Graption: Graph-based P2P Traffic Classification at the Internet Backbone

Marios Iliofotou (UCR), Hyun-chul Kim (CAIDA and SNU), Michalis Faloutsos (UCR),
Michael Mitzenmacher (Harvard University), Prashanth Pappu (Conviva), and George Varghese (UCSD)

***Abstract*—Monitoring network traffic and classifying applications are essential functions for network administrators. Current traffic classification methods can be grouped in three categories: (a) flow-based (e.g., packet sizing/timing features), (b) payload-based, and (c) host-based. Methods from all three categories have limitations, especially when it comes to detecting new applications, and classifying traffic at the backbone. In this paper, we propose the use of Traffic Dispersion Graphs (TDGs) to remedy these limitations. 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. Using TDGs, we develop an application classification framework dubbed Graption (*Graph*-based classific*ation*). Our framework provides a systematic way to classify traffic by using information from the network-wide behavior and flow-level characteristics of Internet applications. As a proof of concept, we instantiate our framework to detect P2P traffic, and show that it can identify 90% of P2P flows with 95% accuracy in backbone traces, which are particularly challenging for other methods.**

I. Introduction

An important task when monitoring and managing large networks is classifying flows according to the application that generates them. Such information can be utilized for network planning and design, QoS and traffic shaping, and security. In particular, detecting P2P traffic is a potentially important problem for ISPs that want to manage such traffic, and for specific groups such as the entertainment industry in legal and copyright disputes. Detecting P2P traffic also has particular interest since it represents a large portion of the Internet traffic, with more than 40% of the overall volume in some networks [17].

Most current application classification methods can be naturally categorized according to their level of observation: payload-based signature-matching methods [26], [24], flow-level statistical approaches [9], [28], or host-level methods [22], [35]. Each existing approach has its own pros and cons, and no single method clearly emerges as a winner. Relevant problems that need to be considered include identifying applications that are new, and thus without a known profile; operating at backbone links [23], [22]; and detecting applications that intentionally alter their behavior. Flow-level and payload-based classification methods require per application training and will thus not detect P2P traffic from emerging protocols. Behavioral-host-based approaches such as BLINC [22] can detect traffic from new protocols [22], but have weak performance when applied at the backbone [23]. In addition, most tools including BLINC [22] require fine-tuning and careful selection of parameters [23]. We discuss the limitations of previous methods in more detail in §IV.

In this paper, we use the network-wide behavior of an application to assist in classifying its traffic. To model this behavior, we use graphs where each node is an IP address, and each edge represents a type of interaction between two nodes. We use the term **Traffic Dispersion Graph** or **TDG** to refer to such a graph [16]. Intuitively, with TDGs we enable the detection of network-wide behavior (e.g., highly connected graphs) that is common among P2P applications and different from other traffic (e.g., Web). While we recognize that some previous efforts [5], [8] have used graphs to detect worm activity, they have not explored the full capabilities of TDGs for application classification. This paper is an extension of a workshop paper [15] and the differences will be clarified in the related work section (§IV).

We propose a classification framework, dubbed **Graption** (*Graph*-based classific*ation*), as a systematic way to combine network-wide behavior and flow-level characteristics of network applications. Graption first *groups* flows using flow-level features, in an unsupervised and agnostic way, i.e., without using application-specific knowledge. It then uses TDGs to *classify* each group of flows. As a proof of concept, we instantiate our framework and develop a P2P detection method, which we call **Graption-P2P**. Compared to other methods (e.g., BLINC [22]), Graption-P2P is easier to configure and requires fewer parameters.

The highlights of our work can be summarized in the following points:

- **Distinguishing between P2P and client-server TDGs.** We use real-world backbone traces and derive graph theoretic metrics that can distinguish between the TDGs formed by client-server (e.g., Web) and P2P (e.g., eDonkey) applications. Section: §II-B.
- **Practical considerations for TDGs.** We show that even a single backbone link contains enough information to generate TDGs that can be used to classify traffic. In addition, TDGs of the same application seem fairly consistent across time. Section: §II-C.
- **High P2P classification accuracy.** Our framework instantiation (Graption-P2P) classifies 90% of P2P traffic with 95% accuracy when applied at the backbone. Such traces are particularly challenging for other methods. Section: §III-B2.
- **Comparison with a behavioral-host-based method.** Graption-P2P performs better than BLINC [22] in P2P identification at the backbone. For example, Graption-

TABLE I
Set of backbone traces from the Cooperative Association for Internet Data Analysis (CAIDA). Statistics for the TR-ABIL trace, are reported only for the first five-minute interval since IPs were anonymized differently at each five-minute sample.

Name	*Date/Time*	*Duration*	*Flows*
TR-PAY1	2004-04-21/17:59	1 hour	38,808,604
TR-PAY2	2004-04-21/19:00	1 hour	37,612,752
TR-ABIL	2002-09 /(N/A)	1 month	2,057,729

P2P identifies 95% of BitTorrent traffic while BLINC identifies only 25%. Section: §III-C.

- **Identifying the unknown**. Using Graption, we identified a P2P overlay of the Slapper worm. The TDG of Slapper was never used to train our classifier. This is a promising result showing that our approach can be used to detect both known and unknown P2P applications. Section: §III-D.

The rest of the paper is organized as follows. In §II we define TDGs, and identify TDG-based metrics that differentiate between applications. In §III we present the Graption framework and our instantiation, Graption-P2P. In §V we discuss various practical issues. In §IV we discuss related work. Finally, in §VI we conclude the paper.

