Trajectory Sampling for Direct Traffic Observation

N. G. Duffield M. Grossglauser

AT&T Labs - Research
180 Park Ave, Florham Park NJ 07932, USA
{duffield, mgross}@research.att.com

ABSTRACT

Traffic measurement is a critical component for the control and engineering of communication networks. We argue that traffic measurement should make it possible to obtain the spatial flow of traffic through the domain, i.e., the paths followed by packets between any ingress and egress point of the domain. Most resource allocation and capacity planning tasks can benefit from such information. Also, traffic measurements should be obtained without a routing model and without knowledge of network state. This allows the traffic measurement process to be resilient to network failures and state uncertainty.

We propose a method that allows the direct inference of traffic flows through a domain by observing the trajectories of a subset of all packets traversing the network. The key advantages of the method are that (i) it does not rely on routing state, (ii) its implementation cost is small, and (iii) the measurement reporting traffic is modest and can be controlled precisely. The key idea of the method is to sample packets based on a hash function computed over the packet content. Using the same hash function will yield the same sample set of packets in the entire domain, and enables us to reconstruct packet trajectories.

1. Introduction

The efficiency of resource allocation and the quality of service provided by IP networks depends critically on effective traffic management. Traffic management consists of short-term *traffic control* and longer-term *traffic engineering*. Traffic control operates on a time-scale of seconds and without direct human intervention. Examples of traffic control functions include congestion control, automatic recovery in case of link or router failures, or admission control. Traffic engineering operates on time-scales from minutes to weeks or months, and typically with some degree of human intervention. Its goal is to optimally allocate network resources, such as link capacity, to different classes of network traffic in order to ensure good service quality and high network efficiency. Examples of traffic engineering functions include traffic characterization (e.g., trending), accounting (e.g., for pricing), and capacity planning and provisioning.

All of these functions represent feedback loops on a wide range of time-scales and of varying spatial extent, and *traffic observation* or *measurement* is therefore an integral component of these functions. The importance of traffic measurement capabilities is compounded by the fact that IP networks do not maintain per-flow state. By contrast, in circuit-switched networks, the traffic is essentially "observable for free", because per-call state exists along each node on the call's path. In a sense, the scalability of the stateless IP networks has been bought at the expense of observability.

Virtually all traffic engineering functions, such as route optimization or planning of failover strategies, rely on an understanding of the spatial flow of traffic through the domain. For example, suppose we observe that some link in the backbone is overloaded. Appropriate corrective action requires an understanding of which ingress points the traffic observed on this link originates and where it is headed, what customers are affected by the congestion, and what the traffic mix is; without this information, effective remedies (e.g., rerouting of part of that traffic) cannot be taken [11]. Also, it should be possible to infer what fraction of traffic entering the measurement domain at a certain ingress point traverses each link in the network, for example to focus on how the traffic of a specific customer flows through the domain, and to diagnose which link might be the reason for a performance problem experienced by that customer. Domain-wide spatial traffic information is also a prerequisite for the establishment of label-switched tunnels [3], or to decide which potential ingress point is best to connect a new customer to the domain.

We distinguish between direct and indirect measurement methods. Conceptually, an indirect measurement method relies on a network model and network status information to infer the *spatial flow of traffic* through the domain. For example, suppose that the traffic is observed only at network ingress points (e.g., by computing statistics on the distribution of source-destination pairs). In order to infer how that traffic flows through the domain, timely and accurate information about the state of the routing protocol and link states has to be available. If assumptions about traffic routing have to be made in order to obtain the traffic flow matrix, then the use of an outdated routing table can lead to erroneous inferences, and suboptimal allocation of network resources.

