Detecting P2P Traffic from the P2P Flow Graph

Jonghyun Kim
Electrical and Computer Engineering
University of Delaware
Newark, USA
Email: jonghyunl@udel.edu

Khushboo Shah
Electrical and Computer Engineering
University of Delaware
Newark, USA
Email: Khushboo.H.Shah@gmail.com

Stephan Bohacek
Electrical and Computer Engineering
University of Delaware
Newark, USA
Email: bohacek@udel.edu

***Abstract*—Accurate identification of P2P (peer-to-peer) applications' flows is important for network capacity planning, provisioning, application traffic engineering, network service pricing, traffic shaping/policing, and flow prioritization. To this end, many identification methods have been developed based on the transport layer port, analysis of packet payloads, statistical observation on flows and graph-based structural properties. One deficiency of those methods is that they concentrate only on a decision of whether a single flow is P2P flow or not. This paper presents methods for detecting P2P flows by constructing a graph where each flow is a vertex. Edges are constructed by applying various rules that consider the ports used by previously detected P2P flows. In this graph, we find that around 90% of the P2P flows are within a large connected component. The remaining 10% is composed of many smaller connected components. Edges between the large connected component and these other components can be constructed with some heuristics. The methods proposed are tested on traffic traces that included signature matching, so that we are able to ensure that significant P2P applications are detected.**

***Index Terms*—Application identification, P2P, flow graph.**

I. Introduction

Detecting P2P traffic is important for a range of network management activities. For example, since web surfers places a high priority on file transfer time [1] and P2P users do not, a larger number of P2P users could drive the network to a state where web surfers experience poor performance. Therefore, if P2P flows can be identified, network administrators might limit network resources provided to P2P during times of congestion. Also, in some settings P2P applications are security risk. For example, P2P networks can be used to distribute copy-righted and confidential material. Botnets [2] are also known to make use of P2P networks. Therefore, some network administrators might desire to detect P2P users. And finally, network planning will be improved if the applications used by network users are known.

While detecting P2P traffic is desirable, because of encryption and evolving protocols, identifying P2P flows is challenging. Nonetheless, a range of techniques is available for consideration. For example, the TCP and UDP ports of many P2P system are known, and hence some P2P flows can be detected by simply examining the ports. Sophisticated firewalls can detect P2P by examining byte strings signatures [3], [4]. And finally, the behavior of P2P is known, and hence schemes can be developed to detect this type of behavior. It is important to point out that schemes that use byte string signatures require expensive deep packet inspection (DPI) equipment. One important finding of this paper is that P2P can be detected by considering a graph of flows between multiple users. Specifically, we find effective iterative methods that discover new P2P flows based on previously found P2P flows. Insight into the proposed approach is provided by modeling P2P flow detection as traversing a graph where flows are vertices and the detection methods define edges between flows. Among other findings, we find that most P2P flows are within the same large connected component of this graph. Consequently, it is straightforward to find one flow in this connected component, and then all flow in the component can be found. Flows in other components can also be found by linking different components together with a different P2P detection algorithm.

The behavior-based detection [5], [6], [7] has previously been applied to P2P detection. For example, [5] introduced flow-based heuristic methods. TCP/UDP pairs method tags flows generated by two hosts that use both TCP and UDP protocols for their communication. However, in order to limit the number of false positives, the user must develop a white list of applications that behave as servers. Unfortunately, this list is expansive, and we found that the list provided in [5] resulted in a large number of false positives.

Researchers have developed more computational complex methods including methods that employ clustering algorithms [8], machine learning [9], [10], and traffic profiling [11], relying on statistical observations of more detailed information in a flow (e.g., total packet size, total number of packets, flow duration, average inter-arrival time between packets, size of the first data packet, etc.) or on graph-based structural properties [12].

[13] showed transport-layer interactions (social behaviors) for representative applications visually. In their classification, they ignored port numbers, which cannot be relied on to detect all P2P flows, but can assist in detecting P2P flows. For example, if you know that a host's P2P application listens on a specific port, then incoming flows to that port are likely to be P2P flows. Another drawback of [13] is that it has difficulties when a host runs multiple applications simultaneously. [14], [15] showed that the graphs (TDGs or TAGs) constructed by P2P applications have the property of one large giant

connected component (GCC) that can be quickly formed with relatively small number of connected components that are formed based on different P2P ports, each component consisting of hosts and edges (an edge between two hosts exists if P2P communication between them has occurred). This gives us an insight that tracing a confirmed P2P port may result in finding the GCC as well as P2P flows readily. Our graph analysis here is different from [14], [15] in that the vertex is a P2P flow and its traced P2P flows. As far as we know, this paper is the first to address P2P flow-tracing graph analysis.

