

ElephantTrap: A low cost device for identifying large flows

Yi Lu, Mei Wang, Balaji Prabhakar
Dept. of Electrical Engineering
Stanford University
Stanford, CA 94305
yi.lu,wmei,balaji@stanford.edu

Flavio Bonomi
Cisco Systems
175 Tasman Dr
San Jose, CA 95134
flavio@cisco.com

Abstract

This paper describes the design of ElephantTrap, a device which aims to cache the largest flows (the “elephants”) on a network link. ElephantTrap differs from previous work on identifying large flows in one crucial sense: it does not attempt to accurately estimate the size of the flows it is trapping. This leads to an extremely lightweight design and a surprisingly good performance. ElephantTrap can be employed in the line cards of switches and routers and be used for diagnostics, anomaly detection and traffic engineering.

1 Introduction

A key function of network performance monitoring is determining how bandwidth is used by flows; in particular, determining *which flows* use the most bandwidth. Network monitoring devices, such as Cisco Systems’ NetFlow [8], provide information to operators regarding the “top talkers” in the network, where top talkers are loosely defined (see [9]) as “the flows that are generating the heaviest system traffic.” From this loose definition, one can define two problems: (i) Identify the largest flows *without* getting an accurate estimate of their rates, and (ii) identify the largest flows and state an accurate estimate of their rates. While almost all research, including products like NetFlow, addresses problem (ii), our work focuses on problem (i). By attempting to solve the lesser problem we obtain a device, called ElephantTrap, that is much easier to implement. ElephantTrap can be used to perform a variety of different functions such as traffic engineering, load balancing, network forensics (e.g. after a denial-of-service attack or a network outage) and anomaly detection. It can also be adapted to identify bursts on a wide range of time scales; i.e., the so-called “microbursts.”

Before proceeding to describe our objectives and give a high-level description of ElephantTrap, it is helpful to sur-

vey some related literature. Initial attempts [7, 6] to *exactly* measure *all* flows proved unscalable: the cost and speed of the required memory proved prohibitive. This led to research that used some form of flow or packet sampling to reduce the complexity of measurement and monitoring devices [3, 2]. The paper by Estan and Varghese advocated a shift away from per-flow monitoring and towards monitoring only those flows that consume a significant amount of bandwidth; i.e., the so-called *elephants*. The short flows, the *mice*, are ignored. Due to the heavy-tailed nature of network traffic [1], there are few elephants compared to mice and the scalability problem is vastly reduced. They identify elephants using a pre-filter based on multiple hashing on an array of counters. Once an elephant is identified, all further packets from it are counted, a technique they called “sample and hold.” While this approach is certainly simpler, it is not simple enough to be readily implemented. It requires an array of counters for the pre-filter, and per-flow counters for tracking sampled flows, which include the elephants and therefore can be more than 15-20% of all flows.

Our goal is to be even simpler to implement. In order to achieve this we start with a lesser aim: merely identify the largest flows; there is no need for an accurate count of the amount of data they send. This allows us to work with ElephantTrap caches of a small size, say 32 or 64 entries, and yet be able to trap the dominant flows.

It is useful to distinguish flows according to both size and rate. Thus, a large flow (one which sends many bytes) may have a small rate, and vice versa.¹ By an “elephant flow” in this paper we mean a flow which is both large and sends data at a high rate. We are now ready to give the following informal description of ElephantTrap, deferring a precise description and pseudocode to Section 2. ElephantTrap is a cache which stores flow IDs and has a counter for each flow. Arriving packets are sampled and the flow ID corresponding to a sampled packet is inserted into the cache. If

¹It is perhaps better to classify flows according to size as “elephants and mice,” and according to rate as “cheetahs and turtles.” Since this would introduce unfamiliar terminology, we avoid it in this paper.

the flow is already present in the cache, its count is incremented. Eviction occurs when a new flow is sampled and the cache is full. In this case an existing flow with a small count value is evicted.

The design philosophy of ElephantTrap is: Sample for size and hold for rate. It is very easy to sample large-size flows by sampling packets. Indeed, by sampling each packet with a probability of say p , flows with at least $1/p$ packets have a very good chance of being sampled. On the other hand, it is harder to sample according to rate because the rate of a flow is not determinable instantaneously. Once a flow is in the cache, the counters help to monitor the rate (as in the cache eviction policy: least frequently used, or LFU). By choosing a small sampling probability, we can observe rate bursts on a time scale of tens of minutes, i.e., bursts caused by long flows, and make counter updates and evictions infrequent. On the other hand, we can choose a large sampling probability, and focus on bursts on a time scale of micro-seconds.

