An Application-level Content Generative Model for Network Applications

Victor C. Valgenti
School of Electrical Engineering and Computer Science
Washington State University
Pullman, Washington 99164-2752, U.S.A.
vvalgent@eecs.wsu.edu

Min Sik Kim
School of Electrical Engineering and Computer Science
Washington State University
Pullman, Washington 99164-2752, U.S.A.
msk@eecs.wsu.edu

ABSTRACT

Generating application-level content within network simulations and/or testbed environments tends toward an ad-hoc process reliant primarily on evaluator expertise. Such ad-hoc approaches are laborious and often fail to capture important aspects of how content is distributed within traffic. Further, while many tools allow for the generation of a wide-range of content types, there exists no coherent model for populating these tools with the necessary data. In this work, we create a content generative model for identifying, harvesting, and assigning application-level content to simulated traffic. This model ties consumers of content to the producers of the content as well as to a particular content category. This approach then allows for said content to be tied to a workload generator or simulator of choice to evaluate a given network application.

Categories and Subject Descriptors

C.2.0 [**COMPUTER COMMUNICATION NETWORKS**]: General—*Security and Protection*; C.4 [**PERFORMANCE OF SYSTEMS**]: Modeling Techniques

Keywords

Simulation, Content Generation, Traffic Generation

1. INTRODUCTION

A primary difficulty in evaluation and educational testbed environments is the adequate generation of application-level content to support simulated network traffic. Such content is used to evaluate applications, such as a caching system or an Intrusion Detection System (IDS) for example, or to create the atmosphere of a live network that can be used to train individuals in handling a variety of network scenarios. The quality of the content utilized in these evaluations can directly impact the value provided by the testbed environment. Unfortunately, the ubiquity of potential content as well as the large variance of such content encountered across different live networks makes populating testbeds with representative content problematic.

Simutools 2012, March 19-23, Desenzano del Garda, Italy

DOI 10.4108/icst.simutools.2012.247735

Common remedies to this problem range from ignoring it altogether by generating static content (i.e. random characters or static files) to laboriously recreating a network through the import of documents harvested from the Internet, or a business intranet, to simple packet capture replay. While static and random files may prove adequate for benchmarking certain applications, as we previously demonstrated [24], they offer little value to complex evaluations requiring traffic more closely aligned with a particular target network. Harvesting documents from other sources and recreating an external environment with a tool such as the Security Assessment Simulation Toolkit (SAST) [10] or the Scalable WORkloaD generator (SWORD) [1] can tune a testbed environment to closely match a target network. However, gathering enough content and configuring the environment can require extensive labor (as much as 160 to 240 hours of labor in the author's experience) for a single test. Even worse, content gathered in this way may fail to properly represent the targeted environment in many respects such as the relative diversity of content and distribution of content amongst users. Finally, packet capture replay can recreate the conditions of a target network and provide some meaningful evaluation but suffers from myopia, limited variance, embedded network phenomenon which may, or may not, be representative of the network, and potential privacy issues. Essentially, there exists no clear model for gathering and generating content in a testbed environment.

In order for content to adequately support a network simulation a content generative model must meet several constraints.

1. Appropriateness: Content should approximate that seen on a targeted network, at least in general distributions and context.
2. Variability: Any model must allow for perturbations to the content to explore potentials beyond the evidence provided from a targeted environment.
3. Privacy: Any model should protect user privacy and, where possible, not propagate user-specific sensitive data.

In this research, we propose a multi-step approach for generating content that meets these constraints. Given a sample from a target network, we derive the major clusters of content dependent on the contextual distance between separate documents. This data is then abstracted to create a series of document content clusters that can be used to automatically harvest similar documents for generation using the *b*-model Surfer (see Section 6). Further, the behaviors of content consumers (users) and content producers (servers) are captured and clustered into like classes of consumers and producers. This information ensures that not only the contextual trends of the target network are captured, but that the user trends associated with those contexts are likewise retained. Armed with these data, it becomes possible to create the generative pyramid where any given

Simutools 2012, March 19-23, Desenzano del Garda, Italy
Copyright © 2012 ICST
DOI 10.4108/icst.simutools.2012.247735

content cluster can be tied to a particular consumer or producer. In this manner, any workload generator can be used to create a workload of traffic where the generative pyramid can assign content, a producer, and a consumer to each flow created thus matching content to the workload.

2. BACKGROUND AND RELATED WORK

The generation of realistic background traffic can have significant impact on the evaluation of network applications [21, 24, 26]. This stems from both the content of the traffic as well as the traffic patterns employed. Current evaluation methodologies for network applications span a wide variety of tools. A common tactic is to employ packet captures from a proprietary network, or from a publicly available repository [6, 8, 16], and replay these captures onto the testbed network with a tool such as tcpreplay [23]. While this tactic provides the simplest means of creating application-level content and effecting a simulation of a targeted network, it fails in many respects. First, the traffic within the capture may contain sensitive information and require *scrubbing* in order to mitigate potential privacy issues. Secondly, traffic regenerated in this manner will exhibit behaviors and phenomenon existing when the traffic was captured but not necessarily realistic when regenerated in the testbed environment. Finally, replay of traffic captures is not well-suited to variety in that the only real control that an evaluator may exert over the traffic capture is the speed of retransmission.

A multitude of traffic generators have been developed that employ varying levels of simulation in order to address these issues. Traffic generators like Harpoon [17], SWING [27], or the Distributed Internet Traffic Generator (D-ITG) [3] provide a means to generate stream-level traffic capable of better simulating network flows. However, these tools do not offer any models or techniques for harvesting content for use within the streams. Some tools specifically designed for Intrusion Detection System (IDS) evaluation, like the black-box IDS Stimulator [13], or MACE [19], provide a means to inject malicious traffic into a stream but offer no more content generation than that. It is possible to use a combination of packet captures and these tools, along with a significant amount of scrubbing of the packet capture, to generate traffic that will mimic a targeted network while retaining the ability to extrapolate on the traffic by inserting malicious traffic [20, 21]. However, these tools still do not provide a solid content model under which to operate and thus the harvesting of content is an ad-hoc process.

Fully fledged application-level simulators like NESSI [4] or ns-3 [14], or even complete distributed testbed simulation environments like SAST [10] or SWORD [1], offer the ability to simulate any kind of network and any kind of behavior. However, the content generation models for these tools are still dependent on the expertise of the evaluators, not any defined methodology. The burden of populating a simulation with content remains the responsibility of the evaluator with no models or tools to create output to match a desired environment.