The paper is organized as follows. Section II discussed the flow data and presents mathematical definitions used throughout the paper. Section III explains our set of detection methods. Section IV discusses the performance of the detection methods. To gain further insight into the performance, Section V develops and investigates a graph model of P2P flow detection. Finally, Section VI presents our conclusions.

II. Flow Data

Network flows are obtained by the monitoring machine located between dormitory networks of a university and external networks in the spring of 2008. In total, 206 millions of TCP and UDP flows were collected from 3161 users. Flow information consists of source IP, destination IP, source port, destination port, transport protocol, flow start time, and event ID, which we denote by SIP, DIP, SP, DP, PR, ST, and EID, respectively. For TCP flows, SIP is the IP of the host that sends the first SYN packet. Similarly, for UDP flows, the SIP is the source IP in the first packet observed from the stream of UDP packets exchanged between two hosts. Unidirectional flows were removed from the data set. Because of the network topology, all traffic to and from a host within the dormitories passed through our monitor. EID indicates whether the payload of the packet matches the P2P byte-string signature. This identification was performed with byte signatures similar to those available online [16]. While this signature approach has a low false alarm probability, it has a high false negative probability, as we had signatures from only a limited number of P2P protocols. Specifically, we were able to detect BitTorrent, Gnutella, and FastTrack/Kazaa applications.

We represent a flow as a tuple of (SIP, DIP, SP, DP, PR, ST). Thus, a flow ϕ has components ϕ_{SIP}, ϕ_{DIP}, ϕ_{SP}, ϕ_{DP}, ϕ_{PR}, and ϕ_{ST}. Sets of flows are denoted by capital letters. For example, $\Phi = \{\phi | \phi_{PR} = tcp, \phi_{DP} = 80\}$ is the set of flows, which use TCP and have destination port 80, extracted from all flows.

III. Identification Methods

A. Overview of methods

Two classes of methods are utilized. The first class of methods identifies some initial P2P flows (i.e., flows between two instances of a P2P application). The second class of methods uses these initial flows to detect more P2P flows. This second class of methods is applied iteratively. Given a set of P2P flows, these methods detect other P2P flows. With these new P2P flows found, the methods can be reapplied, discovering more flows. Hence, these methods are iteratively applied until convergence. These second set of methods give rise to an graphical representation of P2P flows, which will be further discussed in Section V. In the next two sections two methods are provided to find an initial set of P2P flows. Then, sections III-D-III-F present three methods that can be used iteratively to discover P2P flows. Section III-G summarizes the methods.

B. Degree-Based P2P Detection

One of the hallmarks for P2P file sharing is that a single file is downloaded by downloading different pieces from multiple hosts. Thus, one end-host usually connects to multiple peers simultaneously and often many peers will connect to a single host. We define the *out-degree* to be the number of hosts that a host connects to and the *in-degree* to be the number of hosts that connect to a host.

Out-degree can be determined as follows. Suppose that a host with address IP initiates a connection ϕ to host ϕ_{DIP}, with $\phi_{PR} \in \{tcp, udp\}$, and over port ϕ_{DP} where $\phi_{DP} \notin WL$, which is a white list of ports[1]. Then, the out-degree from this host at time t is

$$OD\left(IP, t, WL, T\right) : \#\left\{\phi_{DIP} | \phi_{SIP} = IP, \phi_{DP} \notin WL, \right. \\ \left. \phi_{PR} \in \left\{tcp, udp\right\}, \left|\phi_{ST} - t\right| < T\right\},$$

where $\#A$ is the number of unique elements in set A. Note that the out-degree depends on the current time, t, and the time window T.