II. Studying the TDGs of P2P Applications

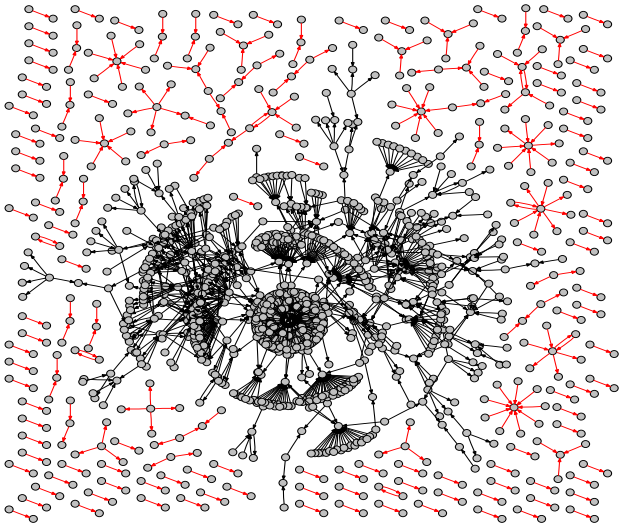
A. Traffic Dispersion Graphs (TDGs)

Definition. Throughout this paper, we assume that packets can be grouped into flows using the standard 5-tuple {`srcIP, srcPort, dstIP, dstPort, protocol`}. Given a group of flows S, collected over a fixed-length time interval, we define the corresponding TDG to be a directed graph $G(V, E)$, where the set of nodes V corresponds to the set of IP addresses in S, and there is a link $(u, v) \in E$ from u to v if there is a flow $f \in S$ between them.

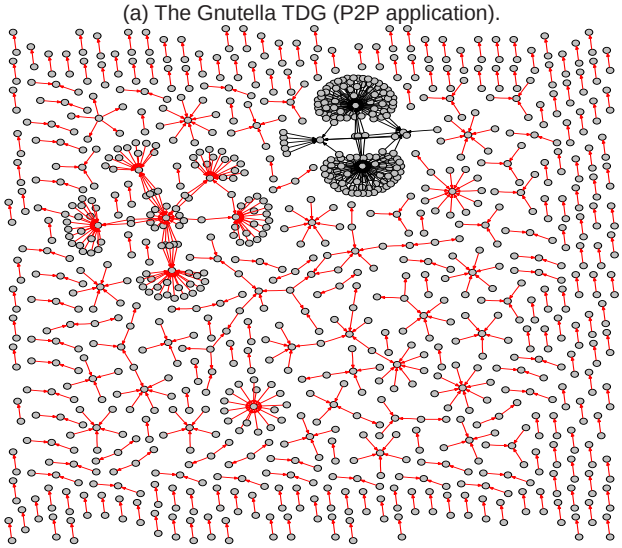
In this paper, we consider bidirectional flows. We define a TCP flow to start on the first packet with the SYN-flag set and the ACK-flag not set, so that the initiator and the recipient of the flow are defined for the purposes of direction. For UDP flows, direction is decided upon the first packet of the flow.

Visualization Examples. In Figure 1, we show TDG examples from two different applications. In order to motivate the discussion in the rest of the paper, we show the contrast between a P2P and a client-server TDG. From the figure we see that P2P traffic forms more connected and more dense graphs compared to client-server TDGs. In §II-B, we show how we can translate the visual intuition of Figure 1 into quantitative measures that can be used to classify TDGs that correspond to different applications.

Data Set. To study TDGs, we use three backbone traces from a Tier-1 ISP and the Abilene (Internet2) network. These traces are summarized in Table I. All data are IP anonymized and contain traffic from both directions of the link. The TR-PAY1 and TR-PAY2[1] traces were collected from an OC48 link of a commercial US Tier-1 ISP at the Palo Alto Internet eXchange (PAIX). To the best of our knowledge, these are the most recent backbone traces with payload that are available to researchers by CAIDA [4]. The TR-ABIL trace is a publicly available data set collected from the Abilene (Internet2) academic network connecting Indianapolis with Kansas City. The Abilene trace consists of five randomly selected five-minute samples taken every day for one month, and covers both day and night hours as well as weekdays and weekends.



(a) The Gnutella TDG (P2P application).



(b) The HTTPS TDG (client-server application).

Fig. 1. Two TDG visualization contrasting a P2P with a client-server application. Largest component is with bold edges - hence the graphs are best viewed on a computer screen or a colored print-out.

Extracting Ground Truth. We used a Payload-based Classifier (PC) to establish the ground truth of flows for the TR-PAY1 and TR-PAY2 traces. Both traces contain up to 16 bytes of payload in each packet, thereby allowing the labeling of flows using the signature matching techniques described in [23], [22]. Running the PC over the TR-PAY1 and TR-PAY2 traces we find 14% of the traffic to be P2P, 27% Web, 7%

[1]The authors thank CAIDA for providing this set of traffic traces. Additional information for these traces can be found in the DatCat, Internet Measurement Data Catalog [7], indexed under the label "PAIX".

DNS, and the rest to belong to other applications. A detailed application breakdown is summarized in Table II. For our study, we remove the 1% of traffic that remained unknown and the 28% that contained no payload.

B. Identifying P2P TDGs

Identifying the right metrics to compare graph structures is a challenging question that arises in many disciplines [27]. Our approach is to consider several graph metrics, each capturing a potentially useful characteristic, until a set of metrics is found that distinguishes the target graphs.

To select an appropriate set of metrics, we generate a large number of TDGs using all our traces (Table I), thus observing TDGs over two different locations at the backbone. For the TR-PAY1 and TR-PAY2 traces, we use the payload-based classifier (PC) in order to select which flows belong to each TDG. Since the TR-ABIL trace does not have any payload information, we use port numbers [23] to assign flows to applications. We can use port numbers for the TR-ABIL trace since it was collected in 2002 where most P2P applications used their default port numbers [12], [20]. We only use the TR-ABIL trace to verify our TDG observations over a second location in the backbone and we do not use it in the final evaluation of our classifier. By using the month-long TR-ABIL trace, we can study the consistency of TDGs over different times of the day and over weekdays and weekends.

