Traffic Analysis using Dynamic Traffic Dispersion Graphs: A study in Time and Space

[UCR-CS-2009-06220. June 22, 2009]

Marios Iliofotou
UC Riverside
marios@cs.ucr.edu

Michalis Faloutsos
UC Riverside
michalis@cs.ucr.edu

Michael Mitzenmacher
Harvard University
michaelm@harvard.edu

ABSTRACT

Traffic Dispersion Graphs (TDGs), which represent network traffic in a "social network" form, have recently been proposed as an alternative way to model, interpret, and visualize network traffic. While previous studies examine static snapshots of TDGs, the goal of this paper is to explore their full potential by explicitly considering the dynamic nature of network traffic, extending TDGs to dynamic Traffic Dispersion Graphs, or d-TDGs. Our first contribution is a systematic methodology for studying the properties of these graphs. For example, we present graph metrics that capture concisely the dynamic nature of the graphs, and address practical issues on how to create and use such graphs in practice. Our second contribution is a massive study of the spatiotemporal properties of d-TDGs based on real backbone traces. A key conclusion is that d-TDGs are both sensitive enough to capture the critical communication patterns, and robust over time and location of observation. Finally, we show strong evidence of how useful d-TDGs can be in traffic monitoring tasks focusing on application classification.

1. INTRODUCTION

Our work is motivated by the need for better tools to analyze, visualize, and classify network behavior and traffic. Specifically, we need better tools for modeling and visualization of traffic that can assist in general tasks such as anomaly detection and pattern discovery and more specific tasks such as application classification and intrusion detection. While there are many current and powerful methods and tools, we need to continually work to improve them, especially considering the ongoing battle between those who wish to use these tools and those who try to actively obfuscate traffic and avoid detection.

A recently introduced way of analyzing traffic is to examine network-wide behavior using 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. The term *Traffic Dispersion Graph* or *TDG* is used to refer to such a graph [15]. TDGs appear to have excellent descriptive power, but at the same time 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 studies with TDGs focused on security issues, such as intrusion detection [34] and worm propagation [9, 39]. A more recent study argues for wider capabilities of TDGs. Previous work, however, has focused on graphs derived from "static snapshots", or a single graph representing a short period of time. For TDGs based on static snapshots to be useful, however, they must be consistent over time and space. That is, one needs to find properties of TDGs that remain consistent when considering snapshots from different periods of time and from different locations. Questions remain regarding the stability of such TDGs and their characteristics taken over longer periods, and in different locations.

In this paper, we explore the full potential of TDGs by explicitly considering the dynamic nature of these graphs, extending the previous notion of TDGs to dynamic traffic dispersion graphs, or **d-TDGs**. We propose a systematic set of methods for analyzing d-TDGs, and we conduct arguably the first massive study of d-TDGs at the backbone. The key message of our work is that, if we use the right graph metrics to analyze d-TDGs, we can develop powerful techniques for modeling and analyzing network behavior.

In more detail, our work consists of three thrusts. First, we develop a systematic approach for using d-TDGs. On the one hand, we address methodological issues, such as the introduction of graph metrics that can capture the concisely the dynamic nature of the graphs. For example, we capture graph volatility in terms of nodes and edges, which can help us distinguish d-TDGs of persistent infrastructure based versus ephemeral applications. At the same time, we also address practical issues of how to utilize d-TDGs in practice. For example, we thoroughly discuss how to set the observation window in order to maximize the usefulness of d-TDGs.

Second, we conduct a large-scale study on the properties of d-TDGs. The high-level conclusion is that d-TDGs are both sensitive enough to capture critical communication patterns and robust enough over variations in time and location of observation to be useful for practical applications. Our study uses multiple backbone traces within the years 2002-2008 collected from five different backbone locations (two Tier-1 Commercial ISPs, two links from Internet2, and one backbone link from the WIDE academic network). In ad-

dition, we study a large range of applications including the Web, DNS, SMTP, NTP, POP3, eDonkey, Gnutella, FastTrack, WinMX, MP2P, etc. An interesting observation is that the persistency of an edge is better predicted by the degrees of its endnodes than its load, such as the number of bytes/packets transferred over that edge.

Finally, we focus on how d-TDGs can form the basis for powerful methods and tools for practical questions by using d-TDGs as the basis of a system to classify traffic according to application type. Specifically, we find an intuitive set of of rules that characterizes P2P d-TDGs over 4 different backbone locations.

The rest of the paper is organized as follows. In §2, we formally define d-TDGs and present a series of graph metrics used in this work. In §3, we provide a systematic method for generating d-TDGs in practice. In §4, we present a massive study of the properties of d-TDGs in time and space. In §5, we show the usefulness of of d-TDGs in practice. In §6, we discuss related work.

2. DEFINITIONS AND GRAPH METRICS

Here we provide the basic definitions underlying our work.

We refer to an ordered sequence of network packets as data trace. A TDG is a graph $G(V, E)$ that represents the network-wide interaction (say "who talks to whom") from a data trace. Typically, a node corresponds to a distinct IP address, and an edge signals an interaction. 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 §3. In their more general form, TDGs are directed and weighted graphs as we discuss below and in §3.

In practice, our data trace consists of a set of packets for a particular **interval of observation**, which we denote by T. Given an edge filter, we can create a TDG that represents all the edges that matched the filter during the interval. 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 edge. In addition, the edge can be associated with a weight or other associated information, which could represent useful information such as the number of packets exchanged.

A d-TDG denotes a series of TDG graphs $G_1, G_2, ..., G_n$ with a common edge filter. Each graph G_i corresponds to a particular interval of observation, and we refer to it as a **snapshot** of the d-TDG. 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 adopt the framework that that a node that is inactive (with respect to the edge filter) in a particular interval does not appear in the graph for that interval, so nodes as well as edges can vary among the G_i. Although it is not our primary motivation for introducing d-TDGs, we suggest that the successive snapshot framework is also a useful practical way to manage and store data in many scenarios.

We now present graph metrics that we use to describe and compare graphs. Several metrics are not commonly used in the measurement community currently, and our choices represent experience based on time-consuming trial and error.

Quantifying Static Graph Properties.

We start by reviewing standard graph terminology and terminology used in previous studies of TDGs [15]. 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.

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). As we will see later, a large fraction of such nodes is a sign of P2P activity. 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**).

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 $MED(e) = min(d(u), d(v))$. Some well known graph metrics are based in the use of the degrees of nodes. 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 of a randomly selected node to have degree k, and is defined as $P(k) = n(k)/n$ for $k = 1, k_{max}$. The entropy of the degree distribution $H(X)$ is defined as $\sum_{k=1,k_{max}} P(k)log(P(k))$. For measuring the uniformity of the distribution, we use the **Relative Uncertainty** $RU = H(X)/\log_2 k_{max}$ metric as defined in [40]. (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.

To quantify the connectivity of a graph, we use the size of its Giant Weakly Connected Component (**GWCC**). If we consider again the graph as undirected, the GWCC is the size of the largest connected component; we report these quantities as a percentage of the total number of nodes in the TDG.