The rest of the paper is organized as follows. The basic version of ElephantTrap is presented in Section 2, followed by an analysis in Section 3. Some variations of the basic algorithm are presented in Section 4. Section 5 presents simulation results and Section 6 concludes the paper.

2 Basic Algorithm

In this section, we describe the basic version of the algorithm. This version is not optimized in terms of implementation cost, but it includes the fundamental idea of sampling and eviction. In short, ElephantTrap samples for size and holds for rate.

The basic algorithm uses a cache that can be built in an SRAM or a CAM. Each line of the cache includes a flow ID (e.g. source-destination pair) and a counter. The cache is also equipped with an eviction pointer, which records the next location to check for eviction. The structure is shown in Figure 1.

There are four operations in the cache: insertion, eviction, incrementing counters (hit) and aging. All operations can be based on packets as well as bytes. For the latter, one simply needs to make the sampling probability proportional to the number of bytes in the packet, and increment the counter by number of bytes or byte-chunks. For clarity, we will assume that all units are in packets in the following sections.

Insertion: When flows arrive at the network node, each incoming packet is sampled with probability p , whose value is a design parameter. If the flow is not already present in the cache, the algorithm will insert the flow ID into the cache if there is space, or will seek to evict an existing flow in the cache, and assign the emptied line to the new flow.

Eviction: When there is no space in the cache for a

Cache of Flow ID	counter	
Flow ID 101	1	← EVICTION POINTER
Flow ID 2	23	
Flow ID 30	98	
Flow ID 41	34	
Flow ID 5	0	
Flow ID 6	0	
Flow ID 17	21	
...		
Flow ID 29	3	
Flow ID 38	2	

Figure 1. Cache structure and eviction pointer for ElephantTrap algorithm.

new flow, the eviction pointer goes through the flows in the cache, and checks if their counters are below a preset threshold. Once such a flow is identified, it is “evicted” and its line is assigned to the new flow.

Hit: When a packet is sampled, the flow cache is consulted to determine if the flow to which the packet belongs is already present in the cache. If the flow is present, the counter in the same line as the flow ID is incremented; this event is called a “hit.” An “elephant” flow can be reported when the counter value exceeds a certain threshold.

Aging: As the eviction pointer goes through the flows to search for one below threshold, it halves the counts that are above threshold before moving to the next line. The exponential decrement is mainly to combat the effect of flows arriving and departing. In a nutshell, when a long flow terminates, we want to quickly bring down its large accumulated counts in order to empty space for new arrivals; on the other hand, when a new flow arrives, we want to provide enough time for it to accumulate counts.

A precise description of the algorithm is contained in Exhibit 1.

3 Analysis

In this section, we consider the scenario of infinitely long-lived flows to gain some insight into the algorithm, and the relationship between the parameters. The consideration of infinitely long-lived flows allows us to focus on the “rate” aspect of ElephantTrap. We will look at the more realistic scenario of finite-sized flows in subsection 3.1.

Assume that there are a total of N infinitely long-lived flows, each occupying a fraction s_i ($1 \leq i \leq N$) of the

Exhibit 1 Basic Algorithm

```

1: Get next packet  $\leftarrow a$ 
2: Generate a random number,  $r \in (0, 1)$ 
3: if  $r < p$ , then
4:   Search cache for flow ID of packet  $a$ 
5:   if cache.line[ $i$ ] ==  $a$ .flowID then
6:     increment cache.counter[ $i$ ]
7:     if cache.counter[ $i$ ] > report-threshold then
8:       report the flow ID
9:     endif
10:  else
11:    if there is empty line in cache then
12:      insert  $a$ .flowID into the line
13:      initialize counter to 0
14:    else
15:      while (counter pointed to by eviction pointer
16:         $\geq$  threshold) or (eviction pointer has not gone
17:        through entire cache)
18:        current counter is divided by 2
19:        eviction pointer ++
20:      endwhile
21:      if (counter pointed to by eviction pointer
22:        < threshold)
23:        then
24:          evict current entry
25:          insert  $a$ .flowID into current line
26:          initialize counter to 0
27:        endif
28:      endif
29:    endif
30:  endif
31: endif
32: go to line 1

```

total bandwidth C in packets. Further assume the flow rate distribution follows a power-law according to

$$\mathbb{P}(s_i C > x) = Ax^{-\alpha}, \alpha > 0, x \geq 1.$$

The power law distribution of flow rates was first observed in [4]. Similar observations have been made in [5] using traces from a variety of locations including a 100Mbps link at a national laboratory, a T3 peering link between two providers, and a FDDI ring connecting a modem bank at a dial-up provider to a backbone network.

