Tracking Long Duration Flows in Network Traffic

Aiyou Chen*, Yu Jin†, Jin Cao*, Li Erran Li*
* Bell Laboratories, Alcatel-Lucent † Computer Science Dept., University of Minnesota

***Abstract*—We propose the tracking of long duration flows as a new network measurement primitive. Long-duration flows are characterized by their long lived nature in time, and may not have high traffic volumes. We propose an efficient data streaming algorithm to effectively track long duration flows. Our basic technique is to maintain only two Bloom filters at any given time. In each time duration, only old flows that appear in the current time duration get copied to the current Bloom filter. Our basic algorithm is further enhanced by sampling. Using real network traces, we show that our tracking algorithm is very accurate with low false positive and false negative probabilities. Using multi-faceted analysis, we show that more than 50% of hosts participating in long duration flows (duration no less than 30 minutes) are blacklisted by various public sources.**

I. Introduction

Network administrators need to monitor their network traffic timely and effectively to respond to network anomalies and security problems or help with traffic engineering decisions. Due to the sheer volume of network traffic, they increasingly rely on sophisticated data streaming algorithms to characterize their network traffic or extract a subset of flows of interest, where a flow is a sequence of packets that share the same flow identifier (e.g. source/destination IPs, source/destination ports and protocol). There have been many studies on summary statistics such as flow distribution [5], packet feature (IP addresses and ports) distribution [6], and entropy (e.g. entropy of the packet distribution over various ports) [6], [7], [1].

Prior work on extracting a subset of flows has focused on tracking heavy hitters and super-spreaders. Heavy hitters are top ranked hosts by traffic volume [3]. A super-spreader is defined as the host that communicates with a large number of peers (at least k) [9]. Little attention has been paid on long lived flows. A flow is long-lived if it persists for a long period of time, say 30 minutes. From the interaction with some operators, it seems that the conventional wisdom is that long duration flows are not interesting as they mostly belong to p2p traffic or video traffic of normal Internet hosts. However, as we show in our analysis, this is not the case. First of all, the proportion of flows from chat applications increases as flow duration increases. In the traces we analyzed, long duration flows from chat applications can be as high as 48%. Second, more than half of all hosts participating in long duration flows get blacklisted by real-time black lists. We have identified most of the blacklisted hosts participate in chat applications (port 5190 for AOL, 1863 for MSN, 5050 for Yahoo and 6667 for IRC). In one trace where the timing coincides with Storm worm outbreak, we have also found that 10% of hosts that generated long duration flows belong to Storm worm botnet.

We propose the use of long-duration flows as a network monitoring primitive. It is useful for both anomaly detection and traffic engineering. On the anomaly detection side, long duration flows which are low in traffic volume have a high probability of being participating in botnet activities. This has been confirmed by our analysis on real network traces. Thus, new appearance of these long duration flows can help quickly identify botnet traffic, and measures taken can prevent botnet attack before it is launched. On the traffic engineering side, long-lived flows are significant in traffic volume, e.g., in one university trace we analyzed, 23% (bytes) of the traffic are contributed by long-duration flows, and these flows may not fall into the category of heavy hitters. Information on these flows can aid in traffic engineering decisions.

In this paper, we are interested in tracking long-lived flows in network traffic. Given a duration threshold d, say 30 minutes, what flows last longer than d? Existing data streaming techniques work on a per-epoch basis. They do not apply to this problem which tracks flow for many epochs. A naive approach is to directly track all the flows, i.e., by maintaining the start time and last time of a packet arrival for each flow in a hash table, and periodically removing flows with duration less than d from the hash table, when the corresponding FIN packets are received or the flows are inactive for some time. Here, we propose a simple yet effective technique for identifying long-duration flows. Our basic idea is to only maintain two Bloom filters (B_1 and B_2) and one small hash table at any given time. More precisely, at the starting time interval t_0, we use Bloom filter B_1 to record all the flows that appear during t_0. At the next time interval t_1, we only add new flows or flows that appear both in B_1 and t_1 to B_2. We then iterate the process by switching the roles of B_1 and B_2 on the subsequent time intervals. When a flow is identified to be longer than d, we add it to the hash table. Our basic algorithm is further enhanced by sampling techniques. We show that our approach can reduce the memory requirement of the naive algorithm by orders of magnitude. See complete analysis of false positives and false negatives in our full paper at http://ect.bell-labs.com/who/aychen/ldfmain.pdf.