We observe TDGs over 5-minute intervals. This interval length gives good classification results and stability of TDG metrics over time as we show later in this section. For each TDG we generate a diverse set of metrics. Our metrics capture various aspects of TDGs including the degree distribution, degree correlations, connected components, and distance distribution. For additional details about these metrics we refer the reader to [16], [27].

To select the right set of metrics we use various graph visualizations and trial and error. Finding a less ad hoc approach is beyond the scope of this work. Two TDG visualization examples are shown in Figure 1. We see that FastTrack (P2P) has a denser graph than HTTPS, or a higher **average degree**, where the average node degree $\bar{k}$ is given by $\bar{k} = 2|E|/|V|$.

TABLE II
Application breakdown for TR-PAY1 and TR-PAY2 traces. Values in parenthesis show the percentage of each P2P application over the entire identified P2P traffic (All-P2P).

Name	*% in Flows*	*% in Bytes*	*% in Pakts*
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
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	20.66	44.42	36.27

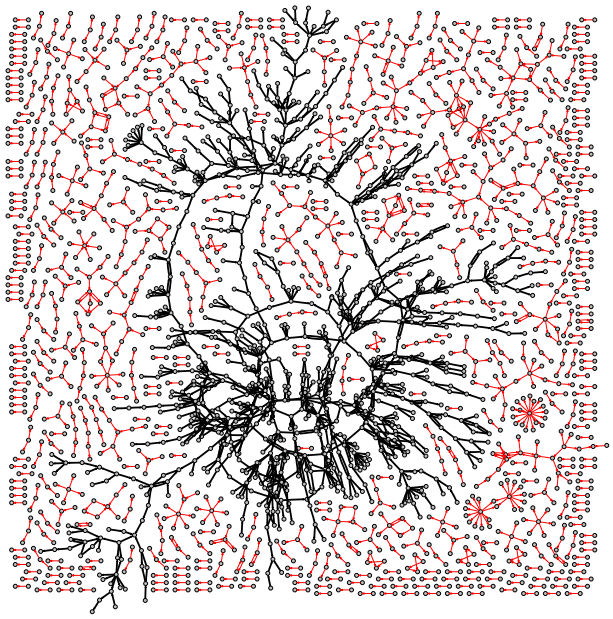


Fig. 2. Graphical representation of the BitTorrent TDG with 3,000 IPs, showing the formation of long paths connecting P2P hosts. The largest connected component is highlighted with darker color edges.

We utilize two other metrics that capture the directionality of the edges in the graph and the distances between nodes. The directionality is useful since we know that pure clients only initiate traffic, pure servers should never initiate traffic, and that some P2P nodes play both roles. To capture this quantitatively, we define **InO** to be the percentage of nodes in the graph that have both incoming and outgoing edges.

The distance between two nodes is defined as the length of their shortest path in the graph. The diameter of a graph is defined as the maximum distance between all pairs of nodes, which is sensitive as a metric [27]. For a more robust metric, we use the **effective diameter** (EDiam), which we define as the 90-th percentile of all pairwise distances in the graph.

We show the high graph diameter of the BitTorrent P2P TDGs in Figure 2. The graph represents the BitTorrent interactions between 3,000 nodes from the TR-PAY1 trace. The visualization is suggestive of a spider's web with large paths between nodes at opposite edges. This intuition is precisely captured by the large value of the effective graph diameter. In general TDG visualizations are suggestive, but the metrics make the intuitions precise and allow classification.

From our measurements, we empirically derive the following two rules for detecting P2P activity. **Rule 1**: $\bar{k} > 2.8$ and $InO > 1\%$; **Rule 2**: $InO > 1\%$ and $EDiam > 11$. With these simple rules, we can correctly identify *all P2P TDGs from both backbone locations* (Abilene backbone and Tier-1 ISP). Intuitively, P2P hosts need to be connected with a large set of peers in order to perform tasks such as answering content queries and sharing files, which can explain the higher average degree compared to client-server applications. An additional characteristic of P2P applications is the duality of

roles, with many hosts acting both as client and server. The duality of roles is in turn captured by the high InO value. We further speculate that the decentralized architecture of some P2P applications (such as BitTorrrent), can explain the high diameters in some P2P TDGs.

Distinguishing collaborative applications from P2P: Some well-known applications other than P2P exhibit collaborative behavior, such as DNS and SMTP. This is not surprising since in these applications servers communicate with each other and with other clients (high $\bar{k}$), and servers act both as clients and servers (high InO). This is exactly what our metrics are set out to detect. It has been reported recently [23] that port numbers are fairly accurate in identifying such legacy applications, although they fail to identify P2P and other applications with dynamic use of port numbers. Therefore, one could use legacy ports to pinpoint and isolate such collaborative applications and then use graph metrics on the remaining traffic. In addition to port inspection, we can also examine the payload of a flow in order to verify that it follows the expected application-layer interactions. As a future work, our goal is to select metrics that can further help to separate between collaborative applications (e.g., DNS) and P2P. We discuss similar topics again in §V.

We do not claim that our thresholds are universal, but our measurements suggest that small adjustments to these simple parameters allow our methodology to work on different backbone links.

C. Practical Considerations for TDGs

1) Stability of metrics over time: If thresholds derived from TDGs changed significantly with time, then the classifier must be trained constantly and that would detract from its value. Fortunately this does not appear to be the case. We show the stability of TDGs in time and space using traces from different points in time at two different backbone locations for Abilene and a commercial Tier 1 ISP (Section II-B).

To test stability over time, we measured P2P TDGs from our longest trace (TR-ABIL) that span over one month and include samples taken over day and night hours, and both weekdays and weekends. To the best of our knowledge such month-long traces with payload are not available. Stability was also observed in all the one-hour long traces (TR-PAY1, TR-PAY2) but more detailed results are omitted due to space limitations.

