Exploiting Dynamicity in Graph-based Traffic Analysis: Techniques and Applications

Marios Iliofotou
UC Riverside
Riverside, CA 92521
marios@cs.ucr.edu

Michalis Faloutsos
UC Riverside
Riverside, CA 92521
michalis@cs.ucr.edu

Michael Mitzenmacher
Harvard University
Cambridge, MA 02138
michaelm@harvard.edu

ABSTRACT

Network traffic can be represented by a Traffic Dispersion Graph (TDG) that contains an edge between two nodes that send a particular type of traffic (e.g., DNS) to one another. TDGs have recently been proposed as an alternative way to interpret and visualize network traffic. Previous studies have focused on static properties of TDGs using graph snapshots in isolation. In this work, we represent network traffic with a series of related graph instances that change over time. This representation facilitates the analysis of the dynamic nature of network traffic, providing additional descriptive power. For example, DNS and P2P graph instances can appear similar when compared in isolation, but the way the DNS and P2P TDGs change over time differs significantly. To quantify the changes over time, we introduce a series of novel metrics that capture changes both in the graph structure (e.g., the average degree) and the participants (i.e., IP addresses) of a TDG. We apply our new methodologies to improve graph-based traffic classification and to detect changes in the profile of legacy applications (e.g., e-mail).

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network monitoring; C.2.5 [**Local and Wide-Area Networks**]: Internet

General Terms

Measurement, Experimentation, Security

Keywords

Behavioral approach, network-wide interactions, dynamic graphs, network monitoring

1. INTRODUCTION

A recently introduced way of analyzing network-wide behavior of traffic uses the "social" interaction of the network. This approach leads to a directed graph, where each node is an IP address, and each edge represents a particular interaction between two nodes. For example, we can use the DNS interactions to form a graph with the IP-hosts involved in the DNS protocol. The term *Traffic Dispersion Graph* or *TDG* is used to refer to such a graph [17]. TDGs appear to have excellent descriptive power, but they require novel metrics and tools to extract and utilize the information they can provide.

A number of studies have explored the capabilities of TDGs. Early graph-based efforts focused on specific problems, mainly security issues such as intrusion detection [36] and worm propagation [10, 40]. Network-wide interactions at the backbone were studied in [41], targeting the automatic grouping and profiling of network applications using information about their degree and port distributions. A more recent study [16] argues for wider capabilities of TDGs, with direct application in traffic classification. In [16], the graph signature of P2P applications is used in order to distinguish P2P applications from other network applications. A recent work by Jin et al. [18] studies the dominant communities within a graph making it easier to interpret the graph and characterize its structure. A fundamental characteristic of all previous work, however, is that they focus on graphs derived from static snapshots, or a single graph representing a short period of time. Even though network traffic is by nature dynamic, previous efforts do not incorporate the impact of time in the graph representation of interactions.

Our goal is to provide a set of traffic analysis techniques useful to network administrators, to aid them in managing, provisioning, and protecting their networks. To give focus in our work, we specifically consider two problems: (a) How can we classify traffic to its originating application? (b) Can we detect **polymorphic blending**, i.e., when a new application (e.g., a new P2P protocol) attempts to hide within an existing application (e.g., e-mail)? Although polymorphic blending can be seen as a part of the general traffic classification problem, it is an interesting problem in its own right, but has received very little attention. Both these problems are of interest in practice as we mentioned above and further elaborate later on.

In this paper, we expand on the potential of TDGs to address these problems by explicitly considering how they change over time. To achieve this, we shift away from looking at single graph snapshots and instead associate to each TDG a sequence of static graph instances collected over different time periods. To analyze these graph sequences, we need novel metrics and techniques. As our key contribution, we propose a systematic set of methods for quantifying the

CoNEXT'09, December 1–4, 2009, Rome, Italy.

changes between snapshots and develop methods for the two traffic analysis tasks we mentioned above. To the best of our knowledge, this is the first work to address the problem of analyzing network traffic using graphs that captures the dynamics over periods of time. However, we note that such graph representations have been successfully used in different contexts [32]. The key message of our work is that considering the dynamic nature of TDGs can open the doors to new, powerful techniques for traffic analysis.

Our work makes the following specific contributions:

(a) We present a systematic way to generate and analyze TDGs that capture changes over time. In §2, we introduce a framework (metrics and methodologies) to generate, model, and analyze TDG snapshots over time. Our work complements existing graph-based methods by introducing metrics that explicitly capture the inherently dynamic nature of network traffic.

(b) We apply our methods to the problem of traffic classification. In our previous work [16], we showed how TDGs can be used to identify P2P flows in a network traffic trace. More details about [16] are given in §5. In §3, we go beyond the previous results by showing that using the dynamic changes of TDGs can isolate the traffic of P2P applications as well as distinguish *between* the traffic of legacy applications (e.g. DNS vs. SMTP) and some network games, which are particularly notorious in classification. Our new method operates on behavioral aspects of the applications using no port specific information. Note that other behavioral approaches, such as BLINC [21], require port numbers to achieve the granularity of classification we achieve here, e.g., to separate DNS traffic from P2P [22]. We believe that our findings can be used to enhance other traffic classification methods [3, 11, 26] that are based on the automatic grouping of related network flows.

(c) We show our methods are successful in detecting polymorphic blending. In §4, we use graph metrics to profile the TDGs of well known applications and use this profile to detect polymorphic blending. Using five different P2P protocols as intruders, we show that we can detect such behavior more than 68% of the time, even when a P2P application tunnels as little as 10% of its traffic. The key results from our experiments are that changes in the dynamic behavior of TDGs are better than changes in static behavior for detecting small-scale blending, and further that a combination of dynamic and static behavioral changes provides better results than each type separately. We believe that our methods can open the door to detect other types of behavioral changes and anomalies.

The rest of the paper is organized as follows. In §2, we formally define TDGs and present an array of static and dynamic graph metrics. In §3, we present graph-based heuristics that can be used to isolate all P2P TDGs from all our backbone locations. In §4, we show how we use TDGs to identify polymorphic blending by detecting changes in the profile of legacy applications. A discussion of related work and our conclusions appear at the end of the paper.

2. DEFINITIONS AND GRAPH METRICS

In this section, we provide definitions and an array of graph metrics we use throughout the paper. The definitions of static snapshots and of many metrics can be found in [17, 16], but we include them here for completeness. The metrics capturing dynamicity do not appear in previous work. Readers may wish to skip the details of §2.1 and return to it as a reference when needed.

In [17, 16], a TDG refers to a graph $G(V, E)$ that represents the network-wide interaction (say "who talks to whom") from a data trace[1]. In a TDG, each node corresponds to a distinct IP address, and an edge signals an interaction between a pair of nodes. The power of TDGs lies in the flexibility of deciding what constitutes an interaction, which can be implemented in practice by what we refer to as **edge filter**. An interaction could correspond to a simple packet exchange, or could be determined by a complex rule, such as "at least three TCP packets at port 25 were exchanged". We discuss edge filters in more detail in §2.2. In their more general form, TDGs can be a sequence of directed and weighted graphs as we discuss below.

Static TDG snapshots. Given an edge filter, we can create a TDG snapshot that represents all the edges that matched the filter during a fixed **interval of observation**, which we denote by T. Depending on the edge filter, we can direct the edges; for example, we might have the sender of the first packet in an interaction be the head of the directed edge. In addition, the edge can be associated with a weight or other associated useful information, such as the number of packets exchanged. A real-world example of a static snapshot is shown in Figure 1, representing the FTP traffic over a five-minute interval.

Extended definition: TDGs as a sequence of static graph snapshots. To capture dynamic behaviors, we represent TDGs as series of graph snapshots $G_1, G_2, ..., G_n$ with a common edge filter. Each graph G_i corresponds to a particular interval of observation T_i. In practical terms, the initial data trace is split into subintervals and we apply the same edge filter to create a graph for each subinterval. We emphasize that we have now extended the definition of a TDG from a single graph (as given in [17]) to include also a sequence of graphs; we believe the meaning remains clear and this should cause no confusion. We adopt the framework that a node that is inactive (with respect to the edge filter) during a particular interval will not appear in the graph for that interval. Therefore, nodes as well as edges can vary among the G_i. Next, we describe metrics that can explicitly quantify the changes between graph snapshots of a TDG.