More generally, indirect measurement methods suffer from the uncertainty associated with the physical and logical state of a large, heterogeneous network [11]. This uncertainty has several sources. First, the exact behavior of a network element, such as a router, is not exactly known to the service provider and depends on vendor-specific design choices. For example, the algorithm for traffic splitting among several shortest paths in OSPF is not standardized. Second, there are deliberate sources of randomness in the network to avoid accidental synchronization, e.g., through active queue management disciplines [12] or randomized timers in routing protocols [13]. Third, some of the behavior of the network depends on events outside of the control of the domain; for example, how traffic is routed within an autonomous system (AS) depends in part on the dynamics of route advertisement to this AS by neighboring domains [18]. Fourth, the interaction between adaptive schemes operating at different time-scales and levels of locality (e.g., QoS routing, end-to-end congestion control) may simply be too complex to characterize and predict [26]. Finally, with increasing size and complexity, the likelihood increases for faults and misconfigurations to disrupt the normal operation of the network. Often, traffic measurement is one of the potential tools to detect and diagnose such problems; however, this benefit is mitigated if traffic measurement requires correct network operation.

A *direct* method does not rely on a network model and an estimation of its state and its expected behavior. Rather, it relies on direct observation of traffic at multiple points in the network. As such, it does not suffer from the sources of uncertainty discussed above. In this paper, we describe a direct method for traffic measurement, called trajectory sampling. The method samples packets that traverse each link (or a subset of these links) within a measurement domain. The subset of sampled packets over a certain period of time can then be used as a representative of the overall traffic.

If packets were simply randomly sampled at each link, then we would be unable to derive the precise path that a sampled packet has followed through the domain from the ingress to the egress point. The key idea in our proposal is therefore to base the sampling decision on a deterministic hash function over the packet's *content*. If the *same* hash function is used throughout the domain to sample packets, then we are ensured that a packet is either sampled on *every* link it traverses, or on no link at all. In other words, we effectively are able to collect *trajectory samples* of a subset of packets. The choice of an appropriate hash function will obviously be crucial to ensure that this subset is not statistically biased in any way. For this, the sampling process, although a deterministic function of the packet content, has to resemble a random sampling process.

A second key ingredient of our proposal is that of *packet labeling*. Note that to obtain trajectory samples, we are not interested in the packet content per se; we simply need to know that *some packet* has traversed a set of links. But to know this, it is sufficient to obtain a unique packet identifier, or label, for each sampled packet within the domain and within a measurement period. Because the label is unique, we will know that a packet has traversed the set of links which have reported that particular label. We propose to use a second hash function to compute packet labels that are, with high probability, unique within a measurement period. While the size of the packet labels obviously depends on the specific situation, note that labels can in practice be quite small (e.g., 20 bit). As the measurement traffic that has to be collected from nodes in the domain only consists of such labels (plus some auxiliary information), the overhead to collect trajectory samples is small.

Trajectory sampling has several important advantages. It is a direct method for traffic measurement, and as such does not require any network status information. The spatial flow of traffic through the domain can be inferred from trajectory samples, i.e., paths taken by a pseudo-random subset of packets through the domain. Trajectory sampling does not require router state (e.g., per-flow cache entries) other than a small label buffer. The amount of measurement traffic necessary is modest and can be precisely controlled. Multicast packets require no special treatment - the trajectory associated with a multicast packet is simply a tree instead of a path. Finally, trajectory sampling can be implemented using state-of-the art digital signal processors (DSPs) even for the highest interface speeds available today.

This paper is structured as follows. We define notation and formally define trajectory sampling in Section 2. We discuss the choice of parameters for the hashing functions, and demonstrate their statistical properties in Section 3. We give an example of traffic measurement based on an extensive packet trace in Section 4. In Section 5, we discuss implementation issues and possible extensions of trajectory sampling. Section 6 concludes the paper.

2. Formal Description of Trajectory Sampling

For simplicity, let us describe the scheme assuming that all packets are of size S bits. We represent the measurement domain as a directed graph $G(V, E)$, where V is the set of nodes and E is the set of directed links. Packets enter the measurement domain at an *ingress node*. They traverse several links to leave the measurement domain at an *egress node* (or several egress nodes in the case of a multicast packet[1]). A packet can potentially be dropped at an intermediate node. We let $x_i(P_k)$ denote the *content* of a packet k at link i, i.e., the sequence of bits making up the IP header and the IP packet content. When there is no risk of ambiguity, e.g. when considering a stream of packets at a single link, we refer to a packet P and its content $x = x(P)$ interchangeably.

Consider all the packets $P_1, \ldots, P_N$ entering the domain within a measurement interval of length T. The *trajectory* of packet P_k is the set of links traversed by packet P_k. In the case of a unicast packet, the trajectory is a path from the ingress node to the egress node or to the node where the packet is dropped. In the case of a multicast packet, the trajectory forms a tree rooted at the ingress node.

