A Flow Trace Generator using Graph-based Traffic Classification Techniques

Peter Siska
University of Bern
psiska@students.unibe.ch

Marc Ph. Stoecklin
IBM Research – Zurich
mtc@zurich.ibm.com

Andreas Kind
IBM Research – Zurich
ank@zurich.ibm.com

Torsten Braun
University of Bern
braun@iam.unibe.ch

ABSTRACT

We propose a novel methodology to generate realistic network flow traces to enable systematic evaluation of network monitoring systems in various traffic conditions. Our technique uses a graph-based approach to model the communication structure observed in real-world traces and to extract traffic templates. By combining extracted and user-defined traffic templates, realistic network flow traces that comprise normal traffic and customized conditions are generated in a scalable manner. A proof-of-concept implementation demonstrates the utility and simplicity of our method to produce a variety of evaluation scenarios. We show that the extraction of templates from real-world traffic leads to a manageable number of templates that still enable accurate re-creation of the original communication properties on the network flow level.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement techniques, Modeling techniques, Performance attributes; C.2.3 [**Computer-communication Networks**]: Network Operations—*Network monitoring*

General Terms

Measurement, Performance

Keywords

Trace Generation, Network Flows, Self-parameterization, Traffic Dispersion Graphs

1. INTRODUCTION

Traffic-monitoring systems are widely used in corporate and service provider networks to gather network-related information of business critical applications, analyze prevalent communication patterns of the traffic, collect data for accounting, or detect abnormal traffic patterns. In general, the systems use flow-based information of the network traffic, e.g., NetFlow or IETF IPFIX, exported by routers or switches. Even though these protocols summarize observed traffic flows into a compact representation, the sheer amount of flows as well as particular traffic conditions may lead to an overload and inaccurate analysis results produced by traffic-monitoring systems.

Systematic testing of monitoring systems to ensure accuracy and performance is therefore crucial, but, at the same time, poses a number of challenges. The evaluation of a system is typically performed on a collection of traces known to contain complex constellations such as attacks, high item cardinalities, or very high flow rates. Many traffic conditions detrimental to a monitoring system's performance are, however, difficult to collect as they are observed rarely or produce overloads on the collector side. Critical traffic scenarios must be generated synthetically. This is achieved with a set of manually parametrized scripts to imitate expected traffic conditions. As a consequence, the flow attributes and the structure of the traffic are limited by simplifications and assumptions resulting in unrealistic traffic structure. Evaluations using these traces often do not provide satisfying results. Even though a packet-level generator combined with a flow-exporting probe may as well produce flow traces, the overhead incurred, the risk of potential measurement errors arising in the probe, as well as their focus on preserving packet-level properties are not justified for the evaluation of flow-level monitoring systems.

In this paper, we consider the problem of flow-level trace generation to support the performance evaluation of monitoring systems under realistic conditions. We propose a template-based approach using graph-theoretic metrics to define traffic patterns present in traffic scenarios such as attacks, anomalies, and borderline cases. By combining templates and streaming the generated traffic trace to a monitoring system, the performance of the system can be studied under various conditions. Our approach enables easy-to-use customization of traffic features and characteristics in terms of the number of hosts and flows as well as the evaluation time. Moreover, we present a self-parametrization technique that extracts templates from real-world traffic traces. These templates enable the recreation of flow traces that closely resemble those of the original traffic. The template library created can be used as background traffic and combined with customized templates to produce specific evaluation scenarios.

The contribution of our work is twofold. First, we introduce the concept of traffic templates, a compact representation of traffic conditions using a set of graph metrics and their evolution over time. By means of the templates, network flow traces containing desired traffic conditions can be composed and generated. Second, we present a self-parametrization technique that extracts template parameters from existing flow traces and enables the creation of a library of background traffic. We demonstrate that the flows generated from these templates exhibit characteristics close to those in the original traffic.

This paper is organized as follows. In Sect. 2, we review existing techniques to produce evaluation traces as well as techniques related to our work. We specify our objectives and present observations that justify our design choices in Sect. 3. Section 4 presents our template-based approach and self-parametrization technique. The flow generation and the self-parametrization are evaluated in Sect. 5.