The paper's structure is as follows. We briefly discuss related work in Section II. In Section III, we propose our data streaming algorithm. Performance evaluation and analysis are presented in Section IV and Section V respectively.

II. Related work

Our work falls in the broad scope of data streaming algorithms for network traffic analysis. For real time analysis, streaming algorithms must process the data in one pass as storing the data would entail a large delay. Most of existing

algorithms are not concerned about flow durations. One notable exception is the work of Whitehead on flow duration tracking [10]. Their problem is on tracking all flow durations. They use a set of Bloom filters, each of which represents flows with duration falling in a certain range. An aging process allows Bloom filters to track longer duration flows. Our algorithm is much more efficient and exploits the fact that we are only interested in long-lived flows.

III. Data Streaming Algorithm for Tracking Long Duration Flows

Let Δ be the predefined time-out value and $d\Delta$ be a threshold value that defines a long duration flow (LDF), where d is assumed to be an integer. The straightforward solution mentioned in the introduction needs to track all flows and thus is not an efficient way due to large memory requirement and the fact that typically a majority of flows are not LDFs.

In this section, we develop a simple and memory-efficient algorithm for tracking LDFs. The basic idea is to make use of two counting Bloom filters to 'store' the candidate set of LDFs that are active in the past and current unit time intervals, respectively. The unit time interval is chosen to be the timeout value Δ, so that past candidate flows that do not have packets in the current time interval will no longer be stored in the current Bloom filter. To reduce the false positive rate of identified LDFs due to collisions in Bloom filters, we also use a hash table to 'store' the already identified LDFs so that new flows that collide with LDFs will not be mistaken as LDFs. This basic scheme can be enhanced by using a sampling procedure to filter out a significant proportion of short duration flows at the expense of affordable false negatives. Our online flow duration estimation is similar to the Count-Min sketch devised by [2] for heavy hitter detection (minimum value of the hashed entries), but our (Bloom Filter) update uses the maximum of its current hash entry and an estimate from past, which is different from Count-Min sketch and standard CBF.

A. Basic Algorithm for Tracking LDFs

Let B_1, B_2 be two CBF each with m counters that are used to store the durations (in units of Δ) of candidate LDFs of the past and current interval respectively. Let $(h_{i1}, \cdots, h_{iK})$ denote K hash functions associated with B_i, $i = 1, 2$, where each hash function maps a flow ID to one of the m counter locations uniformly. Let f denote a flow ID. The duration estimate of f for each $B_i(f), i = 1, 2$, is the same as that for a Count-Min sketch:

$$B_i(f) = \min_{1 \le k \le K} (B_i[h_{ik}(f)]) \quad \text{(duration estimate)} \qquad (1)$$

where $B_i[h]$ is the counter-value of B_i at location h[1]. To make the counting Bloom filters B_1, B_2 amenable for duration counting, during each time interval, B_2 is updated from B_1 in the following manner. First, initialize B_2 with zeroes. Upon

[1]For simplicity, we slightly abused the notations, where $B_i(f)$ denotes duration estimate of flow f, and $B_i[h]$ denotes the counter value at location h.