2.1 Quantifying TDGs

Graph metrics are used to describe and compare graphs. The set of metrics presented here are naturally divided in two groups: static or **unary** metrics metrics that operate on individual graph snapshots, and dynamic or **binary** metrics that compare between snapshots of a particular TDG. Several metrics introduced here are not commonly used in the measurement community currently, and our choices represent experience based on time-consuming trial and error.

Unary Metrics: Quantifying Static Snapshots.

We start by reviewing standard graph terminology and terminology used in previous studies of TDGs [17]. Let us consider a TDG $G(V, E)$, with V the set of nodes and E the set of edges. We use $|X|$ to denote the cardinality of a set. For any edge (u, v) the nodes u, v are called the **endnodes** of the edge. In Figure 1 we show a graph visualization example of a TDG snapshot. The main goal of the metrics we de-

[1]We refer to an ordered sequence of network packets as data trace.

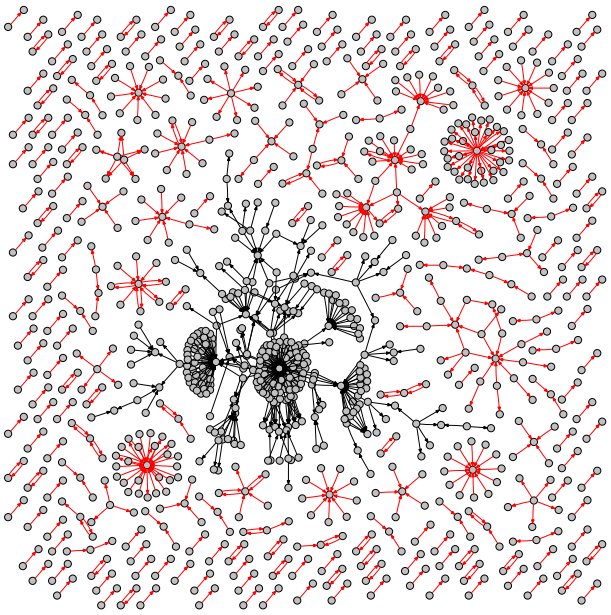


Figure 1: A static TDG snapshot of the FTP application using a 5 minute interval of observation. The largest connected component is located in the center of the figure.

scribe next is to translate the visually meaningful aspects of a graph into quantitative measures that can automate the processes of comparing graphs (§3) and detecting changes (§4).

TDGs are typically directed, and hence we can group nodes based on the direction of their edges. Sinks (V_{snk}) are nodes that have only incoming edges and sources (V_{src}) are nodes that have only outgoing edges; we also refer the set of nodes having both incoming and outgoing connections as V_{InO}, or In-and-Out (**InO**). These subsets V_{InO}, V_{snk}, and V_{src} partition V (so $|V| = |V_{InO}| + |V_{snk}| + |V_{src}|$). We say that a pair of nodes u, v has a *bidirectional edge* if and only if $(u, v) \in E$ and $(v, u) \in E$. To quantify the symmetry of a graph, we use the percentage of communicating node-pairs that have a bidirectional edge (**BiDir**).

To quantify the connectivity of a graph, we use the size of its Largest Weakly Connected Component (**LWCC**). If we consider again the graph as undirected, the LWCC is the size of the largest connected component; we report these quantities as a percentage of the total number of nodes in the graph. The LWCC for the graph visualization example of Figure 1 is located in the middle of the figure and is highlighted with darker colored edges (better viewed on a computer screen or colored print-out). From the figure we see that TDG snapshots can be disconnected. We have observed that measuring the size of the top 10 LWCCs is a good metric for characterizing TDGs as we show in §4 in more detail.

The neighborhood of a node u is the set of nodes adjacent to u and is denoted by $\Gamma(u)$. The degree of u is defined as $d(u) = |\Gamma(u)|$. The minimum endnode degree (**MED**) of an edge $e = (u, v)$, is defined as $MED(e) = min(d(u), d(v))$. We define the **average degree** of a TDG as $\bar{k} = \sum_{u \in V} d(u)/|V|$. We denote the number of nodes with degree k as $n(k)$. We denote the maximum degree of the graph as k_{max}. The **degree distribution** of a graph captures the probability that a randomly selected node has degree k, and is defined by $P(k) = n(k)/n$ for $k = 1, k_{max}$. The entropy of the degree distribution $H(X)$ is defined as $-\sum_{k=1,k_{max}} P(k)\log(P(k))$, with each term in the sum being 0 if $P(k) = 0$. For measuring the uniformity of the distribution, we use the **Relative Uncertainty (RU)**; given by $RU = H(X)/\log_2 k_{max}$ as defined in [41]. (The maximum entropy is achieved with the uniform distribution, so an RU value of 1 denotes the uniform distribution.) Note that all the above metrics can also be defined for the marginal distributions of incoming and outgoing edges of nodes, e.g., in-degree distributions, average in-degree, etc.

The **assortativity coefficient** r is a summary metric of the correlation of the degree between endpoints of edges in the graph [27]. The value is the Pearson correlation coefficient of the degrees of the endnodes of edges and lies in the range $[-1, 1]$. If $r = 0$, the graph appears to have random degree correlations, and there is no linear relationship between the degrees of neighbor nodes. If $r > 0$ then the graph is assortative, and high degree vertices are likely to connected to other high degree vertices. Conversely, if $r < 0$ then the graph is disassortative, and high degree vertices are likely to connect to low degree vertices.

Binary Metrics: Quantifying Dynamic Changes.

We can quantify the dynamic behavior of TDGs by comparing successive snapshots or snapshots separated in time. Toward this end, we consider metrics that capture differences between graphs. All previously described metrics in this section can be consider as "unary" since they operate on a single graph instance. We now present "binary" metrics that operate on two snapshots of a TDGs. The use of binary metrics in this context is one of our contributions. In what follows we use V_1 and V_2 (and similarly E_1 and E_2) to refer to node (and edge) sets corresponding to two graphs G_1 and G_2, which in context generally correspond to two snapshots over distinct intervals.

We can compare graphs using either labeled or unlabeled representations. Thinking of unlabeled graphs, where nodes do not have identities, we focus at the *structure* of the graph using metrics such as average degree, InO, RU, etc. Using the labeled representation, we can study the membership consistency (i.e., if a node/edge is present in both snapshots), and status consistency of individual nodes and edges (e.g., if the maximum degree node of instance A is also the max degree node in instance B). Both frameworks can offer insight; note that two graphs can be isomorphic, but still differ significantly in their underlying nodes.

For quantifying changes in node and edge membership consistency, we use the following graph metrics. The **Relative Inclusion** (RI) of nodes of a graph G_1 relative to another graph G_2 is defined as $RI(V_1, V_2) = |V_1 \cap V_2|/|V_1|$. For edges the definition is $RI(E_1, E_2) = |E_1 \cap E_2|/|E_1|$ (in general our metrics can be defined both for edges and nodes). Comparing a current snapshot (G_1) with a snapshot of the graph after 2 hours (G_2), the RI captures the percentage of nodes (or edges) that remained in the graph. These definitions also hold for comparing appropriate subsets of nodes or edges. For example, we can also consider the relative inclusion of InO (V_{InO}) nodes in the graph.

