Synthetic Trace Generation for the Internet

W. Shi M. H. MacGregor P. Gburzynski

Department of Computing Science
University of Alberta
Edmonton, AB, T6G 2E8, Canada
{wgshi, macg, pawel}@cs.ualberta.ca

Abstract

We consider the distribution of destination addresses in IP packets arriving at an Internet router and show that the spatial locality of those addresses is well characterized by an empirical power law function. We demonstrate how the LRU stack model implied by this function can be used to generate synthetic IP traffic, e.g., for experimental studies of routing and caching protocols. We also show how this model (which was originally devised to generate synthetic memory reference strings of programs) can be modified to better capture the temporal locality of destination addresses in aggregate IP traffic. Our observations are illustrated by comparing the footprints of real and synthetic traces, and by simulating routing table lookups driven by both types of traces.

1 Introduction

It is well established that memory reference strings of computer programs exhibit spatial and temporal locality [1]. This locality is the motivating concept behind the use of instruction and data caches, which play a critical role in improving the performance of contemporary computer systems. In this paper, we apply the concepts of locality and workload modeling originally developed for investigating program memory references to the characterization of traces of Internet traffic.

Internet routers use locally-stored routing tables to look up the correct outgoing interface for each incoming IP packet, based on the best match for the destination address extracted from its header. In this sense, the destination address is the index to the routing table, just as a virtual memory address is an index to the page directories and page tables. The sequence of destination addresses in the IP packet trace constituting the input to this lookup process exhibits both temporal and spatial locality. This similarity has been exploited, e.g., in [2], to accelerate routing table lookups by harnessing for this purpose the caching hardware used by virtual address translation.

To some extent, the IP addresses in a packet trace can be viewed as 32-bit virtual addresses representing memory references issued by a hypothetical program during its execution. If we could characterize the locality patterns exhibited by such reference strings with simple formulas, we could generate synthetic traces mimicking the relevant statistical properties of real traces. This would facilitate and simplify the performance studies of routing algorithms (especially caching algorithms) by providing a low-cost experimental testbed for their evaluation.

The remainder of this paper is organized as follows. In Section 2, we give a brief review of the previous relevant work on characterizing the behavior of programs and traffic patterns with respect to their locality. Section 3 briefly discusses the model used to generate synthetic reference traces. In Section 4, we demonstrate that the original model well captures the spatial locality of IP traces, and show how it can be modified to account for the temporal locality patterns observed in real traces. Section 5 summarizes our work.

2 Previous work

Previous work on program behaviors is abundant. From the viewpoint of generating synthetic memory reference strings, the model of Thiebaut, Wolf and Stone [3] is relatively simple, computationally

inexpensive, and well matches the relevant properties of real reference strings, e.g., in terms of the cache miss ratio. With this model, the following hyperbolic function:

$$u(n) = An^{1/\theta} \quad (1)$$

is used to describe the number of unique references $u(n)$ observed at reference number n, where A is a program-dependent constant and θ gauges the spatial locality of the trace. The larger the value of θ, the more spatial locality is present in the trace.

There have been numerous efforts to model and characterize computer network traffic, with the stress on temporal statistics of packet arrival in both single-source and aggregate environments. It was shown that data traffic differs from voice traffic [4] in that it exhibits a high temporal variability but is autocorrelated in the long run. Fractal models are often more appropriate than Poisson models to describe the arrival of data traffic.

Jain [5] proposed a "packet train" model as being appropriate for the LAN environment, where the traffic was shown to drastically depart form the traditional Poisson arrival model. In the train model, packets between a pair of nodes form a series of trains, with every train consisting of a number of "cars." The basic parameter in this model is the Maximum Allowed Inter-car Gap (MAIG).

Mogul [6] studied locality at a per-peer level by passively monitoring LAN traffic. Persistence [7], rather than temporal locality, was used to characterize the tendency of packets to arrive for the same destination process. It has been shown that in as many as 66 percent of all cases, an incoming packet is intended for the process that sent the previous packet. The work by Claffy, Braun and Polyzos [8] identifies "flows" in the traffic stream based on various temporal and spatial locality conditions. The flow profiling in their work was motivated by Jain's packet train model. The parameters of a flow include a *timeout* value and a flow specification. The *timeout* value is used to determine whether a flow is still active.

Those previous efforts attempt to characterize network traffic, and often try to exploit its locality, particularly temporal locality, to improve system performance. In contrast, our goal is to develop a method for generating synthetic network traces fitting some of the spatial and temporal locality patterns present in real aggregate Internet traces.