2. RELATED WORK

A vast number of packet-based network traffic generators and models have been proposed over the past years. The techniques preserve traffic properties on the packet-level, such as inter-packet gaps, file size distributions, or traffic burstiness. They have been applied in application-specific traffic generation [1], soft- or hardware performance evaluation [2], and the evaluation of anomaly detection systems [3, 4]. We focus on flow-level rather than packet-level trace generation: our aim is to produce traces that preserve traffic properties on the network flow level (e.g., connectivity patterns), which are essential in the evaluation of flow-based network monitoring systems.

Several open-source and commercial tools for generating traces in NetFlow format exist [5, 6]. Although these tools can be configured with IP address and port pools, the traces produced do not reflect realistic scenarios. Moreover, no mechanism is provided to extract traffic properties from real traces to simplify the configuration. Another framework proposes the manipulation of existing flow traces by injecting flow records into existing traces to evaluate the effectiveness of anomaly detection systems [7]. Our approach differs from their work as it generates new traffic from structural properties extracted as opposed to manipulating existing traces.

Sommers et al. proposed Harpoon, a configurable packet-level traffic generator [8, 9]. Their approach aims at generating packet sequences that exhibit similar octet and packet counts as well as temporal and spatial characteristics to those in real traffic. Related to our work is the ability of Harpoon to self-configure by extracting empirical distributions of file sizes, inter-connection times, IP addresses, and the number of active sessions from NetFlow logs. In our approach, we model and generate flow records (as opposed to packet-level traffic) and preserve flow-level properties (as opposed to per-flow properties), such as distinct connection structures and traffic patterns on service ports (applications). This is suited for the evaluation of flow-based monitoring systems using directly generated flow traces, whereas Harpoon aims at generating traffic loads (in terms of packet rate) to test network hardware, such as routers.

3. TRAFFIC DISPERSION GRAPHS

Flow-level network-monitoring systems are faced with very high flow export rates from one or more export devices (e.g., routers) in the network. The systems are equipped with high-speed insertion algorithms and specially designed in-memory database systems. The performance of the flow processing is bounded by hardware constraints as well as the structure of the monitored traffic. If the analyzer process cannot keep up with the rate of incoming flow records, the quality of the results degrades. A proper evaluation of the performance limits of monitoring systems is therefore indispensable, both in terms of export rates and nature of the traffic observed.

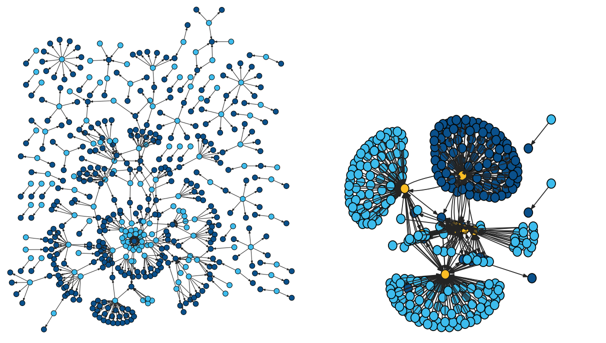

(a) HTTP (80) (b) DNS (53)

Figure 1: Visualization of port-based TDGs from two service ports. The direction of edges defines the service initiators (from) and providers (to).

3.1 Graph-based Connectivity Modeling

Our work has been inspired by the work of Iliofotou et al. [10]. They analyze the interactions between hosts on a given service port by means of graph-based metrics in *Traffic Dispersion Graphs* (TDG). A TDG is defined as a graph $G = (E, V)$ that consists of a collection of vertices V (hosts) and a collection of edges E (connections) that connect pairs of vertices. However, while they use the graph metrics for application classification, we use them as a basis for trace generation. We focus on *port-based* TDGs and add a directed edge between two vertices when a flow between the two hosts is first seen on a given service port. The initiator of a connection and thus the direction of an edge is determined by a heuristic along with a database of popular service ports.

In a preliminary study, we explored how many port-based TDGs are needed to achieve a high coverage of the connectivity patterns. We analyzed flow traces from an average-sized campus network and from a large hosting environment during different periods of time, ranging from hours up to days. We observed that the majority of flow records are associated with only a small subset of dominant service ports. On average, 90.16% ($\sigma = 5.91$) of all flows are related to only 50 service ports. Similarly, 80.99% ($\sigma = 17.59$) of all hosts and 95.1% ($\sigma = 4.2$) of all octets transmitted can be attributed to sets of 50 ports. We found that the combination of the top k service ports (e.g., with $k = 50$) in terms of the number of flows and the number of unique hosts provides a sufficiently high coverage of the connectivity patterns present in the traces.