Detailed Edge Similarity. We introduce metrics that more finely quantify the similarity of the edges between two

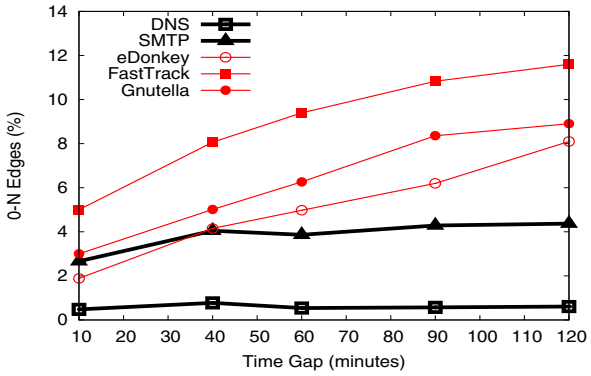


Figure 2: The "edge volatility" of five different applications over time for the PAIX trace (see Table 1). All P2P TDGs change more over time compared to the SMTP and DNS TDGs.

graphs G_1 and G_2. We have $RI(E_1, E_2)$ representing the percentage of edges in G_1 that are also present in G_2. We can group the edges $e \in E_1$ of G_1 into 4 different sets based on whether both, only one, or none of the endnodes of e belong in G_2. The groups are as follows: (a) the *common* (C) edges $E_1^C = E_1 \cap E_2$; (b) the *2N*-edges with both endnodes present in V_2, $E_1^{2N} = \{(u,v) \in E_1 : u, v \in V_2\}$; (c) the *1N* -edges with exactly only one endnode in V_2, $E_1^{1N} = \{(u,v) \in E_1 :$ (one of u or v) $\in V_2\}$; (d) the *0N* -edges with both nodes not present in V_2, $E_1^{0N} = \{(u,v) \in E_1 : u, v \notin V_2\}$.

Detailed edge similarity can be seen as a more informative alternative to the **graph edit distance** for labeled graphs, which is defined as $edit(G_1, G_2) = |V_1| + |V_2| - 2|V_1 \cap V_2| + |E_1| + |E_2| - 2|E_1 \cap E_2|$. (One could also consider weighted versions of the edit distance, with the vertex terms in the sum weighted differently than edge vertices; we use the unweighted edit distance here.) For comparing labeled sets, one can also use the **Jaccard Index** (JI) defined as $|V_1 \cap V_2|/|V_1 \cup V_2|$ or $|E_1 \cap E_2|/|E_1 \cup E_2|$ for nodes and edges respectively. The JI and edit distance quantify the changes in the graphs in a single value. These metrics can be used to detect graph anomalies as shown in [32].

Edge Volatility. Binary metrics offer the potential to capture the dynamic trends of TDGs. For example, suppose that at $t = 0$ we create the reference graph $G_{t=0}$ using a five-minute interval of observation, and our goal is to measure how much the edges of $G_{t=0}$ change over time. Intuitively, we expect that the changes from graphs taken back-to-back ($G_{t=0}$, $G_{t=5min}$) compared to the changes between graphs taken few hours apart ($G_{t=0}$, $G_{t=2hours}$) will be smaller for legacy applications, such as DNS and SMTP, compared to P2P activity where graphs change more with time due to ephemeral interactions. Here, we use the Detailed Edge Similarity (DES) defined above. Intuitively, as time evolves, it will be more likely to see both endnodes of an edge become inactive (i.e. not present after some time) in a P2P TDG compared to DNS/SMTP. An example of this process is shown in Figure 2 showing the percentage of 0N-edges (over all the edges of $G_{t=0}$) as we increase the gap between a graph snapshot and the reference graph ($G_{t=0}$) for the PAIX trace (see Table 1). Even if the initial percentage of 0N-edges is lower for some P2P applications (e.g., eDonkey), the rate of increase is higher than SMTP and significantly higher than DNS.

Discussion: While we have focused on metrics for TDGs based on comparing snapshots, given a complete data trace one could consider further dynamic graph properties not based solely on such comparisons. For example, one could ask if the TDG becomes more connected (higher LWCC) or if it gets more dense (higher average degree) as we increase the interval of observation. In this work we consider the utility of comparing snapshots, but examining other metrics on TDGs remains an interesting direction for future work.

2.2 Edge Filters

For completeness, we present a set of general edge filters that can be used in isolation or in combination to select the right set of flows depending on the focus of the study as discussed in previous work [17, 16, 10].

With a **port-based filter**, we collect traffic for a fixed destination (source) port. If the port number corresponds to a well-behaved single-port application, this can correspond to monitoring the traffic of that application. Signature or **content-based filters**, match string patterns in the payload of the packet. These filters can be very effective, but assume that we have the signature of the desired traffic, and we have access to the content of the packet. These assumptions are not always true for all applications or all traces. Using a **flow-level filter**, we create a graph with flows that meet certain flow-level features, such as specific packet sizes, packet inter-arrival times, or number of packets. In fact, the grouping of flows into TDGs can be automated (unsupervised). This can be achieved with machine learning and clustering algorithms [22, 39]. The advantage of these filters is that they can be made to work with no a priori information of how applications behave. We have used this approach successfully for application classification in our work on graph-based traffic classification [16]. Similar methods have also been used successfully in other works [2, 3, 11, 12, 26, 29].

How do we decide which edge filters to use? This depends on the focus and purpose of the measurement study. If the protocol we want to observe (e.g. DNS) operates under a default port (e.g., port 53) we can use port-based filters. If the target application is a P2P protocol that uses ephemeral port number then we can choose to use a content-based or flow-level filter. We further discuss the selection and use of edge filters in §2.3 and in the context of application classification in §3.

Some practical considerations. Throughout this paper, we assume that packets are grouped into flows using the standard methods based on the five tuple (`SrcIP`, `SrcPort`, `DstIP`, `DstPort`, `Protocol`). For a TCP flow, we create a directed edge starting at the node that sent the `SYN` packet. In the case where we only observe the `SYN/ACK`-packet, the direction of the link is reversed and set to point from `dstIP` to `srcIP` (as directly derived from the packet). For a UDP flow, we create a directed edge starting from the sender of the first packet in our trace.

2.3 Data Sets

We present a summary of the backbone traces that we use in this study in Table 1. All traces with the exception of PAIX are **publicly available** from NLANR/CAIDA [5]. Using publicly available traces allows other researchers to extend and verify our findings and contributions, and we attempted to use a rich set of currently publicly available

Alias	*Start Date/ Time*	*Duration*	*IPs*	*Bytes*
CLEV	2002-08-14/ 09:00	2 hour	232,579	777.2 GB
KSCY-MON	2002-09-01/00:00	1 month	258,535	5.3 TB
KSCY	2002-08-14/ 09:00	2 hour	198,752	683.05 GB
CLEV-DAYS	2002-08-27/ 09:00	5 days	171,359	898.5 GB
PAIX	2004-04-21/17:59	2 hour	258,636	891 GB
OC48	2002-08-14/09:00	3 hour	480,637	864.7 GB

Table 1: The IPs report the average values over 5-minute long intervals. Traces (except PAIX) can be found at `http://pma.nlanr.net/Special/` and `http://www.caida.org/data/passive/`.

backbone traces. All traces are IP-anonymized and contain traffic from both directions of the link. The traces were collected during both night and day hours as well as workdays and weekends, accounting for around 10 terabytes of raw TCP/UDP traffic.

We use six data sets from four different links. Traces OC48 and PAIX (Palo Alto Internet eXchange) are collected from backbone links of two Tier-1 commercial ISPs. The other four traces are collected from two different backbone links of the Abilene (Internet2) academic network. The {KSCY-MON, KSCY} traces are from a link connecting Kansas City to Indianapolis and {CLEV, CLEV-DAYS} are from a link connecting Cleveland to Indianapolis. The CLEV-DAYS and KSCY-MON traces, are comprised of several five-minute-long samples taken daily over five consecutive days and one month respectively. Even though the OC48 trace spans over three hours, it contains two hours of actual packet collection; between 09:00 - 10:00AM and 11:00-12:00PM. All the remaining traces contain uninterrupted consecutive collections of packet traces.

Ground truth. In this work, we use a combination of signature-based and port-based filters to define TDGs and we establish the ground truth using existing methods [20, 4, 34]. The PAIX trace contain up to 16 bytes of payload from each packet, thereby allowing the classification of flows into applications using standard signature matching techniques [19, 21, 34]. The payload classifier managed to classify 99.3% of the flows while only 0.7% did not match a known signature.