Let S be the cache size in number of entries, p be the sampling probability, and t be the time taken for the eviction pointer to go through the entire cache. Time is measured in number of packet arrivals.

Note that the rate at which the packets of a flow enter the cache is $ps_i C$. The flow will remain in the cache if it manages to accumulate a count no less than the threshold value within time t . For this analysis, we set the threshold

value to 1. Hence a flow will remain in the cache if

$$ps_i C t \geq 1.$$

Define the functions

$$f(t) = |\{s_i : ps_i C t \geq 1\}| \text{ and}$$

$$g(t) = \sum_i s_i C \mathbb{I}(ps_i C t < 1),$$

where \mathbb{I} is the indicator function evaluating to 1 when $ps_i C t < 1$ and 0 otherwise. $f(t)$ is the number of flows that remain in the cache and $g(t)$ is the sum rate of the flows that are slow; i.e., the flows whose rates are insufficient to keep them in the cache; as a result, they keep entering and leaving the cache.

Note that with infinitely long-lived flows, if the rate of a flow is above the threshold, it will always remain in the cache, while if the rate is below the threshold, it will never accumulate a hit before the eviction pointer returns. In other words, the threshold draws a *hard* line between the “fast” and “slow” flows. Hence when a slow flow enters the cache, which happens with rate $pg(t)$, a hit never happens and it always needs to evict another slow flow. Since at any time, there are $S - f(t)$ lines occupied by slow flows, we have

$$t = \frac{S - f(t)}{pg(t)}. \quad (1)$$

Let N be the total number of flows. We will compute $f(t)$ and $g(t)$ as follows:

$$f(t) = N\mathbb{P}(s_i C > \frac{1}{pt})$$

$$= NA(pt)^\alpha,$$

and

$$g(t) = \mathbb{E}(s_i C | ps_i C t < 1) N\mathbb{P}(ps_i C t < 1)$$

$$= N \int_1^{\frac{1}{pt}} \mathbb{P}(x < s_i C < \frac{1}{pt}) dx$$

$$= N \int_1^{\frac{1}{pt}} Ax^{-\alpha} - A(pt)^\alpha dx$$

$$= \frac{AN}{\alpha - 1} (1 - \alpha(pt)^{\alpha-1}).$$

We obtain the following relation from equation (1)

$$t = \frac{(S - NA(pt)^\alpha)(\alpha - 1)}{pAN(1 - \alpha(pt)^{\alpha-1})},$$

and with simple algebraic manipulation, we get

$$AN(pt)^\alpha - AN(pt) + S(\alpha - 1) = 0.$$

When the rate distribution and total number of flows are fixed, choosing the cache size S determines the product pt ,

and hence the number of flows that will remain in the cache $f(t) = NA(pt)^\alpha$.

It is interesting to note that the choice of the sampling probability p does not affect the functioning of Elephant-Trap in the case of infinitely long-lived flows. The variable t will adjust itself for different values of p such that the same high-rate flows will be captured in the trap. Also, the actual value of C does not come into the equation. Only the traffic distribution α matters. In fact, the consumed bandwidth can vary over time, and the same high-rate flows will be captured so long as the flow distribution remains the same.

3.1 Real Network Scenarios

In the case of infinitely long-lived flows, each flow will be sampled eventually regardless of the value of p . However, this is not the case in real networks where flows are of finite length. Hence the value of p determines a sampled sub-population of flows which will enter the ElephantTrap. Another way to view this is in terms of the size of a time window. The smaller p is, the larger the average size of the sub-population, and correspondingly, we are searching for “bursts” or “high-rate” flows over a large time-scale, where the large-size flows reside. A larger p , on the other hand, focuses on a much smaller time window, e.g., micro-seconds, and identifies bursts at this scale. Although a large p seems to include almost all flows in consideration, within each small time-window, the number of active flows is small.

