Profiling the End Host

Thomas Karagiannis[1], Konstantina Papagiannaki[2],
Nina Taft[2], and Michalis Faloutsos[3]

[1] Microsoft Research
[2] Intel Research
[3] UC Riverside

Abstract. Profiling is emerging as a useful tool for a variety of diagnosis and security applications. Existing profiles are often narrowly focused in terms of the data they capture or the application they target. In this paper, we seek to design general end-host profiles capable of capturing and representing a broad range of user activity and behavior. We first present a novel methodology to profiling that uses a graph-based structure to represent and distill flow level information at the transport layer. Second, we develop mechanisms to: (a) summarize the information, and (b) adaptively evolve it over time. We conduct an initial study of our profiles on real user data, and observe that our method generates a compact, robust and intuitive description of user behavior.

1 Introduction

Profiling a behavior refers to the act of observing measured data and extracting information which is representative of the behavior or usage patterns. Profiling is useful in developing a model of the behavior and in deriving guidelines of what is normal and abnormal within that context. Examples of successful uses of profiles include profiling of traffic patterns on server links to uncover DoS and flash crowd events [4], web-server profiling [11], power usage profiles for efficient power management [10], profiling end-to-end paths to detect performance problems [8], profiling of traffic patterns on aggregated gateway and router links to facilitate accurate application classification [5], etc.

While there has been research on profiling web server traffic [4,11], and gateway and backbone links (i.e., highly aggregated traffic) [5,12], end-host profiling has received little attention. One work in this area is [7] in which the authors build end-host profiles with the goal of defending against worm attacks. Their profile describes the community of hosts an end-system normally interacts with.

We believe that observing the host behavior at the transport layer can reveal a wealth of information, such as: behaviors on who tries to talk to the host, who the host communicates with, the mix of applications used, the dispersion (or randomness) of the destinations contacted for a particular application, the pattern of port usage, the evolving mix of protocol usage, and so on. A number of security applications have identified particular features, derivable from packet header fields, as useful for detecting specific attacks. For example, many IDS

S. Uhlig, K. Papagiannaki, and O. Bonaventure (Eds.): PAM 2007, LNCS 4427, pp. 186–196, 2007.

systems will declare a machine compromised if the number of simultaneous TCP connections exceeds a predefined threshold [2,1]. Yet other systems look for a change in the dispersion on destination IP addresses to find anomalies [6].

All aforementioned work tends to define profiles within the bounds of their intended use. In this work, we are trying to formalize the concept of profiling transport layer information and identify desirable properties. We believe that a profiling mechanism, focused on end-hosts, should meet the following goals.

- *Goal 1:* It should be able to identify dominant and persistent behaviors of the end-system, capturing repeatable behaviors over time.
- *Goal 2:* It should be a compact enough representation, avoiding excess detail that may correspond to ephemeral behavior. This is important in a networking setting due to the use of ephemeral ports by certain applications.
- *Goal 3:* It should be stable over short-time scales avoiding transient variability in the host behavior.
- *Goal 4:* It should evolve by adding new behaviors and removing stale ones.
- *Goal 5:* It should be able to capture historical information to illustrate typical ranges of values for features.

Our main contribution is a novel approach to profile end-host systems based on their transport-layer behavior. First, we propose the use of a graph-based structure which we call a *graphlet*, to capture the interactions among the transport-layer protocols, the destination IP addresses and the port numbers. Note that *graphlet*s have two key properties: (a) they are extensible, since through annotations of the nodes or links one could achieve a lossless representation of all flow information through a single graph, and (b) they provide intuitive and interpretable information. The notion of *graphlet*s was introduced in [5] for application classification. Building on [5], we extend this original idea in a number of nontrivial ways. Second, we design a two step method that is based on an unsupervised online learning process. In the first step, we build and continuously update *activity graphlets* that capture all the current flow activity. The second step contains mechanisms to (a) compress the large *activity graphlets* to retain only essential information, and (b) to evolve this latter summary in a way that reflects changes over time. The output of this process is called a *profile graphlet*.