Figure 1 shows visualizations of TDGs for two service ports. The number of unique hosts has been limited to 300 to improve the quality of visualization of the underlying traffic structure. The graphs exhibit distinct structures that are characteristic for each service port. The TDG of service port 80 (HTTP) consists of many hosts (clients) that connect to one or a few other hosts (web servers). Other service ports, such as service port 53 (DNS), exhibit denser structures containing many low out-degree vertices connecting to only a few high in-degree vertices (DNS servers). Moreover, we find that in many TDGs a large number of vertices have either zero out-degree (sinks) or in-degree (sources) and only a small portion of the hosts act as both service initiators and providers (e.g., peer-to-peer).

3.2 Graph Degree Properties

We analyzed the degree properties of various port-based TDGs. In Fig. 2, we visualize the graph vertices in a scatter plot, where the number of out-degrees and in-degrees defines the position of each vertex. We find that graphs for most ports (e.g., port 80), exhibit a

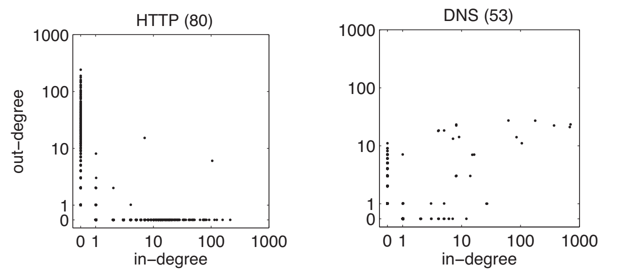


Figure 2: Scatter plots of out-degree and in-degree values of each TDG vertex for different ports on a log-log scale over a one-hour interval.

separation of vertices along both axes. These vertices have either zero out-degree or zero in-degree. Other services, such as DNS, are also characterized by vertices in the first quadrant of the Cartesian plane, reflecting vertices with non-zero in- and out-degrees.

The vertex degree values in TDGs reflect the communication patterns between hosts. We use this observation to define a method to capture the traffic structure for a given port. Due to space restrictions, we limit the presentation of the results to HTTP and DNS traffic; the two applications are distinct in terms of their connectivity patterns and representative for the majority of applications.

4. METHODOLOGY

Our flow trace generation technique consists of four building blocks. We model the connection patterns on distinct service ports by *partitioning* the joint degree distribution. We introduce the concept of *traffic templates* to express these patterns and additional traffic characteristics. Flow traces are generated from a collection of such traffic templates by a *trace generator*. Moreover, we describe a *self-parameterization* technique to extract template parameters from an existing set of flow traces.

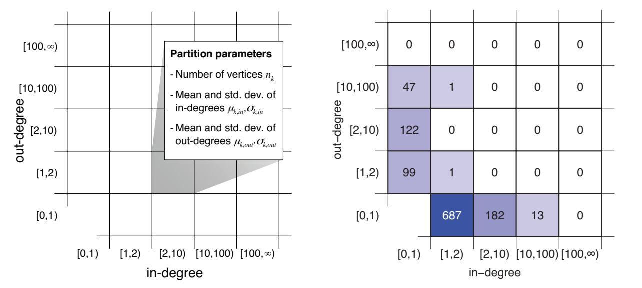
4.1 Partitioning

We divide the plane spanned by the out-degree and in-degree values of vertices into fixed-sized partitions (quantization). We chose the partition boundaries such that vertices are separated into meaningful partitions to distinguish between zero, low, medium, and high degree vertices. The partitions are determined by the non-linear left-closed intervals $[0, 1)$, $[1, 2)$, $[2, 10)$, $[10, 100)$, and $[100, \infty)$. This leads to a set K of 24 relevant[1] partitions as shown in Fig. 3(a). For example, a web server accessed by a few clients falls into one of the low in-degree partitions. A popular web server is probably to be assigned to a high in-degree partition (e.g., $[100, \infty)$), while clients likely exhibit zero in-degrees, but non-zero out-degrees.