As stated earlier, content generation should be appropriate to the desired simulation, variable so as to allow exploration of a desired purpose, and should not pose a privacy risk to users of a system. This can be performed in several steps. Appropriateness requires that the simulated traffic must be capable of utilizing statistics pulled from an actual network. This requires the classification of the content within a targeted network as well as the classification of usage trends. Methods to accomplish this goal are detailed in Section 3 and Section 4. The classification of content and usage trends provides the necessary infrastructure for generating new content as needed—content that will approximate the original network and support a simulation. Variability is the power to methodically adjust the model so that the content generated can explore different scenarios. The pyramid generative model illustrated in Section 5 allows adjustments to be made to any of the three primary planes within the model: content, consumers, or producers. Thus, the evaluator may adjust these attributes of the model and realize changes in the generated content. Further, the *b*-model as presented in Section 6 may be used to simulate clustering patterns found in consumers, producers and content. Finally, the privacy of individuals is maintained inasmuch as the model will not attribute the content to individual users as a result of the harvesting methods employed in Section 6. The content generated from this model can be coupled with testbed simulators or other regeneration tools to create the necessary traffic within a simulated environment. The end result is a model that bridges the gap between the infrastructure traffic generation tools and the application-level content desired on the testbed network.

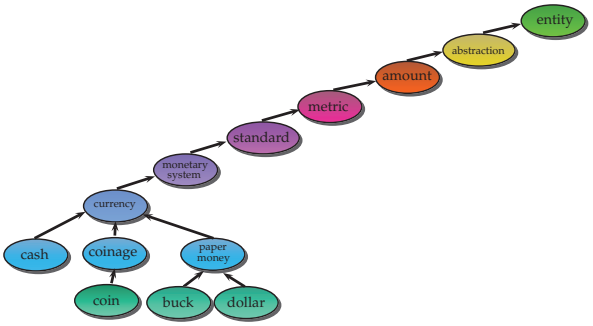


Figure 1: Example WordNet hypernym tree.

3. CLASSIFYING, CATEGORIZING, AND QUANTIFYING CONTENT

We created ContextNet [25], a system for clustering documents by the relative distance in the context of those documents in order to support our generative model. For completeness, we will summarize the function of ContextNet in this section.

ContextNet works by evaluating a set of samples. For the purpose of ContextNet, a sample is a bounded region of text. These samples could be derived by any number of means ranging from a local corpus to pulling the samples directly from network traces, or even indirectly using augmented classification mechanisms such as sound [9,12], or video [2,5]. Regardless the method to collect these samples, ContextNet assumes the existence of this set of samples.

ContextNet functions in the following manner. First, each sample in the set of samples is reduced to the proverbial "bag of words" which is the distinct set of words in the sample; each word associated with a count indicating the number of times that word occurred. Several techniques are used to limit this "bag of words" to only those words most relevant to the context of the document—please refer to [25] for more details. The words are then ordered by frequency within the sample, and the most frequent words are used as a primary feature vector for describing that sample.

A second feature vector is created by using WordNet [11] on the "bag of words". WordNet is a tool, developed by George Miller, that allows the exploration of lexical dependence between words. In particular, WordNet allows the examination of *hypernyms*. A *hypernym* is a word or phrase with a meaning that encompasses the meaning of another word. For example, Figure 1 illustrates a *hypernym* tree for the terms "cash", "coin", "buck", and "dollar" where each term higher in the tree encompasses the meaning of those terms lower in the tree. ContextNet exploits the *hypernym*

relation of words in order to gain broader categories by which the content of a sample may be described. A sampled word is queried for hypernyms in WordNet. The resultant hypernym is then queried for its hypernym, and so on until no more hypernyms are available thus creating a tree. The trees for all words in the "bag of words" are aggregated into a single tree. Since a sample will often contain many similar words that will demonstrate similar *hypernym* trees then there is a natural aggregation to common *hypernyms* for a sample. ContextNet prunes away all the lightly followed branches leaving behind only the most common paths. The intuition is that these paths represent clusterings of meaning and imply shared context. ContextNet takes the most frequent of these *hypernyms* and builds a second feature vector for each sample.

Once these feature vectors exist for every sample, ContextNet compares each sample's feature vectors to the feature vectors of every other sample. This comparison examines the amount of intersection between two feature vectors and determines the distance between the two samples. This creates a graph between all the samples where the nodes represent samples and the edges are weighted with a value indicating the distance between each sample. A distance of zero indicates that a sample is identical to, or completely encompassed within, another sample while an increasing distance quantifies the level of increasing dissimilarity between two samples. In the case where two sample vectors share no commonality, Dijkstra's algorithm is employed to find the shortest path between that sample and others.

After the distance to every document is calculated, k-center clustering is employed to derive k clusters of content for some value of k. The end result is k content clusters, the choice for k dependent on the evaluator's needs. From these content clusters, it is possible to ascertain the most common words and categories for a given content cluster. These can then be directly used as dictionaries for harvesting content specific to a particular content cluster as is illustrated in Section 6.

Each cluster maintains a trio of attributes. First, each cluster maintains a list of the most frequent words and categories within the cluster. This is simply an aggregation of the feature vectors of the samples. Secondly, each cluster maintains the ratio of samples from this cluster to all the samples. This may be used to determine the relative popularity of a cluster. Formally, each cluster D is a triple $D(\alpha,\beta,\delta)$, where α is a finite list of keywords aggregated from the cluster's keyword feature vectors, β is a finite list of categories aggregated from the cluster's categories feature vectors, and δ indicates the density of this cluster in comparison to other clusters. Further, we designate the set of these clusters as D_{all} with $D_{\text{all}} = \{D_1, D_2, \ldots, D_k\}$ where k indicates the total number of clusters. Figure 2 illustrates this formalism. This data is now sufficient for the generative pyramid.

It should be noted that it is not necessary to perform the classification process as outlined here. The content cluster set D_{all} is all that is needed for the generative pyramid (see Section 5), along with the data from Section 4, to provide a functioning generative pyramid. This allows the generative pyramid to vary output simply through modifying the data in D_{all}. In other words, the evaluator could simply create k clusters and populate them with arbitrary keyword and category vectors and an arbitrary density. So long as the set D_{all} is maintained in structure then the generative pyramid will function. The model still offers a defined framework within which to operate (the set D_{all}) despite the fact that arbitrary clusters may not match a target network. Thus, even when generating purely theoretical content, the generation of such content need not be ad-hoc or without structure. We address this more fully in Section 6.

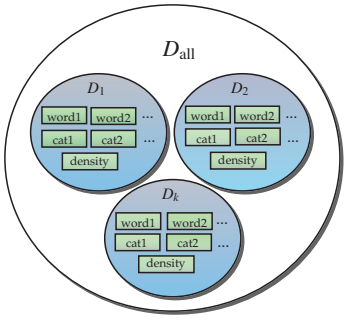


Figure 2: Illustration of the set of derived values from the content clusters.

4. PRODUCERS AND CONSUMERS

Deriving the producers (servers) and consumers (users) for a particular network requires gathering statistics about communication trends. This aspect is also a popular mechanic of network traffic classification. First, we note that within a traffic capture it is possible to capture the set of hosts simply as the set of unique Internet Protocol (IP) addresses within the capture. However, distinctions need to be made between these IP addresses to determine if a particular IP address is a consumer, a producer, or both. Further, it is necessary to preserve clustering behaviors amongst consumers and producers. Finally, the consumers and producers need to link to the content groups. We will address each of these points in turn.