The **assortativity coefficient** r is a summary metric of the correlation of the degree between endpoints of edges in the graph [22]. 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.

Quantifying Dynamic Graph Properties.

In studying d-TDGs, a fundamental question is to quantify the evolution, which can be done by comparing successive snapshots. We therefore consider various metrics that compare two snapshot graphs. 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 unlabeled labeled representations. Thinking of unlabeled grpahs, 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). Operating at both levels, we can have more detailed insight into the graph behavior. For example, structural similarity does not imply membership consistency and vice versa.

For quantifying changes in node and edges 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 G_1 that are also present in G_2.

Detailed Edge Similarity. We introduce metrics that quantify in finery granularity the similarity of the edges between two graph, G_1, G_2. The $RI(E_1, E_2)$ represents 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, $E_1^{1N} = \{(u, v) \in E_1 : (\text{one of } u \text{ or } v) \in V_2\}$; (d) the *0N* -edges with both nodes not present in V_2, $E_1^{0N} = \{(u, v) \in E_1 : u, v \in V_2\}$.

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 [29].

Is useful to compare graphs by taking under consideration the "ranking" of nodes and edges. For example, it might be more important for a high degree (e.g., server) node to to change than for a low degree node (e.g., client). For ranking nodes we could use many features such as degree of nodes and volume metrics using the number of flows/bytes/packets. We say the top ranked node (e.g., node with maximal degree) has rank one, the seconds has rank two, and so on. Intuitively, by comparing sorted sets of elements we can identify differences in a finer granularity compare to aggregate reports over the entire set of edges or nodes. Such methods were also used previously for the detection of anomalies on the Web [29].

Ranked Detailed Edge Similarity (RDES). For the definition we use an example based on ranking edges $e \in E_1$ by the number of bytes sent between its endnodes, denoted as $b(e)$. We sort the edge set E_1 in non-increasing order, $E_1 = \{e_1, e_2, ..., e_n\}|\ b(e_1) \geq b(e_2) \geq ... \geq b(e_n)$. We define $E_1[i, j] = \{e_i, ..., e_j\}$ to be an ordered subset of the edges of the graph G_1. We will be using the 2N-edges for this definition but it can be used for other subsets of edges. We define $E_1^{2N}[i, j]$ to be the 2N-edges that belong in the ordered subset $E_1[i, j]$.

With RDES we can for example report the percentage of the top 10% ranked in bytes edges of E_1 that are also found in E_2. Note that the definition does not require the edges in E_2 to be also in the same rank range.

While we have focused this discussion on metrics for d-TDGs based on comparing snapshots, given a complete data trace one could consider further dynamic graph properties not based on such comparisons. For example, one could ask about the rates of various events over time on each of the edges, and cluster nodes or edges based on these rates. In this work we consider the utility of snapshots, but examining these other metrics on d-TDGs remains an interesting direction for future work.

2.1 Data Sets

We present the vital statistics of the backbone traces that we use in this study in Table 1. All traces are IP-anonymized and contain traffic from both directions of the link. We used 8 data sets from 5 different links collected from 2002 to

No.	*Alias*	*Start Date/Time*	*Duration*	*IPs**	*Flows**	*Bytes*	*Packets*	*Mbps**	*Payload*	*Type*
1	CLEV	2002-08-14/ 09:00	2 hour	232,579	1,857,709	777.2 GB	1,008 M	863.7	No	Consec.
2	KSCY-MON	2002-09-01/00:00	1 month	258,535	2,535,199	5.3 TB	7,742 M	1374.4	No	Samples
3	KSCY	2002-08-14/ 09:00	2 hour	198,752	1,782,979	683.05 GB	886 M	789.4	No	Consec.
4	CLEV-DAYS	2002-08-27/ 09:00	5 days	171,359	2,128,738	898.5 GB	1,289 M	1041.8	No	Samples
5	PAIX	2004-04-21/17:59	2 hour	258,636	888,750	891 GB	1529.2 M	997.1	16 bytes	Consec.
6	OC48	2002-08-14/09:00	3 hour	480,637	3,375,899	864.7 GB	1,414 M	960.8	No	Consec.
7	WIDE	2008-03-18/00:00	3 days	148,713	452,388	819.02 GB	1,293 M	75.8	No	Consec.
8	WIDE-2007	2007-01-01/00:00	1 year	114,824	263,325	540 GB	1,432 M	27.6	No	Samples.

Table 1: For all the fields with * we report the average values over 5-minute long intervals. Traces (except No.5) can be found at `http://pma.nlanr.net/Special/`, `http://www.caida.org/data/passive/`, and `http://tracer.csl.sony.co.jp/mawi/`.

2008. Traces were collected during both night and day hours as well as workdays and weekends accounting for around 10 Tera Bytes of raw TCP/UDP traffic. All traces with the exception of PAIX are publicly available from MAWI [23] and NLANR/CAIDA [4].

Traces OC48 and PAIX (Palo Alto Internet eXchange) are collected from backbone links of two Tier-1 commercial ISPs. The PAIX contain up to 16 bytes of payload from each packet thereby allowing the classification of flows into applications using standard signature matching techniques similar to the ones described in [16, 18, 32]. The payload classifier managed to classify 99.3% of the flows while only 0.7% did not match a known signature.

The data set also includes traces collected from the Abilene (Internet2) academic network, from a link connecting the Indianapolis router towards Kansas City (KSCY-MON, KSCY) and Indianapolis towards Cleveland (CLEV, CLEV-DAYS). Traces WIDE and WIDE-2007 contain data from a transpacific link connecting the Japanese academic network (WIDE) with the US . WIDE-2007 contains 15-minute long samples from each day in 2007 taken at exactly 2pm. Traces CLEV-DAYS and KSCY-MON, comprised of several five-minute-long samples taken daily over five consecutive days and one month respectively. The starting point of each sample was uniformly chosen every 4-5 hours. All other traces of type *Consec.* (Table 1) contain uninterrupted consecutive collection of packet traces. The duration of each trace is given in Table 1. We used CoralReef [3] for analyzing and processing all traces.

Application Set. In this paper, we study the network-wide behavior of many different applications. To be able to establish long-term patterns, we study applications that were present in all our data sets (with the exception of the WIDE traces, where P2P activity was minimal). The WIDE traces allow the study of legacy applications (e.g., DNS, SMTP, HTTP) over longer time intervals. By considering applications present in all traces, we compare the behaviors of each application at different points in time and space. The set of applications used in this study are the following: DNS, SMTP, POP3, NTP, HTTP, MSN, SSH, FTP, HTTPS , Streaming, eDonkey, MP2P, WinMX, FastTrack, Gnutella. We used a combination of port-based techniques, payload signatures, and commonly used methods [17, 3, 32].

Some P2P applications, for example Soribada, GoBoogy, and BitTorrent, were only present in the PAIX trace. Unfortunately, BitTorrent was not present in 2002, where most of our traces were collected. The WIDE traces have very low P2P activity and they are not useful for studying P2P traffic [6].