We can extend our understanding of the infinitely long-lived flows to the real network scenario by letting the number of active flows, N , and the current rate distribution, $Ax^{-\alpha}$, vary with time. The following figures plot the number of captured “high-rate” flows $f(t)$ against N and α respectively. The line labelled “var 2” will be explained in Section 4. The cache size S is fixed to 64 for this plot.

Figure 2 shows that for a fixed rate distribution, the number of flows that remain in the cache decreases as N increases. This is because more slow flows are entering and leaving the cache, making the turnaround faster and hence the rate threshold higher.

Correspondingly, Figure 3 shows that the more heavy-tailed the rate distribution is (in the direction of α decreasing), the larger number of flows remain in the cache. This is due to both the abundance of fast flows, and the lack of interference from the slow flows entering and leaving the cache.

4 Variations of Algorithms

In hardware implementations, tossing a coin on every incoming packet can be too costly, hence we propose the first variation of the algorithm:

Toss a coin on one in ten incoming packets.

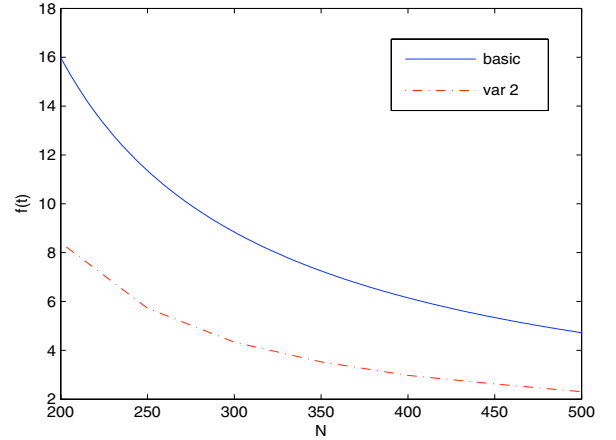


Figure 2. Plot of the number of captured flows against the number of active flows in a time window. The flow rate distribution is fixed to $2x^{-2}$.

Since the sampling probability can be adjusted ($\times 10$) for this variation such that it is the same as in the basic version, the analysis remains the same.

In the basic algorithm, the eviction pointer goes through the entire cache until a counter below threshold is found. This operation is costly and also varies in amount of time required, making pipelining difficult. We propose the second variation of the algorithm:

Check at most 2 entries before eviction.

With this variation, if no flow below threshold is found in the next two entries, the new flow is discarded. This process with infinitely long-lived flows is governed by the following equation:

$$t = \frac{S}{\left(\frac{q(t)}{2} + 2\left(1 - \frac{q(t)}{2}\right)\right)pg(t)},$$

where

$$q(t) = \frac{S - f(t)}{S}.$$

We assume that with equal chance the eviction pointer will encounter a flow below threshold at the first step and at the second step, conditioned on the fact that such a flow is found.

With $f(t)$ and $g(t)$ as defined in Section 3, we obtain the following equation:

$$AN(pt)(3S + AN(pt)^\alpha)(1 - \alpha(pt)^{\alpha-1}) = 2S^2(\alpha - 1).$$

The number of flows captured with N or α varying for variation 2 is also plotted in Figures 2 and 3. From the

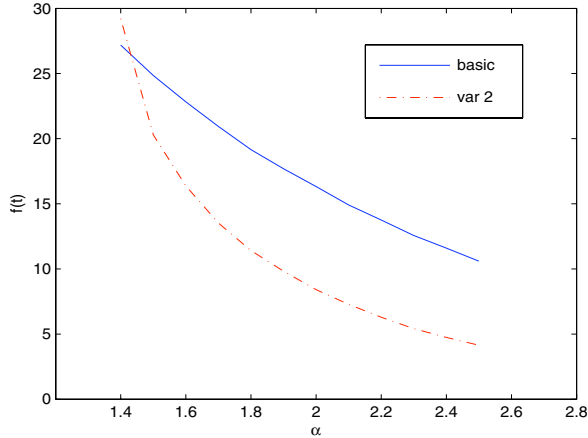


Figure 3. Plot of the number of captured flows against the heaviness of the tail. The number of sampled flows is fixed to 150.

comparison, we see that variation 2 is catching fewer top talkers than the basic version when the tail is light or when too many flows are sampled, a tradeoff for the reduction in complexity. The direction of change with respect to N and α remains the same.