The *invariance function* ϕ is a function of the packet content whose output depends of the invariant packet content, i.e., the bits of the packet that are not modified upon forwarding, as described below. An invariance function does not depend,

[1]Strictly speaking, several copies of a multicast packet could enter the measurement domain at multiple ingress nodes; for our purposes, we can simply consider each copy of the multicast packet entering the domain as an independent packet.

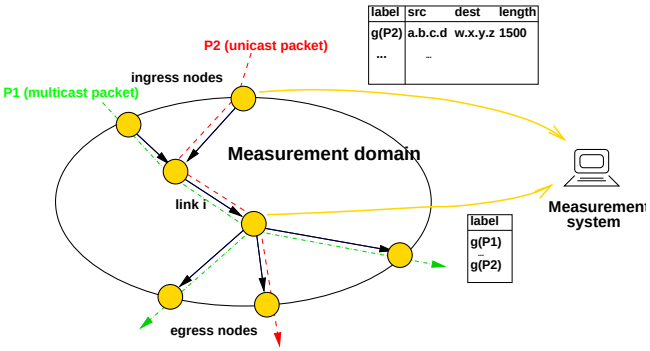

Figure 1: SCHEMATIC REPRESENTATION OF TRAJECTORY SAMPLING. A measurement system collects packet *labels* from all the links within the domain. Labels are only collected from a pseudo-random subset of all the packets traversing the domain. Both the decision whether to sample a packet or not, and the packet label, are a function of the packet's invariant content.

for example, on the TTL field, which is decremented at each hop. Without loss of generality, we assume here that the function ϕ simply extracts *all* the S_c invariant bits from the packet.

$$\phi : \{0,1\}^S \rightarrow \{0,1\}^{S_c} \quad (1)$$

The basic idea of trajectory sampling is to decide whether to sample a packet P based on a deterministic function of the invariant packet content $\phi(x(P))$; we call this deterministic function the *sampling hash function* h, defined as

$$h : \{0,1\}^{S_c} \rightarrow \{0,1\}. \quad (2)$$

A packet P is sampled if $h(\phi(x)) = 1$. Note that we use the same sampling hash function h on each link in the measurement domain. In this way, a packet is either sampled everywhere on its trajectory or not at all, and the sample data lets us reconstruct the trajectories of the sampled packets.

In principle, a node could send the entire content of a sampled packet to the measurement collection system. However, this is very inefficient; note that to identify trajectories, we are not interested in the content of the packet per se; we only need an identifier to distinguish a given packet from other sampled packets, in order to obtain unambiguous samples of packet trajectories. Therefore, we use an *identification hash function* g to compute a compact packet identifier on the constant part of the packet.

$$g : \{0,1\}^{S_c} \rightarrow \{0,1\}^m \quad (3)$$

In this way, we only have to send m bits per sampled packet per link to the collection station.

In its most basic form, trajectory sampling performs the following simple operation at each link in the domain: for each observed packet of content x, if $h(\phi(x)) = 1$ then send the label $g(\phi(x))$ to the measurement collection system. While this suffices to identify packet trajectories, additional information about a sampled packet (such as its length and its source and destination addresses) are required for many measurement purposes. It is sufficient to collect this additional information once per sampled packet. For example, ingress nodes could be configured to retrieve this information along with the labels, while all other nodes *only* collect labels (cf. Fig. 1).

2.1 Packet Identity and Invariant Content

The definition of the invariance function ϕ is completed by identification of the invariant packet content. Here we consider only packets in IP version 4. We first consider candidate parts with the first 20 bytes of the packet; this comprises the packet header, or the first 20 bytes of a packet with IP options. We exclude TTL (bits 64–71) which is decremented per hop, and the SERVICE TYPE field (bits 8–15) since certain of its bits may be changed in transit, e.g. during Explicit Congestion Notification [20], and by operation of Differentiated Services [4]. The HEADER CHECKSUM (bits 80–95) is recalculated on changes of each of these and must hence also be excluded.