Figure 3 summarizes the stability study for TR-ABIL showing the average degree metric. As we see, average degree takes values in a small range over the entire month. Soribada tends to have very small graphs ranging from 30 to 1,000 nodes and hence the higher variability. Stability was also observed for the diameter and InO metric but results are not shown here due to space limitations. For this study, we used TDGs representing five-minutes of traffic. In §II-C2 we show experiments with TDGs where we vary the observation interval from 5 seconds to 15 minutes. Note that BitTorrent is not shown in Figure 3 since it was not well-known in 2002 and resulted only in a handful of flows in the TR-ABIL trace.

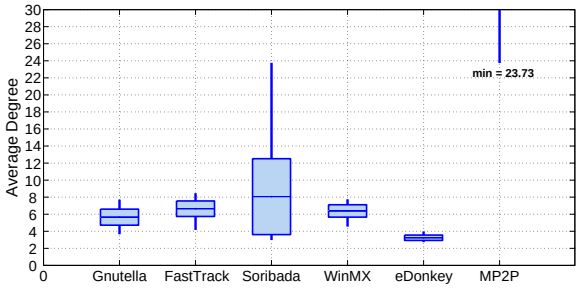


Fig. 3. The average degree for various P2P protocols over one month in trace TR-ABIL. Candle sticks show the maximum and minimum recorded values together with the average (horizontal line) and ± the standard deviation. For visualization purposes, for the MP2P application we show only the minimum value which is very high (23.73).

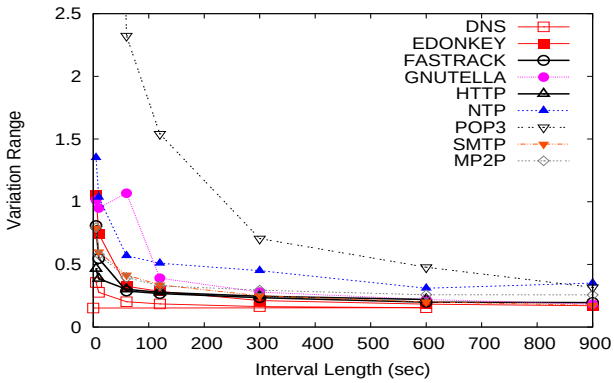


Fig. 4. The effect of changing the interval of observation for TDGs ranging from 5 seconds up to 15 minutes over a large set of protocols. To reduce variability we can choose to use longer intervals. After 300 seconds the reduction in variability is not significant.

2) Selecting the interval of observation: The effect of the observation interval on the graph metrics. We observe that by increasing the observation interval, graph metrics are more stable across successive TDGs of an application. To show this, we measure the variation of all our graph metrics over time as a function of the interval of observation. We vary the interval from one second up to fifteen minutes. For each interval length, we generate consecutive graph snapshots with non-overlapping time intervals. For each interval, we extract the graph and calculate the average degree, InO, and effective diameter metrics.

We examine the variability of each metric over all intervals of observation. We use the commonly-used *range* metric, which is the difference between the maximum and minimum observed values over all intervals for each graph metric. For each metric, we then normalize the range by the average value calculated over the time series. This normalization makes the range for different metrics somewhat comparable. For each application, we report the average range over all three graph metrics, and we plot this average range versus the duration of the observation interval in Figure 4. Our experiments showed very high range values for intervals smaller than 60 seconds. The range shows significant decrease for intervals larger than

a minute. Increasing the interval from 5 minutes to 10 and 15 does not significantly effect the range. This trend is shown in Figure 4, where we report the average range over all metrics for a set of our applications. Even though we have variability in our results, there was a clear trend in all our measurements. Similar trends we observed in all our traces (Table I). For the rest of our study, we use five minute intervals.

III. The Graption Framework

The Graption traffic classification framework consists of the following three steps.

Step 1: Flow Isolation. The input is network traffic in the form of flows as defined in §II. The goal of this first optional step is to utilize external information to isolate any flows that can already be classified. This knowledge could be based on payload signatures, port numbers, or IP address (e.g., exclude flows from a particular domain such as `google.com`).

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. Eventually, the output of this step is a set of groups with each group ideally containing flows from a single application (e.g., Gnutella, NTP, etc.). However, at this step, the exact application of each group is not known.

Step 3: Group Classifier. For each group of flows, we construct a TDG. Next, we quantify each TDG using various metrics. The classifier uses these metrics to identify the application for each group of flows. For the classification decision, we use a set of rules which in general depend on the focus of the study.

Although this paper focuses on P2P detection, Graption can be used for general application classification by choosing metrics and parameters appropriately. We next describe how we specialized Graption to detect P2P traffic (Graption-P2P).

A. Implementation Details of Graption-P2P

Step 1: This is an optional step in our methodology. Experiments without this step are discussed later in the section. Recent work [23] 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, in this study, we isolated flows with port 80 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. In our traces, the proportion of P2P actually using one of these ports is as low as 0.1%.

Step 2: To implement flow grouping we use the fact that application-level headers are likely to recur across flows from the same application. Therefore, payload similarity can be used to group flows. In Graption-P2P, we only use the first sixteen bytes from each flow. As we show, sixteen bytes are sufficient to give very good classification results. This observation agrees with findings in [26], [24]. Even though we use the payload bytes, our grouping is agnostic to application semantics, as each byte is considered as a single independent categorical feature. We consider each byte as a single categorical feature in the range $\{0, 1, ..., 255\}$.

The flow grouping step comprises two substeps: cluster formation and cluster merging.

a. Forming Clusters. Given the set of discriminating features, the next step is to cluster "similar" flows together. 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 to form **groups**, which produces the final output.

Feature-based clustering is a well-defined statistical data mining problem. For this task we used the popular **K-means** algorithm [33]. This algorithm has been commonly used for unsupervised clustering of network flows [9], [26], with very good results and low computational cost. K-means operates with a single parameter that selects the number of final clusters (k). As we show later in our evaluation, our classifier gives very good results over a large range of k.