3 Synthetic trace generation

Synthetic traces are desirable for many reasons. Firstly, real traces are often difficult to obtain, mostly because of security or privacy concerns. For this reason, publicly available collections of traces are often "sanitized" by hiding the real source and destination addresses of all packets. Although such sanitized traces may still be useful for many studies (e.g., dealing with the interarrival time or length distribution of packets), they are completely useless in those cases when the actual IP addresses are needed, e.g., to investigate the behavior of caching algorithms.

Secondly, with the increasing speed of Internet routers, it becomes more and more expensive to collect and store full traces of a meaningful duration without affecting the router's performance. The situation begins to resemble the problems of collecting memory reference strings of programs in execution. The complete string of a sizable CPU bound program cannot be collected and stored in real time without drastically impairing its execution time.

Thirdly, Internet traffic patterns are likely to undergo significant changes due to new applications and user activity patterns that cannot be anticipated at present. For example, Web transactions (which are considerably idiosyncratic) became one of the major components of the Internet traffic almost overnight. Such changes will render real trace collections obsolete.

Finally, real traces are not very convenient from the viewpoint of performance studies because they are not parameterizable. Given two different real traces, it is difficult to say in a precise manner how much they actually differ. It is even more difficult to expose a new protocol or mechanism to a graded change in the nature of the traffic. Although some properties of the offered load may seem easy to change (e.g., the arrival intensity), such changes must often be correlated with other changes (possibly quite difficult to conceive) to result in a realistic and meaningful trace. Consequently, the extrapolation of real traces to new conditions is practically impossible.

Synthetic trace generation is appealing because it is fast, easy, and controllable. The key issue is the fidelity of a synthetic trace, i.e., how well it mimics the relevant statistical properties of a real trace. The exact nature of those properties depends on the intended study. For example, the relevant

property from the viewpoint of caching is locality.

Thiebaut, Wolf and Stone [3] have developed an LRU stack model to generate representative synthetic traces of program memory references. The stack is initialized with a set of unique addresses. Given the values of A and θ in Formula 1, a random offset into the stack can be generated. This offset is called a hit index. The address at the hit index is moved to the top of the stack, and this address is output as the next address in the synthetic trace. A pointer, called LRU_POINTER, keeps track of the size of the working set of addresses. LRU_POINTER is initially zero, and it is incremented by one whenever a hit index is generated that is larger than the current value of LRU_POINTER. Owing to the hyperbolic nature of the footprint function, the likelihood of updating the LRU_POINTER decreases very rapidly with the increasing number of memory references.

Notably, our method of deriving the parameters of the model differs from that used in [3], where the authors visually estimate A and θ from the hyperbolic miss rate curve. Given the large number of samples in our traces, we were able to to calculate the model parameters by linear regression.

4 Simulation results

We compare synthetic traces generated by the infinite cache model described in [3] to those generated by selecting destination addresses at random from a set of unique addresses. The fidelity of these synthetic traces is assessed by comparing their footprints, as defined by Formula 1, to those of real traces.

The real traces used in our experiments were obtained at the University of Alberta (UofA) and from the Finnish University and Research Network (FUNET). The set of addresses for the random trace generator is the set of unique destination IP addresses in the real trace to which the random trace is being compared.

The values of θ calculated from the two real traces are 2.239 (UofA) and 2.631 (FUNET). The values for A are 11.95 and 23.25 respectively.

Figures 1 and 2 show that the synthetic traces approximate the real traces much more closely than the random traces. In particular, in both figures, the number of unique addresses reaches the size of the total address space far more quickly for the randomly generated trace than for either the synthetic or the real trace.