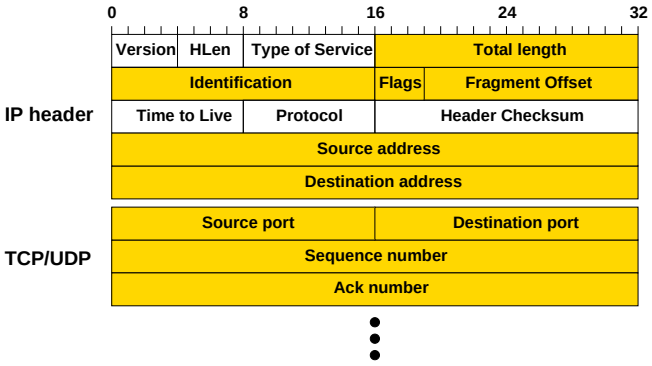

Figure 2: INVARIANT PACKET CONTENT. The hash functions are computed over a subset of header fields and part of the payload. Included fields are shaded.

VERSION (bits 0–3), HEADER LENGTH (bits 4–7) and PROTOCOL (bits 72–79) are either constant or take one of a small number of values; there is little gain in their inclusion in the invariant packet content.

SOURCE AND DESTINATION IP ADDRESS (together bits 96–159) are included in the invariant packet content. We also include the IDENTIFICATION field (bits 32–47). The presence of tunneling will impact packet identity through encapsulation behind a tunnel header. In some types of tunnel the original header could be recovered from the tunnel payload upon through appropriate offsetting; see e.g. IP in IP Tunneling [23] and Multiprotocol Label Switching (MPLS) [6]. This approach lets us match up samples inside and outside the tunnel. If tunnel endpoints are confined to the network edge, then one can simply sample consistently in the network interior.

FLAGS (bits 48–51) and FRAGMENT OFFSET (bits 52–63) are likewise mutable through fragmentation. Indeed, fragmentation raises potentially a larger issue, since it provides a mechanism by which the notion of a single identifiable packet becomes corrupted. However, we expect fragmentation to by confined to the network edge, with an edge-to-edge notion of packet identity remaining valid. In this case we can include TOTAL LENGTH, FLAGS and FRAGMENT OFFSET within the invariant content.

The remainder of the packet following the first 20 bytes completes the invariant packet content. In certain IP options packets, such as packets with a record route option, these following bytes may change hop by hop. However, since such packets are rare, we believe the effect on sampling can be ignored.

2.2 Ambiguous Trajectories

We discuss how to infer trajectories from the labels collected from the network over a measurement period. The measurement period T is chosen as an upper bound of the packet lifetime (e.g., 10 seconds). We assume that all the packet observations made within the same measurement period can only be distinguished by their label, not by their arrival time within the measurement period. As labels are allocated pseudo-randomly to sampled packets, their is obviously a chance of *label collision*, i.e., of two or more packet trajectories having the same label in the same measurement period. The question we address in this subsection is under what circumstances we can disambiguate these trajectories.

It is useful to introduce the concept of a *label subgraph* associated with a label i and a measurement period. The label subgraph is simply the graph of the network domain, where each link is annotated with the number of times label i has been generated by that link in the measurement period; links with zero are deleted. A label subgraph basically represents the superposition of all the trajectories in the measurement period that had this label.

We restrict this discussion to unicast packets and to acyclic label subgraphs. First, note that in the trivial case where a label subgraph stems from a single trajectory, that trajectory can always be inferred unambiguously. Intuitively, this is because a packet is either sampled everywhere in the domain or nowhere. Thus, if we observe label i on exactly one inbound and one outbound link of a node, it must be the same packet[2]. By induction, the entire trajectory can be reconstructed without ambiguity.

Second, let us consider the case where the label subgraph is the superposition of several trajectories. A few examples of superpositions of two trajectories are given in Figure 3. The examples (a) through (e) are unambiguous, while examples (f) through (h) are ambiguous.

The following property holds: a label subgraph is unambiguous if each connected component of the subgraph is either (a) a source tree, or (b) a sink tree such that for each node on the sink tree, the degree of the outbound link is the sum of the degrees of the inbound links. Note that example (e) is unambiguous because the only connected component is a source tree; it is also a sink tree, but the degree condition does not hold.