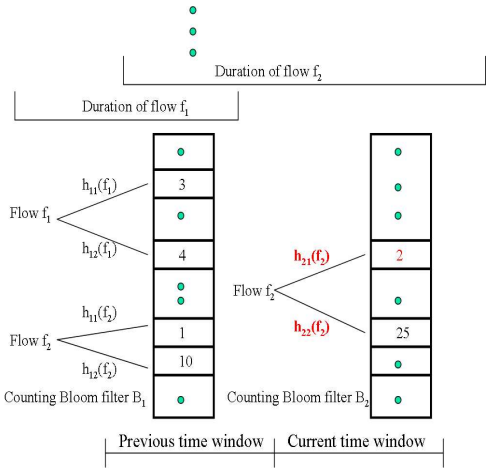


Fig. 1. Diagram illustrating the basic flow tracking algorithm with two counting Bloom Filters.

the arrival of a packet with flow ID f, if $B_2(f) > B_1(f)$, then this implies that f already exists in B_2 and thus we skip to next packet. If not, then we update

$$B_2[h_{2k}(f)] \leftarrow \max(B_1(f) + 1, B_2[h_{2k}(f)]) \quad \text{(update)}. \qquad (2)$$

The intuition for using the maximum in the above is that if $B_2[h_{2k}]$ exceeds $B_1(f)+1$, then the counter value must reflect the duration of a flow other than f.

Figure 1 illustrates the basic algorithm with B_1 and B_2, using $K = 2$ hash functions and two flows f_1 and f_2, where f_1 terminates at the current interval, but f_2 spans both previous and current intervals. At the previous interval, duration estimates for f_1, f_2 are 3 and 1 respectively. Since no packets from f_1 appear in the current interval, f_1 is not recorded in B_2. However, the duration estimate of f_2 is incremented to 2 by updating corresponding hash entries in B_2 since at least one of its packets appeared in the interval. Obviously the entry value 25 in B_2 is contributed by a flow other than f_2.

To reduce the false positive rate of the identified LDFs due to short flows colliding with long duration flows, we use a a hash table T to store the identified long duration flows (i.e., those whose duration estimates exceed $d\Delta$), that are active in the current interval. The following describes the implementation of the basic algorithm using both B_1, B_2 and hash table T on a case by case basis.

Upon the arrival of a new packet with flow ID f in the time interval t, suppose that we first obtain $B_1(f)$ and check whether $f \in B_1$. Then we obtain $B_2(f)$ and check whether $f \in B_2$. $B_i(f) = 0$ iff $f \notin B_i$. If $B_2(f) > B_1(f)$, f already exists in B_2 and thus we skip to next packet. Otherwise, $B_2(f) \le B_1(f)$, i.e. f has not been updated in B_2, then there are four possibilities:

(C1) If $B_1(f) = 0$, f is a new flow. Insert it into B_2 and update $B_2[h_{2k}(f)]$ by $\max(1, B_2[h_{2k}(f)])$ for each k.

(C2) If $0 < B_1(f) < d - 1$, f may already exist (collision may occur) and we update $B_2[h_{2k}(f)]$ by $\max(B_1(f) + 1, B_2[h_{2k}(f)])$ for each k.

(C3) If $B_1(f) = d-1$, f has duration d (collision may occur), update $B_2[h_{2k}(f)]$ by $\max(B_1(f)+1, B_2[h_{2k}(f)])$ for each k, and insert f into T with value d;

(C4) If $B_1(f) \geq d$, f may be an existing LDF (collision may occur), and so we check whether $f \in T$. If yes, insert f into T with value $B_1(f)+1$ (existing LDF); Otherwise, skip to next packet (f may not be a new flow, in which case, its previous information is lost).

By the end of the time interval, replace B_1 by B_2, reset B_2 with 0s, and eliminate any flows in T that do not have packets in the interval. If the next packet arrives at or after $t+2$ intervals, both Bloom filters are reset to 0s.

B. Enhancements using Sampling

The majority of network flows have very short duration. For example, web transactions are typically short, most less than one minute. In addition, one packet flows due to for example scanning also has a significant presence in network traffic. Since the counting Bloom filters B_1 and B_2 have to record all active flows, these flows waste lots of resources. Therefore, the basic LDF tracking algorithm outlined above can be further enhanced by using a flow sampling procedure described below to filter out a significant proportion of the short duration flows.