Using enterprise network traces, we find that user activity is successfully captured in our profiles. In particular, our profiles capture roughly 70-90% of all user activity, yet are about 80-90% smaller in size relative to the uncompressed activity graphlets. This result demonstrates that our profiles are efficient and compact while still remaining highly descriptive. Our initial findings indicate that profiles can vary greatly across users and this motivates the use of end-host profiles for security, diagnosis and classification applications. One of our interesting findings is that over short time scales (e.g., 15 minutes) the profiles evolve slowly typically experiencing small changes, yet over longer periods of time (e.g., a month), the majority of the profile content may change. This indicates that most parts of the profile are apt to change, and further underscores the need for adaptivity.

2 Data Description

We collected packet header traces within a secure enterprise network environment. Using the *CoMo* monitoring tool [9], we capture all traffic on the access link of our office building. Two traces were collected; one spans the entire month of October 2005, and the other a two week period in November 2005. We monitor the traffic of roughly 200 distinct internal IP addresses, that collectively represent user laptops and desktops, as well as network infrastructure equipment (e.g., NFS or DNS servers).

3 Methodology

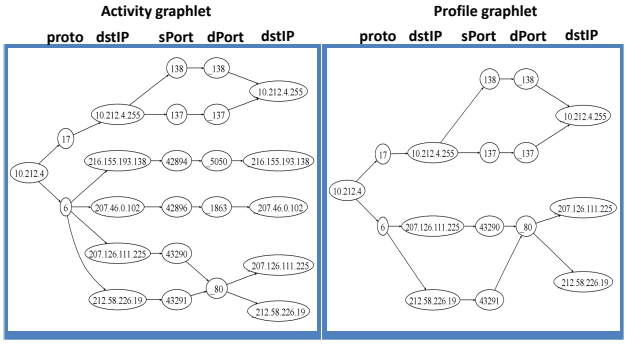
3.1 Capturing Host Activity Via Graphs

A fundamental element of our profiling methodology is the special purpose graph, called ***graphlet***. The concept of *graphlet*s was first introduced by the *BLINC* methodology [5] to capture the distinct transport layer footprint of different applications, termed as "application *graphlet*s". Application *graphlet*s were further used for the classification of the traffic observed at a traffic aggregation point into applications. Our use of *graphlet*s in this work is significantly different from the original BLINC work. We extend the definition of a graphlet, introduce graphlet annotations and manipulate the graphlet in different ways (in terms of learning, updating, compacting, etc). First, the intended goal and use of *graphlet*s in our work is substantially different compared to BLINC. BLINC's goal was to identify application footprints in traffic streams in a *supervised manner* according to the pre-defined *application graphlet*s. On the contrary, we study *graphlet*s with the goal to profile hosts in an *unsupervised way*, i.e., we learn (and update) an unknown user behavior on-line. Second, as described below, we extend the graphlet definition to include additional elements. Third, we introduce ideas for summarizing and creating compact *graphlet*s.

A *graphlet* is a graph arranged in six columns corresponding to: (srcIP, protocol, dstIP, srcport, dstport, dstIP). Fig. 1(top left) presents an example graphlet that was derived from one of our *LAB* hosts and plotted using the graphviz tool [3]. The BLINC *graphlet*s consisted only of the first 5 columns; the additional sixth column here is critical to our methodology.

Each ***graphlet* node**[1] presents a *distinct* entity (e.g. port number 80) from the set of possible entities of the corresponding column. The lines connecting nodes imply that there exists at least one flow whose packets contain the specific nodes. This way, each flow creates a directed ***graphlet* path** starting from the host IP address on the left and traversing the appropriate entities in each column. Note that we define a **flow** by the 5-tuple of the packet header, and the flow can consist of one or more packets. Similarly, a *graphlet* path can correspond to a multitude

[1] The term node indicates the components of a *graphlet*, while the term host indicates a communicating device.