Also note that ambiguity as defined here pertains only to the *trajectories* followed by packets. For example, example (e) is unambiguous because there is no ambiguity about the two trajectories followed by the packets. However, if we have collected other attributes of the two packets (at the ingress node, say), then we have no way of knowing from (e) which packet was dropped in the middle, and which one made it to the egress node. In contrast, there are several possible sets of trajectories that can result in the label subgraphs (f) to (h).

[2]We view packets generated by routers (e.g., routing updates) as coming from a virtual ingress node connected to that router.

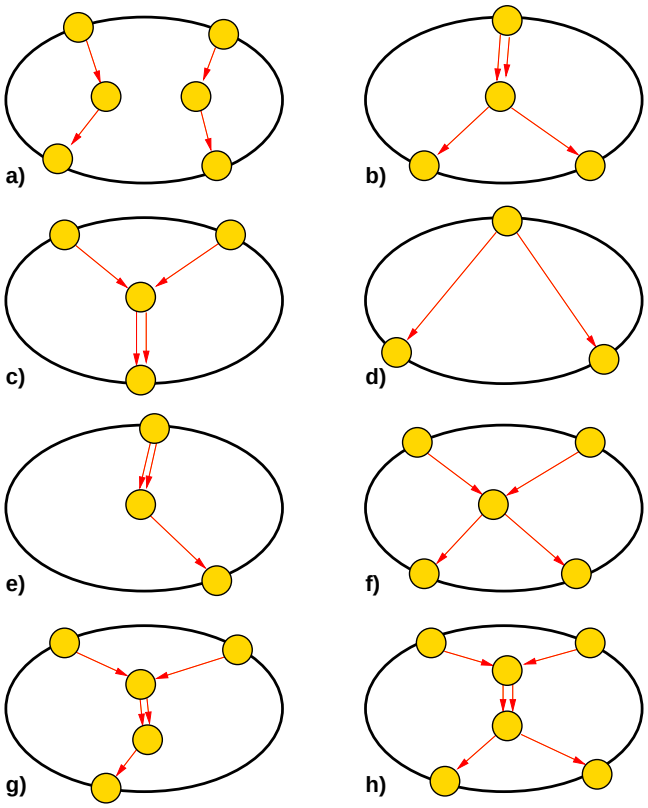

Figure 3: TRAJECTORY DISAMBIGUATION. Examples of unambiguous (a-e) and ambiguous (f-h) label subgraphs. For (e) and (g), a packet is dropped at an interior node.

3. Performance of Trajectory Sampling

In this section, we study the performance of trajectory sampling. Our overall goal is to obtain as many pseudo-random trajectory samples from the network as possible, without using too many resources (network bandwidth, collection system memory). We first describe calculation of the hashes. We then demonstrate that the hashes appear statistically independent from the original packet content, thus enabling unbiased sampling. We then compute the optimal choice of the total number of samples to be collected from the network and the number of bits per sample, subject to a constraint on the network bandwidth available for traffic measurement.

3.1 Specification of Hash Functions

We regard the ordered bits of a packet x and of its invariant part $\phi(x)$ as binary integers. We use the sampling hash

$$h(\phi(x)) = \begin{cases} 1 & \text{if } \phi(x) \leq r \mod A \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

for positive integers A and r. The modulus A is chosen in order to avoid collisions arising from certain structural properties of the the packet contents. For example, we expect

to find complementary sets of packets in which source and destination IP addresses are interchanged, arising from the two way flow of traffic in TCP sessions. The hash function, and hence the modulus, must be chosen to avoid collisions in which a pair of packets that differ little by such an interchange are mapped onto the same remainder. Knuth (see §6.4 in [17]) formulates a condition for avoidance of such collisions, namely that $q^k \pm a \neq 0 \mod A$ for small a, k where q is a radix of the alphabet used to describe the header. Including $q^k = 2^{32}$ in this criterion suppresses collisions of the type described above. Moduli obeying these conditions can be selected from tables of primes. r determines the granularity of sampling; A must be chosen sufficiently large in order that the smallest available sampling rate, namely $1/A$ for $r = 1$, is sufficiently small.