For each application, we extracted a large set of static graph metrics as defined earlier in this section: assortativity, RU for degree/in-degree/out-degree distributions, average degree/out-degree/in-degree, max degree, max in-degree, max out-degree, GWCC, InO (%), Sinks (%), 75th and 95th percentile of the degree/in-degree/out-degree distribution, BiDir(%).

3. A SYSTEMATIC APPROACH

We discuss essential and subtle practical issues on how to define and use d-TDGs in practice.

3.1 Edge Filtering

Edge filtering is a critical element in using d-TDGs effectively, as we mentioned earlier. Each edge filter gives rise to the creation of a d-TDG graph, and each such graph provides a focused view to the network traffic. We present a set of general edge filters that can be used in isolation or in combinations in order to select the right set of flows to form an application specific d-TDG (e.g., Gnutella d-TDG, DNS d-TDG, etc.).

a. Port-based filters. These filters monitor 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. We will often use the name of the dominant application at a port number to refer to a port-based d-TDG.

b. Signature or content-based filters. These filters match string patterns in the payload of the packet. These filters can be very powerful, but assume that: (a) we have the signature of the desired traffic, and (b) we have access to the content of the packet. These assumptions are not always true for all applications or all traces.

c. Flow-based filters. These filters create a graph with flows that meet certain flow-level features, such as packet size, packet inter-arrival, number of packets. In fact, the

grouping of flows into TDGs can be agnostic and based on machine learning and clustering algorithms [38] [6]. The advantage of these filters is that they can be made to work with no a priori information of how applications behave[1].

How do we decide which edge filters we need? 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 signature filter. To go one step further, if the application uses encryption then we use flow-based filters. We further discuss the selection and use of edge filters in the context of application classification in §5.

In this work, we focus on modeling and understanding the differences in the behavior of applications. Given the available traces, we use a combination of signature-based and port-based to define d-TDGs and we establish the ground truth using existing methods [17, 3, 32], as we described in §2.

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.

In addition, we consider that each d-TDG graph is a simple graph, e.g., we do not allow multiple edges in the same direction between two nodes. Note that with the majority of the filters that we use, this is a non-issue anyway.

3.2 Interval of Observation

How long should the observation interval be? The creation of a d-TDG snapshot requires a time interval over which the data is collected. A short interval will create sparse graphs that do not contain useful and stable information. On the other hand, a long interval could create large graphs that may make the detection of interesting phenomena difficult and increase the computational complexity. Ideally, we would like the minimum interval that can provide sufficient information for the purpose of a study.

Our work makes two key observations:

a. For our backbone traces, 5 minutes is a sufficient observation interval for reasonably stable graphs.

b. We propose an event-based definition of the observation interval, as an alternative way for creating d-TDGs. With this definition, we let the graph grow until a fixed number of nodes or edges. The idea is that such graphs are less sensitive to the intensity of the traffic at the time of observation. In fact, we show that a traffic increase may sometimes give the false impression of a behavioral change in the observed graph.

[1]We have used this approach successfully for application classification in our going work [37], and in other works, [2, 25, 21, 10, 11] Here, we do not use this approach, and this creates a clear separation between our two works.

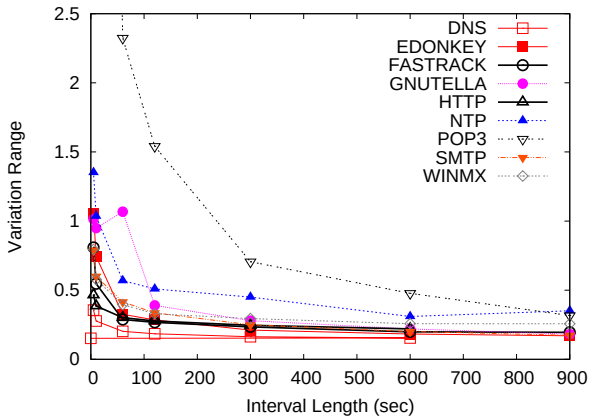


Figure 1: The effect of changing the interval of observation for TDGs ranging from 5 seconds up to 15 minutes over a large set of protocols. To reduce variability we can choose to use longer intervals. After 300 seconds the reduction in variability is not significant.

3.2.1 *Selecting the Observation Interval*

We examine the effect the interval of observation on: (a) the graph metrics, and (b) the usefulness of the resulting graph in application classification.

a. The effect of the observation interval on the graph metrics. First, 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 graph metrics, which we list in §2.1.

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. In fact, we then report the average range over all graph metrics, and we plot this average range versus the duration of the observation interval in Figure 1. 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 1, 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 1).

b. The effect of the observation interval on application classification. We conduct one more test, which further verifies the conclusion that 5 minutes is a sufficient interval for our backbone traces. Specifically, we show that 5 minutes can enable a Naive Bayes [38, 14] classifier to classify applications with higher accuracy compared to smaller intervals. Note that our interest here is not the specifics of the method, but the effect of the observation interval on the method's ability to perform well. The key idea is that high variability of graph metrics will introduce classification errors.

In our test, we train and then test the classifier over graphs generated with 5 seconds, 1 minute, and 5 minute intervals from traces: CLEV, OC48, KSCY. Each TDG snapshot is represented as a vector in an n-dimensional space where each dimension represent on graph metric taken over the graph. A complete list of applications and the corresponding metrics used is listed in §2.1.

For the classifier, we use the commonly-used 10-fold cross-validation process [14]. For statistically meaningful results, this process is iterated 10 times with a different random sample (10%) at each iteration. Therefore, for each experiment, the classification is executed 100 times using a random sample of 10% for testing and the remaining 90% for training. With smaller than 30 seconds intervals the error was higher than 6%. The error can below 1% for 60 seconds and below 0.3% for 5 minutes. Clearly, a larger choice of interval can reduce the variability allowing the structure of a graph to become more stable.

In the remaining of the paper, we use 5-minute long intervals for generating graphs, unless otherwise stated.

3.2.2 *Using Event-Based Intervals*

A change in a graph metric in a d-TDG could be the result of: (a) a change in the nature of the behavior, or (b) a side-effect of a change in the traffic intensity. We show how we can separate the two with the use of event-based intervals, where we let the graph grow until a fixed number of nodes or edges is collected.

In Figure 2, we show the average degree of the DNS d-TDG (using port-based filter) from trace samples taken each day at exactly 2pm over one year from WIDE-2007. The top plot, shows the time series over the entire year for Mondays' and Sundays' samples. For both days, d-TDGs have been collected over fixed 5-minute long intervals. Larger observation intervals showed qualitatively similar differences between the two days.