The vertex population in a partition $k \in K$ is characterized by five parameters $G_k = (n_k,\ \mu_{k,out},\ \mu_{k,in},\ \sigma_{k,out},\ \sigma_{k,in})$, where n_k is the number of vertices in partition k, $\mu_{k,out}$ and $\mu_{k,in}$ are the mean out-degree and in-degree of the vertices, and $\sigma_{k,out}$ and $\sigma_{k,in}$ are the respective standard deviations.

Figure 3(b) shows an example of a partitioning for HTTP traffic (port 80) measured in a 300-s interval. The value (and color intensity) in each partition indicates the number of vertices n_k. The zero out-degree partitions (bottom row) contain most likely web servers, where popular servers with high in-degrees are in partitions on the right-hand side and less frequently visited servers are in

[1]We ignore the partition $[0, 1) \times [0, 1)$ in the bottom left corner in Fig. 3(a) as by definition, a TDG does not contain isolated vertices.



Partition parameters
- Number of vertices n_k
- Mean and std. dev. of in-degrees $\mu_{k,in}, \sigma_{k,in}$
- Mean and std. dev. of out-degrees $\mu_{k,out}, \sigma_{k,out}$
out−degree
in−degree
[100,∞)
[10,100)
[2,10)
[1,2)
[0,1)
0 0 0 0 0
47 1 0 0 0
122 0 0 0 0
99 1 0 0 0
687 182 13 0

(a) Degree plane partitioning (b) Partitioning of HTTP (80)

Figure 3: Left: The out-degree and the in-degree plane is divided into 24 distinct, fixed-sized partitions. Right: Partitioning of a 300-s interval for traffic analyzed showing the number of vertices in each partition.

partitions on the left-hand side. Similarly, the clients are in the zero in-degree partitions (left column), with clients connecting to several web servers in the upper and clients accessing few servers in the lower partitions.

4.2 Traffic Templates

A traffic template captures the structural properties of the connection patterns and the flow attribute value distributions for a time period and a given service port. A template consists of two parts: a representation of the degree distribution partitioning and a collection of distribution parameters defining admissible flow attributes. Moreover, a period length T is assigned to each template to maintain the temporal dimension associated with the template definition. Table 1 depicts the list of all template components.

The analysis of parameters from partitions generated over a number of time intervals of varying length allows us to determine the temporal behavior of the number of hosts active on a given service port p_{dst} and of their connectivity properties. With increasing numbers of distinct hosts observed, the connectivity patterns change. For example, for DNS traffic, the in-degree of servers generally increases with the number of clients observed whereas the number of servers remains constant. For web traffic, on the other hand, many new distinct servers appear in low in-degree partitions, whereas only the in-degree of popular servers increases.

To address the dependency of the traffic structure on the host population, we express the partition parameters as a function of the number of hosts. As such, we approximate the parameters in G_k of each partition $k \in K$ with a polynomial function. Each partition is then expressed as a collection of coefficients $\vec{a}_{g,k}$ for each partition parameter $g \in G_k$. From the resulting polynomials we derive the partition parameters for a desired number of hosts when generating flow traces.

4.3 Flow Trace Generation

The process of flow trace generation consists of three steps. (1) The templates to produce a traffic scenario are selected by the user and customized if desired. (2) For each template selected, a TDG of the underlying traffic structure, whose set of edges E represents admissible connections, is generated. (3) The flow records, whose attribute values have been populated, are shuffled across all templates and output.

4.3.1 Template Customization

The customization of traffic templates consists of the (a) parametrization of flow record fields, such as the number of packets, octets, or flow duration, (b) scaling the number of flows, and (c)

Table 1: Traffic Template Parameters

Parameter	Description
T	Period length associated with the template parameters.
$\vec{a}_{g,k}$	Polynomial coefficients for each partition parameter $g \in G_k$ for every partition $k \in K$.
p_{dst}	Destination (service) port for service initiators.
P_{src}	Source port range for service initiators.
IP_{src}, IP_{dst}	Ranges for source and destination IP addresses.
$d_\mu, d_\sigma, p_\mu, p_\sigma, b_\mu, b_\sigma$	Lognormal distribution parameters for the duration, packets, and octets flow record field.
μ_H, σ_H	Mean value and standard deviation of the number of hosts.
μ_F, σ_F	Mean value and standard deviation of the number of flows.

adjustment of the number of hosts.