3. TDG CLASSIFICATION

In this section, we answer the following question: *Can TDGs be used to distinguish between different classes of applications, for example, identify the set of graphs that correspond to P2P applications?* The key observation is that static metrics provide useful information about the behavior of an application but they have their limits. Our dynamic metrics can further capture inherent behavioral properties of the underlying applications which makes them easy to understand by network administrators and (debatably) harder to hide by an application. For example, a P2P overlay will be hard to sustain a graph structure with the stability over time of the DNS TDGs (see Figure 2).

To provide a concrete problem, we focus on the detection of P2P applications. P2P applications are particularly challenging due to their continual change of characteristics (upgrades with additional functionality) and the continual appearance of new P2P protocols (especially in Asia). Our main goal in this section is to isolate P2P TDGs from the rest of the Internet applications. We want to achieve this by identifying a minimal set of discriminating features for separating TDGs of P2P protocols from those of other applications. A traffic classifier based on TDGs was the topic of an earlier work [16]. In [16] we show how TDGs can help to classify individual network flows. More details about our previous work [16] can be found in §5.

Collaborative Applications. We here use the term *collaborative applications* to include all Internet protocols where hosts: (a) are required to interact (collaborate) with a relative large number of other hosts and (b) have interchangeable roles with hosts acting both as clients and servers. All known P2P applications, as well as legacy application such as DNS, NTP, and SMTP have these two features in common. For example, in DNS servers collaborate with other servers in order to resolve DNS queries. To achieve this, DNS requires the servers to be "connected" with one-another as well as operate both as servers (replying to queries) and clients (issuing queries).

Our classification methodology has two steps. First, we start by isolating all typical client-server applications (e.g, POP3, Web, FTP) from the remaining types of *collaborative applications*. This is achieved by using a small set of static graph metrics which have been shown to work well in other works [17, 16]. At the second step, we use metrics on dynamic behavior to further isolate P2P TDGs from the bulk of collaborative applications.

We show that trying to isolate P2P from other collaborative applications using only static graph snapshots can be a challenging task. This fact highlights the benefit of using a combination of unary and binary metrics. Our final set of classification rules can be used to improve current traffic classification methodologies such as Graption [16], as we discuss next in more detail.

3.1 Experimental Setup

Applications Involved. We used the following applications, grouped as being collaborative or non-collaborative. Collaborative applications include: P2P (Gnutella, FastTrack, eDonkey, MP2P, WinMX), DNS, SMTP, NTP, and HalfLife (Game). The Non-Collaborative includes: Web (http and https), POP3, SNMP, FTP, SSH, Streaming (Real Time Streaming Protocol). Discussion about other application (e.g., BitTorrent) that existed in a small subset of our traces are given later in the section.

Traces Used. To evaluate our findings we tested all our rules on four two-hour-long, one week-long and one month-long traces from four deferent locations. When we report results on static (unary) metrics, we include all our traces. Since our dynamic (binary) metrics require the IP anomymization to be consistent across all intervals, we do not include KSCY-MON and CLEV-DAYS. These traces, comprise of five minute samples with different randomization of IPs which does not allow graph comparisons between samples.

TDG Creation. For selecting the right *edge filter* we can use any of the current methods to automatically group related flows. Such methods are described in for example [11, 16, 26]. A complete traffic classification system is out of the scope of this work. As we show next, we have incorporated our methodology to our work on Graption [16] allowing it to operate entirely in the "dark" without relying on any port- or signature-based information. Our main goal here is to identify the set of behavioral properties that can distinguish between applications. For the remainder of the

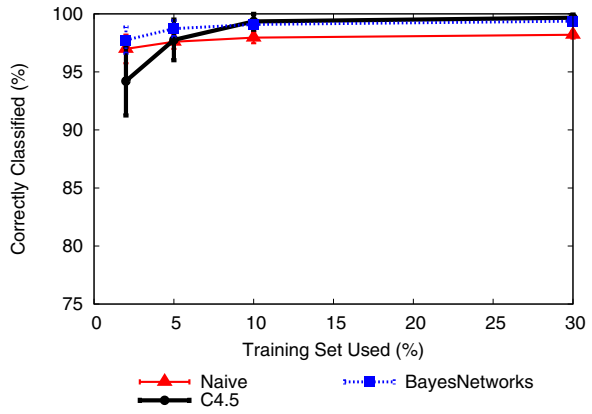


Figure 3: Collaborative vs Non-Collaborative applications: graph classification performance using static metrics. For each experiment we executed 50 runs with different random training samples.

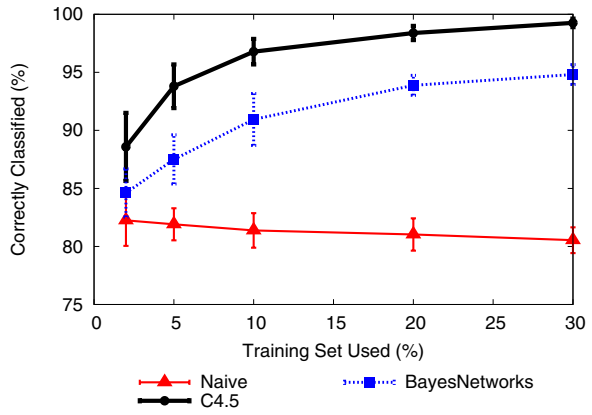


Figure 4: P2P vs Rest: Graph classification performance using static metrics. We need a larger training size in order to achieve good results compared to the Collaborative vs Non-Collaborative classification.

section, we assume that application specific edge filters are known (e.g., extracted by methods such as in [16, 3, 26, 11]). We generate all TDG snapshots over five-minute intervals. Five-minute intervals have been used before [16, 17, 41] and give good results here.

3.2 Deriving Classification Rules

For selecting the set of discriminating features, we pose the following requirements: (a) the metrics should be intuitive, making them easy to be used by network operators, (b) they must apply to different locations, avoiding particularities of a single locations (e.g., a metric to detect the Web TDG could be the existence of a high degree server, which might not be visible from all locations).

3.2.1 Static Behavior Heuristics

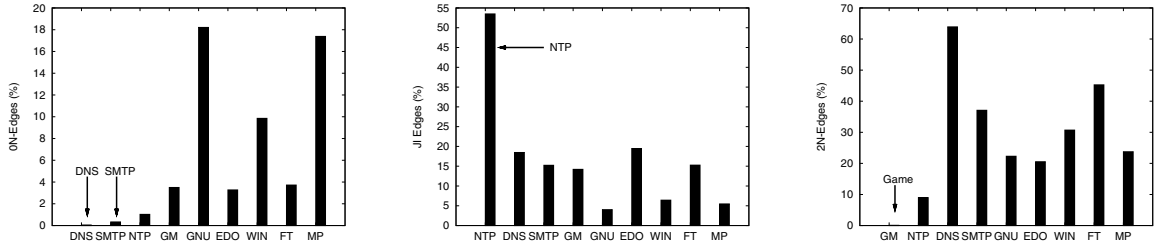
Our experience and findings from prior work [17, 16] show that static metrics can be used to efficiently distinguish between collaborative and non-collaborative applications. Intuitively, most collaborative applications have more dense graphs (higher average degree), less skewed degree distributions, and a higher percentage of InO nodes.

Static Metrics Used. For each application, we extracted a large set of static graph metrics as defined earlier in §2: assortativity, RU for degree/in-degree/out-degree distributions, average degree/out-degree/in-degree, max degree, max in-degree, max out-degree, LWCC, InO (%), Sinks (%), 75th and 95th percentile of the degree/in-degree/out-degree distribution, and BiDir(%). For the classification process, each TDG snapshot is represented as a vector of numerical values each representing a distinct graph metric. We used TDG snapshots over all five-minute long disjoint intervals from all six traces. Our final set contained more than two thousand graphs.