Here, we test whether this behavior is due to the difference in the traffic intensity between weekends and weekdays. We use an event-based interval, where the interval of observation is such that each d-TDGs graph has exactly 28,000 nodes. This value is approximately the same as the average number of nodes for the DNS d-TDG during the weekdays. The results are shown in the bottom plot of Figure 2. Clearly, the

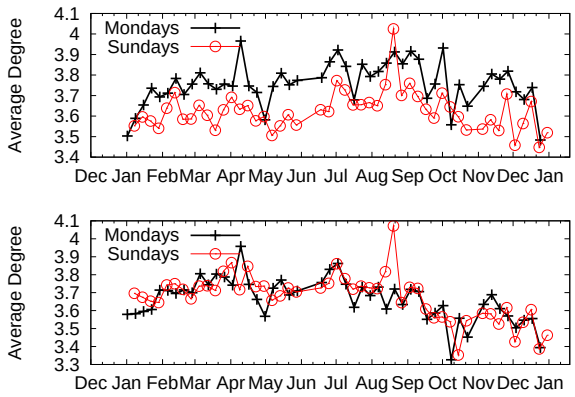


Figure 2: Top: Changes to the average degree over the year 2007 for TDGs grown over fixed time interval of 300 sec. Bottom: The same TDGs now grown over fixed number of nodes 28,000. Over fixed time interval due to lower activity over the weekends we have less dense graphs. On the other hand, if we grow the graphs over fixed number of nodes, we can clearly see that DNS TDG follows the same changes and patterns over the year on different days.

two time series are much closer with each other and seem to follow the same trends over the year. We obtain similar result with other graph metrics such as assortativity, Relative Uncertainty, and GWCC. We have similar trends for all the days of the week compared to weekends, and not just for Mondays compared to Sundays. We also observe the same phenomenon for SMTP and HTTP on this trace, as well as with other year-long traces from MAWI [23]. Since we don't have traces spanning so many days from the other backbone locations, we could not verify this observation with the other traces.

Isolating traffic-related from fundamental behavioral changes. The above observation suggests that the difference between the weekend and weekday graphs is due to a difference in the traffic, and not a fundamental change in the nature of the application. Simply put, the graph metric captures the decrease in traffic, which leads to less interactions and less dense graphs. This is an important distinction to make when one studies the behavioral properties of applications, and we will revisit it in the next section.

4. SPATIO-TEMPORAL D-TDG PATTERNS

Before we use d-TDGs in network analysis, we need to understand the sensitivity of d-TDGs across time and space (from different locations at the backbone). Indeed, if they vary too widely and unpredictably across time and space, their utility will be sharply limited.

Our work suggests that d-TDGs can be a useful way for analyzing network traffic. This is based on the following observations: a. d-TDG metrics vary relatively smoothly over time. b. d-TDG metrics exhibit strong 24 hour periodicity, while a time difference of 12 hours corresponds maximizes

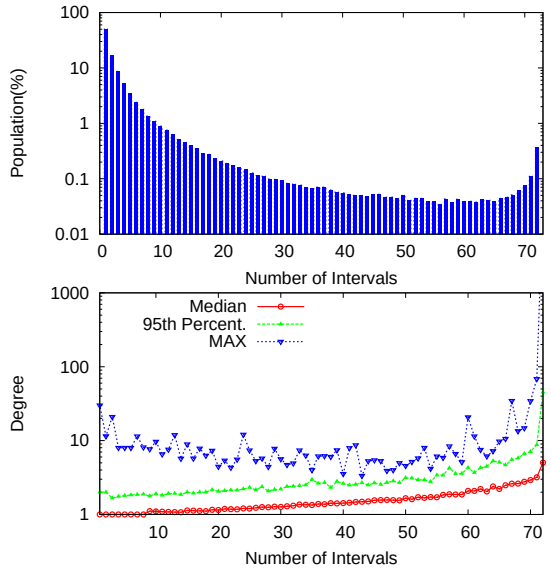


Figure 3: Top: PDF of the number of intervals that each DNS node appeared in. Bottom: The degrees of nodes that appear in X many intervals. Clear correlation between the degree of a node and its consistency in the graph.

the difference. c. Node and edge membership is very dynamic. For example, even infrastructure-based applications do not have large ever-present set of nodes. d. The likelihood of an edge appearing in a subsequent interval is better predicted by the degrees of its endnodes than its traffic load. e. d-TDGs of applications exhibit sufficiently similar behavior in several different backbone locations.

As we show in §5, the above key observations are sufficient to achieve separation of applications across multiple backbone links using a common set of simple rules.

4.1 d-TDGs in Time

For the first set of measurements, we use the WIDE trace. Even though we don't have access to payload, a very recent study by Kim et. al [6] showed that the amount of P2P traffic transferred over this link is very low (lower than 0.6%). In addition, in [6], they show that by using port numbers they could classify all legacy applications with very high accuracy. Next, we show results for the DNS d-TDG behavior. We also discuss similarities and differences with SMTP and Web traffic collected on the same link.

a. Time-based observation interval. All of our measurements show that the graph structure changes little between consecutive snapshots, with larger changes as the time gap between snapshots increases. The graph structure shows also shows periodicity with a period of (roughly) 24 hours as shown in Figure 4-TOP, and the largest differences are between graph snapshots taken roughly 12 hours apart. We observe this periodicity by varying a sliding widow of various lengths and comparing graph snapshots taken at both ends of the window. This observation is also visually apparent in the plot. The plot also shows that even though graph metrics have a periodic variation, the changes are very smooth.

We obtain similar plots for other applications such as SMTP and HTTP, which we do not show due to space limitations.

b. Event-based observation interval. To remove the effect of traffic volume variations, we generated the same plots using event-based intervals, as defined in the previous section. Here, we require all the graphs to have the same number of nodes (20,000 nodes). The results are shown in Figure 4-BOTTOM. It is easy to see that the top and bottom plots are qualitatively similar exhibiting smooth evolution and strong 24-hour periodicity.

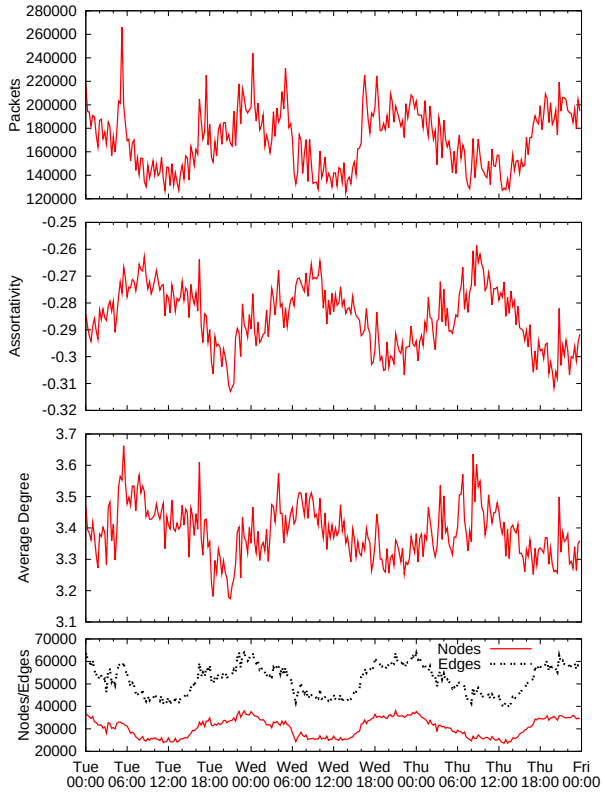
4.1.1 Static and Dynamic Graph Components

If we consider the identity of each node, a natural question to ask is the following: *"Do we see the same nodes edges all the time?"*. A typical d-TDG behavior is shown in Figure 3, which shows the probability distribution function for the number of intervals that each DNS node appeared in for the WIDE trace. For this experiment, we monitored snapshots of DNS d-TDGs taken every hour over the 72 hour of the trace (WIDE). The plot shows how many nodes appeared in a single interval, two intervals, etc. As we can see, the majority of nodes $> 50\%$ remain active only in a single interval. Only 0.3% of the nodes appear in all intervals. We refer to those nodes as the **core** of the graph. We see qualitatively similar behavior for the vast majority of graphs we studied. The bottom plot in Figure 3 shows a positive correlation between the degree of a node and the number of times it is present in a graph snapshot. The figure shows the median, 95^{th} percentile and the maximum degree of all the nodes that appear in k intervals as a function of k. The degree is averaged across all graph snapshots that a node is present. This was a consistent behavior for all applications. The only exception was graphs that included scanning activity: scanners are both ephemeral and of high degree.