To parametrize flow record fields, the parameters of the number of packets (p_μ, p_σ) and octets (b_μ, b_σ) as well as for flow duration (d_μ, d_σ) are set. The number of flows (and its variability over time) to be generated during a time interval is controlled by the parameters μ_F and σ_F. Similarly, the number of hosts is customized with μ_H and σ_H.

4.3.2 Generating TDGs from Templates

We establish a TDG from the joint degree distribution given by the partitioning. The parameters, i.e., the number of vertices in each partition and the mean degrees and deviations, are derived from the polynomial functions evaluated for the parametrized number of hosts. To establish a TDG, we apply a random graph algorithm, the *matching algorithm* proposed by Newman et al. [11], which efficiently generates random graphs with arbitrary degree distributions.

First, all graph vertices are initialized for each partition. Each vertex in the graph is associated with a unique IP address drawn randomly from either IP_{src} or IP_{dst} depending on their degree setting. Then, a set of out-stubs or in-stubs (or both) is assigned to each vertex; a stub can be considered as an open end of outgoing or incoming edges. The number of stubs for each vertex in a partition k is drawn from the normal distribution defined by $\mathcal{N}(\mu_{k,out}, \sigma^2_{k,out})$ and $\mathcal{N}(\mu_{k,in}, \sigma^2_{k,in})$. Finally, the out-stubs and in-stubs are picked randomly in pairs and joined to form edges in the graph. In order to generate graphs without self-loops and multiple edges between two vertices, we use a modified version of the matching algorithm proposed by Milo et al. [12].

4.3.3 Flow Record Generation

The number of flows generated for consecutive time intervals of length T is drawn from a normal distribution defined by the template parameters μ_F and σ_F. The flow generator randomly chooses connections from the lists of admissible connections (set of edges E in a TDG) from all templates and creates two flow records for each entry[2]. Values for duration, packets, and octets flow record fields are chosen by sampling from the lognormal distribution defined by the parameters d_μ, d_σ, p_μ, p_σ, and b_μ, b_σ respectively. The end timestamps of flows are sampled from a uniform distribution over a period of length T. The derived end timestamps, in conjunction with the duration values, yield the start times and thus the time-dependent interleaving of flows. Finally, the port fields in the records are updated with the destination port p_{dst} and source port chosen from the port range P_{src}.

[2]As a simplification, we assume that for every flow between two hosts a responder flow exists. The attributes of the responder flow are set by reversing the source and destination IP addresses and ports.

4.4 Self-Parametrization

In addition to manually defining traffic templates, our approach provides the ability to automatically extract the template parameters from existing flow traces. We refer to this process as *self-parametrization*.

First, the top k service ports in terms of the unique number of hosts as well as the top k ports with respect to the number of flows are established from the flow traces. The union of the two sets of ports yields a collection of relevant service ports that account for the major portion of the traffic as described in Sect. 3. The parameter extraction is performed on flow trace segments of time intervals of length T. For each relevant service port, the lognormal distribution parameters are estimated over all intervals using maximum likelihood estimation. Furthermore, the average number of flows μ_F (and hosts μ_H) is established, as well as the standard deviation values σ_F (and σ_H) observed over all intervals. The collection of source and destination IP addresses (IP_{src}, IP_{dst}) is derived by counting the number of unique addresses, and grouping them into network prefixes.

For each time interval analyzed, a TDG is created and the associated partitioning of the joint out-degree and in-degree distribution of vertices in the graph is established. In this process, the partitions for each interval consist of the traffic structure accumulated over all previously analyzed intervals. As a consequence, the partitionings capture the dependency of the traffic structure on the (increasing) number of hosts observed. The polynomial coefficients $\vec{a}_{g,k}$ for the partition parameters $g \in G_k$ are finally estimated using nonlinear least squares regression. We use a polynomial of degree 4 that approximates the evolution of partition parameters depending on the number of hosts well.

5. EVALUATION

In this section, we evaluate the flow generation and self-parametrization technique. We assess the accuracy of graph generation algorithm by creating TDGs from partition parameters and comparing the graph metrics of the generated graphs and graphs from real traffic traces. Moreover, we show that the flow records generated by our technique exhibit similar structural properties as the records from original traffic, and we provide performance measurements of the record generation. In a case study, we demonstrate the ability of using background traffic templates in combination with user-defined templates to generate traces containing abnormal traffic events.