To automate the classification of TDG snapshots, we use standard machine learning (ML) methods: Naive Bayes, Bayesian Network, and C4.5 Decision Tree classifier. For these algorithms we used the standard implementations of WEKA [39]. The Naive Bayes is the simplest of classifiers. Bayesian networks are more complex but offer more power when the independence assumption does not hold. Finally, the Decision Tree can be easily transformed to IF-THEN rules which are easy to understand by humans (e.g., network administrators). These machine methods have been also used in the past for classifying network flows [22, 31]. We emphasize that the use of ML methods provides a rigorous framework to determine what metrics are most appropriate for classification, and hence our utilization of a large number of possible metrics is not as ad hoc as it may first appear. As we show next, these basic classifiers achieve very good results. However, the Decision Tree and Bayes classifier can overfit when large training sizes are used. The use of more advanced ML methods, such as Support Vector Machines (SVMs) [39], can further improve the classification performance as well as eliminate the problem of overfitting and this is included for future work.

Static metrics: Separating between Collaborative vs Non-Collaborative applications. The classification results for all three ML methods are shown in Figure 3. For this experiment, we increased the size of the training size, shown here as a percentage over the data set, and calculated the instances classified correctly. For each training set size, each algorithm is executed 50 times with different randomly selected samples of the training set. The figure shows the average value over the 50 runs and the standard deviations. As we see, even for a 5% training size we can extract classification rules that clearly separate between Collaborative vs Non-Collaborative applications with above 99% correct classifications. The two dominant features as selected by the C4.5 decision tree classifier are: Average Degree and InO nodes. Using only these two metrics we can distinguish between our two classes of applications in all 4 backbone link locations. Moreover, our automatically extracted rules capture inherent properties of collaborative applications. The high average degree represents collaboration (communication) between multiple hosts and InO captures the interchanged roles of the hosts.

Static metrics: identifying P2P is harder. Using static graph metrics to further distinguish between applications is harder. To show this we use the same classification methodology to try to distinguish P2P snapshots from our set of graphs. The classification results for this experiment are shown in Figure 4. Comparing Figure 4 and 3, we can



(a) Edge Volatility: DNS and SMTP change much less over time compared to P2P TDGs.

(b) Edge Consistency: NTP has the higher percentage of identical edges.

(c) Long Stay or Early Departure: Games appear to have very low number of 2N-edges.

Figure 5: Our three dynamic heuristics for the KSCY trace. The applications separated at each heuristic are reported first (leftmost side). Notations: Game (GM), Gnutella (GNU), FastTrack (FT), WinMX (WIN), eDonkey (EDO), MP2P (MP).

clearly see that separating P2P graphs is a much harder task. Even though the decision tree algorithm can extract classification rules that yield 98% correct classifications, it requires a large training set ($> 30\%$) to achieve this and results in a large tree (twenty-seven rules involving eight metrics). The classification problem is much harder for the two Bayes classifiers where classification accuracy remains lower than 95% for 90% training size. In contrast, in our previous experiment (Figure 3) we achieve 98% accuracy using the simple naive Bayes classifier with 5% sample size.

3.2.2 *Dynamic Behavior Heuristics*

Next, we introduce a set of heuristics that capture characteristic dynamic behaviors for specific applications. We show that such heuristics can easily be used to further separate between user-generated P2P file-sharing TDGs from the remaining collaborative applications, including DNS, SMTP, NTP, and a network game application (Half Life). For the following heuristics, we use the binary operators for annotated graphs as described in §2. For all experiments, the two compared graph snapshots are taken to be two hours apart from each other. Since traces KSCY-MON and CLEVDAYS comprise of only 5-minute-long samples, we do not include them in our results. For presentation purposes, we will report results for the KSCY trace. Our results on other traces are similar.

Edge Volatility: Even though DNS/SMTP form TDG snapshots similar to file sharing P2P applications, we would expect them to have a dynamic behavior that is more stable than end-user formed overlays. Intuitively, DNS/SMTP are based on an "infrastructure" of commercial servers, thus we would expect these protocols to have TDGs with lower volatility. To measure Edge Volatility, we repeat the same process as in §2 (see also Figure 2). The results of Edge Volatility are summarized in Figure 5(a). We see that DNS and SMTP have the smallest volatility values over all collaborative applications. Note that DNS is generally less volatile than SMTP so we can always use this distinctive behavior of DNS to distinguish between the two. We also observe that eDonkey is the P2P protocol with the lowest volatility. We speculate this is because eDonkey utilizes a hybrid architecture of both long-lived servers and a standard P2P architecture. We see this as evidence that our approach may be able to distinguish between distributed and centralized P2P architectures, but we leave this for future work.

Edge Consistency: A distinctive characteristic for some applications is the consistency of their interactions. We capture this behavior using the Jaccard Index on the edges of the graphs. Intuitively, TDGs that don't change significantly over time have a large number of identical edges. The NTP has this property since NTP nodes constantly communicating with a small subset of their peers leading to graphs that change very slowly over time. In Figure 5(b), we show the Edge JI for our set of collaborative applications. From the figure we can see that NTP stands out from the remaining of collaborative applications.

Long Stay or Early Departure: On all four backbone locations we had a strong presence of the Half Life online game which allowed us to study this game application in more detail. Our findings suggest that a known behavior of online gamers can also be observed using TDGs. In particular, gamers are known to query a large number of game servers in order to identify a server that satisfies their requirements for game type, round trip delay, and number of active players [14, 25]. If the player finds a server she likes, then she will continue to play the game, otherwise she will leave the system. This behavior is captured by the small number of 2N-edges, as seen in Figure 5(c), where Half Life has the smallest percentage of 2N-edges.

3.3 TDG Classification Process

Our proposed classification process is graphically illustrated in Figure 6. The decision tree is derived by hand based on the following rule. From top to bottom, at each level of the decision tree we classify the application(s) that is(are) more easily identified. At the first stage, we use static graph properties to group applications into collaborative and non-collaborative sets. We then use binary graph metrics for dynamic behavior to identify NTP, Games, and DNS/SMTP. Using this process, we are able to correctly classify all TDGs from our 4 different backbone locations using a **common set of thresholds**. An automatic process for deriving such classifications trees would be a subject for further work.

To further evaluate our TDG classifier, we have incorporated all our heuristics on the Graption P2P traffic classifier [16] using our 3 traces to train the classifier and the PAIX trace for testing. Our classifier identified P2P traffic with above 92% accuracy using only behavioral-based rules without using port-based heuristics (which existed in the original version of the system). Note that BLINC [21] achieves 85% accuracy on the same trace. BLINC requires port-based fil-

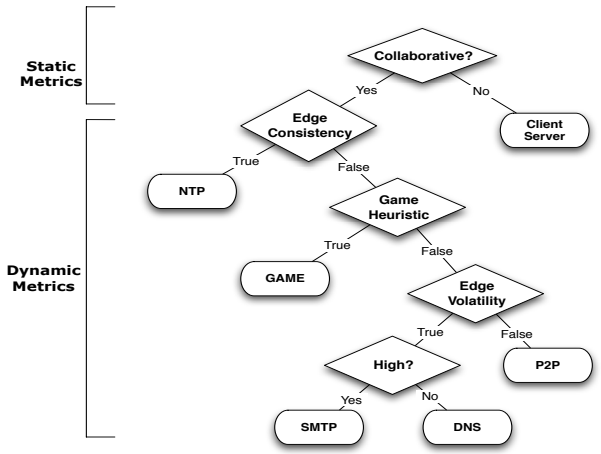


Figure 6: Final TDG-based classification process combining static (unary) and dynamic (binary) metrics. From top to bottom, at each level of the decision tree, the classifier selects the easier to identify application.

tering for the DNS application and 27 other parameters in order to achieve this accuracy [22].

The BitTorrent (BT) protocol has dynamic behavior and an InO threshold similar to other P2P applications, but it shows lower average degree. This could be due to early stages in the use of BT in 2004. Since we only observe BT from a single location we cannot generalize our findings. Obtaining more recent traces with ground truth would allow a more robust characterization. Even with BT we can create a simple classification rule. To show this, we included BT in or P2P set and repeated the aforementioned process. The C4.5 decision tree increased by one rule to include BT by using two graph metrics (95th percentile of the graphs in-degree and the max out degree). The classifier correctly classified (99.83%) of the graphs without affecting the size of required training set.