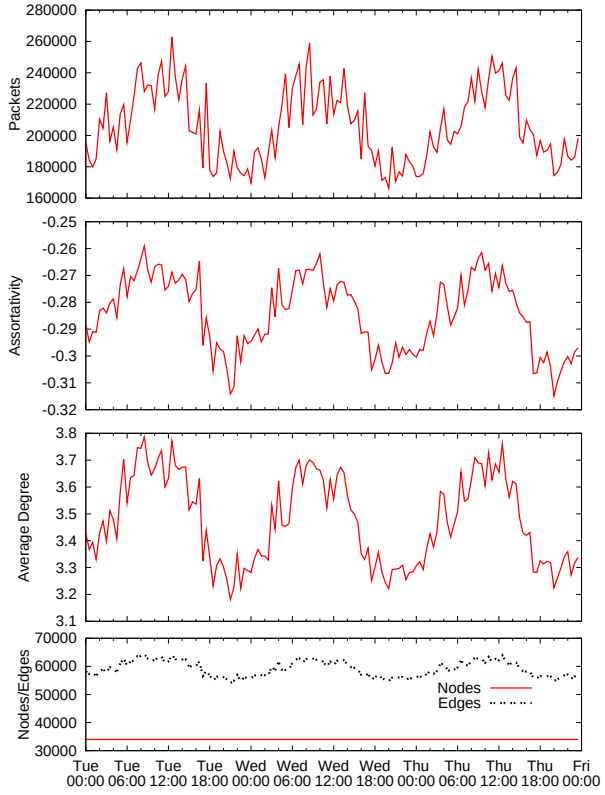
Discussion. The relatively small size of the core shows how dynamic the nodes that participate in an applications are. A possible use of the core is that perhaps storing a core set of nodes for each TDG is not as impractical as it may sound, since the relative size of the core is much smaller compared to the overall number of participating nodes. By storing the core for each application, we can possibly detect anomalies such as the removal of few core nodes from the graph.

4.1.2 Node and Edge Volatility

We now consider whether useful insights can be obtained by using labeled graph comparisons. To start, in Figure 5.a, we show the Relative Inclusion for nodes and edges for a TDG $G_{t=0}$ taken at time t=0 (the beginning of the trace) compare to a three *target* graphs taken (a) back-to-back , (b) 12 hours apart, and (c) 24 hours apart. As expected, the higher the gap the smaller the overlap between the nodes.



(a) TDGs formed over 5 minute intervals.



(b) TDGs formed to have 34,000 nodes.

Figure 4: Periodicity over 3 consecutive days in 2008 for the WIDE trace using: (a) time-based and (b) event-based observation intervals.

Going one step further, Figure 5.b shows the RI of $G_{t=0}$ for nodes with different "roles". At consecutive intervals, we have higher similarity in *sources*, which we hypothesize represent ephemeral clients. The InO nodes and the *sink* nodes have more consistent presence over time. We surmise such nodes to represent DNS servers that form the core of the graph.

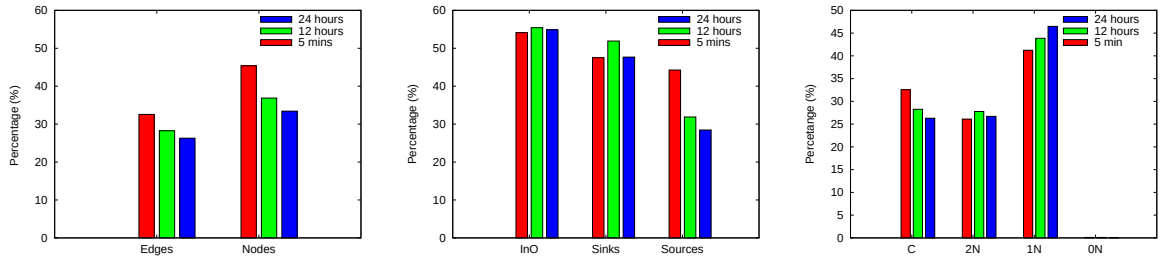
For showing the edge volatility, we present the Detailed Edge Similarity (§2) in Figure 5.c. The 0N-edges of $G_{t=0}$ are those that have both endnodes not present in the target graph. The most important observation is that the number 0N-edges is very low ($< 0.5\%$). This is one observation for DNS that appeared in all our backbone traces. We also observe that even though the number of common (C) edges is decreasing over larger time gaps, the number of edges that have their two endnodes common (2N) remain fairly stable. This shows that common edges are mostly replaced by other edges with one common endnode (1N). Such edges are likely to belong to ephemeral clients contacting a server node.

4.1.3 Understanding changes in finer granularity

By using more descriptive metrics (as we show in Figure 5.c) to compare graphs, we have better understanding of what changed and what remains stable in the graph of an application. The next step is to rank edges (or nodes) based on a feature. Such a feature can be for example the number of total bytes transferee over the edge. Having the set of edges (or node) ordered by bytes we can ask queries like: "How many from the top 100 heavy hitter edges (ranked by bytes) have remained present after two hours?".

Towards this end, we use the Ranked Detailed Edge Similarity (RDES), as defined in §2, to understand traffic changes in a finer granularity. We now describe the main steps for generating what we will refer as **Ranked Detailed Edge Similarity plot**. This visualization can help us isolate the changes in the edges of a graph over time. For this plot, we place ranked edges into bins of fixed size. For example, we can have 10 bins each one collecting 10% of the edges. The first 10% ranked of the edges are placed in bin 1, the next 10% in bin 2, and so on. For each bin, we compute the Detailed Edge Similarity labeling each edge accordingly (C, 2N, 1N, ON).

We show an example with 10 bins where we rank the edges according to the number of bytes in Figure 6.a. This plot shows the changes in the edges of the eDonkey d-TDG (OC48) in snapshots one hour apart. In Figure 6.a, the edges are ranked by bytes. It is interesting to see that for the top 10% edges (bin 1), we have roughly 18% of 0N edges. Recall that 0N are the edges in the first graph with both endnodes not present after one hour. We observed this behavior in all our P2P TDGs. We conjecture that these edges are transfers of files between peers that finished within an hour. After the downloading is over, these nodes did not appear to communicate with other peers.