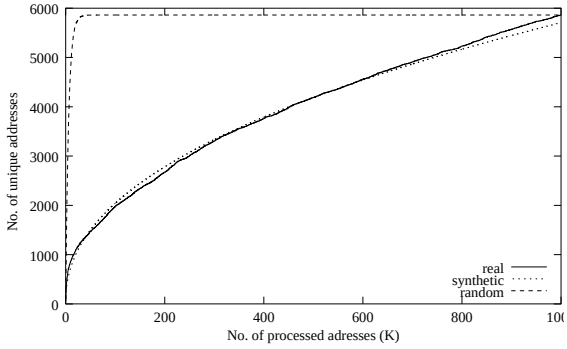

Figure 1: UofA traces

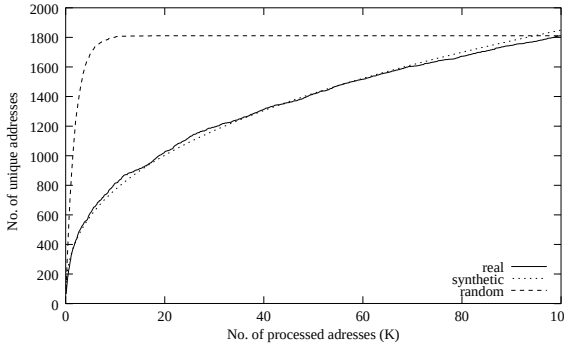

Figure 2: FUNET traces

Cache miss ratio is one of the most important parameters in characterizing the performance of a memory reference system. In our experiments, we have used both synthetic and real traces to drive a routing table lookup algorithm comparing the observed miss ratios for different caching policies. Note that in this simulation study we are concerned with the data cache only. This makes sense from the viewpoint of IP address caching at a router, where the program itself is short and can be entirely stored in any reasonable instruction cache, while the implementation of the data structures for the address cache is a primary concern.

The routing table lookup code was extracted from the FreeBSD [9] 4.3-Release kernel, which uses a radix tree [10] data structure to organize the routing table. The routing table itself was obtained from the same University of Alberta router at which the real UofA trace was collected. The simulation tool used was SimpleScalar[11].

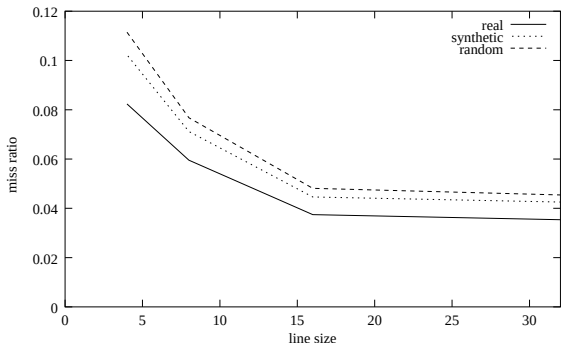

Figure 3: Simulation of a 16K direct-mapped cache

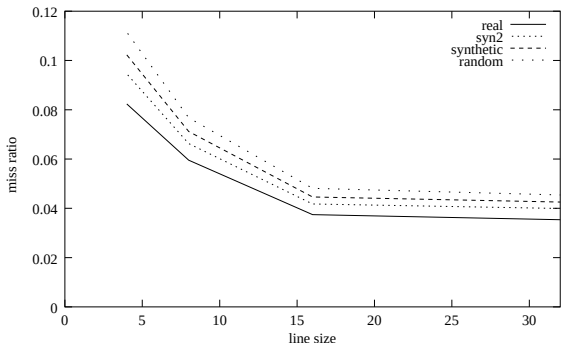

Figure 5: Comparison of miss ratio

Figure 3 shows the result for a 16KB direct-mapped cache with cache line sizes of 4, 8, 16 and 32 bytes. The miss ratios of the real trace are consistently lower than those of both the random trace and the synthetic trace. The miss ratios of the synthetic trace are between those of the real trace and those of the random trace. Notably, the synthetic trace curve is closer to that for the random trace than it is to the real trace curve.

This discrepancy can be explained by the fact that the synthetic trace underestimates the temporal locality of the real trace, i.e., there is more temporal locality in the real trace than in the synthetic trace. In other words, while the synthetic trace generator well approximates the spatial locality of the real trace, it fails to capture the temporal locality. To test this hypothesis, we calculated for each trace a crude measure of its temporal locality, namely, the number of times when a destination address was the same as the previous address. This situation occurs 147272 times in the UofA trace, 155 times in the random trace and never (!) in the synthetic trace. The results for the FUNET traces were very similar.

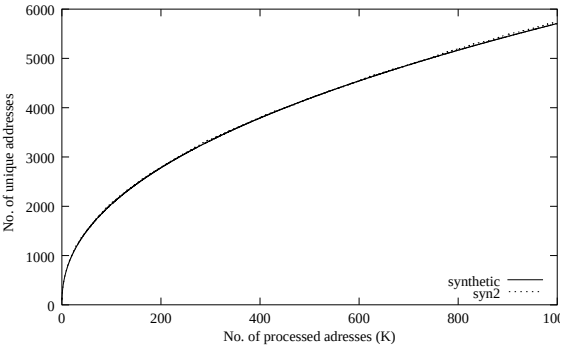

Figure 4: Footprints of two synthetic traces

Temporal locality means that once an address has appeared in the reference string, it is highly possible that the same address will be seen again in the near future. As the synthetic trace generator is only concerned with making sure that the total number of different addresses in the reference string obeys a certain power law (i.e., the spatial locality), it does not explicitly account for the temporal locality, i.e., relative closeness of identical addresses in the string.