The evaluation testbed consists of a desktop-class commodity Core 2 Duo processor with 3 MB shared L2 cache running at 3.06 GHz with 4 GB RAM. We implemented a prototype of our technique in Perl 5. We use two different data sets for the evaluation of our framework. The first data set consists of 10 days of NetFlow records of the internal traffic (LAN) from an average-sized campus network, collected between May 1, and May 9, 2009. The second data set comprises 7 days of NetFlow records collected at a large hosting environment between April 14, and April 20, 2008.

5.1 Graph Generation

We evaluate the graph generation technique in terms of the number of vertices and edges as well as the degree distribution of TDGs created from the partition parameters G_k. First, we generate the graphs directly from static partition parameters. Then, we study the effect of the approximation of the partition parameters using the polynomial function.

We analyzed flow traces from the hosting provider data set for a time period of one hour, divided into 12 intervals of 300-s length, worth of 5.80 million flows and 172 752 unique IP addresses. For each interval, we extract the partitioning parameters G_k from the traffic accumulated over the analyzed intervals. Then, we generate a TDG for each partitioning. We repeat the random graph algorithm for different ports for five times. In Fig. 4 we show the relative difference in the number of vertices $|V|$ and edges $|U|$ between the original and the generated graph for each interval for DNS (53) and HTTP (80) traffic. We observe that the random graph algorithm introduces an average error of 0.62% ($\sigma = 2.03$) for DNS and 0.02% ($\sigma = 0.05$) for HTTP in the number of vertices, and 1.36% ($\sigma = 2.78$) for DNS, 0.47% ($\sigma = 0.15$) for HTTP in the number of edges. In some rare occasions a higher difference in the number of vertices and edges can be observed such as for DNS in Fig. 4. We found that this particular case was caused by a mismatch between the number of in-stubs and out-stubs (when rounding the values sampled from distributions) which lead to isolated vertices and fewer connected edges.

Now, we apply the self-parametrization technique to extract the polynomial coefficients $\vec{a}_{g,k}$ over the same intervals. Based on the coefficients, the partition parameters G_k are computed from the polynomials and the TDGs are generated. We observe that the graphs generated in this manner exhibit slightly higher average errors, 0.28% ($\sigma = 0.60$) for DNS and 0.62% ($\sigma = 1.44$) for HTTP in the number of vertices, and 2.49% ($\sigma = 4.18$) for DNS and 1.47% ($\sigma = 0.71$) for HTTP in the number of edges. We justify the error introduced by the polynomial with its ability to achieve still good approximations of the partition parameters for an arbitrary host population size.

We further compare the degree distribution of the underlying undirected graph for both the original and generated graphs on different ports by computing the empirical Complementary Cumulative Distribution Function (CCDF) for the degree of each vertex $v \in V$. In Fig. 5, we depict the CCDF for the original graphs and those generated from partition parameters, both created from an analysis of one hour of traffic trace from the hosting provider data set on two popular service ports. To show the stability of our measurements, we repeated the graph-generating algorithm 5 times and plot all curves. We find that our approach reproduces the original degree distribution and thus the host connectivity properties accurately. The number of vertices with a specific out-degree and in-degree in the generated graphs is close to the values in the original graphs.

5.2 Flow-Generation Performance

We divide the evaluation of the flow-generation performance into two parts. First, we measure the time needed to create the TDGs for all templates and to construct the connection lists. Then, we quantify the flow generation rate achieved when iterating over the connection list, setting record attributes, and sending NetFlow v5 packets to a collector.

The graph generation and the preparation of the connection list for 5000 unique hosts present in 72 templates took 4.2 s ($\sigma = 0.39$) on average. Increasing the number of hosts for a period to 22 000 hosts yields a mean processing time of 28.7 s ($\sigma = 0.74$). After the pre-processing step, the flow records are generated from the connection list. We measure an average rate of 98 218 flows/s ($\sigma = 3907$). Varying the number of flows generated up to 10M flows leaves this rate unchanged. We conclude that the graph generating algorithm scales well in terms of the number of hosts and that the generator achieves high flow rates.