Discussion: Our study shows that TDG-based features can be used to discriminate among different applications even when TDGs are observed from different backbone locations. This observation suggests that our selection of metrics and observed properties are not specific to a single location. In addition, we acknowledge that, in some cases, it is easy to identify a group of flows as belonging to a legacy application such as DNS without using TDGs. For example, one could use port numbers or IP addresses of well-known servers. However, these methods have their limitations. For example, port numbers can be easily faked and IP addresses may not be usable in an anonymized trace.

4. DETECTING CHANGES IN TDGS

Network administrators need to be able to detect changes and anomalies. Here we focus on polymorphic blending, an anomalous behavior in which an evading application such as a P2P file sharing protocol, attempts to pass as a legacy application. Currently, because of rate limiting policies enforced by many ISPs, P2P applications use polymorphic blending by typically changing their port to a number used by a legacy application (e.g., port 80). We expect such attacks to increase in intensity and sophistication making the problem of detecting these anomalies more challenging for network administrators.

We emphasize that our goal here is not to find specific misbehaving nodes, although TDGs should be useful for such a task. Rather, here we are showing only that TDGs can be useful to detect the change in the profile of a legacy applications and inform the network administrator about the event. Extending the approach to the task of identifying misbehaving entities remains a promising subject for future work.

4.1 Experimental Setup

To detect polymorphic blending, the monitoring system must first generate a profile of each application under normal conditions. Then, it must monitor the TDGs of a set of applications (e.g., DNS), looking for behaviors that deviate from their expected profile. Throughout this section, we use the term **target** applications to refer to the set of legacy applications that we monitor and the term **polluting** applications for the "intruding" applications.

Applications Involved. Our set of target applications includes {HTTP, DNS, SMTP, NTP} and the set of polluting applications includes {Gnutella, eDonkey, FastTrack, MP2P, WinMX}. All target and polluting applications are present in all our backbone locations. This allows us to compare the detection ability of each metric over the same application at different points in time and over different backbone locations. In a practical setting, we will also be interested to detect changes in a small set of legacy applications. The four target applications we inspect here account for more than 50% of the flows in all our traces.

Traces Used. For the experiments that follow, we use traces PAIX, KSCY, and CLEV which contain two hours of contiguous traffic. Such traces are required for applying our dynamic (binary) metrics. We assume that our traces represent the behavior of the application under normal (without pollution) conditions. We verified this using payload analysis and methods from [22, 16].

Graph Metrics. For each five-minute snapshot we compute all static (unary) metrics as listed in §3.2.1. From the binary metrics, we compute the Jaccard Index (JI) for: nodes, edges, V_{InO}, V_{snk}, and V_{src}; the 2N-edges, 1N-edges, 0N-edges; and the graph edit distance. We also compute each of the above metrics for particular subsets of the edges: (a) the top 10% edges from each graph ordered using the MED (which we defined to stand for minimum endnode degree of an edge), (b) the bottom 10% edges from each graph ordered using MED, (c) top 10% edges from each graph ordered using the number of bytes transfered over the edge, and (d) bottom 10% edges from each graph ordered using the number of bytes. Intuitively, by sorting edges according to their MED and number of bytes gives a more focused view to the network by isolating edges with high/low endnode degrees and high/low traffic intensities respectively.

For unary metrics, we calculate all metrics over each five-minute interval resulting to 24 values for each target application over a two hour-long trace. The binary metrics are applied to five-minute snapshots with a 30 minutes gap; using a sliding window of 30 minutes and a 5 minute step. Time gaps between 20 to 30 minutes (depending on the trace) give similar results and are omitted due to space limitations. Using a 30 minute window, for each target application, we get 18 values for each binary metric over the two hour trace.

Experimental Methodology. For evaluation, we use standard methods from anomaly detection [6] and count the number of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) alarms. For each target application, we first generate a sequence of snapshots that contain no injected P2P edges and compute all metrics (unary and binary). For each metric, we use half of the values to learn the behavior of the target application under normal (no pollution) conditions. The remaining values are used to count for any wrong alarms (FP) and for no-alarms (TN). We measure the level of pollution using the ratio of edges from the P2P TDG that we inject to the target application's graph; which we refer to as **pollution intensity**. We "pollute" all target applications each time with a different polluting P2P and different pollution intensity {0.001, 0.01, 0.1, 0.5, 1}. For each experiment, we have a single target and a single polluting application. Using the previously extracted profile of the target application, we look for deviations that can trigger an alarm. Given that there is pollution, all alarms are counted as TP and all no-alarms as FN. We elaborate more on the metrics (unary and binary) that we used and our detection processes later in the section. We here note that for injected P2P traffic, we use the actual P2P IPs and edges present in the interval we are polluting. For example, to pollute the first five minute DNS snapshot using Gnutella, we will use the actual Gnutella edges from the same trace that are present during the first five minutes. In this way, we simulate a real-world scenario where the Gnutella application changes its traffic to look like DNS, say in terms of port number or flow characteristics, and thus, appears in the same TDG as DNS.

4.2 Experimental Results

Analyzing both static and dynamic graph properties can result in a large collection of metrics. Our first goal is to select the best metrics for the task at hand.

Let us see an example to get the visual intuition of the process. In Figure 7 we show an example of two candidate features, namely the average degree (bottom) and the number of edges (top). Our goal here is to detect the pollution by Gnutella to the Web TDG over a month long trace[2]. Until 08/09 we do not inject any Gnutella traffic, so the normal and polluted traffic streams are the same. At 12/09 we begin injecting Gnutella traffic thereby causing the metrics for the normal (without Gnutella) and the polluted Web TDGs (with Gnutella) to deviate. We see that the average degree appears to offer better discrimination between the normal and polluted streams, and hence rates to be more useful in designing a test for anomalies. It is clear that we need to select our metrics carefully.

Selecting Relevant Metrics. To evaluate each metric we use **Accuracy** which is defined as $(TP + TN)/(TP + TN + FP + FN)$. For these experiments, an alarm is triggered when the value of a metric deviates X number of standard deviations from the average value over the training period (under no pollution). To rank metrics we do the following. We calculate the average accuracy over a series of experiments where we vary: the target application, the polluting application, the pollution intensities, and the detection threshold (1,2,3 and 4 standard deviations). From our results we observed that entropy metrics have lower

[2]Because of technical problems, for this trace, there is no data between the period 08/09 and 12/09.

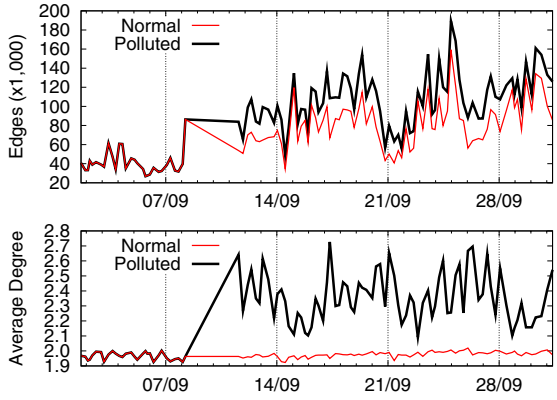


Figure 7: The changes in average degree (bottom) and the number of edges (top) of the Web TDG after we inject Gnutella traffic. Plots show both the values of the metrics under no pollution (normal) and after pollution (polluted).

accuracy compared to other metrics, such as: directionality (InO), component sizes, and average degree. Moreover, component-based metrics work well, and the size of the 10 LWCCs (Top10) is consistently among the best metrics selected. Component-based metrics work well since, in most cases, P2P nodes tend to form their own components that are disconnected from the components of the target application. We repeated the same experimental process using all our dynamic metrics. It is interesting to note that the dynamic metrics appear to be more application-specific. Given the distinct dynamic behavior of different TDGs (as shown in §3) this observation is expected. For example, NTP is more sensitive to the JI of edges, as we have seen in §3.2.2, whereas DNS is more sensitive to 0N-Edges. This suggests for a more "personalized" selection of metrics for different target applications.