The similarity between two flows is measured by the *Hamming* [33] distance calculated over the 16 categorical features (i.e., the payload bytes). Even though more involved similarity measures such as edit-distance (also known as Levenshtein distance) exist, Hamming distance has been used successfully before [14] and performs very well in our application.

b. Cluster merging. During clustering, it is likely that the same application generates multiple clusters. For example, many P2P protocols exhibit a variety of interaction patterns, such as queries (UDP flows) and file transfers (TCP flows), each with significantly different flow and packet characteristics [20]. This motivates mergings clusters that we expect to belong to the same application into groups. This grouping provides a more complete view of the application and aids in understanding the structure of the P2P protocol, as we show in §III-B.

Cluster merging cannot be based on the chosen set of flow-level features that were already used to create the clusters originally. Instead, in the case of a P2P protocol, it is natural to assume that the TDGs corresponding to each cluster of the same protocol would share a large number of common nodes (IP addresses).

Based on these observations, we use an Agglomerative (Hierarchical) Clustering Algorithm that recursively merges clusters with significant similarity in IP addresses. We used the following metric to calculate similarity between clusters: $Sim(C_1, C_2)$ = (Number of flows having their source or destination IPs present in both clusters) / (The number of flows of the smaller cluster). 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 **similarity threshold (ST)**. As we show later in our evaluation, our classifier gives very good results over a large range of similarity thresholds.

Step 3: The outcome of the previous step is a set of groups of flows, with each group consisting of flows that we hope stem from a single application. In order to classify each group, we generate a TDG on the group in the same way as described

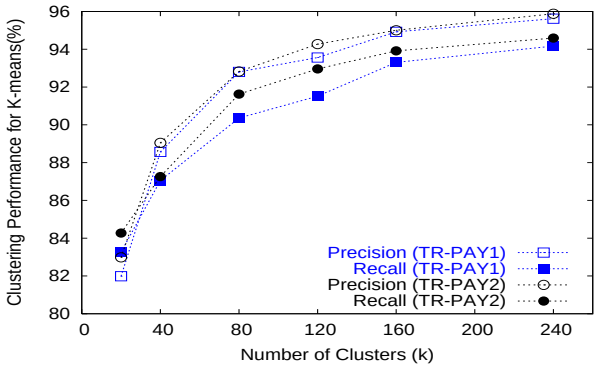


Fig. 5. Evaluating K-means: With sufficiently large k (>120) K-means can efficiently separate the flows of different application.

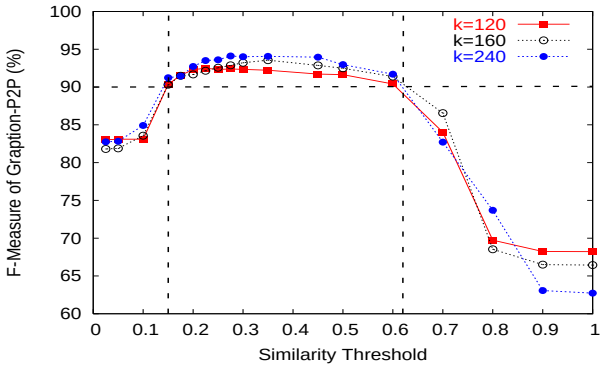


Fig. 6. Graption-P2P achieves $> 90\%$ F-Measure over a large range of similarity thresholds and number of clusters (k).

in §II. Each group yields a TDG that can be summarized using graph metrics. To identify P2P TDGs, we used the rules extracted from §II-B. When a group is labeled as P2P then all the flows of that group are classified as P2P flows.

B. Evaluating Graption-P2P

To evaluate Graption-P2P, we use traces TR-PAY1 and TR-PAY2, where we have the ground truth using the payload classifier (§II). We compute the True Positives, False Positives, and False Negatives. The True Positives (TP) measures how many instances of a given class are correctly classified; the False Positives (FP) measures how many instances of other classes are confused with a given class; and the False Negatives (FN) measures 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** [33], defined as $F = 2P \cdot R/(P + R)$, combining P and R.

1) Forming Clusters: The K-means algorithm generates k clusters and assigns each flow to a different cluster. We first test the effectiveness of K-means to form **pure** clusters. A pure cluster will ideally contain flows from a single application. To label the clusters, we use the **dominant heuristic** [2]. In this heuristic, using the ground truth of flows we label each cluster as belonging to the application with the majority of flows in the cluster. All the flows of a cluster are then classified to belong to this dominant application.

Using the dominant heuristic we assign clusters to applications and calculate the precision (P) and recall (R) for different values of k. The clustering results as we increase k for both traces are shown in Figure 5. We observe that with sufficiently large k (> 120) we achieve very good results with P and R above 90%. By increasing k, on one hand we have more "pure" clusters but on the other we make the cluster merging step harder. In the extreme case, each cluster will contain a single flow giving 100% P and R, but making cluster merging challenging. We will return to this topic later in §III-E.

The Graption-P2P classifier 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 k. As we show next, with k in the range 120-240 Graption-P2P has high classification performance.

2) Identifying P2P traffic: Using Graption-P2P, we achieve high P2P F-Measure over a range of values of k (K-means) and similarity thresholds (ST). We show this in Figure 6, where we vary the ST from 0.01 to 1 and use a sufficiently large k (see Figure 5). All experiments are averaged over each disjoint 5 minute interval of both traces. Intuitively, by using a very large ST, the clusters of an application are not grouped together, which results in TDGs that are harder to classify as P2P. On the other hand, with a very small ST, clusters belonging to different applications are merged together leading to poorer classification performance. The results in Figure 6 show that we achieve good classification performance ($> 90\%$ F-Measure), over a large range of similarity thresholds and number of clusters (k).

In Figure 7, we compare our approach with labeling each cluster using the ground truth (i.e. without merging any clusters and labeling each cluster using the dominant heuristic). Intuitively, for a given clustering of flows, the ground truth shows the best that our cluster labeling mechanism can achieve. For merging, we use a ST of 0.5. From Figure 7, we see that Graption-P2P deviates only slightly from labeling clusters using the ground truth. In the same plot, we also compare Graption-P2P without the cluster merging step, highlighting the benefit of merging clusters of the same application together.