(a) Nodes and Edges (b) Per node role (c) Detailed Edge Similarity

Figure 5: Comparing the graph changes of DNS when snapshots are taken: back to back, 12 hours, and 24 hours apart.

Edges between high degree nodes are more likely to survive. Surprisingly perhaps, for the the vast majority of d-TDG (including P2P applications) using graph metrics can give more predictive ability than volume based metrics. In Figure 6.b, we present the ranked detailed edge similarity for the eDonkey d-TDG (OC48) where we rank the edges using the minimum endnode degree (MED). Note that the highly ranked edges of bin 1 have zero 0N edges, and are more likely to continue to exist (roughly 30% do) compared to under 20% in bin 1 of Figure 6.a. In Figure 6.b, if we compare across all bins, the higher MED edges in bin 1 are more likely to remain in the graph. We verified this observation over a large range of applications (DNS, P2P, etc.).

In the next section, we show how we used the edge volatility as a measure to distinguish between applications. For the rest of the paper, we use MED to rank the edges in a graph.

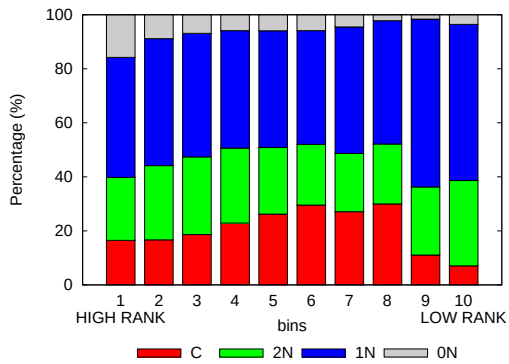
4.2 Spatial Properties

In this section, we study d-TDG characteristics over space, from traces taken at different backbone links.

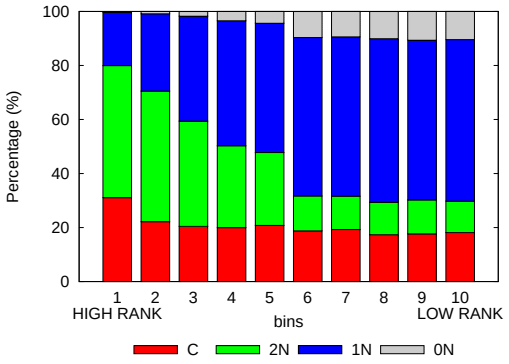
a. The fundamental properties of graph instances of the same application remain comparable across locations. Our finding are very promising and show that a single point of observation (a backbone link) can provide sufficient information, in that d-TDGs from different locations give qualitatively similar results for many key applications. Clearly, some deviations will exist depending on the location of a link. However, the key idea is that the carefully defined metrics will not change. For example, the highest node degree may be different but the distribution of degrees will be comparable. Due to space limitations, we do not show any plots here. Instead, in the next section, we show that the same rules can distinguish the d-TDGs of many different locations.

b. The evolution of d-TDGs is similar across locations. The dynamic nature of the graphs is one of the key properties that show high similarity across different locations. To illustrate this, we show an example using the RDES plot for the WinMX P2P application taken from traces OC48 (commercial ISP) and KSCY(Interne2). The comparison is shown in Figure 7. At first we see that both figures are visually very similar. WinMX represents a P2P applications with a decentralized architecture and this is reflected in its highly dynamic behavior. We observed qualitatively very similar visualization over the range of protocols we studied. In §5 we will show how we can use the graph volatility of P2P applications in order to facilitate separation between classes of d-TDGs.

As a side note, we can also compare WinMX with eDonkey in Figure 6. In all our traces, eDonkey shows a more stable behavior than most other P2P d-TDGs. We attribute this behavior to the architecture of eDonkey which is based in the se of centralized servers. This finding can lead to methods to distinguish between applications of the same family (e.g., WinMX vs eDonley). We include this in our future



(a) Ranked by traffic volume (bytes)



(b) Ranked by Minimum Endone Degree (MED).

Figure 6: Ranked Detailed Edge Similarity (RDES) for the eDonkey protocol (OC48). Snapshots compared are taken 1 hour apart.

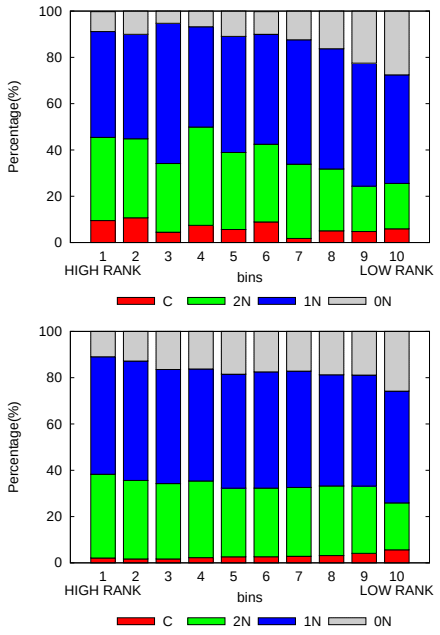


Figure 7: Comparing the d-TDG changes of WinMX using RDES from two backbone locations (top: OC48, bottom: KSCY). Both locations show signs of highly volatile graphs.

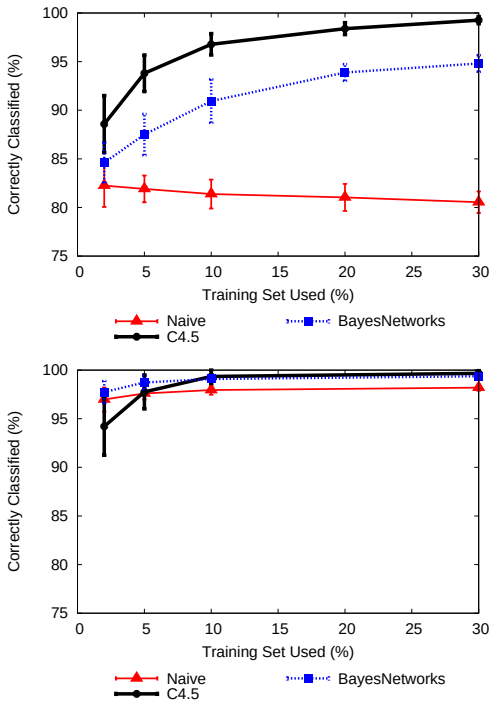


Figure 8: Top: Classification results when all applications are included. Bottom: Classification results when DNS/SMTP/NTP are not used. For each experiment we executed 50 runs with different random training samples.

directions.

The similar behavior of d-TDGs at different locations will further be verified in §5, were we explicitly show that few common rules can be extracted in order to distinguish between applications in from multiple backbone locations.

5. CASE STUDY

In this section, we show that d-TDGs can be the basis for a new generation of monitoring tools. In general, d-TDGs and tools based on them can provide novel insight and complement existing methods.