To remedy this problem, we modified the trace generator by introducing one more parameter, the so-called *self-repeat ratio*. If the hit index is below the LRU_POINTER, i.e., the address to be generated has been seen already, that address is made identical to the previous one with the probability equal to the self-repeat ratio. Otherwise, with the remaining probability, the address at the hit index is issued—as before. The modified algorithm is listed in Figure 6. The four lines starting at "R = Random2" are our modification to the original generator.

It is natural to set the self-repeat ratio for the generator to the measured value of this ratio in the real trace which the generator is meant to approximate. In particular, the self-repeat ratio observed in the UofA trace is 0.147272. Note that with this modification, the generator will tend to issue not only repeating pairs of addresses, but also runs of three, four, and so on. The results for this modified address generation algorithm are shown in Figures 4 and 5.

Figure 4 presents the footprint of the trace generated with the modified algorithm (syn2). Notably, it matches that of the synthetic trace from the original method. This is because we only consider generating repeating addresses when the hit index is below LRU_POINTER. Thus, in terms of the total

```
{--- fill LRU stack with unique items ---}
InitializeStack;

{--- generate "SyntheticTraceLength" synthetic addresses ---}

for count:= 1 to SyntheticTraceLength do
      begin
      {--- generate index from uniform random real in [0, 1] ---}
       U:=Random1;
       if (U<1/Theta) then
            index := round((U/((A^Theta)/Theta)))^(1/(1-Theta)));
       else
            index := round(random * Cc);

      {--- adjust ---}
      if index < 1 then
            index := 1;

    {--- determine the address to output ---}
    R = Random2
    if index<LRU_POINTER and R<SelfRepeatRate then
        address = PreviousAddress;
    else
        {--- move item at "index" to front of stack. ---}
        UpdateLRUStack(index, address, LRU);

      {-- process new synthetic address --}
      UerProcess (address );
      end; {for}
```

Figure 6: The synthetic trace generation algorithm

number of different addresses that have been issued up to a given point, the two methods produce exactly the same outcome. This also means that the spatial locality of the strings produced by them is identical.

Figure 5 shows that the miss ratios of the trace from the modified algorithm (syn2) are closer to those of the real trace (real) than those from the original algorithm (synthetic). This is due to the improvement in temporal locality in the synthetic trace.

5 Conclusions

We have demonstrated that the results of previous studies on locality of memory references can be applied to locality models of network traffic. By introducing a modest modification to a generator of memory reference strings based on a simple power law, we have turned it into a generator of synthetic IP traces at a router. The synthetic traces appear to well capture the locality aspects of real traces with similar measured parameters.

Additional parameters are still needed for characterizing other aspects of the traffic, so that the synthesized traces can be controlled as needed for specific studies. For example, a parameter reflecting the degree of temporal locality may be needed. Other researchers [8] have suggested additional parameters.

6 Acknowledgment

We would like to thank Dr. Thiebaut for discussions about the method used to determine the values of A and θ in [3].

References

[1] P. J. Denning. Working set model for program behavior. *Communications of the ACM*, 11(5):323–333, 1968.

[2] T. Chiueh, P. Pradhan. High performance ip routing table lookup using cpu caching. In *Proceedings of IEEE INFOCOM'99*, 1999.

[3] D. Thiebaut, J. L. Wolf , H. S. Stone. Synthetic traces for trace-driven simulation of cache memories. *IEEE Transactions on Computers*, 41(4):388–410, 1992.

[4] W. Willinger, V. Paxson. Where mathematics meets the internet. *Notices of the American Mathematical Society*, 45(8):961–970, 1998.

[5] R. Jain, S. Routhier. Packet trains-measurements and a new model for computer network traffic. *IEEE Journal of Selected Areas in Communications*, SAC-4(6):986–995, September 1986.

[6] J. Mogul. Network locality at the scale of processes. In *Proc. ACM SIGCOMM '91*, pages 273–285, 1991.

[7] C. L. Williamson. Dynamic transport-level connection management in a distributed system. pages 315–322, 1989.

[8] K. C. Claffy, H. W. Braun, G. C. Polyzos. A parameterizable methodology for internet traffic flow profiling. *IEEE Journal of Selected Areas in Communications*, 13(8):1481–1494, 1995.

[9] The FreeBSD Project. http://www.freebsd.org.

[10] K. Sklower. A tree-based packet routing table for berkeley unix. In *USENIX Winter '91*, Dallas, TX, USA., 1991.

[11] SimpleScalar: Simulation Tools for Microprocessor and System Evaluation. http://www.simplescalar.org/.