Sampled packets are encoded using a similar hash function

$$g(\phi(x)) = \phi(x) \mod B, \tag{5}$$

with the modulus $B \neq A$ in order that the identification hash is uncorrelated with packet sampling.

3.2 Identical Packets

As hashing is a deterministic function, if two packets are exactly identical, then the sampling decision and their label will be identical as well. Therefore, identical packets are not sampled pseudo-randomly by our method, which can lead to biased estimators. We therefore have to convince ourselves that identical packets are rare in practice. We call the occurrence of identical packets in a trace *collisions*.

More generally, we are interested in the frequency with which a *prefix* of a certain length l (i.e., the first l invariant bytes) of a packet is not unique within a large set of packets. If we can identify a packet prefix length for which collisions are rare, then it is sufficient to compute the sampling and the identification hash over this prefix. In a sense, the prefix generates sufficient "entropy" to make the sampling and labeling processes look random.

We have computed the number of collisions in a trace of one million packets, as a function of the packet prefix length; see Fig. 4. It is clear that relying only on the packet header is not sufficient for trajectory sampling to work well, as identical headers appear too frequently ($l = 20$ bytes). However, increasing the packet prefix length to take into account a few bytes of the payload quickly decreases the collision probability to below 10^{-3}. Increasing the packet prefix length beyond about 40 bytes does not reduce collisions any further; the remaining collisions are due to packets that are indeed exact copies of at least one other packet[3]. However, collisions are sufficiently rare to be inconsequential.

3.3 Evaluation of Hash Functions

We explored the statistical properties of hashing algorithms on packet traces. The traces were gathered using the `tcpdump` utility [16] on a host attached to a local area network segment close to the border of a campus network. Analysis was performed on four traces each comprising 1 million IP packets. Except in one case, the traces involved traffic between

[3]We note that the majority of these residual collisions are due to TCP duplicate acknowledgment packets, which are indeed exact copies of each other.

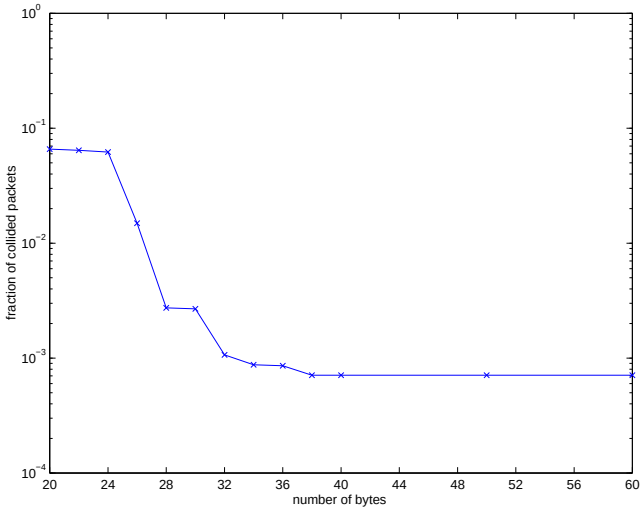

Figure 4: PACKET COLLISIONS. The fraction of packets whose prefix is not unique, as a function of the prefix length l. The smallest value for the prefix length (20 bytes) corresponds to using only the packet header.

about 500 distinct campus hosts and about 3000 distinct external hosts. The exception was a trace of a single ftp session set up between two campus hosts.

The hash functions were implemented in 32 bit integer arithmetic by long division over 16 bit words. Thus, a given number $z = (z_k, z_{k-1}, \ldots, z_1) = \sum_{i=0}^{k} z_i 2^{16i}$ has its modulus $z \mod A$ calculated through iteration of

$$(z_k, z_{k-1}, \ldots, z_0) \mod A = (z_{k-1} + 2^{16}(z_k \mod A), \ldots, z_0) \mod A. \tag{6}$$

Since the word size is 16 bits, $z_{k-1} + 2^{16}(z_k \mod A)$ fits within a 32 bit unsigned integer.

A desirable property of sampling hash function is that packet sampling should appear independent of a proper subset of the packet content. Consequently, the distribution of any variable attribute of the packet (such as source or destination IP address) should be the same for sampled packets as for the original population. We now perform tests of the independence hypothesis, based on chi-squared statistics calculated from the samples and the original traces.