Significant-Node Set

Node	6	10.212.4.255	_80
In-Degree	-	2	2
Out-Degree	4	2	2

Fig. 1. Example of a host activity and profile graphlets and the significant node set

of flows with the same 5-tuple definition. The destination IP appears twice at the third and sixth column in the *graphlet*. This redundancy is critical, since it allows us to observe *all* pairwise interactions between the most information-heavy fields of the 5-tuple: destination IP address ($dstIP$), the source port ($srcport$) and destination port ($dstport$).

If many flows traverse a node, the node will most likely have a high degree. By construction, all edges in a *graphlet* are between nodes of adjacent columns. If we traverse a *graphlet* from left to right by following a path, we define a direction in visiting the nodes. This way, we can define the **in-degree (out-degree)** of a node as the number of edges on the left (right) side of the node. The in- and out-degree of a $dstIP, srcport, or\ dstport$ node abstracts its interaction with the other two types of nodes. For example, the out-degree of a node representing port 80, captures the dispersion of addresses visited using web applications.

Because we are building profiles for a single host, there is only one source IP address and hence this field is not included in what we are terming the "heavy information fields" of the 5-tuple. We point out that a graphlet is a directed graph. When the host is the source, the directed edges flow from left to right in our depictions. If the directed edges flow from right to left, then our host is the recipient of incoming flows. Note that although conceptually each profile consists of two directed graphs, in practice a single data structure can be designed to capture all the needed information.

For example, Fig. 1 presents an "activity" graphlet which resulted by observing all the incoming and outgoing flows of a host during a specific time window. The "profile" graphlet refers to our distilled and compact version of the activity graphlet. (Activity and profile graphlets, along with significant node sets are discussed in Sec. 3.3).

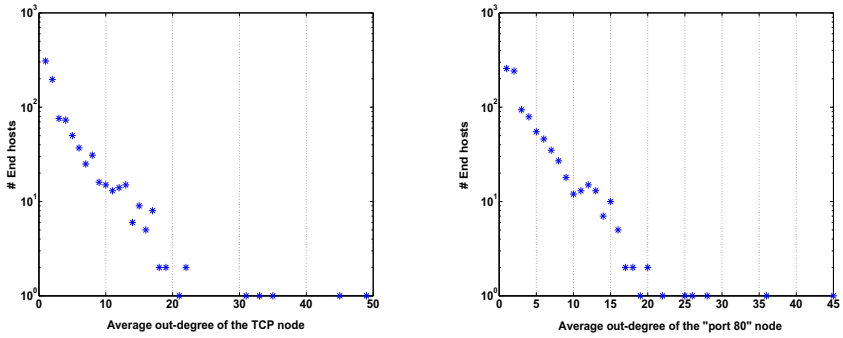


Fig. 2. Histograms of the average out-degree of two different nodes (TCP and "port 80" node) in the client *graphlet*s computed every 15 minutes. Significant variations in the number of out-degrees across clients point towards client personalized profiles.

3.2 The Advantages of *graphlet* Profiling

We believe that graphlets are an interesting approach to end-host profiling for a number of reasons. First, one could imagine keeping per host flow records in order to compute statistics regarding its behavior. A database of flow records is an enormous amount of information. Instead, our graphlet achieves a representation of important information in a compact form limiting the redundancy. Second, such flow records are not interpretable without further processing. However, the paths, nodes and node properties in graphlets are easy to interpret.