In-degree is defined similarly,

$$ID\left(IP, t, WL, T\right) := \#\left\{\phi_{SIP} | \phi_{DIP} = IP, \phi_{DP} \notin WL, \right. \\ \left. \phi_{PR} \in \left\{tcp, udp\right\}, \left|\phi_{ST} - t\right| < T\right\}.$$

Note that using in-degree as a P2P detection method is challenging since typical servers also have high in-degree. However, using it should not be neglected since large in-degree is also a possible indication of P2P activity. To reduce false positive, it is critical that the white list (especially for servers) be complete[2]. The times when a host with address IP is possibly engaged in P2P activity is given by

$$PAC_{T,R}\left(IP\right) := \{t | \alpha \times ID\left(IP, t, WL, T\right) + \\ OD\left(IP, t, WL, T\right) > R\}.$$

Because of false positives, we reduce the effect of in-degree by utilizing $\alpha < 1$ and do not use PAC alone to detect P2P flows. Instead, we use it along with a method discussed in the next section.

As compared to in-degree, out-degree is a better indication of P2P activity. For example, as mentioned, servers often have high in-degree, leading to false positives. Moreover, in-degree may be zero when some P2P users are behind NATs or

[1]The white list we used was TCP ports $<$ 1024, and TCP posts 1025, 1755, 2967, 3268, 3724, 5050, 5190, 5351, and 8080, and UDP ports $<$ 1024, and UDP ports 1755, 3268, 3724, and 5351.

[2]In tightly control enterprises or in military networks, all allowed applications are known, and hence white listing might be complete.

firewalls, and hence cannot accept incoming connections. On the other hand, out-degree has some drawbacks. For example, it is possible that the host is engaging in P2P activity while at the same time using some application that is not included in the white list. While such false positives can be eliminated only with a better white list, we note that these false positives require two events to occur, the simultaneous use of P2P and some other application and that other application is not on the white list.

Considering the above, we define a detector based on out-degree as follows.

$$AC_{T,R} := \{\phi | \phi_{ST} \in \{t | OD(\phi_{SIP}, t, WL, T) > R\} \cup \{t | OD(\phi_{DIP}, t, WL, T) > R\}, \phi_{DP} \notin WL\},$$

where $\{t | OD(\phi_{SIP}, t, WL, T) > R\}$ is the set of times where each host with IP address equal to ϕ_{SIP} has a large out–degree. Thus, $\{t | OD(\phi_{SIP}, t, WL, T) > R\} \cup \{t | OD(\phi_{DIP}, t, WL, T) > R\}$ is the set of times where either the source or destination of ϕ has large out-degree.

The parameters R and T can be set in a conservative fashion, which will miss many P2P flows, and yet, after applying the second class of methods, will still detect all P2P flows.

C. Known Port

Often a P2P application has a specified default port. These ports are often published on the web or can easily be determined by monitoring the application. We call these ports *known P2P ports* (KP)[3]. Of course, unknown P2P applications might use unknown ports. Moreover, known P2P applications can use different ports besides the standard know P2P ports. In these cases, P2P flows can be detected using degree-based P2P detection method and the second class of methods. The detection of P2P applications where the port is unknown is also discussed in Section V-B.

We have found that using a list of P2P ports to detect flows results in some false positives when other applications use the ports. For example, GroupWise uses randomly generated UDP source ports. These ports sometimes match known P2P ports. These false positives can be reduced by requiring both the usage of a known P2P port and P2P behavior, as described in the previous section. Thus, we have the following detector,

$$KPF_{T,R} := \{\phi \ | \phi_{DP} \in KP, \phi_{ST} \in TP\} \cup \{\phi \ | \phi_{SP} \in KP, \phi_{PR} = udp, \phi_{ST} \in TP\},$$

where TP is $PAC_{T,R}(\phi_{SIP}) \cup PAC_{T,R}(\phi_{DIP})$. Note that if P2P applications use known P2P ports, TCP uses known ports for listening, while UDP may use the ports for listening and for initiating connections (e.g., UDP-based host caching protocol in Gnutella [17]). For this reason, $KPF_{T,R}$ is composed of two terms. Also, the P2P activity detected by the PAC method can be at the source or the destination, hence, we use the union $PAC_{T,R}(\phi_{SIP}) \cup PAC_{T,R}(\phi_{DIP})$.

[3]Default ports used for each P2P application in our research are as follows. BitTorrent: 6881~6889, 6969, 2710, Gnutella: 6346~6349, Edonkey: 2323, 3306, 4242, 4500, 4501, 4661~4674, 4677, 4678, 7778, 1214, 1215, 1331, Freenet: 19114, 8081, and Soulseek: 2234, 5534.