5 Simulation

We run simulations on a real Internet trace collected by the NLANR MNA team on Aug 14, 2002. It consists of ten-minute contiguous bidirectional packet header traces collected at the Indianapolis router node (IPLS). It contains 4.96 million packets and a total of 161367 flows.

Due to space constraint, we will demonstrate the use of ElephantTrap on a large time scale, i.e., to catch long and fast flows, and compare the performance of the three variations. We choose the cache size S to be 32 entries since this is a very small cost to the implementors. In practice, we do not have any notion of the instantaneous rate or size distribution. Hence we choose the sampling probability p according to the following rule of thumb for the three variations of the algorithm.

Rule of Thumb

We introduce a further parameter L , which is the guessed top talker length. The idea is as follows. Since we do not know the exact distribution of flows in the network, we *guess* the size of the flow who is consuming a significant part of the bandwidth. And we compute p based on L as follows:

Basic:

$$p = \frac{2}{L}$$

Var 1:

$$p = \frac{20}{L}$$

Var 2:

$$p = \frac{20S}{8L} = \frac{5S}{2L}$$

The simulation results for different guesses of top talker size are plotted in Figs. 4 to 7.

For all figures except Fig. 4, we plot the cache size (constant line fixed at 32), the actual “big” flows (whose sizes are above the *guessed* top talker size) and the cached “big” flows. Since some of the actual “big” flows might not pass the instantaneous rate screening, the line for actual “big” flows is always above the cached “big” flows.

When we guess the top talker size to be 10000 packets, all the “big” flows are captured in the cache soon after they start and trapped thereafter, regardless of the variation of the algorithm we use. Only one plot of captured “big” flows is shown since all three are indistinguishable.

As the guess of top-talker size decreases and the sampling probability increases correspondingly, we are simulating the scenario when we do not have a good knowledge of sizes of flows in the network and decide to cast our net wide. As discussed in Section 3.1, this is equivalent to focusing our attention on a smaller time window, and hence is not helpful for identifying the real top-talkers, who live on a much larger time scale. Not surprisingly, as we compare the actual number of flows above the guessed top-talker size and the number captured, the latter is only a fraction of the former. The situation becomes the worst in Fig. 7, when the guessed top-talker size is 2000 packets.

However, even when only a small portion of flows with size above 2000 packets are cached, this portion proves to be the *real* top-talkers, i.e., flows who occupy the most significant portion of the bandwidth. Table 1 shows the counter values for the ten biggest flows from variation 2. A high counter value indicates that the corresponding flow is trapped in the cache for a long time.

In summary, ElephantTrap is able to capture and track the top-talkers in the Internet trace over the span of 5 million packet arrivals. As forecasted by analysis, variation 2 sets the rate threshold higher for the same cache size and sampling probability, hence captures fewer flows than the other two versions of the algorithm. Regardless of the guessed top-talker size, the small number of real top-talkers are always captured by ElephantTrap.

flow ID	total packets	hits
1	13750	777
2	13489	665
3	12082	577
4	10929	432
5	10621	257
6	9564	197
7	9533	181
8	8594	161
9	8551	137
10	8445	283

Table 1. Counter values for the ten biggest flows when the guessed top-talker size is 2000 packets.

6 Conclusion

In this paper, we presented ElephantTrap, an easily implementable and low-cost device for identifying large flows in the network. With a small cache size, a very low sampling rate, and a readily pipelined simple eviction scheme, it is able to identify the flows that occupy the most significant portion of the bandwidth. ElephantTrap samples for size and holds for rate. Since its definition of the rate threshold is implicit, no actual estimates of the flow rates need to be made. By tuning the sampling probability and aging rate, it can also be extended to track the bursts caused by mid-sized flows on a much smaller time scale.

References

- [1] M. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, 1997.
- [2] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a better netflow. *ACM SIGCOMM*, 2004.
- [3] C. Estan and G. Varghese. New directions in traffic measurement and accounting. *Proceedings of the 2001 ACM SIGCOMM Internet Measurement Workshop*, Nov, 2001.
- [4] R. Mahajan and S. Floyd. Controlling high-bandwidth flows at the congested router. *Proceedings of ICNP*, November 2001.
- [5] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker. Approximate fairness through differential dropping. *ACM Computer Communication Review*, January 2002.
- [6] S. Ramabhadran and G. Varghese. Efficient implementation of a statistics counter architecture. *ACM Sigmetrics*, 2003.
- [7] D. Shah, S. Iyer, B. Prabhakar, and N. McKeown. Analysis of a statistics counter architecture. *Hot Interconnects 9*, 2001.
- [8] C. Systems. Netflow. <http://www.cisco.com/warp/public/732/Tech/netflow>.
- [9] C. Systems. Netflow mib and top talkers. http://www.cisco.com/en/US/products/sw/iosswrel/ps1838/products_feature_guide09186a0080259533.html#wp1131383.