Let r be a pre-specified sample rate. Let t be the index of the time interval. For each flow f, we sample the pair (f,t) independently with the rate r until f is picked. Once f is picked, the flow ID will be recorded by its current CBF and all later packets will be picked by the counter Bloom Filters B_1 and B_2. It is easy to see that for a flow with a very short duration, the chance that it will be picked is small, and hence we can save space without recording these flows in B_1 or B_2. On the other hand, if the sampling rate is high enough, LDF will almost always be picked. The following extends the algorithm description (C1) under the proposed sampling scheme.

(C1') If $B_1(f) = 0$, f is a new flow and with probability r it is inserted into B_2 and update its buckets $B_2[h_{2k}(f)]$ by $\max(1, B_2[h_{2k}(f)])$ for $k = 1, \cdots, K$.

An outline of the algorithm with sampling is given in Figure 2, where the time unit is the predefined timeout value Δ.

Notice that the memory requirement of our algorithm can be further reduced by applying count truncation techniques such as Counter Braids [8], or making use of FIN information. We note that sampling may cause under-estimation of flow durations and thus results in false negatives for the identification of LDFs. Figure 3 (a,b) shows a simulation regarding the tradeoff between false positives and false negatives with different sampling rates where flow duration follows a geometric distribution.

IV. Performance evaluation

In this section we evaluate the performance of the proposed LDF tracking algorithm with real network traces. We report the false positive and false negative errors with given memory requirements. The first trace is based on a one-day packet trace of a large corporation network, and the second one is based on a one-hour packet trace of a large university network.

```
1. Initialize an empty hash table T, two counting Bloom filters
   B1, B2 with 0s, t0 = 0, and threshold d (assume Δ = 1)
2. For each incoming packet with flow ID f and arrival time t
3.     Switch (t − t0),
4.         Case t − t0 ≥ 2:
5.             reset B1, B2 to 0s and T to be empty
6.         Case 1 ≤ t − t0 < 2:
7.             update B1 by B2 and reset B2 to 0s, and
               remove time-outed LDFs from T
9.         Case 0 ≤ t − t0 < 1:
10.            look up v1 = B1(f), v2 = B2(f)
               (set 0 if no match)
11.            If v2 ≤ v1,
12.                Switch (v1),
13.                    Case v1 = 0: # C1' #
14.                        insert f into B2 with sampling rate r
15.                    Case 0 < v1 < d − 1: # C2 #
16.                        update B2[f] by (2), and
                           OUTPUT f, T[f]
18.                    Case v1 = d − 1: # C3 #
19.                        update B2[f] by (2), and
                           insert T[f] = d
20.                    Case v1 ≥ d: # C4 #
21.                        If f ∈ T,
22.                            update T[f], B2[f] by (2)
23.                        Endif
24.                Endswitch
25.            Endif
26.    Endswitch
27.    update t0 = t0 + ⌊t − t0⌋
28. Endfor
29. OUTPUT T at the end of a measurement epoch.
```