4.1 Distinguishing Consumers and Producers

First, it is important to note that a variety of means may be used to create consumer and producer clusters. One method is to use a list of clients and servers on the system as provided by network infrastructure documentation. Optionally, usage data could be derived from generalized user behaviors captured at a gateway. These data offer a ready-made means to cluster users into logical groupings. However, the actual behaviors of these users as demonstrated in network-level phenomena may exhibit high variance. As an alternative approach, we adopt a method to cluster consumers and producers by the relative "social" behavior of each identifiable host similar to that demonstrated in BLINC [7]. This ensures that clustered consumers and producers have demonstrated similar behaviors. Of course, it is possible to apply behavioral clustering after a logical clustering has already been applied to not only capture business-level (i.e. logical) behaviors but also usage trends (behavior). However, since logical clustering is purely ad-hoc in nature and dependent on an individual target network, we elaborate only on the behavioral clustering.

BLINC [7], a tool for classifying network traffic without payload inspection, examines traffic in a hierarchical fashion by utilizing the "social", "functional", and "application-level" behaviors of hosts within the network. In particular, BLINC makes the observation that most standard services (producers) will use a single source port when delivering content for multiple requests. Conversely, user machines (consumers) will use an operating system generated port for each distinct network flow. Thus, it becomes possible to identify consumers and producers by the ratio of of distinct source ports over the total number of flows for a particular host. The smaller this ratio, the greater the likelihood a particular host is a producer. While the larger the ratio then the greater the chance that the host is a consumer. In this manner it is possible to divide the set of hosts into two sets: consumers and producers.

However, there may also exists a set of hosts that do not neatly

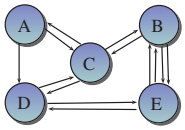


Figure 3: Example multi-graph of a network capture for a network with hosts A, B, C, D, and E.

align to consumers or producers. This is often the case with hosts that are part of a P2P network [7] which act as both consumer and producer. While this may seem like a disadvantage, it is actually an added piece of information that can be well exploited. If a particular host rests in the middle, between the extremes, then it is possible to assume that the host is actually both producer and consumer. That host can then be added to both lists. During generation, some care must be made to prevent a consumer from consuming its own content but otherwise this scheme allows for a clean segregation of consumers and producers. For the purpose of clearly delineating consumers and producers, all hosts with a ratio of source ports to total flows above $1/3$ are added to the consumer lists and all hosts with a ratio less than $2/3$ are added to the producer list. Applying this methodology to one week of network statistics collected at the gateway to the Networking Research Laboratory (NRL) at Washington State University demonstrated only a very small amount of overlap between producers and consumers (about 2.6%). Otherwise, the method worked quite well in segregating the consumers from the producers.

4.2 Clustering Consumers and Producers

Once the consumers and producers have been identified, it is necessary to cluster each group based upon the general behaviors of those particular consumers or producers. BLINC [7] made use of the social aspect of hosts by clustering hosts according to popularity determined by the number of other hosts with which a particular host communicated. We expand on this idea by examining what we term the "super structure" of the network graph. Consider a network capture as a directional multi-graph where each vertex represents a host and each edge a unidirectional communication from one host to another, with potentially multiple edges between the same two hosts. Individual edges are defined by the Protocol and the TCP/UDP source port and destination port with the direction determined by the IP source and destination addresses. In this manner, it is possible to build a multi-graph of the network as illustrated in Figure 3.

Once the multi-graph is created, there are certain host behaviors that will exhibit themselves within the graph. First, more active hosts will maintain a much larger number of communications that will manifest as increased in-degree and out-degree at each node. Thus, it becomes possible to cluster the set simply by the degree of activity attached to a node. This is further refined to include the amount of data into and out of the node to better differentiate heavier data flows. Thus, the in-degree, out-degree, and total bytes into/out of a host stand as three dimensions that can be applied to each consumer or producer. As such, a simple k-means clustering can be applied to identify k clusters within each set of consumers and producers. An example of this is derived from one week of network statistics collected at the gateway to the NRL. The resulting data was segregated into consumer and producer groups using the techniques mentioned earlier, and then k-means clustering was applied to each set. The results are represented in the Figures 4 and 5. Note that the values for in-degree, out-degree, and bytes are normalized.

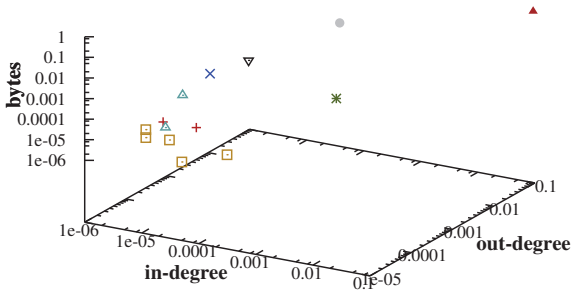


Figure 4: Producers from NRL. Symbols mark clusters.

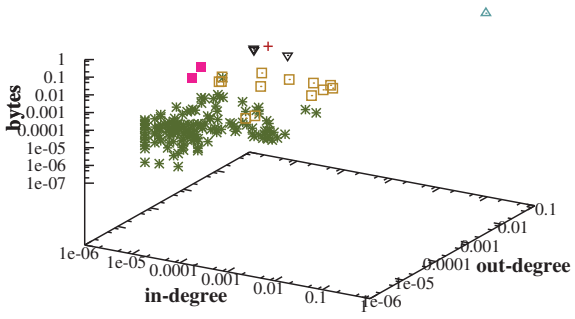


Figure 5: Consumers from NRL. Symbols marks clusters.

Clustering the data in this way preserves several phenomena embedded within the data. First, there is an unequal distribution of hosts to each cluster. In other words, most hosts are gathered into a few clusters, while most of the clusters have only a few values. Such unequal distributions are common to network behaviors and not necessarily captured when arbitrary grouping is employed. Second, choosing the correct value for k can often prove quite difficult for the most accurate classification. However, the goal in this clustering of data is simply to lump like with like. Thus, there is no need for a fine-grained classification of hosts. In our experiments with the NRL data we found little real difference in clusters produced with a value of k between 3 and 9—only more small clusters, no real change to the overall shape of the clusters. While the exact clustering will depend largely on the data set, the real objective of this clustering is to ensure that all hosts are part of a cluster more than that these clusters meet rigorous classification criteria. Ultimately, the evaluator must decide the level of precision necessary for an actual test. Finally, we note that this clustering method could serve to augment approaches that utilize available business data such as user groups. Thus, it becomes possible to cluster within a large logical category and retain both the logical underpinnings of a group as well as the network behaviors.

4.3 Assigning Content

Once the consumer and producer clusters are created, it is necessary to connect those clusters to individual content clusters as

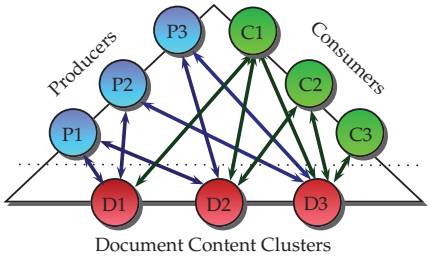


Figure 6: Illustration of the Generative Pyramid.