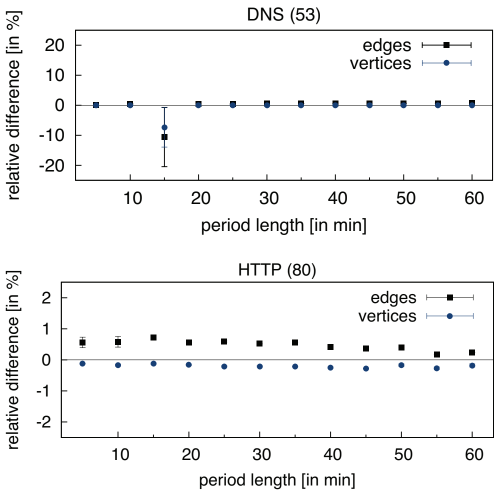


Figure 4: Relative difference between the number of connected vertices and edges of the original and the generated graphs (5 runs for each period length).

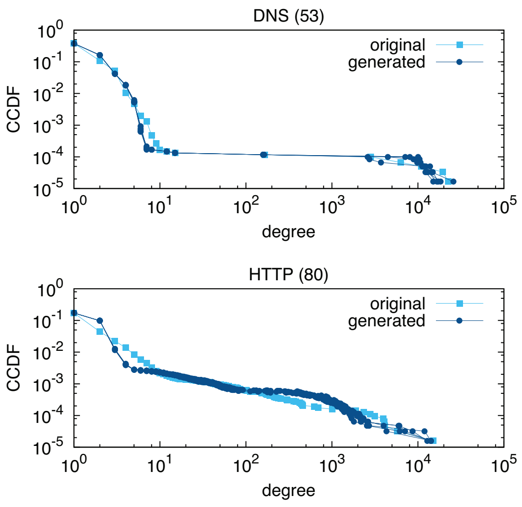


Figure 5: CCDF, $P(X > x)$, of the degrees of each vertex in the original and the generated graphs.

5.3 Traffic Structure

We use the self-parametrization on the LAN traces to extract the traffic template parameters. The flow records analyzed span over 48 consecutive 300-s intervals, comprising a total of 5.15 million flows and 43 385 unique IP addresses. We generate traffic traces for a period of 4 hours from the templates and create the TDGs. Similarly, we establish the TDGs from the original traffic traces (with equal period length) from three different days of the LAN data set. Then, we compare the TDGs by means of 8 graph metrics, that capture the structure of network traffic, which have been introduced by Iliofotou et al. [10] and recently refined [13].

Table 2 shows a comparison of the graph metrics of the original and generated traces for port 53 and 80 (10 runs). The Max Degree Ratio (MDR) is the maximum vertex degree in the graph normalized by the total number of vertices minus one. The directionality of a graph is captured by the percentage of sources (OnlyOut), sinks (OnlyIn), and vertices with both incoming and outgoing edges (InO). The Largest Weakly Connected Component (LWCC) quantifies the connectivity of a graph and is indicated

Table 2: Comparison of graph metrics of TDGs established from original and generated traffic.

Port	Trace type	Vertices	Edges	Average Degree	InO, %	OnlyOut, %	OnlyIn, %	LWCC, %	MDR	RU	r
53	original	7 275	8 991	1.239 (0.028)	0.648 (0.062)	19.196 (1.248)	80.156 (1.191)	99.992 (0.011)	0.803 (0.012)	0.049 (0.005)	-0.567 (0.005)
	generated	7 592	9 484	1.249 (0.001)	0.632 (0.000)	20.138 (0.005)	79.229 (0.005)	99.818 (0.055)	0.783 (0.003)	0.051 (0.000)	-0.532 (0.006)
80	original	8 399	26 259	3.127 (0.090)	0.520 (0.015)	10.349 (0.667)	89.132 (0.657)	99.621 (0.206)	0.067 (0.004)	0.258 (0.006)	-0.373 (0.005)
	generated	8 167	24 865	3.045 (0.010)	0.524 (0.007)	10.608 (0.031)	88.870 (0.035)	98.893 (0.201)	0.036 (0.004)	0.279 (0.004)	-0.336 (0.006)

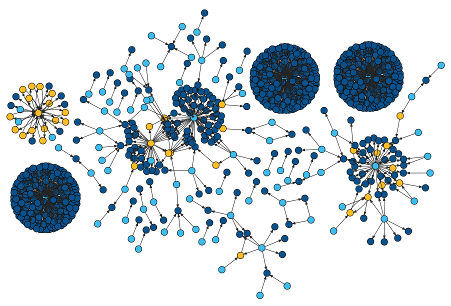


Figure 6: TDG created from generated flow traces containing a user-defined network scan combined with background traffic.

relative to the total number of vertices. The Relative Uncertainty (RU) measures the uniformity of the degree distribution, while the assortativity coefficient r provides a metric of the relationship between the degrees of adjacent vertices. We observe that the differences in the metrics are small. This shows that our approach generates traces with structure highly similar to the original traffic.