Using a ST of 0.5 and $k = 160$, Graption-P2P achieves above 90% Recall and above 95% Precision over all disjoint 5 minute intervals for both traces. To apply Graption-P2P to other backbone links, the same selection process can be repeated to adjust the values of ST and k. Our experiments show that the classification performance can degrade with a bad choice of parameters. However, as shown in Figures 5, 6, and 7, for reasonable choices for k and ST, our method provides very good results. Including methods to automate the selection of parameters (e.g., using a parameter free clustering algorithm) is part of our future directions. The Graption framework is flexible enough to incorporate state-of-the-art clustering methods.

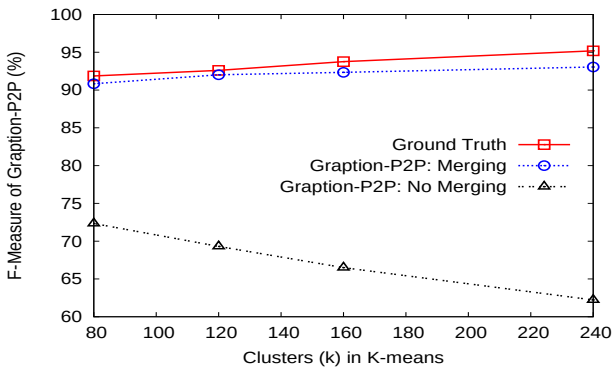


Fig. 7. Graption-P2P compared to cluster labeling based on ground truth. Results are also compared with and without cluster merging.

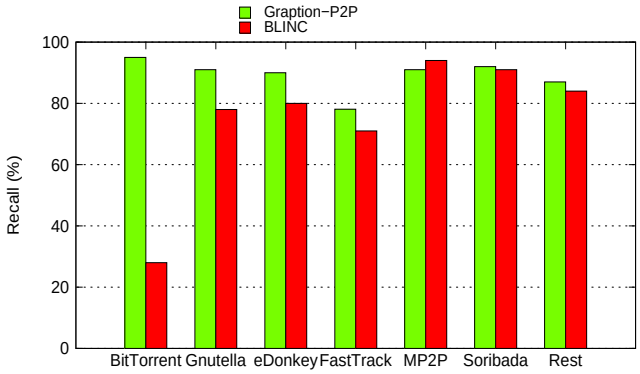


Fig. 8. The percentage of P2P traffic detected by Graption-P2P and BLINC. Precision for Graption-P2P is 95% and for BLINC is 89%.

C. Comparison with BLINC [22]

We used BLINC to classify traffic for both TR-PAY1 and TR-PAY2 traces. BLINC was optimized after several trial and error efforts to achieve its best accuracy for classifying P2P traffic over these traces. A detailed description of the selection processes for the parameters for BLINC is provided in [23]. As reported by Kim et al. [23], tuning the 28 parameters of BLINC is a time consuming processes. This is because small changes to the thresholds have a significant effect on the classification results.

The Recall and Precision for BLINC are 84% and 89% respectively compared to 90% and 95% of our method. The classification performance per P2P protocol for both BLINC and Graption-P2P are shown in Figure 8. Given that both methods do not distinguish between particular P2P protocols (e.g., Gnutella vs BitTorrent), it only make sense to report the portion of identified traffic (recall) for each protocol. Graption-P2P can identify more traffic for all P2P applications tested with the exception of MP2P. We can see from Figure 3 that MP2P has a much higher average node degree than the other P2P applications which can explain the high performance of BLINC over this protocol. Going back to Figure 8, we see that BLINC has significantly lower performance for some P2P applications. For example, Graption-P2P detects 95% of BitTorrent traffic, while BLINC detects only 25%! In addition, Graption-P2P detects 91% of the flows from the three popular P2P applications {BitTorrent, Gnutella, and eDonkey} while BLINC detects only 73%.

Our experiments suggest that BLINC and possible other behavioral-host-based approaches work well when applied at the edge, where a large fraction of host flows are observed and hence enough evidence is collected to profile each node. However, this is not always true for backbone monitoring points which can explain BLINC's lower performance. These observations are also supported by findings in [23].

We acknowledge that BLINC provides a methodology for detecting traffic from multiple classes of applications (e.g., HTTP, FTP, Chat, etc.) and not only for P2P. Our results only show that Graption-P2P does better in P2P detection. Selecting TDG features that can identify other applications is included in our future work.

D. Identifying the Unknown

Our goal here is to answer the following question: *Can Graption identify ports used by unknown P2P applications in our traces?* To achieve this, we configure Graption as follows. For step 1, we isolate legacy flows using the same approach as in Graption-P2P. In addition, we isolate flows from ports that belong to known P2P applications (e.g., port 1214 for FastTrack). At step 2, we cluster flows using their port number. This produces one cluster for each distinct port in the trace. For merging clusters, we use the same similarity threshold as in the Graption-P2P configuration. Finally, for classifying each group (step 3) we use the same rules as in Graption-P2P. We will refer to this configuration as Graption-P2P-Ports.

By manually inspecting the TDGs that Graption-P2P-Ports reports as P2P, we discovered interesting behaviors in the Abilene Trace (TR-ABIL). Our simple method was able to detect three ports (2002, 4156, 1978) that we initially thought were false positives. On further research, we found that these three ports were used by the Slapper[2] worm. Slapper has been reported to use these ports in order to form a P2P botnet overlay for launching DDoS attacks. Even though Graption-P2P-Ports was never explicitly trained to detect the Slapper overlay, it used the TDG profile of known P2P applications (e.g., eDonkey, Gnutella, etc.) to identify it. This is an advantage of using network-wide behavior to classify traffic.

