incomplete views of traffic. We found that a combination of metrics allows for high accuracy with low false positives, and is able to detect the majority of hosts in 10 minutes or less.

Evading detection by distorting inherent behavior is possible, but at a high price. For example, to avoid triggering M_{FC} , a peer could deliberately intersperse successful connections (e.g., to stable servers) with failed P2P connections, but regulating and synchronizing these to reliably taint our detection window is hard and will degrade P2P performance.

We see several important directions for future work. First, we would like to evaluate our methods against data that includes ground-truth based on packet contents to further validate accuracy. Second, we have shown good accuracy while monitoring only a fraction of an ISP’s traffic; we need to quantify how percentage of traffic seen affects accuracy. Finally, we are in the process of applying our approach to estimate the total amount of P2P traffic at a university, and to compare this to traditional, port-based detection schemes.

REFERENCES

- [1] G. Bartlett, J. Heidemann, and C. Papadopoulos. Inherent behaviors for on-line detection of peer-to-peer file sharing. Technical Report ISI-TR-627, ISI, 2006.
- [2] M. Bawa, H. Deshpande, and H. Garcia-Molina. Transience of peers and streaming media. In *Proceedings of the ACM HotNets I*, pages 107–112, Princeton, NJ, USA, October 2002.
- [3] M. Collins and M. Reiter. Finding peer-to-peer file-sharing using coarse network behaviors. In *Proceedings of the European Symposium On Research In Computer Security*, Hamburg, Germany, September 2006.
- [4] F. Constantinou and P. Mavrommatis. Identifying known and unknown peer-to-peer traffic. In *IEEE International Symposium on Network Computing and Applications (NCA)*, pages 93–102, Cambridge, MA, USA, July 2006.
- [5] A. Hussain, G. Bartlett, Y. Pryadkin, J. Heidemann, C. Papadopoulos, and J. Bannister. Experiences with a continuous network tracing infrastructure. In *Proceedings of the ACM SIGCOMM Workshop on Mining network data Mine Net*, pages 185–190, Philadelphia, PA, USA, August 2005.
- [6] T. Karagiannis, A. Broido, M. Faloutsos, and kc claffy. Transport layer identification of p2p traffic. In *Proceedings of the ACM SIGCOMM Workshop on Internet Measurement (IMC)*, pages 121–134, Taormina, Sicily, Italy, October 2004.
- [7] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: Multilevel traffic classification in the dark. In *Proceedings of the ACM SIGCOMM Conference*, pages 229–240, Philadelphia, PA, USA, August 2005.