We can further expand *graphlet*s to annotate nodes with temporal information. For example, we can create time series information for each node (e.g., the time series of the out-degree). This is equivalent to annotating the nodes in the graph and tracking the evolution of the weights. Similarly, we can attach weights to links in the graph in order to track more typical features, such as the number of packets or bytes for all flows transiting that path. Existing security solutions use threshold based-techniques on metrics like the number of TCP connections per destination port per time interval. Recent solutions examine the dispersion of the 3 key fields [6]. All such techniques can be captured within the framework of weight-annotated *graphlet*s. The power of this profiling mechanism is that it goes beyond these methods, since it also incorporates the graph relationships, all in a single structure. We illustrate this here with three examples:

- The out-degree of the TCP node (or any protocol node) reveals the typical number of TCP destination IPs per client. By observing how the out-degree of the TCP node in the graphlet evolves over time, we learn about the typical range for the number of simultaneous destinations contacted through TCP within a window of time (the time scale of the graphlet). For example, Fig. 2 (left) presents a histogram of the average *out-degree* of the TCP node for all our client graphlets every 15 minutes. We observe a wide range of behavior.
- For applications with well-known port numbers, *graphlet*s can reveal what is the typical number of destination IPs contacted for each given application. For

Method: Construct Profile
1. Upon arrival of each packet, update *activity graphlet* if flow information not already included.
2. Every t minutes
   a). identify new significant activity, according to **summarization policy** as candidate to join profile graphlet.
   b). Add new significant activity into profile, if approved by **delayed-accept policy**, using Algorithm 1.
   c). Remove stale parts of profile according to **aging policy**.

Fig. 3. Summary of Method

Algorithm 1: Populate Profile Graphlet with Significant Nodes
Repeat until all significant nodes processed
1. Rank all nodes in *activity graphlet* according to their maximum in-degree or out-degree: $max\{indegree, outdegree\}$
2. Remove the highest degree node and all its edges. Insert into *profile graphlet*.

Fig. 4. Algorithm: inserting significant nodes into profile

example, examining the out-degree of the graphlet node for destination "port 80" reveals the number of destinations typically contacted by an HTTP application. Similarly Fig. 2 (right) presents a histogram of the average *out-degree* of the "port-80" node for all our client graphlets computed every 15 minutes. Again we see considerable variability across hosts.

- Scanning behavior can be easily seen from graphlets. For example, port scanning would appear as an excessively large number of destination ports associated with a single destination address. Similarly if a host initiates an address-space scan for a specific port (worm-like behavior) this would appear as an excessively large number of destination IPs associated with a single destination port.

3.3 Building Profiles

All the information obtained from monitoring a host's communication traffic could lead to an enormous graphlet (called the *activity graphlet*). Recall that, as per our goals described in the introduction, our aim is to capture "typical" or "persistent" behaviors in a compact way that avoids transient noise. We now describe our methods for converting the activity graphlet into a profile graphlet via policies for compression (i.e., summarization) and adaptivity.

Our method is depicted in Fig. 3. The policies used in this method were designed as follows. The intuition behind our **summarization policy** comes from observations on our trace data that activity graphlets do vary dramatically from one host to another, and a summary metric such as *number of nodes in the graphlet* is very volatile. Looking at activity *graphlet*s across many hosts, time intervals and traces, we did find one common characteristic; namely that they feature a small number of high degree nodes ("knots" in the *graphlet*). These nodes result from flows that share at least one *graphlet* node (e.g., distinct web flows that share port 80). At the same time, our activity *graphlet*s featured a number of paths comprising only one-degree nodes (ignoring protocol nodes).

See for example the middle two paths in Fig. 1(top left). Typically, those corresponded to ephemeral flows[2].

Building on this insight, we define the set of **significant nodes** in an activity *graphlet* to be those nodes with an *in-degree or out-degree larger than 1*. The only nodes we retain in our graphlet profiles are the significant nodes. We populate our graphlet profiles using the procedure outlined in Fig. 4. Fig. 1 gives an example of an activity graphlet and the resulting profile graphlet that it generates. We use the term **significant set** to refer to the group of significant nodes of a graphlet. The **profile *graphlet*** consists of the union of all the flows that are affiliated with the significant nodes. As such, the profile *graphlet* is a subset of the initial activity graphlet. Thus, we could say that our profiling consists of two components: (a) the significant set, and the (b) profile *graphlet*.