5.4 Definition of Traffic Scenarios

The availability of specific borderline scenarios such as network scans or abnormal surges of traffic is of special interest when evaluating flow-based network-monitoring systems. We demonstrate that such events can easily be defined using our technique by defining a customized traffic scenario of a network scan performed by three hosts. We manually created a template and set the polynomial coefficients in two partitions to produce three scanners performing a scan of a population of 600 target hosts on port 22. Normal background traffic was taken from the port 22 template obtained in Sect. 5.3. In Fig. 6, we depict the TDG created from the flow records generated from the two templates. The three scanners are reflected distinctively from the background traffic as three vertices with high out-degrees. We use such scenarios, among others, to assess the quality of analysis results reported by the AURORA traffic monitoring system [14].

6. CONCLUSION

In this paper, we proposed a graph-based method to generate network flow record traces using traffic templates to enable systematic evaluation of flow-based monitoring systems. The traffic templates define the structural properties of the connection patterns as well as the flow attribute distributions observed for various applications. By combining a set of templates, traces from desired traffic scenarios can be generated, such as complex monitoring conditions, traffic anomalies, or regression tests. The trace generation allows the parameterization of templates to modulate the number of hosts and flows while preserving the underlying structural properties of the traffic. A self-parameterization technique has been introduced to extract templates automatically from existing flow-level traces to simplify the generation of realistic traffic. The evaluation of our method has shown its performance in generating complex traffic traces at high export rates.

[1] C. Rolland, J. Ridoux, and B. Baynat, “LiTGen, a Lightweight Traffic Generator: Application to P2P and Mail Wireless Traffic,” in *PAM ’07: Proceedings of the Passive and Active Measurement Conference*, 2007.

[2] K. V. Vishwanath and A. Vahdat, “Realistic and responsive network traffic generation,” in *SIGCOMM ’06: Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2006.

[3] A. Rupp, H. Dreger, A. Feldmann, and R. Sommer, “Packet trace manipulation framework for test labs,” in *IMC ’04: Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, 2004.

[4] J. Sommers, V. Yegneswaran, and P. Barford, “A framework for malicious workload generation,” in *IMC ’04: Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, 2004.

[5] Paessler, “Paessler Netflow Generator,” http://www.paessler.com/tools/netflowgenerator/ (last accessed: Apr. 2010).

[6] J. Juping, “Netflow Simulator in C#,” http://sourceforge.net/projects/netflowsim/ (last accessed: Apr. 2010).

[7] D. Brauckhoff, A. Wagner, and M. May, “Flame: A flow-level anomaly modeling engine,” in *CSET’08: Proceedings of the Conf. on Cyber Security Experimentation and Test*, 2008.

[8] J. Sommers and P. Barford, “Self-configuring network traffic generation,” in *IMC ’04: Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, 2004.

[9] J. Sommers, H. Kim, and P. Barford, “Harpoon: a flow-level traffic generator for router and network tests,” in *SIGMETRICS ’04/Performance ’04: Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, 2004.

[10] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, and G. Varghese, “Network monitoring using traffic dispersion graphs (TDGs),” in *IMC ’07: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, 2007.

[11] M. E. J. Newman, S. H. Strogatz, and D. J. Watts, “Random graphs with arbitrary degree distributions and their applications,” in *Phys. Rev. E 64(2)*, 2001.

[12] R. Milo, N. Kashtan, S. Itzkovitz, M. E. J. Newman, and U. Alon, “On the uniform generation of random graphs with prescribed degree sequences,” Eprint arXiv:cond-mat/0312028, 2003.

[13] M. Iliofotou, M. Faloutsos, and M. Mitzenmacher, “Exploiting dynamicity in graph-based traffic analysis: techniques and applications,” in *CoNEXT ’09: Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, 2009.

[14] IBM Research, “AURORA: Traffic analysis and visualization,” http://www.zurich.ibm.com/aurora/ (last accessed: Apr. 2010).