In this section, we focus on the following general problem. We are given a set of d-TDGs and we are asked to classify the d-TDGs to applications. Note that we assume that the creation of TDGs has already been done using appropriate edge filters. We have already discussed that the edge filters can be content-based, port-based, or flow-based or an appropriate combination of those. In fact, we have already described how we create the d-TDGs and how we establish the ground truth for the different traces we have.

As a case study, we will focus on separating P2P applications (e.g. file-sharing), from all other applications, which is an interesting and important question in practice. To make the problem more challenging, we would like to derive "rules" that can be applier successfully to traces from several different locations in order to ensure that our solution is not customized and specific to a single point of observation.

We used six traces collected from four different locations in the backbone (OC48, PAIX, KSCY, KSCY-MON, CLEV, CLEV-DAYS). For traces taken in 2002, we used port-based edge filters since the use of random port numbers was not common in 2002 [13, 16]. On the other hand, for the 2004 trace, we used payload signatures filters.

Our aim is to classify d-TDGs into two groups: P2P (Gnutella, Fastrack, WinMX, eDonkey, MP2P) and non-P2P (DNS, SMTP, POP3, NTP, Http, MSN, Https, Streaming) which we denote as non-P2P. At first, we use *static TDGs* in an effort to capture the structural graph characteristics of P2P applications. We used TDG snapshots over all 5-minute long disjoint intervals from all 6 traces. For each TDG we calculated all the static graph measures as discussed in §2.1. For the classification process, each TDG is represented as a vector of numerical values each representing a distinct graph metric. Our final set contained more than two-thousand graphs.

To classify TDGs, 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 [38]. 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 [6, 27].

The classification results for all three ML methods are shown in the top plot of Figure 8. For this experiment, we

increased the size of the training size, shown here as a percentage over the data set, and calculate the classified correctly instances. 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 values and the standard deviations. Even though the decision tree algorithm can extract very good classification rules that perform 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). As we show next, by using intuitive rules and standard visualization of our data, we managed to achieve better results using fewer training instances. In data mining, simple rules are much more attractive than complicate solutions. This is because, in many cases, more complicate solution are usually signs of over-fitting [38, 14].

Profiling P2P Applications. To visually analyze our data, we used the data visualization tool from WEKA [38], which provides a compact way to inspect and analyze 2-D scatter plot combinations of all our metrics. We observed that P2P applications share some common graph features. In particular, we have observed that P2P TDGs have: (a) mode densely connected graphs with higher average degrees, (b) high percentage of nodes with dual roles having both incoming and outgoing edges (InO), and (c) high RU metric (§2).

Identifying conflicting applications. From the 2-D scatter plots visualizations, we also observed that SMTP, DNS, and NTP protocols have similar behavior with P2P in the above measures. This observation also agrees with our classification experiments, where these three applications are often misclassified as P2P. This is not surprising since the protocols internally use P2P interactions with servers communicating with each other and acting both as clients and as servers.

Correctly identifying SMTP/NTP/DNS improves the classification. We want to quantify the improvement in the classification if we can correctly identify SMTP/NTP/DNS. We separate DNS, SMTP, and NTP protocols from the non-P2P class and run our three classifiers again comparing P2P with non-P2P graphs. The classification accuracy increased significantly for all methods even with smaller training sets; as we show in the bottom plot of Figure 8. In addition, even with small sample sets (5%) the decision tree managed to derive a very simple set of IF-THEN rules. In Figure 9, we show the 2-D scatterplot between the two dominant features as selected by the C4.5 decision tree classifier. On the same figure, we show the decision boundaries extracted by C4.5 using 5% training set. As we can see, using only two graph metrics (Average Degree and InO nodes) we can distinguish between classes of applications in all 4 backbone link locations.

Using d-TDGs to distinguish DNS/SMTP/NTP from P2P. Even though DNS/SMTP/NTP form static TDGs similar to file sharing P2P applications, we would expect them to have a dynamic behavior that is more stable than end-user formed

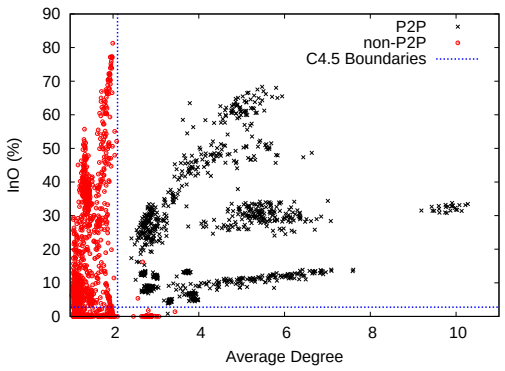


Figure 9: Scatter plots showing the average degree and InO for each TDGs in our 6 traces data set. Each point in the scatterplot corresponds to a 5 minute long TDG snapshot. For readability, we also excluded MP2P which had a much higher InO and Average Value than all other graphs.

overlays. Intuitively, DNS/SMTP/NTP are based on an "infrastructure" of commercial servers, thus we would expect these protocols to have d-TDGs with lower volatility.

To quantify the rate of change, we do the following. At t=0 we create the reference graph $G_{t=0}$. 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 DNS and SMTP compared to P2P activity where graphs change more with time (ephemeral interactions).

Here, we use the Detailed Edge Similarity (DES) defined in §2. The percentage of 0N-edges metric worked good in all test traces. 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 d-TDG compared to DNS/SMTP/NTP. To give an example of this process, we shown in Figure 10 the percentage of 0N-edges as we increase the gap between a graph snapshot and the reference graph ($G_{t=0}$). We show this for three P2P applications compare to DNS/SMTP for the PAIX trace. Even if the initial population of 0N-edges is lower for some P2P applications (e.g., eDonkey), the rate of increase is higher than SMTP and significantly lower than DNS. For the WIDE trace, here we have day-long consecutive traces, we observe that even if we increase the interval beyond many hours, the percentage of 0N-edges increases very slowly, as we have seen in the previous section (Figure 5.c). Similar observations we have for the SMTP protocol on the same trace.

In some traces, we have observed that absolute percentage of 0N-edges can be sensitive as a metric. We noticed this especially for SMTP where the percentage of such links can be higher than other protocols, such as FastTrack and eDonkey, even after two hours of observations. Isolating these ephemeral edges, we found many of the connections to correspond to failed connection attempts at the well known

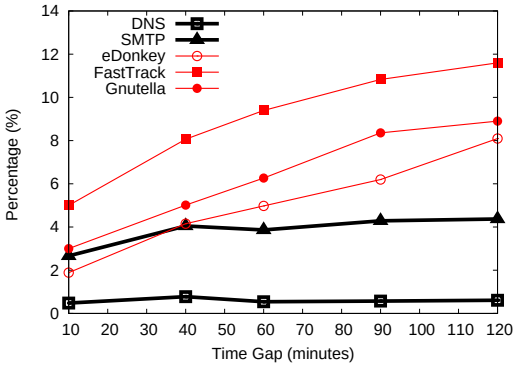


Figure 10: Effect of time to the volatility of three P2P protocols compared to DNS and SMTP. For P2P applications the percentage is increasing with a higher rate than in DNS/SMTP.