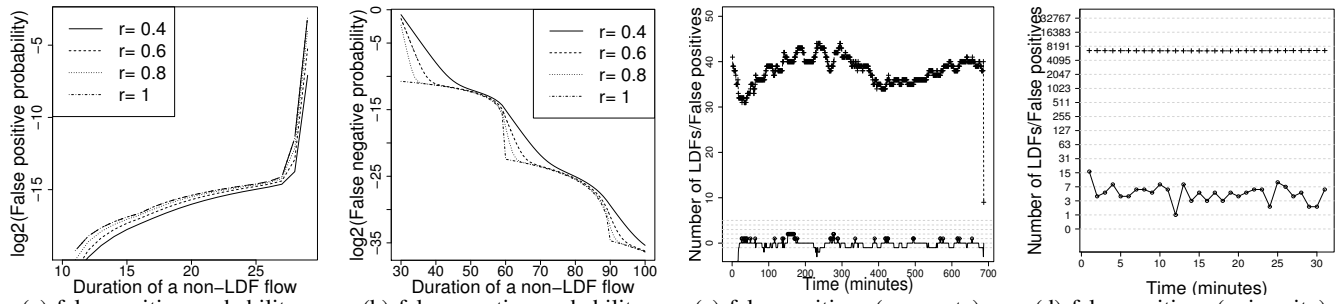
Fig. 2. Tracking LDF

A. A corporate network

The corporate trace covers 12 hours on a weekday in January of 2007. There are about 31.5 million packets and 0.61 million flows, most of which are one-directional flows. Figure 4 (b) shows the distribution of flow durations across the 12 hours, and Figure 4 (a) shows the distribution of active flow durations in a one-minute time interval randomly picked from the last 8 hours. The number of active flows in each one-minute interval varies from 800 to 1500. We apply $m = 2,048$ buckets and $K = 3$ hash functions to each CBF. Without sampling we obtain 103 LDFs, where we have only 5 false positives and no false negatives. Here a flow with duration greater or equal to $d = 30$ minutes is called a LDF.

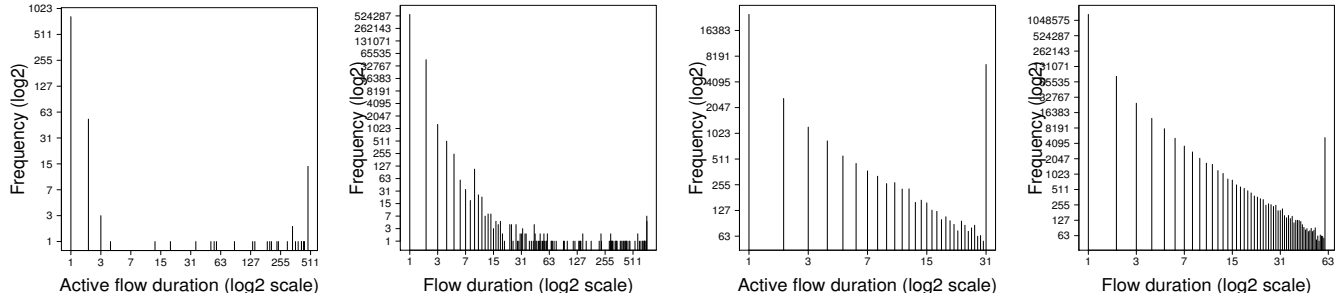
We also calculate the number of false positives and number of active LDF flows for each time interval, see Figure 3 (c), which shows the time series of both sequences (the first 30 minutes are not included since there are no false positives). The number of active LDFs falls into the range [30,45], while the number of false positives is mostly 0, except occasionally 1 and rarely beyond 1.

We then apply the sampling algorithm with $r = 0.5$. With $m = 2,048$ and $K = 3$, we obtain only 4 false positives and 1 false negative. Our investigation finds that the 1 false negative flow in fact has duration 31 minutes but estimated to be 29 minutes due to delay caused by sampling. When we set



(a) false positive probability (b) false negative probability (c) false positives (corporate) (d) false positives (university)

Fig. 3. (a,b): How sampling affects false positive and false negative probabilities; (c,d): Time series of the number of active long duration flows (in '+' with the dotted line) and the number of false positives (in circles with the solid line) for a large corporate network with no false negatives but 3 false positives over the whole trace time period with $m = 2048$ and $K = 3$ (c), and for a large university network with 36 false positives and only 1 false negative over the whole trace time period with $m = 262,144$ and $K = 3$ (d) (no sampling)



(a) Active flows (corporate) (b) Flow durations (corporate) (c) Active flows (university) (d) Flow durations (university)

Fig. 4. Flow duration distributions of a corporate network with a one-day packet trace (a,b) and a university network with a one-hour packet trace (c,d)

$d = 60$ minutes, there is no LDF whose duration is around d. Applying sampling rate $r = 0.5$ does not introduce any false negatives but decreases the number of false positives from 4 to 3. In this case, we only have 69 flows with duration greater or equal to 60 minutes. Note that the memory requirement in the above is only about 2.5K bytes.