D. Repeated Communication between Known P2P Peers

Once two hosts are identified as P2P peers, we assume that any other communication between the hosts is also a P2P flow since in [12] nearly 99.5 % or more of flows between two hosts is from a single application. This detection method is defined by

$$G(\theta) := \{\phi | \phi_{DIP} = \theta_{DIP}, \phi_{SIP} = \theta_{SIP}\} \cup \{\phi | \phi_{DIP} = \theta_{SIP}, \phi_{SIP} = \theta_{DIP}\},$$

where θ is the initial P2P flow. This method is extended to take a set of known P2P flows as an input argument, $G(\Theta) := \Theta \cup \bigcup_{\theta \in \Theta} G(\theta)$.

E. P2P Port Identification and Port-Based P2P Detection

The information of P2P ports that hosts are using leads to the detection of more other P2P flows. For example, the incoming flows to the P2P port identified are also likely to be P2P flows. The identification of P2P ports is straightforward from a confirmed P2P flow θ. If a confirmed P2P flow θ is formed by TCP connection from peer A to peer B (i.e., peer A sent the first SYN packet), we identify a peer B's P2P port that is the destination port θ_{DP} since the P2P application on peer A assigns the destination port the peer B's incoming P2P port. On the other hand, if θ is formed by UDP connection, we identify not only a peer B's P2P port ($= \theta_{DP}$) but also a peer A's P2P port ($= \theta_{SP}$) if the P2P application on peer A uses UDP for P2P network controlling purposes. We observed many times that the peer A's UDP source port is the same port open for incoming P2P flows.

Given a TCP (or a UDP) confirmed P2P flow θ, three types of flows will be identified based on the identified P2P port θ_{DP} via

$$M(\theta) := \{\phi | \phi_{DIP} = \theta_{DIP}, \phi_{DP} = \theta_{DP}, \phi_{PR} = tcp\} \cup \{\phi | \phi_{DIP} = \theta_{DIP}, \phi_{DP} = \theta_{DP}, \phi_{PR} = udp\} \cup \{\phi | \phi_{SIP} = \theta_{DIP}, \phi_{SP} = \theta_{DP}, \phi_{PR} = udp\} \cup$$

The first two types identify the incoming flows to $\{\theta_{DIP}, \theta_{DP}\}$ and the third the outgoing flows from it. On the other hand, given a UDP P2P flow θ, six types of flows will be identified based on the identified P2P port θ_{DP} as well as θ_{SP}. Therefore, given any P2P flow, other P2P flows can be detected via

$$H(\theta) := M(\theta) \cup \{\phi | \phi_{DIP} = \theta_{SIP}, \phi_{DP} = \theta_{SP}, \phi_{PR} = tcp, \theta_{PR} = udp\} \cup \{\phi | \phi_{DIP} = \theta_{SIP}, \phi_{DP} = \theta_{SP}, \phi_{PR} = udp, \theta_{PR} = udp\} \cup \{\phi | \phi_{SIP} = \theta_{SIP}, \phi_{SP} = \theta_{SP}, \phi_{PR} = udp, \theta_{PR} = udp\}.$$

One complication of this method is that an application might stop using one port and start using another port. More importantly, some other applications might start using a port that was once used by the P2P application. Thus, for a given P2P flow, we assume that the ports identified by this flow are used by the P2P application for no more than T seconds. For example, suppose that θ is a P2P flow and flow ϕ uses the same identified P2P port as θ. Then, in order for θ to be used to identify ϕ as a P2P flow, we require that $|\theta_{ST} - \phi_{ST}| \leq T$.

Note that $|\phi_{ST} - \theta_{ST}| \leq T$ is omitted for each union set in $H(\theta)$ but assume that it is there and hence $H(\theta)$ becomes $H_T(\theta)$.

F. Triggered P2P Detection

Once a P2P flow is detected, it is likely that other flows started at the same time are also P2P flows. In fact, the behavior-based methods in Section III-B are based on the tendency of P2P flows to start at approximately the same time. Based on this observation, we assume that given a P2P flow, any flow that starts at approximately the same time is also a P2P flow, where we say that flows start at the same time if they start within one second (the time resolution of our measurements). Thus, we define the following detection given a P2P flow θ.

$$\begin{aligned} TA(\theta) := \{\phi | \phi_{SIP} = \theta_{SIP}, \phi_{DP} \notin WL, |\phi_{ST} - \theta_{ST}| \leq 1\} \cup \\ \{\phi | \phi_{SIP} = \theta_{DIP}, \phi_{DP} \notin WL, |\phi_{ST} - \theta_{ST}| \leq 1\} \end{aligned}$$

Note that other thresholds could be used so that flows that start within T seconds of a P2P flow are declared to be P2P flows for some T. Or, by combining the methods of Section III-B, we could insist that the in or out-degree exceeds some threshold. However, we have found that these approaches have little impact on the total number of flows detected, especially after the other methods are also applied.