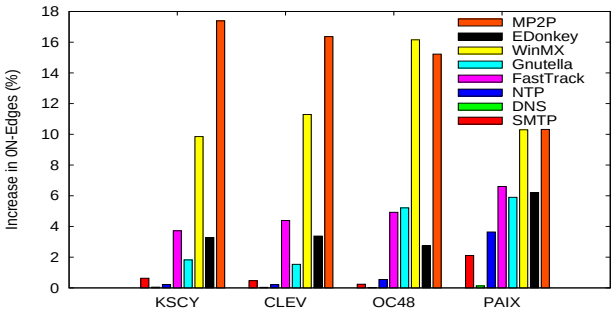


Figure 11: NTP,DNS, and SMTP have lower increase (%) of 0N-Edges over all traces.

SMTP port 25. Such failed connections is a typical sign of scanning activity looking for SMTP servers to send spam and searching for potential security vulnerabilities. We observed this in each of the 4 backbone locations. Using standard methods to detect scanning activity [30], we can further remove the and this is included in our future directions. In contrast to SMTP, in DNS even after few minutes the percentage of such edges is the lowest compare to all applications. In general, DNS seems to have the most stable d-TDG from all studied applications.

In all our traces, we found the relative increase in the percentage *0N-edges*, over sufficient time (e.g., two hours), worked good in separating between P2P applications and DNS/SMTP/NTP. The increase is calculated as the difference between the percentage of *0N-edges* when the graphs are taken back-to-back and when the graphs are taken with a two hour gap. The results are shown in Figure 11. For each individual trace, DNS/NTP/SMTP have smaller increase than P2P applications. This allows the selection of threshold to separate them at each trace. In Figure 11, we can also observe that in all traces, the applications with the highest increase are MP2P and WinMX. This agrees with our observation in §4 that volatility is one of the d-TDGs properties that show stability in space.

Practical considerations. Using our proposed method we distinguish between P2P activity and DNS/SMTP/NTP applications. A system administrator can measure the volatility on its link and set the thresholds to achieve this goal.

Our simple set of metrics for grouping TDG into P2P Vs non-P2P worked well in all our traces. Some protocols not present in all traces are Soribada, GoBoogy (2 very popular P2P protocols in Korea), and BitTorrent in PAIX trace. Soribada and GoBoody, closely follows all the observations we derived here. BitTorrent follows the dynamic behavior and InO threshold, but it shows lower average degree than other protocols. This can 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, is included in our future directions.

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 effecting the size of required training set. However, since we only have BT for a 2 hour long trace we don't include it in our main analysis.

6. RELATED WORK

Many papers use graphs to model various interactions in network settings. For example, trust propagation networks and other social networks also are often expressed as graphs (e.g.,[41]). What separates our paper from others is the focus on graph differences, and the application to network traffic classification. 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 [34, 9]. 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 (the identification of the source of a worm outbreak, the so-called patient zero) using backbone traces [39]. 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].

Previous work on TDGs focused on graphs derived from "static snapshots", or a single graph representing a short period of time [15, 37]. An application using TDGs was developed to group flows of the same application together and then extract application-level signatures using statistical methods. Our work represents a significant advance in several regards: we introduces d-TDGs, allowing us to go beyond a single snapshot; we perform a large-scale study of

d-TDGs with traces over multiple time periods and locations in the backbone; and we show how to use d-TDGs to classify traffic in collaborative and non-collaborative groups.

Another set of related work studies the connectivity patterns of users within enterprise networks for the extraction of Communities of Interest (CoI) [1, 35, 24, 36]. Any particular CoI will contains enterprise hosts with similar behavior (e.g., connections) and habits (e.g., a common mail server). In [35], 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.

Earlier methods show promising results and automated methodologies for the profiling of hosts and port numbers using flow statistics [40]. The entropy of the distribution of source and destination IP address and port numbers was used to profile individual hosts and application (at fixed port numbers). Statistical techniques have also been used for anomaly detection by monitoring the feature distribution of flows [19, 28]. Recent anomaly detection methods propose the correlation of events from multiple locations for the detection of volume based irregularities [5].

Tools for the analysis and visualization of network traffic, such the Autofocus tool [12] and Plonka's FlowScan [31] 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 work more closely related to ours by Karagiannis et. al is BLINC [18], a host-based method for traffic classification. In [18], "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. [33], 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. [20], 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. [26] 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.

7. CONCLUSIONS

The key contribution of this paper is to study whether "social graphs" representing network interactions are suitable for capturing critical communication patterns for use in traffic monitoring and analysis tools. In particular, a key concern not adequately addressed in prior work is whether this approach is robust when considering variations in behavior over time and over location of observation.

Our contribution includes the introduction of the concept of d-TDG, with which we obtain as a collection of snapshots of network behavior over time in the form of graphs. We explore metrics that effectively capture and summarize their evolution over time, and address important practical considerations such as the effect of the observation window and the use of both time-based and event-based intervals of observation.

We have backed our establishment of d-TDG with a large-scale study based on multiple traces to explore the properties of d-TDGs, taking into account variations in behavior over time and space. Our conclusions are that d-TDGs provide powerful summary information for network communication and are robust enough in time and space to form the basis for traffic monitoring tools.

To support this final conclusion, we show elements of how useful d-TDGs can be using application classification as a case study and how using d-TDG metrics can help separate the network-wide behavior of different applications. We believe d-TDGs will provide another useful tool in the arsenal of traffic analysis and monitoring techniques.

8. REFERENCES

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

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

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

[5] P. Chhabra, C. Scott, E. D. Kolaczyk, and M. Crovella. Distributed spatial anomaly detection. In *IEEE INFOCOM*, 2008.

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

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

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

[11] 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.

[12] C. Estan, S. Savage, and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. In *ACM SIGCOMM*, 2003.

[13] 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.

[14] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2006.

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

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

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

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

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

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

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

[22] P. Mahadevan, D. Krioukov, B. Huffaker, X. Dimitropoulos, kc claffy, and A. Vahdat. The internet as-level topology: Three data sources and one definitive metric. *ACM SIGCOMM CCR*, 36(1), 2006.

[23] MAWI Working Group Traffic Archive. http://tracer.csl.sony.co.jp/mawi/.

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

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

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

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

[28] G. Nychis, V. Sekar, D. Andersen, H. Kim, and H. Zhang. An Empirical Evaluation of Entropy-based Traffic Anomaly Detection. In *IMC*, 2008.

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

[30] V. Paxson. Bro: a system for detecting network intruders in real-time. *Comput. Netw.*, 31(23-24):2435–2463, 1999.

[31] D. Plonka. Flowscan: A network traffic flow reporting and visualization tool. In *USENIX LISA*, 2000.

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

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

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

[35] 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.

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

[37] Withheld for anonymity. . In , .

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

[39] Y. Xie, V. Sekar, D. Maltz, M. Reiter, and H. Zhan. Forensic analysis of epidemic attacks in federated networks. In *IEEE ICNP*, 2006.

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

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