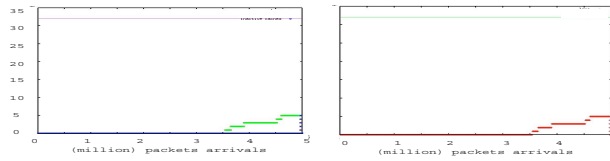


Figure 4. Left: Plot of actual big flows against packet arrivals. Right: Plot of captured big flows against packet arrivals. Gussed top talker size = 10000 packets. All flows with sizes more than 10000 packets are trapped.

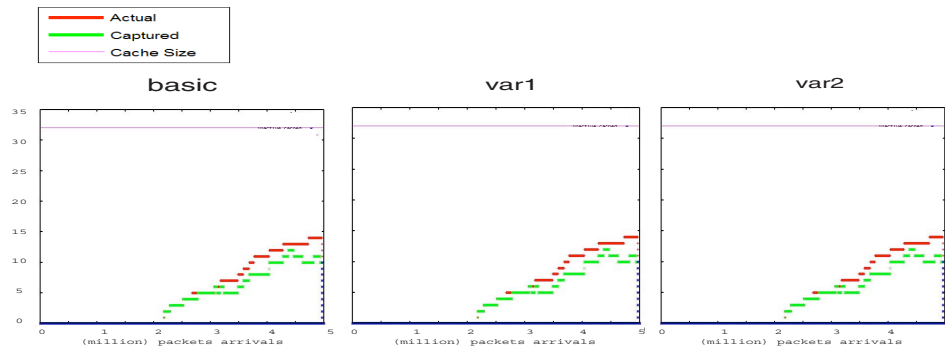


Figure 5. Plots of actual and captured big flows against packet arrivals. Gussed top talker size = 6000 packets. Most flows with sizes more than 6000 packets are trapped.

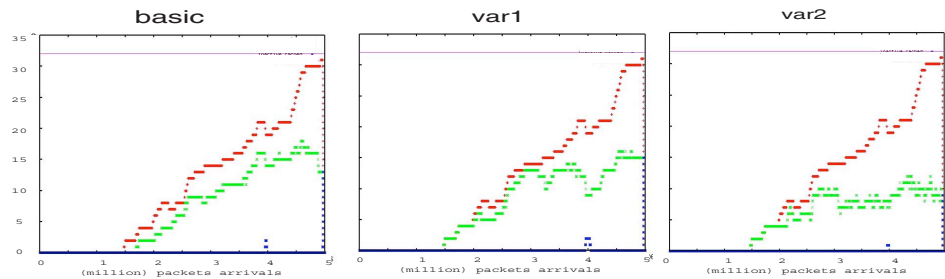


Figure 6. Plots of actual and captured big flows against packet arrivals. Gussed top talker size = 4000 packets. The majority of flows with sizes more than 4000 packets are trapped.

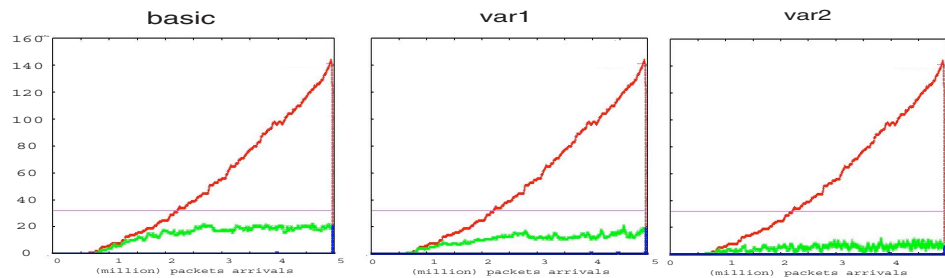


Figure 7. Plots of actual and captured big flows against packet arrivals. Gussed top talker size = 2000 packets. Only a fraction of flows with sizes more than 2000 packets are trapped.