Graption-P2P-Ports also captured an unusual behavior of IPs executing `traceroute` to each other forming a near clique. Such highly connected group of hosts was not observed in any of the P2P applications present in our traces. We speculate these interactions are by host performing active network measurements inside the Internet2 (Abilene) backbone. Given this unique connection pattern, we can always isolate such unusual behavior and distinguish it from a potentially new P2P application. Besides the active measurement traffic and the flows from the Slapper worm, Graption-P2P-Ports reported only 0.3% other flows as P2P. With no payload information, we were not to able to manually match these flows to a known exploit or P2P application.

[2]http://www.cert.org/advisories/CA-2002-27.html

E. Other Graption Configurations

For completeness, we show here preliminary results with other system configurations assuming that access to payload is limited or isolation cannot be used. We proceed by describing these configurations.

Graption-P2P-NI: Without isolation. This is the same configuration as the Graption-P2P, but without using isolation. By increasing k in K-means above 300 we achieve good results; $> 92\%$ precision and $> 85\%$ recall. A larger k was to be expected, since by including all the Web, SMTP, and DNS flows the total number of signatures increased as well. Our preliminary experiments showed that by increasing the number of clusters, it makes the cluster merging step more difficult. This observation highlights the advantage of using isolation.

Graption-P2P-NP: Using only flow-level features. This configuration is the same as the Graption-P2P, but assuming that payload and port numbers cannot be used. We achieved $> 90\%$ precision and $> 88\%$ recall over a range of k and similarity thresholds. For each flow, we extracted more than 40 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. In order to test the relevance of each feature we applied the Information Gain Ranking Filter [33] over various time intervals. For this configuration, we used the most prominent features: packet size information (i.e. min, max, and the size of the first five packets) and protocol (UDP or TCP). A recent work [10] also supports our observation that packet sizes are good features to cluster/classify network flows.

More detailed results are not shown here due to space limitations. The study of Graption over a variety of configurations and backbone traces is included in our future directions.

IV. Related Work

This paper extends our 6-page workshop version [15]. In addition to the work presented in the workshop version, we here include: (a) A study on practical considerations for using TDGs (see §II-C, Figures 3 and 4). (b) We provide additional details regarding our data sets and our classification results (see §II-B, §III-E, Figure 2, and Table II). (c) We provide a more detailed comparison with BLINC (see §III-C and Figure 8). (d) We include additional experiments showing how Graption can be used to detect previously unknown traffic (see §III-D).

Traffic classification. As an alternative to port-based methods, some works used payload [26], [24]. Other approaches use Machine Learning (ML) algorithms to classify traffic using flow features (e.g, packet sizes). For an exhaustive list and comparison of ML algorithms we refer the reader to [28] and [23]. All supervised methods require per application training and will thus not detect traffic from new applications. Our work has more in common with unsupervised data mining methods which group similar flows together. All previous methods [2], [25], [9] 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.

In BLINC [22], the authors characterize the connection patterns (e.g., if it behaves like using P2P) of a single host at the Transport Layer and use these patterns to label the flows of each host. 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. As we show, our approach performs better than BLINC in our backbone traces (see §III-C).

Similar to BLINC, other host-based methods [1], [19] target the identification of P2P users inside a university campus (i.e., network edge). Unlike Graption, in [1], [19], [22] they do not use *network-wide* host interaction. In [6], the authors use a port-based method to identify P2P users, using their temporal appearance and connection patterns in a trace.

The most recent host profiling method is by Trestian et al. [32]. They used readily available information from the Web to classify traffic using the Google search engine. They show very good results for classifying flows for legacy application, but their results are not promising for P2P detection because of the dynamic nature of P2P IP hosts. Our method can thus be used to complement the work in [32].

Worm detection. Graphs have been used for detecting worm activities within enterprise networks [8]. Their main goal was to detect 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 [34]. 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 [5].

Measurements on network-wide interactions. Statistical methods are used in [35] for automating the profiling of network hosts and ports numbers. The connectivity behavior and habits of users within enterprise networks is the focus of many papers, including [31]. In [30], the authors study P2P overlays using passive measurements, but target mainly the profiling of P2P hosts. The most resent work on network-wide interactions in by Jin et al [18]. In their work [18], they use graph-partitioning methods to extract and study the evolution of smaller communities within a TDG. None of the above papers targets P2P detection.

V. Discussion

Enhancing Isolation. To improve isolation we can enforce payload inspection in addition to port-based filtering. For example, we can test all DNS flows at port 53 to see if they also have a DNS payload signature or if another protocol is tunneling its traffic under the DNS port. If payload is

encrypted, then we can choose to use flow-level feature such as packet sizes [23] or white-listed IP addresses [32].

Encryption in P2P applications. If an application encrypts all traffic classification methods using payload [29], [24], [14], [26] will be out-dated. Note, however, that it has been shown before [3] that even if an application encrypts its traffic there are ways to distinguish it from other flows. To show that even in the case where P2P encryption becomes prevalent our framework will still be useful, we also did preliminary experiments with Graption using only flow level features and observed good results with recall and precision of above 88% in both traces.

Obfuscation of P2P applications at multiple levels. In the future, obfuscation might become more common. In the case where IPsec (encryption up to Layer-4 headers) becomes prevalent, transport layer [21], and host-based [22] methodologies will most likely not work. If P2P protocols start using *polymorphic blending methods* [11], [13] to masquerade their traffic at the packet (e.g., packet sizes) and payload level to resemble a legacy application (e.g., Web), then all current Machine Learning [28] methods will not work. In such a scenario, TDGs (IP-to-IP graphs) might still prove to be useful because the interaction patterns seem hard to hide. Using Graption to detect such applications is left for future work.