G. Summary of Detection Methods

Five methods for detecting P2P flows are described above.

- Section III-B defines $AC_{T,R}$, which is the set of flows that start when the destination or source of the flow has a large out-degree.
- Section III-C defines $KPF_{T,R}$, which is the set of flows that use a known P2P port while the source or the destination of the flow has a high in-degree or out-degree.
- Section III-D defines $G(\theta)$, which tags all flows between the end hosts of flow θ to be a P2P flow.
- Section III-E defines $H_T(\theta)$ which tags all flows that use the same ports as θ to be a P2P flow, where use of a port depends on the transport protocol.
- Section III-F defines $TA(\theta)$, which tags all flows that start within one second of θ as a P2P flow.

Note that $G(\theta)$, $H_T(\theta)$ and $TA(\theta)$ can be applied repeatedly. That is, if $H(\theta)$ finds some new P2P flows, it can be applied to each of the newly found flows. To this end, let $GH(\Theta) = G(H(\Theta))$ and let $GH^k(\Theta) = GH\left(GH^{k-1}(\Theta)\right)$, where $GH^1(\Theta) = GH(\Theta)$. Similarly, define $TGH(\Theta) = T(G(H(\Theta)))$ and define TGH^k similarly. Finally, we iterate GH and TGH until convergence via

$$GH^{\infty}(\Theta) := \lim_{k\to\infty} GH^k(\Theta)$$
$$TGH^{\infty}(\Theta) := \lim_{k\to\infty} TGH^k(\Theta).$$

Methods GH^{∞} and TGH^{∞} require an initial set of P2P flows. This initial set of flows can be $AC_{T,R}$, $KPF_{T,R}$ or the union of the two. That is, the P2P flows detected can be $GH^{\infty}(AC_{T,R})$, $GH^{\infty}(KPF_{T,R})$, or $GH^{\infty}(AC_{T,R} \cup KPF_{T,R})$, and GH^{∞} can be replaced with TGH^{∞}. The behaviors of these methods are discussed in the next sections.

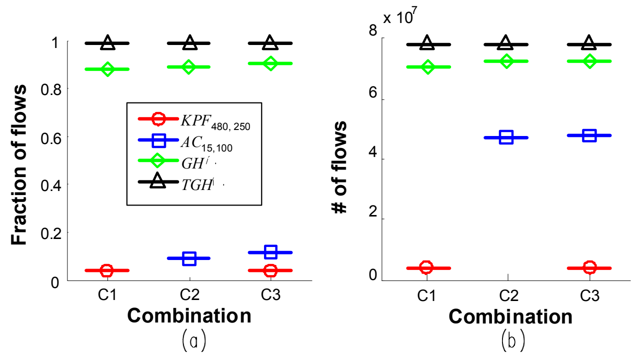


Fig. 1. Flows detected with different combinations of methods. C1 applies $KPF_{T,R}$ and then $GH^{\infty}(KPF_{T,R})$ and finally $TGH^{\infty}\left(GH^{\infty}(KPF_{T,R})\right)$. C2 is similar, but starts with $AC_{T,R}$ while C3 applies $KPF_{T,R}$ then $AC_{T,R}$ and then GH^{∞} and TGH^{∞}. (a) shows the fraction of flows detected that were also detected by the signature method, while (b) shows the total number of all flows detected, which includes flows that were also detected by signatures and flows that were not detected by signatures.

IV. Numbers of P2P Flows Detected

The data used for this analysis includes results from byte-string signatures for some P2P applications. However, as discussed in Section II, the signatures only detect a subset of P2P applications and only consider TCP flows. Thus, these signatures can be only used to determine false negatives (i.e., we can determine if our methods miss P2P flows). Hence, we consider the total number of flows detected as well as the fraction of flows detected by the signature method that are also detected by our methods.