B. A university network

The university trace covers about one hour on a weekday in Sept 2008. There are about 314.1 million packets and 1.37 million flows, most of which are one-directional flows. Figure 4 (d) shows the distribution of flow durations over the whole trace, and Figure 4 (c) shows the distribution of active flow durations in a one-minute time interval randomly picked from the second half hour. The number of active flows in each one-minute interval varies from 30,000 to 55,000. The active flow duration distribution shows that there are many flows with durations around $d = 30$ and thus sampling will introduce a lot of errors from boundary effects. We apply $m = 262,144$ buckets and $K = 3$ hash functions to each CBF. Without sampling we obtain only 36 false positives and only 1 false negatives. (Again a flow with duration greater or equal to $d = 30$ minutes is called a LDF.) Note that the memory requirement is only about 330K bytes.

We also calculate the number of false positives and number of active LDF flows for each time interval, see Figure 3 (d), which shows the time series of both sequences for the second half hour. The number of active LDFs falls into the range $[7900, 8190]$, while the number of false positives is mostly below 7. Note that the sum of false positives is much greater than 36, this is because that a flow that has duration less than d and becomes a false positive will continue to be a false positive until it ends. Our investigation in details found that 99% of such false positives have durations either 28 or 29 minutes and part of the false positive flows with duration 29 minutes are caused by rounding errors since all packet arrival times are discretized and rounded into minutes.

V. Analyzing the causation of LDFs

In this section, we investigate the causation of LDFs from the associated applications and participants. We conduct experiments on real Internet traffic traces from two heterogeneous sources: a large university campus network and one major ISP. Traces from the university include two day-long data sets, *u-tr1* and *u-tr2*. Both traces consist of *unsampled* packets captured at the border router of a large campus network. Each data set contains above 300 million flows[2]. Close examination of these traces reveals a significant amount of p2p traffic from campus hosts and scanning traffic from remote hosts. In comparison, ISP traces are provided by one of the major ISPs. Three ISP traces (*isp-tr1*, *isp-tr2* and *isp-tr3*) consist of *packet sampled* traffic data from three contiguous days. There are around 80 million flows in each ISP trace. The ISP traces contain much more variety of traffic types, such as traffic related to large commercial websites, or global malicious activities like botnets, worms and DDoS attacks, etc. All these traces are different from (in fact much larger than) those used for performance evaluation in Section V.

[2]A flow is an aggregation of *bidirectional* packets with the same 5-tuple.

TABLE I
PERCENTAGE AND NUMBER OF LDFS

	u-tr1	*u-tr2*	*isp-tr1*	*isp-tr2*	*isp-tr3*
#. LDFs	20,462	18,619	820	915	854
Flow pct. (%)	0.0611	0.0739	0.0011	0.0010	0.0011
Byte pct. (%)	23.6	24.6	0.41	0.44	0.45

TABLE II
PERCENTAGE AND NUMBER OF BLACKLISTED (RBL) HOSTS

	u-tr1	*u-tr2*	*isp-tr1*	*isp-tr2*	*isp-tr3*
#. RBL hosts	8,650	9,091	862	956	801
Percentage(%)	54.46	61.44	71.1	72.8	70.1

A. Characteristics of LDFs

We extract *TCP* flows longer than *30 minutes* from all 5 data sets. From the university traces, we identify 15K out of 50K hosts participating in LDFs everyday, while in the ISP traces, around 1.3K out of 4 million hosts are associated with LDFs per day. Such difference is likely due to the packet sampling strategy for collecting the ISP traces, which leads to missing LDFs (note the probability that the packets from a LDF are always captured in every 5-minute time interval decreases exponentially as the flow duration increases)[3].

We observe in Table I that the LDFs only account for a extremely small proportion of flows in each data set e.g., around 20K LDFs everyday in the university traces and around 700 LDFs per day in the ISP traces. Despite the small number of occurrence, these LDFs account for a relatively large proportion of bytes in the traffic. For example, LDFs account for around 0.4% bytes in the ISP traces and above 23% bytes in the university traces, respectively.