In order to evolve, our profiles need to: 1) remove information when it becomes stale, and 2) add new content when it becomes relevant. The time scale of this adaptivity affects both the stability and meaningfulness (i.e., utility) of the profile. If the profiles evolve too quickly, they will be less stable (nodes will be added and removed very frequently); whereas if they evolve too slowly, they will be less meaningful (miss new important nodes and contain stale ones). Let t denote the update period of the profile graphlet. Updating the profile means that the set of significant nodes at a time instance t is the union of the sets at time $t-1$ and t.

We employ a **delayed-accept policy** to control the addition of new nodes. Significant nodes are not inserted in the graphlet profile unless they are active for at least two consecutive intervals t. Such a mechanism is robust to ephemeral nodes introduced by the reuse of port numbers across flows.

We make use of an **aging policy** to remove obsolete information. A significant node is removed from a profile if it is inactive for some period of time. Our timeout period N is measured in days. Inactivity refers to nodes that are currently not significant but were in previous time intervals. Due to space constraints, we do not illustrate the stability and utility tradeoffs we observed for various values of t and N. In short, we found that using an update period t equal to 15 minutes, and aging threshold N of one week achieved a good tradeoff between utility and stability.

4 Properties of the End-System Profiles

Here we describe the properties that establish the robustness of significant nodes as a means of profiling end-user activity. To this end, we examine the extent to which our profiles meet the five goals mentioned in section 1.

Goal 1 - Capturing representative information: We first examine the identities of the nodes that populate the user profiles. Intuitively, the nodes should depict the primary activities of each end-system and if possible also reflect its functional role in the network (e.g., client vs. server).

[2] Note that ephemeral flows refer to a whole path in the *graphlet*, while ephemeral nodes only to the specific node.

Table 1. Profile instances of various end-systems

Host	activity *graphlet* size	significant node set in the profile
Client1	104	dst ports: 22 (SSH), 443 (HTTPS), 80 (HTTP), 2233 (VPN)
Client2	72	dst ports: 993 (IMAP), 137 (NETBIOS), 80 (HTTP), 995 (POP3)
Client3	259	dst ports: 80 (HTTP), 6881, 6882, 6884, 6346, 16881 (P2P)
NFS SERVER	31	src port: 2049 (NFS)
LDAP SERVER	309	src ports: 389 (LDAP), 139 (NETBIOS)

Table 1 presents five profile instances for three randomly picked clients and two servers from our enterprise networks. We observe that *all* significant nodes in the client profiles are destination ports reflecting well-known services accessed by the clients. Note that client 3 appears to run the *BitTorrent* peer-to-peer application and the set of significant nodes reflects common ports used by this application. The significant nodes for the servers, however, reflect the ports where the offered services reside.

Table 2. Most popular significant nodes

dstP = 80	dstP = 5499	dstP = 443	dstP = 2233	dstP = 53	dstP = 1863	dstP = 389	dstP = 22
WEB	CHAT	HTTPS	VPN	DNS	MSN	LDAP	SSH

To examine the identities of our profiles in a broader setting, we looked at the most popular significant nodes across all profiles. Table 2 presents the eight most popular nodes which, similarly, represent services at well-known ports. This initial data exploration indicates that *our profiles are able to capture dominant and meaningful end-system behavior and discriminate its functional role in the network*.

Note that while a number of significant nodes are common in host profiles, these significant nodes can be annotated with a variety of information such as their average out-degree to capture the user variability as shown in Fig. 2.

Goal 2 - Compact representation: To assess the breadth and compactness of the profiles, we define two metrics. *Compression* is defined as the ratio of the number of significant nodes over the total number of nodes in the activity graphlet. *Coverage* is defined as the fraction of flows that the profile captures compared to the total number of the flows generated by the host. (A flow is defined here as a unique 5-tuple.) A good profile should achieve high coverage and high compression because the significant nodes should: a) represent the majority of the activity of the edge-host (high coverage), b) amount to only a small number of the total nodes in the *graphlet* (high compression).