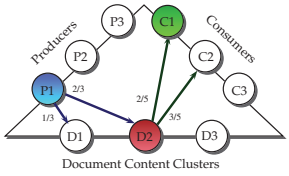
derived in Section 3. Arbitrarily assigning content clusters to consumer or producer clusters would not preserve the distribution of content amongst consumers and producers. However, those distributions may contain important trends or phenomenon. In order to preserve this data, it is necessary to quantify the sample-set such that each sample clustered as per Section 3 must also tie to at least one consumer and one producer. Such a requirement is viable when examining a sample directly from a target network, though may require secondary means of classification when dealing with a corpus of samples not derived directly from network traffic. Regardless, once the content, consumer, and producer clusters are known it is then a simple procedure to connect each consumer cluster to a content cluster and producer cluster to a content cluster. In fact, the relative frequency with which one cluster interacts with another can be used to simulate the likelihood of a particular content cluster applying to a given consumer or producer. This, in fact, is the basis of the generative pyramid discussed in Section 5.

If it occurs that the sample corpus has no direct connection with any consumer or producer, then it becomes necessary to arbitrarily assign consumer and producer clusters to content clusters. A simple model for generating unequal distributions is the *b*-model. The *b*-model is elaborated in Section 6. Essentially, this model allows for the emulation of the Hurst parameter to create self-similar workloads. The benefit of using the *b*-model in this instance is that it would allow an evaluator to systematically adjust the distribution trends between consumers and producers and the content clusters.

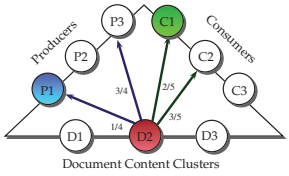
5. GENERATIVE PYRAMID

The generative pyramid is the centerpiece of the content generative model. Essentially, given a set of content clusters as derived in Section 3, a set of consumer and producer clusters as derived in Section 4 and the ability to connect consumers and producers to content, then a pyramid can be formed. Figure 6 illustrates this pyramid with content clusters at the base, consumers on another angle, and producers the last. In fact, connecting the content between consumers and producers creates a bipartite graph. The content clusters serve as the middle ground for consumers and producers. The key to making this model work relies in the connections between consumers and content and producers and content.

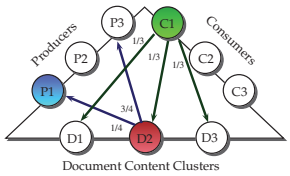
The pyramid works in the following manner. In Figure 6 there exists a connection between $P1$ and $D2$. Notice also that $P1$ is also connected to $D1$. From the assignment of content to producers as per Section 4 it is possible to derive the relative distribution of documents from that producer that fall into content clusters $D1$ or $D2$. Imagine that $1/3$ of all documents produced by $P1$ fell into the content cluster $D1$, and the remaining $2/3$ fall into the content cluster $D2$. As such the directional edge $P1D1$ could be weighted with the value $1/3$ and edge $P1D2$ with the value of $2/3$. Conversely, notice that $D2$ is connected to $P1$ and $P3$ as well as $C3$ and $C2$. Since the producer and consumer sets are known, it is possible to distinguish



(a) Finding content and consumer from producer.



(b) Finding producer and consumer from content.



(c) Finding producer and content from consumer.

between a consumer and a producer. This allows $D2$ to consider its interaction with producers in isolation to consumers. As such, if $1/4$ of $D2$ content comes from $P1$ and $3/4$ comes from $P3$, the the directional edges $D2P1$ and $D2P3$ would be weighted $1/4$ and $3/4$ respectively. In addition, $D2$ might also demonstrate $2/5$ and $3/5$ of its content consumed by $C1$ and $C2$ respectively. Finally, the distribution of content consumed by $C1$ might be evenly distributed amongst all three content clusters $D1$, $D2$ and $D3$.

Once the pyramid is weighted in this manner, it becomes possible to probabilistically determine a combination of consumer, producer, and content by simply starting with one (consumer, producer, or content) and probabilistically choosing edges to other clusters. Figure 5 illustrates three different methods in which the combination of $P1$ $D2$ $C1$ could have been chosen given a different starting point with Figure 7(a) starting from $P1$, Figure 7(b) starting from $D2$ and Figure 7(c) starting from $C1$. Even better, the initial cluster can be chosen based on distributions (i.e. more frequent consumers, more common content, etc). A further benefit of this approach is that a simulator or traffic generator need only determine that a communication is to take place based on low-level workload models. Once the presence of a communication is determined, the pyramid can be used to derive the consumers and producers and content. In the event that the low-level model has already determined a consumer or producer, or even both, it is still possible to use that information to determine the content, or vice versa.

More practically, the generative pyramid can be reduced to a directed incidence matrix with n rows (one row for each producer, consumer, and content cluster) and m columns (one for each edge in the pyramid). The intersection between any edge and one of the vertices of that edge is marked by the probability that given this vertex this edge will be chosen. In other words, it marks the outgoing edges. An example of this incidence matrix is illustrated by Table 1. Since the consumer and producer clusters are known, they can be listed within the columns of the matrix such that all edges for producers come before all edges with consumers. This is a simple method for organizing the matrix and maintaining what amounts to two separate matrices. Note that the probabilities for each consumer and producer cluster sum to 1 and that for each content cluster the sum is 2 when examined across the entire row, but 1, when examined only between consumers or producers.

The reduction of the relationships between consumers, producers, and content clusters to the directed incidence matrix is a powerful abstraction. The methods explored in Section 3 and Section 4 provide a meaningful method to produce the consumers, produc-

ers, and content clusters. Further, the connections between these clusters can be ferreted from network traces. Thus, the directed incidence matrix of the generative pyramid provides all the necessary information to tie content to any simulated flow. Further, this model preserves clustering trends inherent in the consumers, producers, and content of a targeted network. As such, the generative pyramid maintains by default, if the methods from Sections 3 and 4 are employed, appropriateness to a given target network. In the following sections we will explore how this model can be tuned to provide variability and to maintain privacy.

6. VARIABILITY AND PRIVACY

The generative pyramid is a model that is ripe for variability. The simplest variance that may be exerted over the model is changing the probabilities within the directed incidence matrix. This will have an immediate effect of changing the overall content resulting from the tests and may be used to study extremes within content generation. Examples of such tests would be to adjust the probabilities such that all content originated from only one or a few producers and then perform consecutive tests when the distribution of producers moved closer to an equal distribution.

6.1 The *b*-model

However, more useful to our purposes is a simple model that might be used to simulate the relative consumer, producer, and content clusters without the need to actually scour traffic traces or a document corpora in order to derive the necessary statistics. For this task, we adopt a modified version of the *b*-model originally demonstrated by Wang et al. [28]. In short, the *b*-model is a simple recursive algorithm for generating an unequal distribution—often used for generating self-similar workloads. The value of b in this model is the loose equivalent of the Hurst parameter with a value of .5 indicating relatively equal distribution and a value of 1 a single long burst. The algorithm works in a simple manner. Define a space within which the distribution will occur, typically a time period, and a set number of events, y, to occur. The object is then to determine where, within the entire space, those events will occur. This is done by cutting the space in half and sending $b \times y$ events into one half and the remainder of events into the other half; the choice of which half (right or left) chosen randomly with equal probability. This process continues recursively until a minimum period is met (i.e. no more dividing the space). The events are then aligned within the space which may adjust the placement of other events within the space due to spillage off either side of the space. When spillage occurs all events are simply shifted onto the space until all are accommodated which may result in events getting moved from their initial placement.