In addition to the large byte proportion, these LDFs often indicate interesting persistent network activities. We investigate the applications that cause LDFs using the ISP traces, where we have the ground truth in terms of the application associated with each flow. Such information is obtained by matching the application header against predefined expert rules. In the ISP traces, p2p traffic and video traffic account for the majority of flows with duration less than 5 hours (90.12% in total), while the proportion of such traffic drops quickly when the flow duration becomes longer. For example, only 56.83% of the flows with duration from 6 hours to 12 hours are p2p or video flows. Instead, the proportion of the chat flows increases to 24.46%. For the flows with duration more than 12 hours, the proportion of chat flows further increase to 47.76%. The p2p and video traffic totally account for 82.9% of the total traffic volume. Similar observation is made for the university traces using port numbers, which we omit due to space limit.

B. Analysis of LDF participants

We next study the participants of LDFs. We utilize information from two independent sources, the real-time black list (RBL) and a list of bots from the Storm worm botnet.

Querying RBLs. RBLs are publicly available lists of addresses associated with reported spamming activities. We build a RBL query tool to check against 147 public RBLs for the participants of LDFs. We label a host as suspicious if it is listed in at least one of the RBLs. The percentages of blacklisted hosts are illustrated in Table II. The results are very surprising. More than half of all the LDF participants are blacklisted! Most such blacklisted LDF participants are involved in chat applications, where the port 5190 (AOL), 1863 (MSN), 5050 (Yahoo) and IRC (6667) rank the top. It is very likely that these hosts are members of specific botnets which utilize existing chat applications as communication channels to keep persistent connections with the botmaster.

Matching Storm worm botnet list. The ISP traffic traces are captured in 2007, when the Storm worm botnet first becomes publicized as a giant botnet consisting millions of bots and operating in a way as a p2p network. We use a list of bot addresses which is obtained by collecting and decoding the p2p queries in the botnet (similar as the approach in [4]) to match the LDF participants. Interestingly, around 10% out of all the 1.2K LDF participants belong to the Storm worm botnet. Admittedly, Storm worm bots may also participate in normal p2p and video streaming activities. However, the average packet size of the LDFs associated with the Storm worm bots is below 300. This indicates these LDFs are mainly used for persistently transferring small amount of data. We suspect these LDFs are due to continuous queries from storm worm bots for the supernode addresses. In addition, unlike normal p2p activities, there is no dominant port in the flows associated with storm worm bots. This fact coincides with nature of Storm worm botnet, where bots communicate with each other through a p2p network, and random port is chosen for communication between peers to avoid detection.

[3]We argue that, due to the low computational cost, our method can be directly applied to high speed links to identify LDFs from unsampled packets.

REFERENCES

[1] A. Chakrabarti, G. Cormode, and A. McGregor. A near-optimal algorithm for computing the entropy of a stream. In *SODA*, 2007.

[2] G. Cormode and S. Muthukrishnan. What's new: Finding significant differences in network data streams. In *Proc. IEEE Infocom*, pages 1534–1545, 2004.

[3] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proc. ACM SIGCOMM Internet Measurement Workshop*, Nov. 2001.

[4] C. Kanich, K. Levchenko, B. Enright, G. M. Voelker, and S. Savage. The heisenbot uncertainty problem: challenges in separating bots from chaff. In *LEET'08*, 2008.

[5] A. Kumar, M. Sung, J. J. Xu, and J. Wang. Data streaming algorithms for efficient and accurate estimation of flow size distribution. In *ACM SIGMETRICS/Performance*, pages 177–188, 2004.

[6] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. *SIGCOMM Comput. Commun. Rev.*, 35(4):217–228, 2005.

[7] A. Lall, V. Sekar, M. Ogihara, J. Xu, and H. Zhang. Data streaming algorithms for estimating entropy of network traffic. *ACM SIGMETRICS Perform. Eval. Rev.*, 34(1):145–156, 2006.

[8] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani. Counter braids: a novel counter architecture for per-flow measurement. In *SIGMETRICS*, pages 121–132, 2008.

[9] S. Venkataraman, D. Song, P. Gibbons, and A. Blum. New streaming algorithms for superspreader detection. In *Proc. of Network and Distributed Systems Security Symposium*, Feb. 2005.

[10] B. Whitehead. Binned duration flow tracking and symmetric connection detection. *Master Thesis, Carleton University, Canada*, Jan 2007.