Figure 1(a) shows the fraction of flows detected by signatures that were also detected by our methods, while Figure 1(b) shows the total number of flows detected. Here we use $AC_{T,R}$ with $T = 15$ sec and $R = 100$ and $KPF_{T,R}$ with $T = 480$ sec and $R = 250$. Observe that $KPF_{480,250}$ discovers far fewer flows than $AC_{15,100}$. The combination of detection methods labeled C1 starts with $KPF_{480,250}$, then applies GH^{∞} to determine $GH^{\infty}(KPF_{480,250})$, and finally applies TGH^{∞} to determine $TGH^{\infty}(GH^{\infty}(KPF_{480,250}))$. The combination of methods labeled C2 and C3 are similar, but start with initial sets of flows $AC_{15,100}$ and $KPF_{15,100} \cup AC_{15,100}$, respectively.

While the initial sets of flows are different, after applying GH^{∞}, the number of flows detected is approximately the same. Note that this is the case even if the initial set of flows is $KPF_{480,250}$ which is a far smaller set of flows than $AC_{15,100}$. From Figure 1(a) we see that about 90% of all flows that were detected with the byte-string signature were also detected by GH^{∞}. Nearly all remaining flows are detected by applying TGH^{∞}, which detects 99% of the flows that were detected with the byte-string signature.

While false positives cannot be directly observed, we know that the byte-string signature methods have very low false positives probabilities. With this assumption, Figure 1 provides some indication of false positives by comparing the proportions of flows detected in Figure 1(a) to the corresponding proportions in Figure 1(b). Specifically, given a set of flows Θ, let $S(\Theta)$ be the set of P2P flows in Θ detected by the signature method. If we assume that the signature method detects a P2P flow at random with probability α, then, if Θ is an arbitrary set of P2P flows, we have $\alpha|\Theta| = |S(\Theta)|$, where $|\Theta|$ is the number of flows in Θ. On the other hand, if Θ is a set of flows that includes flows that are not P2P flows, then $\alpha|\Theta| > |S(\Theta)|$. Generally, we have $\alpha \geq |S(\Theta)| / |\Theta|$, with equality if and only if Θ only contains P2P flows. Therefore, if our detection methods have no false positives, then

$$\begin{aligned}\alpha &= |S(KPF_{480,250})| / |KPF_{480,250}| \\ &= |S(GH^{\infty}(KPF_{480,250}))| / |GH^{\infty}(KPF_{480,250})| \\ &= |S(TGH^{\infty}(GH^{\infty}(KPF_{480,250})))| / \\ &\quad |TGH^{\infty}(GH^{\infty}(KPF_{480,250}))| .\end{aligned}$$

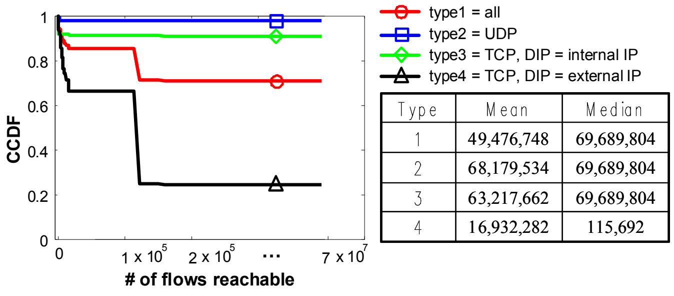
In fact, by comparing the values in combination $C1$ in Figures 1 (a) and (b), we see that these ratios are nearly the same, meaning that Θ detected by our methods nearly does not include non-P2P flows. While this provides some indication of low false alarm probability, this argument relies on the assumption that the signature method detects P2P flows at random, which is not the case (e.g., the signature method does not detect any UDP flows)

V. Graphical Aspects of P2P Flow Detection

A. Large Connected Component

In the previous section we saw that $TGH^{\infty}(KPF_{480,250}) \approx TGH^{\infty}(AC_{15,100})$. This behavior can be understood by modeling P2P flow detection as traversing vertices on a graph. We define the P2P flow graph where each vertex is a P2P flow. The detection methods G, H, and TA define edges between vertices. By the definitions given in Section III-A, it is clear that these edges are bidirectional. Thus, $GH^{\infty}(\theta)$ is the set of flows that are reachable from the flow θ when edges are constructed from G and H. Similarly, $GH^{k}(\theta)$ is the set of flows that are within k hops of θ. $TGH^{\infty}(\theta)$ and $TGH^{k}(\theta)$ can be interpreted similarly but where edges are defined by G, H and TA. For example, $GH^{\infty}(KPF_{480,250})$ is the set of vertices reachable from some flow in $KPF_{480,250}$ and $GH^{\infty}(AC_{15,100})$ is the set of flows reachable from some flow in $AC_{15,100}$