The key to employing the *b*-model for our purposes is to make the observation that a sample space might be considered a binary string x bits long. This binary string can be used to determine the number and size of groups in a very simple way. If a bit is 1, then that indicates that a value exists at that index within the entire sample space. A consecutive string of ones within the bit string indicates a cluster while zeros within the bit string indicate a break in clusters. Thus, it is possible to count the total number of clusters simply by looking for all the distinct clusters of ones within the binary string. Further, this also allows the determination of the size of each cluster through merely counting the number of 1's within a particular cluster. Initially, the bit string is all zeros and is x bits long. The *b*-model is run on this space for y total values (document samples, consumers, producers, etc) with $x > y$. As per the *b*-model above, when a value is added to the bit string at a particular index, that bit is simply flipped from 0 to 1. This results in series of 1's and 0's in the bit string. A substring of 1's, one or more in length, indicates a cluster while each substring of one or more 0's indicates breaks between clusters. Depending on the value for b used, the result is something like Figure 7 which illustrates a bit string x bits long ($x = 15$) and with y=10. Four clusters are evident in this figure though this need not be the case.

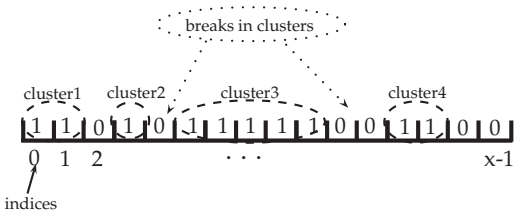


Figure 7: Illustration of the *b*-model filling a bit string x bits long to create clusters.

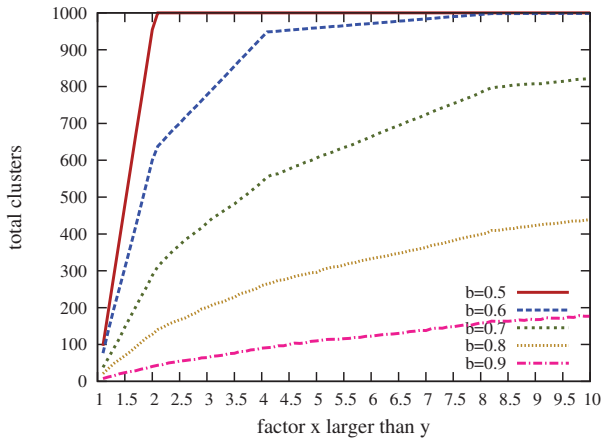


Figure 8: Impact of overall space on number of clusters created for the *b*-model. These results represent 1,000 values placed into a bit string $x * 1000$ large where x is the x-axis.

The *b*-model is impacted by the relative sizes of x and y in addition to the value for b. Obviously, the closer y is to x then the size of clusters will increase and the number of clusters will decrease. This extends from the fact that the values are crowded into a smaller space and are more likely to create contiguous blocks. Conversely, creating a very large space will allow the algorithm enough room to split the space more times and ultimately create more gaps between values which will result in smaller, but more numerous, clusters. Figure 8 illustrates the impact that the size of x and y have on the average number of clusters created by the *b*-model as used here. For this test 1,000 values ($y = 1000$) are inserted into a bit string that is 1,000 $\times$ the x-axis bits wide. The initial value for the x-axis is 1.1 extending out to 10 in increments of 0.1. The y-axis represents the average number of clusters resulting from applying the *b*-model given the above constraints and for the specified b value and aggregated across 100 runs of the algorithm. As is evident, the average number of clusters created increases as the size of the space increases. Further, it also increases dependent on the value for b. Figure 9 illustrates the average cluster sizes for this test. The size of the space has a significant impact on the average size of each cluster. However, as expected, it is the value of b that is the dominant factor in causing variance in cluster sizes. Essentially, the larger the b values, the much more significant the variance in cluster size.

The *b*-model can be used to model the general clustering be-

Table 1: Example of incidence matrix of generative pyramid.

Cluster	P1D1	P1D2	P2D1	P2D3	P3D2	P3D3	C1D1	C1D2	C1D3	C2D2	C2D3	C3D3
P1	1/3	2/3	0	0	0	0	0	0	0	0	0	0
P2	0	0	2/5	3/5	0	0	0	0	0	0	0	0
P3	0	0	0	0	0	0	0	0	0	0	0	0
D1	1/4	0	3/4	0	0	0	1	0	0	0	0	0
D2	0	1/4	0	0	3/4	0	0	2/5	0	3/5	0	0
D3	0	0	0	2/3	0	1/3	0	0	1/6	0	1/2	1/3
C1	0	0	0	0	0	0	1/3	1/3	1/3	0	0	0
C2	0	0	0	0	0	0	0	0	0	3/5	2/5	0
C3	0	0	0	0	0	0	0	0	0	0	0	1

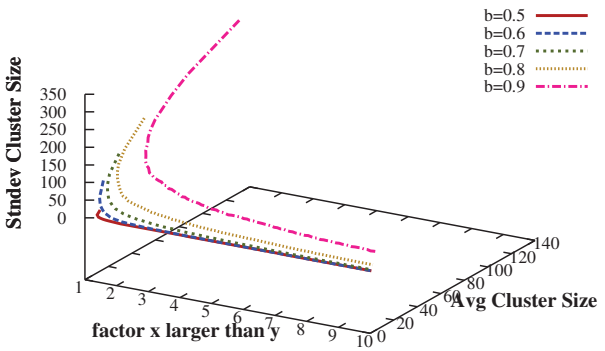


Figure 9: Average cluster size given variation in space size and *b*. Note: the z-axis represents the standard deviation.

havior of consumers, producers, or content. The key to successful modeling is to determine the total number of samples (consumers, producers, or content samples), and then the relative clustering desired. Choosing the values for x and b will depend on the targeted clustering behavior. The most straightforward approach is simply a brute force evaluation using least-squares to identify the best parameters given a desired clustering behavior. This is illustrated in Section 7. More importantly, the *b*-model allows for a range of clustering behaviors to be examined. Systematically manipulating the value for b can be used to investigate the impact of greater or lesser clustering within consumers, producers, or content. In fact, Figure 9 very nicely illustrates the range of clustering behaviors that might be explored with the *b*-model.

6.2 Allowing for Privacy

The *b*-model is useful for creating clusters, but those clusters still need to be populated. For consumers and producers this only requires that a set of IP addresses be designated and then these addresses be added to the clusters randomly, or under certain distribution guidelines. The set of IP addresses could be generated as illustrated by Sommers et al. [18], or could come from network captures as illustrated in Section 4. After which, we assume that there exists a method to map the addresses to clusters. This assumption is not particularly demeaning as the simplest approach is to match the list of IP addresses, in order, to the on-bits in the bit string produced by the *b*-model. In doing so, the IP addresses will naturally be added to clusters. However, we note that there may be subtleties in how the IP addresses need to be distributed and leave the investigation of such to future work.