Fig. 5(left) shows that abstracting the *graphlet* to a set of significant nodes leads to a compression greater than 80% compared to the activity *graphlet*. We also see that the significant nodes often cover more than 90% of the flows sourced at the host. Recall that by definition, "uncovered" flows correspond to those whose path comprises only one-degree nodes in the *graphlet*. We thus conclude that *our set of significant nodes offers both high compression and coverage*.

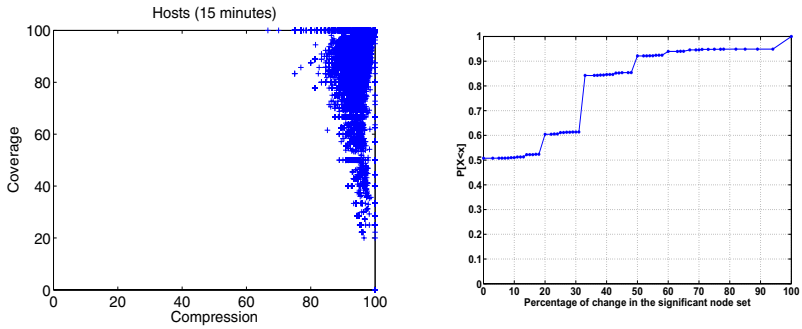


Fig. 5. LEFT: Coverage vs. compression for hourly *graphlet*s (LAB trace). RIGHT: Similarity across consecutive intervals for the significant node set. For approximately 50% of the intervals the significant node set remains the same across time.

Goal 3 - Stability: Recall that our profiles are updated every 15 minutes. We now examine the amount and nature of changes occurring in the profile over time. For each end host, we examined the difference in the set of significant nodes from one time slot to the next. The difference is the ratio of the number of nodes present in both intervals divided by the average number of significant nodes in the two sets. In Fig. 5(right) we present the CDF of these ratios for all hosts over all time slots.

We observe that roughly 50% of the time there is no change at all from one time slot to the next. Also, less than 10% of the time does a node change by more than 70%. We conclude that while there appears to be a reasonable amount of stability from one 15 minute window to the next, every so often the profile can experience a large change. These initial results hint that perhaps over shorter time scales these profiles can remain stable, yet over longer time periods, profiles can experience large amounts of change. This indicates that a notion of stability should perhaps be tied to the amount of evolution occurring in user behavior. This is a subject of our future research.

Goal 4 - Evolvability: Fig. 6(left) demonstrates the impact of the "delayed-accept" and "aging" policies on the total number of significant nodes for all hosts in the network. The upper line corresponds to the total number of significant nodes across all hosts when only "aging" is used, while the bottom line also incorporates the effect of "delayed-accept". During the first week of profiling the number of significant nodes shows a constant increase in both cases. This is the "learning" stage of our approach and lasts approximately 1.5 weeks. While the effect of "delayed-accept" is evident across time, "aging" is observed after the first week due to our choice of "weekly" history. The sum of significant nodes appears not to vary significantly after approximately 2 weeks. Note that while the time interval on the *x-axis* spans a time period of a month, we only observe a few changes. These initial results indicate that our *delayed-accept* and *aging* policies do manage to filter transient behavior while balancing the stability.