Figure 2 shows the complementary cumulative distribution function (CCDF) of $|GH^{\infty}(\theta)|$ for different initial flows θ selected from the set of P2P flows $TGH^{\infty}(KPF_{480,250} \cup AC_{15,100})$. The key observation is that $|GH^{\infty}(\theta)|$ is typically very large. This implies that when edges are defined by G and H, a large number of flows are reachable from θ. This also implies that the graph of P2P flows with edges defined by G and H has a large connected component. Since edges are bidirectional, as long as the initial flow is within this large connected component, we will discover all the other flows in this component. Given



Type	Mean	Median
1	49,476,748	69,689,804
2	68,179,534	69,689,804
3	63,217,662	69,689,804
4	16,932,282	115,692

Fig. 2. CCDF of the number of P2P flows found given a single P2P flow and when $GH^{\infty}(\theta)$ is used to find P2P flows. The number of flows found depends on whether the given P2P flow uses UDP or TCP and whether the destination is within the monitored network when TCP is considered.

the size of this large connected component, a randomly selected flows is likely to be within this component. Thus even though $|KPF_{480,250}| << |AC_{15,100}|$, most of these flows are within the same connected component, and hence $GH^{\infty}(KPF_{480,250}) \approx GH^{\infty}(AC_{15,100})$.

Figure 2 shows that the number of flows reached depends on the type of initial flow. For example, an arbitrarily selected UDP P2P flow is in the large connected component with probability 0.98. As discussed in Section III-E, the identification of a single P2P flow that uses UDP identifies six other types of flows. Thus, we expect that the identification of a P2P flow over UDP would lead to the detection of many flows. Similarly, TCP flows where the destination is within the monitored network (i.e., an internal destination) are in the large connected component with probability 0.90. Hence, these flows are also useful for detecting a large number of P2P flows.

On the other hand, TCP flows where the destination is not in the monitored network (i.e., an external destination) are typically not in the large connected component. Nonetheless, around 20% of these flows are in the large connected component. Note that for this type of TCP flow, we might not observe any other flows to this destination. Thus, the knowledge of the destination's P2P port might not yield any other P2P flows. However, in some cases the destination host is a super node.

An important implication of the existence of a large connected component is that discovering flows in this component is relatively easy. Thus even if we conservatively set the parameters used by methods for the initial set of P2P flows to avoid false positives, those methods will have a good chance to find flows in the large connected component.

B. Connected P2P Flows

Note that the algorithms presented in Section III detect several different P2P applications. However, flows from these applications appear in the same connected component. The reason for this behavior is that some hosts act as bridges between different applications. Specifically, hosts that run applications such as LimeWire [18] support multiple P2P protocols simultaneously. These applications often use the same port for multiple P2P protocols. Thus, method H will discover this port and then find flows for multiple P2P protocols that use this port. As a result, H will construct an edge between

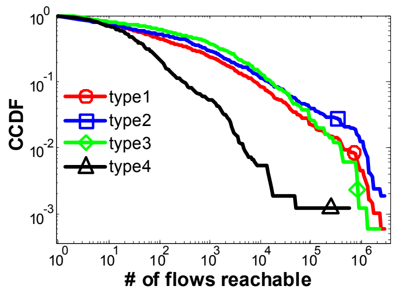


Fig. 3. The degree distribution of P2P flows (i.e., the number of flows detected from $GH^1(\theta)$).

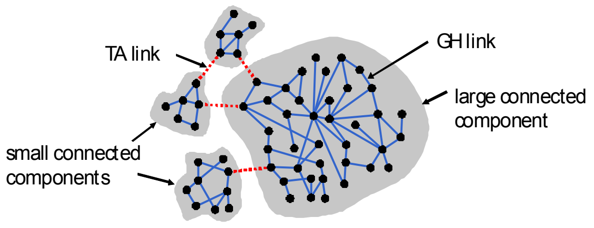


Fig. 4. The graph of P2P flows has a large connected component where edges within the component are defined by GH. There are many other connected components. Edges between the large connected component and these other components can be formed with TA.

flows that use different P2P protocols. Consequently, even if the list of known ports does not include new P2P applications, the existence and continued maintenance of applications that support multiple P2P protocols allow the methods presented here to detect new P2P applications.