Populating the content clusters, however, is somewhat more complicated. The most often adopted approach is to simply use the corpora captured from a live network. If the tests are entirely in-house and properly secured then this may be a viable approach. However, privacy concerns may abound. Also, there exists the potential that the corpora is not sufficiently robust for effective evaluation. To address these issues we developed a content harvesting method we term the *b*-model surfer [25]. The *b*-model surfer derives in part from the PageRank random surfer model [15] in part from the idea of the *b*-model [28], as already discussed, and in part from the ideas of Trestian et al. [22] which consider using other, publicly available, resources to gather information.

In essence, the *b*-model surfer mimics a web user making web searches. First, the *b*-model surfer creates a series of content clusters as per the *b*-model described earlier. Optionally, content clusters as derived in Section 3 may be used. The *b*-model surfer utilizes a search engine to gather content using a randomly chosen key-word, or category, from a selected content cluster. The search engine used may be a public tool or one local to an organization. The results of the search are parsed, and the *b*-model surfer will randomly select one of the URLs and download the document. A number of documents are downloaded in this manner until a particular content cluster is filled. Then the *b*-model surfer will continue with the next cluster until the entire population of samples has been harvested. While our prototype only downloads HTML, the tool could be expanded to include other content types such as images, sound, and video without much difficulty.

The *b*-model surfer offers a simple method for populating the content of an evaluation with publicly available (or locally available) content. The *b*-model ensures that the content will follow a particular clustering policy and the fact that the content is publicly available mitigates privacy issues inasmuch as the content is not tied to users of any particular network. As such, it meets all three criteria for content in that the content will be appropriate to the test (as a result of content clusters derived from actual content), variable (as the *b*-model can be changed), and maintain privacy (as all harvested documents come from a publicly available source). Of course, there are still some issues that need to be overcome with the *b*-model surfer. First, the search engine and search terms used can greatly impact the results. A poor search term or inadequate search engine may return either overly broad results, or no results at all. Secondly, it is possible that the search engine will skew the results of any search. This is partially mitigated by the fact that the *b*-model surfer is not tied to any specific search engine thus evaluators may choose the search engine that best suits their needs. Third, the *b*-model surfer does not mimic link following as per PageRank [15], though this is slated for future work. Finally, while the *b*-model surfer will protect individual user privacy in that no content is linked to real users of a system, it should be noted that the meth-

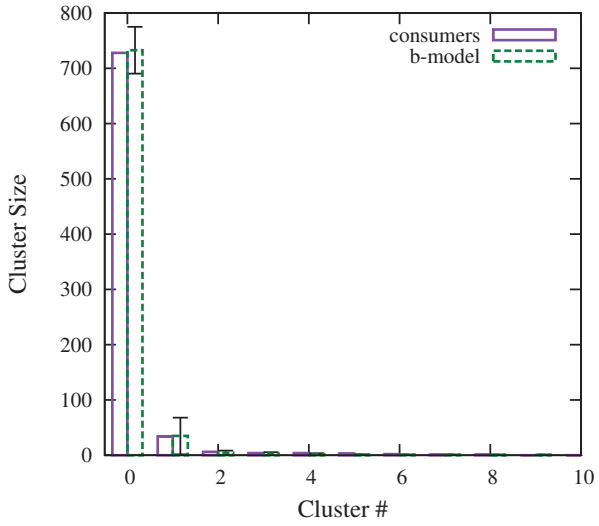


Figure 10: Best fit b-model to consumers from NRL with b=0.85, x = 1.1, and y = 783. Error bars mark 1 standard deviation.

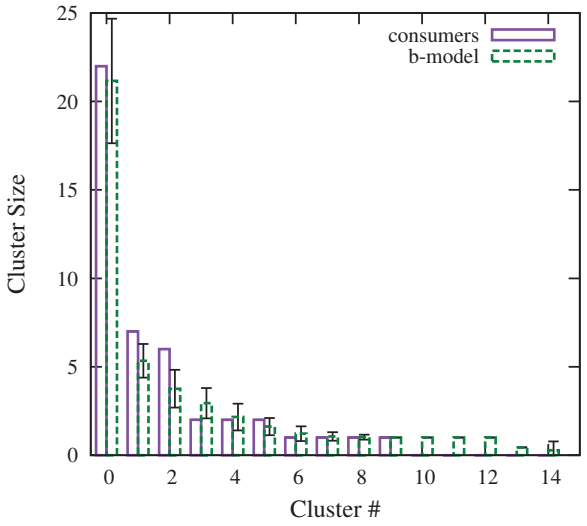


Figure 11: Best fit b-model to producers from NRL with b=0.6, x = 1.3, and y = 45. Error bars mark 1 standard deviation.

ods outlined in Section 3 could well identify trends of a sensitive nature. In other words, sensitive keywords that imply misuse (i.e. pornography) will be retained in any clustering assuming enough such content is encountered. Thus, if there is substantive misuse in the sample network, then keywords representing this should occur in the generative model and impact the output. This is by design, however, though preventing the linking of this information back to actual users would would require procedural safeguards in the handling of the original data. Further, content that ties directly to users on a system is unlikely to be promoted to a keyword by ContextNet. This stems from the fact that only the most common keywords for any cluster are preserved. Thus personal data is unlikely to make any keyword list unless this data appears in an anomalously large number of samples within a cluster. Even in the face of these deficiencies the b-model surfer offers a good tool for harvesting content for clusters.

7. EVALUATION

We evaluate the approach against the 3 primary constraints for generating content as stated in Section 1.

7.1 Appropriateness

The Appropriateness of this approach follows that if the clusters are derived directly from network captures as illustrated in Sections 3 and 4 then they reflect, at least in the general clustering behaviors of content, consumers, and producers, the target environment. As such we offer no further evidence then that already demonstrated. Second, we note that the b-model can be used to simulate the clustering for a target network. To do so, it is only necessary to derive the best fit parameters to fit the model to the desired target environment. Thus, we attempted to fit a b value to the clustering of NRL data as illustrated in Figures 4 and 5. To arrive at the best fit, we simply aggregated the results of running the b-model 50 times for a particular set of b, x, and y values and used least squares to determine the closest fit to the clustering exhibited in the NRL data. Figure 10 illustrates how closely the b-model is able to match to the target for consumers and Figure 11 the same for producers.

As can be seen in the figures, the b-model can be tailored to approximate, with reasonable accuracy, the general cluster size and number of clusters for a particular data set. The calculation of the best parameters required roughly 15 minutes for the consumers and about 11 seconds for the producers. For very large data sets this process could require considerable time. However, this is a task that need only be performed once and is not a direct demerit to the technique. Unfortunately, the randomness involved in the b-model means that there can be considerable fluctuation. First, the b-model has a tendency for creating more clusters than the actual set. The actual number is somewhat random due to the process of the algorithm, but will often exceed the desired number of clusters. Figure 11 illustrates this very clearly. This represents a small failing of the b-model. Luckily, these extra clusters are always small in magnitude (size of 1). These extras can either be ignored, or simply subsumed by the larger clusters. Secondly, the b-model can fluctuate in the cluster sizes of the larger clusters. This is also evident in the figures as the error-bar marks the standard deviation. While this fluctuation will often cause the clusters to not match perfectly to the target, the order of magnitude difference is typically not beyond reason. Further, the general shape of the clusters in terms of largest to smallest clusters is typically preserved. On average, the b-model is capable of approximating a target clustering behavior.