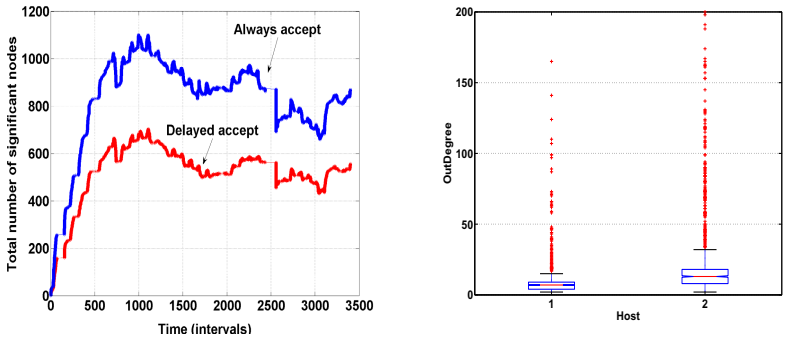


Fig. 6. LEFT: Total number of significant nodes for all network hosts when we use delayed-accept" and "aging". RIGHT: Boxplot of the outdegree time-series for a common significant node for two host profiles.

Goal 5 - Capturing historical information: Recall from section 3 that each significant node in the profile can be annotated with various time-series information. Fig. 6(right) presents such an example in a box plot showing the out-degree of a common significant node (web) across two hosts. Such time-series can be further analyzed to provide insight regarding *typical* individual behavior (e.g., average number of TCP connections), a *range* of behavior (e.g., 90 percentile points), and *outliers* (denoted with the points outside the wedges in the figure). We postulate that this sort of information could be important for anomaly detection applications (benign or malicious).

5 Conclusions–Discussion

In this paper, we present a novel approach to profile end-host systems based on their transport-layer behavior. We introduce the idea of using graphs to capture flow information and inter-flow dependencies. We illustrate that all of a host's flow data can be greatly compressed into a compact representation, that captures dominant user behavior. Initial results suggest that a user's behavior can undergo large changes over time, and this underscores the need for adaptive profiling.

We envision our profiling methodology being used in many different ways depending on the intended goal. Examples include:

- For enterprise network management, to understand user behavior for resource provisioning, load balancing, allowing for user clustering based on similar profiles, etc.
- Monitoring the profile graphlet in comparison to the activity graphlet could be useful for anomaly detection. Abrupt changes in either the normal range of behavior, or outlier events, could signal an anomaly, whether benign or malicious.
- Monitoring the patterns in the out-degrees of protocol-nodes, or other significant nodes, could reveal scanning attempts.

References

1. Intrusion Detection Systems (IDS) Part 2 - Classification; methods; techniques. In *http://www.windowsecurity.com/articles/IDS-Part2-Classification-methods-techniques.html*, 2004.
2. Arbor Networks. http://www.arbor.net/.
3. Graphviz. http://www.graphviz.org/.
4. J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: Characterization and implications for cdns and web sites. In *Proceedings of the 11th International World Wide Web Conference*, May 2002.
5. T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: Multi-level Traffic Classification in the Dark. In *ACM SIGCOMM*, August 2005.
6. A. Lakhina, M. Crovella, and Christophe Diot. Mining Anomalies Using Traffic Feature Distributions. In *Proc. of ACM SIGCOMM*, August 2005.
7. P. McDaniel, S. Sen, O. Spatscheck, J. Van der Merwe, B. Aiello, and C. Kalmanek. Enterprise Security: A Community of Interest Based Approach. In *Proc. of Network and Distributed System Security (NDSS)*, Feburary 2006.
8. V. Padmanabhan, S. Ramabhadran, and J. Padhye. NetProfiler: Wide-Area Networks Using Peer Cooperation. In *Proceedings of the Fourth International Workshop on Peer-to-Peer Systems (IPTPS)*, February 2005.
9. The CoMo Project. http://como.intel-research.net/.
10. G. Theocharous, S. Mannor, N. Shah, B. Kveton, S. Siddiqi, and C.-H. Yu. Machine Learning for Adaptive Power Management, 2006. Intel Technology Journal.
11. Mengjun Xie, Keywan Tabatabai, and Haining Wang. Identifying Low-Profile Web Server's IP Fingerprint. In *IEEE QEST*, 2006.
12. K. Xu, Z.-L. Zhang, and S. Bhattacharyya. Profiling Internet Backbone Traffic: Behavior Models and Applications. In *ACM Sigcomm*, August 2005.