VI. Summary and Conclusions

The underlying theme of our work is to go beyond just monitoring individual packets, flows, or hosts, to *also* monitoring the interactions of a group of hosts. Towards this end, we introduce TDGs and show their potential to generate novel tools for traffic classification. Based on TDGs, we developed Graption, a graph-based framework, which we then specialize (Graption-P2P) for the detection of P2P applications. We show that Graption-P2P classifies more than 90% of P2P traffic with above 95% precision when tested on real-world backbone traces. While our best results are obtained assuming access to some initial payload bytes, Graption still performs well (90% precision with 88% recall) even when payload is not used. We believe the network-wide characteristics of P2P traffic (e.g., InO, average degree, and diameter) are not as easy to hide without significantly altering P2P functionality.

References

[1] Genevieve Barlett, John Heidemann, and Christos Papadopoulos. Inherent Behaviors of On-line Detection of Peer-to-Peer File Sharing. In *IEEE Global Internet Symposium*, 2007.

[2] Laurent Bernaille, Renata Teixeira, and Kave Salamatian. Early Application Identification. In *ACM CoNEXT*, 2006.

[3] Dario Bonfiglio, Marco Mellia, Michela Meo, Dario Rossi, and Paolo Tofanelli. Revealing Skype Traffic: When Randomness Plays With You. In *ACM SIGCOMM*, 2007.

[4] CAIDA. www.caida.org.

[5] M. Patrick Collins and Michael K. Reiter. Hit-List Worm Detection and Bot Identification in Large Networks Using Protocol Graphs. In *RAID*, 2007.

[6] Fivos Constantinou and Panayiotis Mavrommatis. Identifying known and unknown peer-to-peer traffic. In *IEEE NCA*, 2006.

[7] DatCat, Internet Measurement Data Catalog. http://www.datcat.org.

[8] D. Ellis, J. Aiken, K. Attwood, and S. Tenarglia. A Behavioral Approach to Worm Detection. In *ACM CCS WORM*, 2004.

[9] Jeffrey Erman, Martin Arlitt, and Anirban Mahanti. Traffic Classification Using Clustering Algorithms. In *ACM SIGCOMM MineNet*, 2006.

[10] Alive Este, Francesco Gringoli, and Luca Slagerelli. On the Stability of the Information Carried by Traffic Flow Features at the Packet Level. *ACM SIGCOMM CCR*, 39(3):13–18, July 2009.

[11] Prahlad Fogla, Monirul Sharif, Roberto Perdisci, Oleg Kolesnikov, and Wenke Lee. Polymorphic blending attacks. In *USENIX Security Symposium*, 2006.

[12] A. Gerber, J. Houle, H. Nguyen, M. Roughan, and S. Sen. P2P, the Gorilla in the Cable. In *National Cable and Telecommunications Association (NCTA)*, 2003.

[13] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. BotMiner: Clustering Analysis of Network Traffic from Protocol- and Structure-Independent Botnet Detection. In *USENIX Security Symposium*, 2008.

[14] Patrick Haffner, Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. ACAS: automated construction of application signatures. In *ACM SIGCOMM MineNet*, 2005.

[15] Marios Iliofotou, Hyun chul Kim, Prashanth Pappu, Michalis Faloutsos, Michael Mitzenmacher, and George Varghese. Graph-based P2P Traffic Classification at the Internet Backbone. In *IEEE Global Internet Symposium*, 2009.

[16] Marios Iliofotou, Prashanth Pappu, Michalis Faloutsos, Michael Mitzenmacher, Sumeet Singh, and George Varghese. Network Monitoring Using Traffic Dispersion Graphs (TDGs). In *ACM IMC*, 2007.

[17] IPOQUE - Bandwidth Management with Deep Packet Inspection. Internet study: file hosters like rapidshare and streaming services like youtube. 2007.

[18] Yu Jin, Sharafuddin Esam, and Zhi L. Zhang. Unveiling Core Network-Wide Communication Patterns through Application Traffic Activity Graph Decomposition. In *ACM SIGMETRICS*, 2009.

[19] Wolfgang John and Sven Tafvelin. Heuristics to classify Internet Backbone Traffic based on Connection Patterns. In *IEEE ICOIN*, 2008.

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

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

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

[23] Hyunchul Kim, Kimberly Claffy, Marina Fomenkov, Dhiman Barman, Michalis Faloutsos, and KiYoung Lee. Internet Traffic Classification Demystified: Myths, Caveats, and the Best Practices. In *ACM CoNEXT*, 2008.

[24] Justin Ma, Kirill Levchenko, Christian Kreibich, Stefan Savage, and Geoffrey M. Voelker. Unexpected means of protocol inference. In *ACM IMC*, 2006.

[25] Anthony McGregor, Mark Hall, Perry Lorier, and James Brunskill. Flow Clustering Using Machine Learning Techniques. In *PAM*, 2004.

[26] Andrew Moore and Konstantina Papagiannaki. Toward the accurate identification of network applications. In *PAM*, 2005.

[27] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167, 2003.

[28] Thuy T.T. Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys and Tutorials*, 4th edition, March 2008.

[29] Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Accurate, scalable in-network identification of p2p traffic using application signatures. In *WWW*, 2004.

[30] Subhabrata Sen and Jia Wang. Analyzing peer-to-peer traffic across large networks. *IEEE/ACM Transaction on Networking*, 12(2):219–232, 2004.

[31] Godfrey Tan, Massimiliano Poletto, John Guttag, and Frans Kaashoek. Role classification of hosts within enterprise networks based on connection patterns. In *USENIX Annual Technical Conference*, 2003.

[32] Ionut Trestian, Supranamaya Ranjan, Aleksandar Kuzmanovic, and Antonio Nucci. Unconstrained endpoint profiling (Googling the Internet). In *ACM SIGCOMM*, 2008.

[33] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2nd edition, 2005.

[34] Y. Xie, V. Sekar, D. Maltz, M. Reiter, and H. Zhan. Forensic Analysis of Epidemic Attacks in Federated Networks. In *IEEE ICNP*, 2006.

[35] K. Xu, Z. Zhang, and S. Bhattacharyya. Profiling Internet Backbone Traffic: Behavior Models and Applications. In *ACM SIGCOMM*, 2005.