Detection Method. We first measure the average and standard deviation of each metric over the normal period. We set a "violation" to be triggered when a metric value exceeds three standard deviations from the average. Our selection of the threshold works well for all traces and for both static and dynamic metrics. Such basic distance-based detection methods are used in other areas of anomaly detection [6] and have showed success in detecting network anomalies [23]. To reduce false positives, we use a more advanced detection process. We give an alarm after three consecutive "violations" on the same metric, for any of the metrics we use. We can use this very simple heuristic under the assumption that a P2P application that uses polymorphic blending will do so persistently over time (otherwise there is little gain from blending, and relatively little cost of not detecting it). This will result in multiple alarms over time.

For our experiments, we use three configurations, each involving ten metrics: **(a)** monitoring only the top ten static metrics, **(b)** monitoring only the top ten dynamic metrics, and **(c)** monitoring the top ten from both the static and dynamic metrics together (still ten metrics in total). The best metrics are selected using our ranking method (Selecting Relevant Metrics).

Using the combination of static and dynamic metrics gives

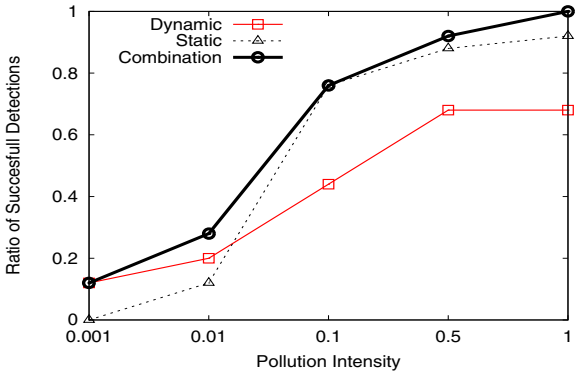


Figure 8: Successful detections over different pollution intensities. For small intensities, our dynamic metrics result in more detections compared to the static. The combination of dynamic and static metrics gives better results in the majority of our experiments.

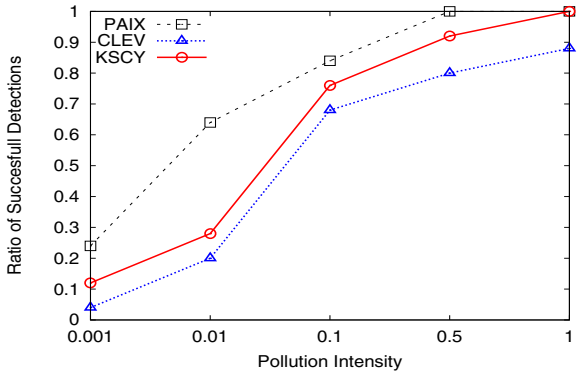


Figure 9: Detection rate per pollution intensity for three different backbone links.

higher detection ratio. The results for all three configurations are summarized in Figure 8 for the KSCY trace. Using the best ten metrics and our detection heuristic resulted in **zero FP in all experiments**. Note that for each pollution intensity we measure the alarms for each of the four target applications for each of the five polluting P2P protocols. The reported successful detection represents the ratio of successful detection over all twenty experiments (one for each target and polluting pair). One important observation from Figure 8 is that the combination of the best static and dynamic metrics works better than using any of the two groups in isolation. We observed this for the majority of our experiments. In Figure 8, we see that our successful detections are high, especially for pollution above 0.1. We also have many detections for pollution intensities as small as 0.01 and 0.001. Considering the large size of most of the target applications compared to the polluting applications our results are very good for most configurations. Similar observations were reported for the other traces as well.

Dynamic metrics result in more detections for small pollution intensities. From the results in Figure 8, we see that for small pollution intensities ($\leq 1\%$) dynamic metrics are more sensitive in detecting anomalies. Most of the detection concern DNS and NTP TDGs. We attribute this to the very characteristic dynamic behavior of these applications compared to P2P (see Figure 5). For example, even a small number of P2P edges can change the edge volatility of DNS making it easy to detect the change. We have observed this behavior in all our traces.

In Figure 9, we show the overall detection performance using the combination of static and dynamic metrics over a range of pollution intensities for three different backbone links. Using the combination, we achieved above 68% detection ratio for pollution intensities as low as 10%, which is very encouraging. Our results show that using this combination of static and dynamic metrics works better in all cases. In the future, we plan to use methods that can better utilize information from all the metrics, both static and dynamic.

Discussion: Other network changes can be caused by misbehaving nodes[3] and upgrades to the existing network infrastructure (e.g., the addition/removal of high degree servers). Incorporating additional causes of anomalies appropriately is a subject for future work.

[3]In [7], the authors show how the size of the LWCC for the Web application changes during the presence of a hit-list worm. Other examples of misbehaving nodes can be triggered by scanning activity.

5. RELATED WORK

Many papers use graphs to model various interactions in network settings. For example, trust propagation networks and other social networks (e.g., [42]) are often expressed as graphs. What separates our paper from others is the focus on dynamically changing graphs, and the application to network traffic classification and anomaly detection. We provide comparisons with the most closely related work below.

The first use of TDGs we know of was for the detection of worm activities within enterprise networks [36, 10]. Their main goal was to detect the tree-like communication structure of worm epidemics within an enterprise. This characteristic of worms was also used for post-mortem trace analysis using backbone traces [40]. 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 [7].

Graph-based Flow Classification (Graption) [16]. In our previous work on TDGs we focused on graphs derived from "static snapshots", or a single graph representing a short period of time [17, 16]. In Graption [16], the static profile of P2P applications was used to classify their traffic. In more detail the method in [16] works as follows. Graption first uses unsupervised methods (clustering) to automate the grouping of network flows into different applications using flow-level features (e.g., payload-signatures, packet sizes, etc.). The output of this step is a set of groups each containing flows from (ideally) a single application but without knowledge of which application is in each group. The network-wide profile of P2P traffic is then used to identify which groups belong to P2P applications. Our current work represents a significant advance in several regards: we introduce dynamic metrics, allowing us to go beyond a single snapshot; and we perform a large-scale study of TDGs with traces over multiple time periods and locations in the backbone.

A recent work by Jin et al. [18] uses graph-partitioning methods to extract and study smaller communities (subgraphs) within a TDG. Such communities can represent communications between known servers (e.g., Google) and clients accessing the particular service. In [18], they also study the temporal properties of the subgraphs showing that they are persistent over time. The work in [18] is different from our study of the dynamic characteristics of TDGs where the changes of the entire graph are used to describe the dynamic behavior of an application (e.g., DNS).

Another set of related work studies the connectivity patterns of users within enterprise networks for the extraction of Communities of Interest (CoI) [1, 37, 28, 38]. Any particular CoI will contains enterprise hosts with similar behavior (e.g., connections) and habits (e.g., a common mail server). In [37], graphs are used as a means of modeling connections and grouping similar hosts within corporate networks. Again, these papers differ substantially from ours in that we focus on using graphs to understand and model network-wide behavior of Internet applications.

Anomaly detection is a well studied research area [23, 6]. However, the problem of detecting polymorphic blending attacks by P2P applications is new and to the best of our knowledge has not been addressed before. Most prior efforts on anomaly detection focused on changes in resource consumption [23], without monitoring the behavior of particular applications. The problem of profiling applications in the backbone was the topic in [41], where they used entropy to summarize distributions and group related applications (e.g., client-server) together. However, the authors in [41] did not address the detection of changes in the profile of network applications. We have included entropy metrics in our feature set for profiling network traffic. Indeed, we build on prior efforts on application profiling [41] and anomaly detection [23] by utilizing unary and binary metrics for TDGs.

Tools for the analysis and visualization of network traffic, such the Autofocus tool [13] and Plonka's FlowScan [33] can infer volume-based anomalies by highlighting patterns of large resource consumption in network data. Our work can be seen as a complementary approach to these tools, by providing a means to visualize, understand, and analyze the static and dynamic network-wide behavior of applications.