C. Vertex Degree and Diameter

Figure 3 shows the degree distribution of the vertices in the graph. Note that there exist vertices with high degree. That is, for some flows, there are many flows that can be detected with a single application of GH. For example, there exists a flow θ such that $\left|GH^1(\theta)\right| > 10^5$. These flows play a critical role in the large connected component. Super nodes are one source of high degree vertices. For example, we found that nearly 1000 different hosts within the monitored network with 3161 hosts connected to a single super node. Moreover, hosts often connect to several super nodes. The result is that when a single node's port is detected, we can detect several super nodes and their P2P ports. These super nodes lead us to a large number of flows as well as to other hosts' ports. These ports lead us to more super nodes and so on. After a few iterations, all super nodes are found.

D. Disconnected Components

Not all of the P2P flows are within the large connected component where edges are defined by G and H. Instead, some flows can only be reached by applying TGH^∞. Specifically, we can model the graph of P2P flows where there are two types of edges. The first type is defined by G or H, and the second type is defined by TA. Recall that by including the edges defined by TA, we detect 99% of the flows detected by the byte-string signatures whereas without TA, approximately 90% of P2P flows were detected. Figure 4 shows a visualization of the P2P flow graphs. Specifically, most flows are within the large connected component and a few flows are within the small connected component. The edges within a component are defined by G and H. However, it is the edge defined by TA that connect any two components.

VI. Conclusion

The paper proposes a novel identification method based on iteratively detecting P2P flows based on P2P ports and other methods. The results show effectiveness of our method, namely, P2P flows and users of P2P applications can be quickly and easily identified. The iterative methods require an initial set of P2P flows, which is expanded during each iteration. We find that even with a small number of initial P2P flows, the iterative methods can detect most P2P flows. Considering a graphical view of P2P detection, we see that most P2P flows are within the same connected component even if the flows are from different P2P applications. Thus, the identification of one flow in this connected component leads to the detection of all flows in this component. This behavior greatly simplifies the tasks such as selecting parameters of detection methods and generating lists of ports used by P2P applications.

References

[1] King, A.B.: Speed Up Your Site: Web Site Optimization. New Riders Press (2003)
[2] Morparia, J.: Peer-to-peer botnets: Analysis and detection. Master's thesis, San Jose State University (2008)
[3] paloalto Networks, http://www.paloaltonetworks.com.
[4] Nevis Networks, http://www.nevisnetworks.com.
[5] Karagiannis, T., Broido, A., Faloutsos, M., Claffy, K.: Transport layer identification of P2P traffic. In: IMC'04. (2004) 121–134
[6] Bartlett, G., Heidemann, J., Papadopoulos, C., Pepin, J.: Estimating P2P traffic volume at USC. Technical Report IST-TR-645, USC/Information Sciences Institute (June 2007)
[7] Wagner, A., Dubendorfer, T., Hammerle, L., Plattner., B.: Flow-based identification of P2P heavy-hitters. In: ICISP. (2006)
[8] Erman, J., Arlitt, M., Mahanti, A.: Traffic classification using clustering algorithms. In: MineNet'06. (2006)
[9] Wang, R., Liu, Y., Yang, Y.X., Wang, H.L.: A new method for P2P traffic identification based on support vector machine. In: AIML. (2006) 13–15
[10] Zander, S., Nguyen, T., Armitage, G.: Automated traffic classification and application identification using machine learning. In: LCN'05. (2005)
[11] Hua, Y., Chiu, D.M., Lui, J.C.: Profiling and identification of P2P traffic. Computer Networks **53** (2009) 849–863
[12] Jin, Y., Duffield, N., Haffner, P., Sen, S., Zhang, Z.L.: Inferring applications at the network layer using collective traffic statistics. In: ITC 2010. (2010)
[13] Karagiannis, T., Papagiannaki, K., Faloutsos, M.: BLINC: Multilevel traffic classification in the dark. In: SIGCOMM'05. (2005)
[14] Iliofotou, M., Pappu, P., Faloutsos, M.: Network monitoring using traffic dispersion graphs (TDGs). In: IMC'07. (2007)
[15] Jin, Y., Sharafuddin, E., Zhang, Z.L.: Unveiling core network-wide communication patterns through application traffic activity graph decomposition. In: SIGMETRICS'09. (2009)
[16] L7-filter, http://l7-filter.sourceforge.net.
[17] Gnutella Specification, http://wiki.limewire.org/index.php?title=GDF.
[18] LimeWire, http://www.limewire.com.