7.2 Variability

The content pyramid is a extremely powerful model where variability is concerned. To demonstrate this, we employ a simple example. We start by implementing a simple caching algorithm and then use the b-model surfer to harvest varying sets of documents with which to test this algorithm. By varying the clustering of the documents it is possible to increase the likelihood that the same page is harvested. The caching algorithm works like this. When a document is requested, the cache is checked to see if the document already exists in the cache. If it does, the caching algorithm counts this as a hit. If not, then the document is added to the end of the cache, potentially removing the first document in the cache if the cache size of 10 has already been reached. The average hit rate is then the number of cache hits divided by the total number of requests.

To evaluate this simple caching system, we utilize the b-model surfer at varying values for b to harvest 100 documents and create

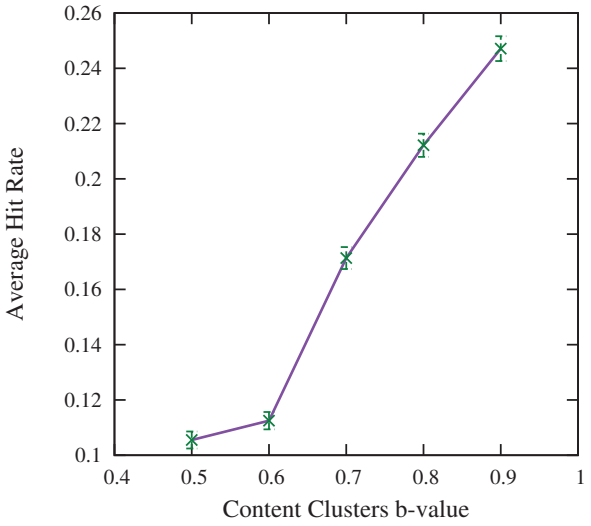


Figure 12: Average cache hit rate for documents gathered under the set value for *b*.

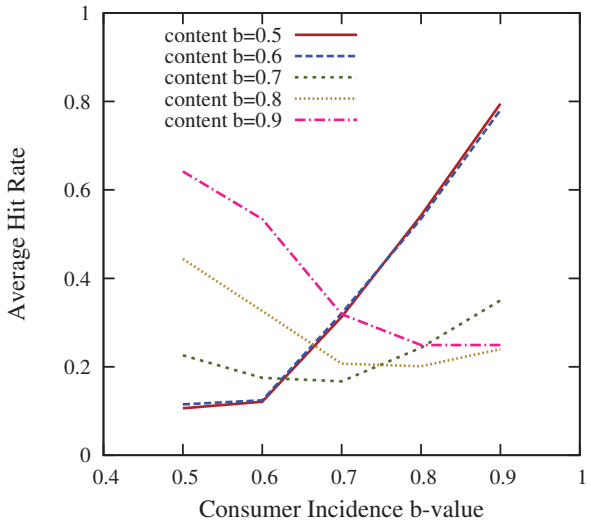


Figure 13: Impact of adjusting consumer incidence probabilities.

a space of documents that a user might request. There are no predefined clusters, only those that result due to the *b*-model. Further, the *b*-model surfer is restricted to a smaller set of search results encouraging the chance that, when a large cluster occurs, a single document may be selected multiple times. Such redundancy is retained within the content clusters—this embeds the likelihood of a particular document occurring more often than others directly into the cluster. We assume a single producer to which all requests are made. Further, we assume a single cluster of consumers making requests, each request utilizing the distribution of documents as the probability for choosing a particular document (i.e. a cluster with more documents will be selected more often). A series of 10,000 web requests are simulated in this manner. The intuition to this test is that as documents become more clustered, the hit rate should climb. In fact, this is the case of the 100 test runs aggregated and illustrated in Figure 12.

Figure 12 represents a good example of how the *b*-model can be used to systematically examine the impact of clustering against a network application. However, the potential for advancing the scenario is even greater. Adjusting the incidence matrix can further impact the results in non-intuitive ways. To illustrate this, we design a more advanced evaluation using the content clusters generated from Figure 12 and then adding multiple consumer clusters and adjusting the incidence matrices for each consumer cluster. To simulate the consumer clusters, we once again adopt the *b*-model to distribute 1,000 IP addresses amongst a varying number of clusters. While the number of consumer clusters will have limited impact on a caching algorithm in most circumstances, the probabilities linking consumer clusters with content clusters will have a significant impact on the results. To simulate this, we also apply the *b*-model to the incidence matrix between each consumer cluster and content cluster. To accomplish this, we use the *b*-model to create a clustering across 100 possible values. We use the size of each cluster generated in this manner to indicate a probability of this consumer choosing content from a particular content cluster. In order to adopt a little more order into the evaluation, we sort these clusters in descending order by size. The content clusters gathered under a particular *b* value are also sorted in descending order by size. These two sorted lists are aligned and the probability at a given index becomes the probability of that consumer choosing that content cluster. The incidence matrix is then filled out with the values derived in this manner. Essentially, this weights the probabilities such that the largest probabilities align with the largest content clusters.

For the test we simulated 1,000 web page accesses by first randomly selecting a consumer from one of the consumer clusters. A random number between 1 and 100 is chosen and given this number, and the incidence matrix, it is possible to determine a target content cluster. A document is then randomly chosen from within the targeted content cluster and lastly the producer is selected (only one producer for this test). Figure 13 illustrates the aggregated results of 10 test runs. The adjustment of the incidence matrix has had a significant impact on hit ratios. First, notice that for incidence probabilities of 0.5 and 0.6, the hit rate for the documents gathered under *b* values of 0.8 and 0.9 are extremely high. The root cause of this phenomena stems from the fact that the clustering of documents in these instances consists of one or two large clusters and then several clusters of a single document. Since each cluster is chosen nearly equally when the incidence matrix probabilities are created using 0.5 or 0.6, then it becomes more likely that the clusters with only a single document are chosen. This directly results in the higher hit rates. However, for documents gathered under a *b* value of 0.8 and 0.9, and when the incidence matrix probabilities are created using 0.8 or 0.9 the exact opposite happens. Essentially, the largest clusters become the target and the results then quickly approach the clustering inherent within the cluster as witnessed in Figure 12. Conversely, as the *b* values used in creating the incidence matrices grows, the documents clustered around lower *b* values (0.5 and 0.6) show increasing hit rates. This is a direct result of the fact that the largest content clusters in these instances contain 2 or 3 documents. As more and more traffic is directed to those clusters, the chance for a cache hit rises appreciably.

This example could be further expanded to include multiple producer clusters as well as different incidence matrices. Of course, the model extends well beyond the testing of caches as illustrated here. The harvesting of content, the clustering of consumers and producers, as well as the incidence matrix serve to allow for a huge amount of variety in potential evaluations beyond that explored here. Regardless, the caching example illustrates the potential of this model.