A host-based method for traffic classification is BLINC [21]. In [21], "Graphlets" were used as method to model the flow level characteristics of particular IP address (host). BLINC hints at the benefit of analyzing the node interaction at the "social" level, but it ultimately follows a different path focusing on the behavior of one node at a time and does not use network-wide graphs or graph differences as we do here.

Passive monitoring of P2P overlays is studied by Sen et al. [35], targeting mainly the profiling of P2P hosts, including the measurement of bandwidth usage, how long they remain active, etc. The goal of the measurement is to support traffic engineering and not for profiling the application. A similar study for large DNS traces [8] uses graphs in the context of classifying DNS servers according to their role in the DNS-hierarchy and for generating a space-efficient DNS traffic summary. Moreover, neither work uses the dynamic nature of the graphs as we define them here.

A recent study by Latapy et al. [24] measured the evolution of TDG-like graphs between all the hosts exchanging a single packet of any type. The high aggregation of this graph is very different from our separate view of the traffic generated by different applications. Meiss et al. [30] used sampled Web flows to extract statistics for the behavior of clients and servers regarding their cardinalities and the level of traffic exchanged between them. While similar in spirit, we believe our work greatly expands and improves on these approaches.

Polymorphic Blending (PB) Attacks. These attacks have been introduced by Fogla et al. [15] in the context of network intrusion detection. The basic idea is very simple. If an attacker mimics the behavior of legitimate traffic, it can be very hard for an intrusion detection system (IDS) to identify the intruder. This is similar to the *adversarial classification* problem [9] where an adversary uses its knowledge about the IDS in order to constantly change its profile and evade detection. A general solution to the problem does not exists. In [15] they suggest a method to address the problem by applying multiple IDSes that operate at different levels. Our approach follows this basic direction by detecting PB attacks at the level of network-wide interactions using TDGs.

6. CONCLUSIONS

The key contribution of this paper is the introduction of the study of dynamic behaviors in graph-based traffic analysis. We achieve this by expanding the definition of TDGs to a collection of graph snapshots of network behavior over time. To the best of our knowledge, this is the first study that uses dynamically changing graphs to characterize and classify network traffic.

We explore metrics that effectively capture and summarize the changes of a TDG over time and apply our approaches to two network monitoring problems: traffic classification and the detection of polymorphic blending attacks. Our results show that when studying TDGs a combination of static (unary) and dynamic (binary) metrics give higher descriptive power to the use of TDGs. Our findings are very promising and strongly suggest that our methods can form the basis for a new wave of techniques for traffic analysis and monitoring.

7. ACKNOWLEDGMENTS

Support for Marios Iliofotou and Michalis Faloutsos is provided by the NSF grant NETS-0721889 and URP grants from Cisco Systems, Inc. Michael Mitzenmacher is supported in part by NSF grant CNS-0721491 and a URP grant from Cisco Systems, Inc. The computer resources provided by SDSC's are founded by the National Science Foundation TeraGrid project. Support for CAIDA's Internet traces is provided by the National Science Foundation, the US Department of Homeland Security, CAIDA Members, and the DatCat system.

8. REFERENCES

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

[2] L. Bernaille, R. Teixeira, I. Akodjenou, A. Soule, and K. Salamatian. Traffic Classification on the Fly. *ACM SIGCOMM CCR*, 36(2):23–26, April 2006.

[3] L. Bernaille, R. Teixeira, and K. Salamatian. Early Application Identification. In *ACM CoNEXT*, 2006.

[4] CAIDA Org. The CoralReef Project, http://www.caida.org/tools/measurement/coralreef/.

[5] CAIDA Trace Project. http://www.caida.org.

[6] V. Chandola, A. Banerjee, and V. Kumar. Anomaly Detection: A Survey. *ACM Computing Surveys*, 2009.

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

[8] C. Cranor, E. Gansner, B. Krishnamurthy, and O. Spatscheck. Characterizing Large DNS Traces Using Graphs. In *ACM IMW*, 2001.

[9] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma. Adversarial Classification. In *ACM SIGKDD*, 2004.

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

[11] J. Erman, M. Arlitt, and A. Mahanti. Traffic Classification Using Clustering Algorithms. In *ACM SIGCOMM MineNet*, 2006.

[12] J. Erman, A. Mahanti, M. Arlitt, and C. Williamson. Identifying and Discriminating Between Web and Peer-to-peer Traffic in the Network Core. In *WWW*, 2007.

[13] C. Estan, S. Savage, and G. Varghese. Automatically Inferring Patterns of Resource Consumption in Network Traffic. In *ACM SIGCOMM*, 2003.

[14] W. Feng, F. Chang, W. Feng, J. Walpole. A Traffic Characterization of Popular On-line Games. *IEEE/ACM Transactions on Networking*, 13(3):488–500, 2005.

[15] P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee. Polymorphic Blending Attacks. In *USENIX Security Symposium*, 2006.

[16] M. Iliofotou, H. Kim, P. Pappu, M. Faloutsos, M. Mitzenmacher, and G. Varghese. Graph-based P2P Traffic Classification at the Internet Backbone. In *IEEE Global Internet Symposium*, 2009.

[17] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, and G. Varghese. Network Monitoring Using Traffic Dispersion Graphs (TDGs). In *ACM IMC*, 2007.

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

[19] T. Karagiannis, A. Broido, N. Brownlee, kc claffy, and M. Faloutsos. Is P2P dying or just hiding? In *IEEE GLOBECOM*, 2004.

[20] T. Karagiannis, A. Broido, M. Faloutsos, and kc claffy. Transport Layer Identification of P2P Traffic. In *ACM IMC*, 2004.

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

[22] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee. Internet Traffic Classification Demystified: Myths, Caveats, and the Best Practices. In *ACM CoNEXT*, 2008.

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

[24] M. Latapy and C. Magnien. Complex Network Measurements: Estimating the Relevance of Observed Properties. In *IEEE INFOCOM*, 2008.

[25] Y. Lee, S. Agarwal, C. Butcher, and J. Padhye. Measurement and Estimation of Network QoS Among Peer Xbox 360 Game Players. In *PAM*, 2008.

[26] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M. Voelker. Unexpected Means of Protocol Inference. In *ACM IMC*, 2006.

[27] P. Mahadevan, D. Krioukov, B. Huffaker, X. Dimitropoulos, kc claffy, A. Vahdat. The Internet AS-Level Topology: three data sources and one definitive metric. *ACM SIGCOMM CCR*, 36(1), 2006.

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

[29] A. McGregor, M. Hall, P. Lorier, and J. Brunskill. Flow Clustering Using Machine Learning Techniques. In *PAM*, 2004.

[30] M. Meiss, F. Menczer, and A. Vespignani. On the Lack of Typical Behavior in the Global Web Traffic Network. In *WWW*, 2005.

[31] A. Moore and D. Zuev. Internet Traffic Classification Using Bayesian Analysis Techniques. In *ACM SIGMETRICS*, 2005.

[32] P. Papadimitriou, A. Dasdan, and H. Garcia-Molina. Web Graph Similarity for Anomaly Detection. Technical report, Stanford University, 2008.

[33] D. Plonka. FlowScan: A Network Traffic Flow Reporting and Visualization Tool. In *LISA*, 2000.

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

[35] S. Sen and J. Wang. Analyzing Peer-to-peer Traffic Across Large Networks. *IEEE/ACM Transaction on Networking*, 12(2):219–232, 2004.

[36] Steven Cheung et al. The Design of GrIDS: A Graph-Based Intrusion Detection System. *UCD Technical Report CSE-99-2*, 1999.

[37] G. Tan, M. Poletto, J. Guttag, and F. Kaashoek. Role Classification of Hosts within Enterprise Networks Based on Connection Patterns. In *USENIX Annual Technical Conference*, 2003.

[38] J. Tolle and O. Niggenmann. Supporting Intrusion Detection by Graph Clustering and Graph Drawing. In *RAID*, 2000.

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

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

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

[42] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. SybilGuard: Defending Against Sybil Attacks via Social Networks. In *ACM SIGCOMM*, 2006.