7.3 Privacy

Finally, we remark on the privacy concerns mitigated by this approach. First, content harvested from public resources can be considered not only realistic (it is actual content), but public and thus not sensitive in nature. However, there are some potential privacy issues that may arise as discussed at the close of Section 6. Still, the methods provided here mitigate the larger concerns of user privacy. However, more advanced testbeds, training simulations for actual network security for example, require extremely realistic traffic. This will typically require email, voice over IP, and potentially other content types. Harvesting such documents can still be handled under the framework provided here—though the documents must undergo an extra stage of cleaning. Essentially, a set of example documents must be harvested however possible. These documents must then be vetted dependent their potential for sensitivity. We posit that ContextNet [25] working with a set of training data could serve to capture most of the sensitive content. The remainder would then need to be vetted by an evaluator before employing the framework provided here. This is not a perfect solution and we seek to further improve this process in future work. Regardless, the research here provides for a systematic framework by which content may be harvested, the relative clustering statistics maintained, and, in most cases, privacy concerns mitigated.

8. CONCLUSION AND FUTURE WORK

The content generative model provided here provides three significant contributions to testbed evaluations. First, it offers a concrete method for quantifying the distribution of content within a target network and capturing those distributions in the content generated. Second, it provides for the connections between producers, consumers, and content. Finally, this model provides a systematic method for not only approximating the distribution of content, consumers, and producers within a target system, but also for exploring wide ranges of variability. There are still several areas that can be improved. Content harvesting can be done easily and relatively privately for media popular on the web. However, specialized content specific to an organization is not as easily harvested. Secondly, the harvesting techniques rely on search engines and can thus be biased by those engines. We hope to address these issues, and the others mentioned in the paper, in the future. Despite these factors, the content generative model here aims to fill the void for attaching content to other simulation models and tools. Since many other simulators and traffic generators provide for content (just not the harvesting of content), then the tactics provided here should enhance those other techniques. Ultimately, the content generative pyramid and the *b*-model for cluster generation are two powerful tools for enhancing simulations and testbed environments.

9. REFERENCES

[1] K. Anderson, J. Bigus, E. Bouillet, P. Dube, N. Halim, Z. Liu, and D. Pendarakis. SWORD: Scalable and flexible workload generator for distributed data processing systems. In *Proceedings of the 2006 Winter Simulation Conference*, 2006.

[2] S. Arifin and P. Y. K. Cheung. A novel probabilistic approach to modeling the pleasure-arousal-dominance content of the video based on "working memory". In *Proceedings of the International Conference on Semantic Computing*, 2007.

[3] A. Botta, A. Dainotti, and A. Pescapè. Multi-protocol and multi-platform traffic generation and measurement. In *INFOCOM 2007 DEMO Session*, 2007.

[4] R. Bye, S. Schmidt, K. Luther, and S. Albayrak. Application-level simulation for network security. In *Proceedings of the First International Conference on Simulation Tools and Techniques for Communications, Networks and Systems, SIMUTools 2008*, Mar. 2008.

[5] T. De Pessemier, T. Deryckere, and L. Martens. Context aware recommendations for user-generated content on a social network site. In *Proceedings of the European Interactive Television Conference*, 2009.

[6] Defcon. `http://www.defcon.org`. Capture the Flag Data Sets.

[7] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: Multilevel traffic classification in the dark. In *Proceedings of ACM SIGCOMM*, 2005.

[8] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, and M. A. Zissman. Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition*, volume 2, pages 12–26, Jan. 2000.

[9] T. Makinen, S. Kiranyaz, and M. Gabbouj. Content-based audio classification using collective network of binary classifiers. In *Proceedings of the IEEE Workshop on Evolving and Adaptive Intelligent Systems*, 2011.

[10] W. Meitzler, S. Oudekirk, and C. Hughes. Security Assessment Simulation Toolkit: SAST. Technical report, Pacific Northwest National Laboratory, 2009.

[11] G. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38:39–41, 1995.

[12] V. Mitra and C. J. Wang. A neural network based audio content classification. In *Proceedings of the Interantional Joint Conference on Neural Networks*, 2007.

[13] D. Mutz, G. Vigna, and R. Kemmerer. An experience developing an IDS stimulator for the black-box testing of network intrusion detection systems. In *Proceedings of the 19th Annual Computer Security Applications Conference*, Dec. 2003.

[14] ns 3 project. The ns-3 network simulator. `http://www.nsnam.org/`.

[15] L. Page. PageRank: bringing order to the web. Technical report, Stanford Digital Library Project, 1998.

[16] B. Sangster, T. J. O'Connor, T. Cook, R. Fanelli, E. Dean, W. J. Adams, C. Morrell, and G. Conti. Towards instrumenting network warfare competitions to generate labeled datasets. In *Proceedings of USENIX Security Workshop on Cyber Security Experimentation and Test*, 2009.

[17] J. Sommers and P. Barford. Self-configuring network traffic generation. In *Proceedings of 2004 Internet Measurement Conference*, Oct. 2004.

[18] J. Sommers and J. Raffensperger. Efficient and realistic generation of ip addresses. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, march 2011.

[19] J. Sommers, V. Yegneswaran, and P. Barford. A framework for malicious workload generation. In *Proceedings of ACM SIGCOMM Internet Measurement Conference*, 2004.

[20] J. Sommers, V. Yegneswaran, and P. Barford. Toward comprehensive traffic generation for online IDS evaluation. Technical report, University of Wisconsin-Madison, 2005.

[21] J. Sommers, V. Yegneswaran, and P. Barford. Recent advances in network intrusion detection system tuning. In *Proceedings of the 40th Annual Conference on Information Sciences and Systems*, Mar. 2006.

[22] I. Trestian, S. Ranjan, A. Kuzmanovi, and A. Nucci. Unconstrained endpoint profiling (Googling the Internet). *SIGCOMM Computer Communication Review*, 38(4):279–290, 2008.

[23] A. Turner. Tcpreplay. Available at: `http://tcpreplay.synfin.net/`.

[24] V. Valgenti and M. S. Kim. Simulating content in traffic for benchmarking intrusion detection systems. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, 2011.

[25] V. C. Valgenti, R. R. Paul, and M. S. Kim. ContextNet: Extending wordnet for ad-hoc contextual classification of network traffic. Technical report, Washington State University Networking Research Laboratory, 2011. `http://nrl.eecs.wsu.edu/pubs/contextnet.pdf`.

[26] K. V. Vishwanath and A. Vahdat. Evaluating distributed systems: Does background traffic matter? In *Proceedings of USENIX Annual Technical Conference*, June 2008.

[27] K. V. Vishwanath and A. Vahdat. Swing: Realistic and responsive network traffic generation. *IEEE/ACM Transactions on Networking*, 17(3):712–725, June 2009.

[28] M. Wang, T. Madhyastha, N. Hang Chan, S. Papadimitriou, and C. Faloutsos. Data mining meets performance evaluation: Fast algorithms for modeling bursty traffic. In *Proceedings of the 18th International Conference on Data Engineering*, 2002.