Consider a given attribute of the packet (or set of packets), e.g. destination IP address. Partition the range of attribute values seen in the full trace into a number I of bins, with n_i values falling in bin i, there being $n = \sum_{i=1}^{I} n_i$ packets in total. Suppose that m_{1i} of the samples have attribute in bin i, there being $m_1 = \sum_i m_{1i}$ samples in total. Likewise, there are $m_{0i} = n_i - m_{1i}$ unsampled packets in bin i, with $m_0 = n - m_1$ unsampled packets in total. We form the 2-by-I contingency table of bin occupancies shown in Table 1.

The chi-squared statistic for Table 1 is

$$T = \sum_{i=0}^{1} \sum_{j=1}^{I} \frac{(m_{ij} - \overline{m}_{ij})^2}{\overline{m}_{ij}} \tag{7}$$

m_{01}	m_{02}	...	m_{0I}	m_0
m_{11}	m_{12}	...	m_{1I}	m_1
n_1	n_2	...	n_I	n

Table 1: 2-by-I table of bin occupancies.

where $\overline{m}_{ij} = m_i n_j / n$ is the expected values of m_{ij} under the null hypothesis that the bin occupied by a given packet is independent of whether or not it is sampled. For a given confidence level c (say $c = 95\%$), we accept this hypothesis if $T < T_c$, the c^{th} quantile of the chi-squared distribution with $I-1$ degrees of freedom. Equivalently, we accept if $C(T) < c$, where the C is the cumulative distribution function of the chi-squared distribution with $I-1$ degrees of freedom[4]. We applied three variants of this procedure in order to test the independence hypothesis.

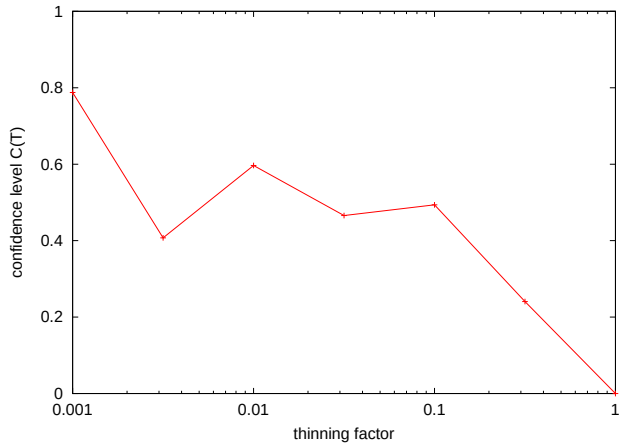

Figure 5: HASH-SAMPLED ADDRESS DISTRIBUTIONS. Confidence levels $C(T)$ from chi-squared statistics of sampled address distributions as a function of thinning factor. In all cases, the sample distribution is consistent with that of full trace down to a 80% confidence level. Sampling hash is calculated on a 40 byte packet prefix.

Address Prefix Distributions. Packets were binned on address prefix. The sampling hash was calculated using a 40 byte packet prefix. Increasing the packet prefix for the sampling hash beyond this point does not decrease the frequency of collisions (see Figure 4), so we expect no further reduction in dependence between sampling hash and packet address.

The experiments reported here used a fixed length 8-bit prefix, yielding $I = 2^8$. We amalgamated bins i with expected occupations $\overline{m}_{1i} < 1$ in order to avoid under-emphasizing contributions to T, which could otherwise lead to optimistic acceptance of the null hypothesis[5]. Of 80 bins occupied in the full trace, nearly half remained occupied at a thinning factor of 10^{-3}. Figure 5 shows $C(T)$ as a function of the thinning factor r/A using modulus $A = 16979$. In all cases, $C(T)$ was less than 0.8; thus the sampled and full trace address distributions cannot be distinguished at 80% or higher confidence level.

We repeated the experiments for two other binning schemes (i) fixed length 16 bit address prefixing; and (ii) BGP address prefixing in which addresses are allocated to bins according to their longest prefix match on a snapshot of the BGP routing table. In both these cases there were roughly 1000 bins occupied by the full trace. The confidence levels $C(T)$ were lower than those reported above, i.e., the independence hypothesis would be more readily accepted.

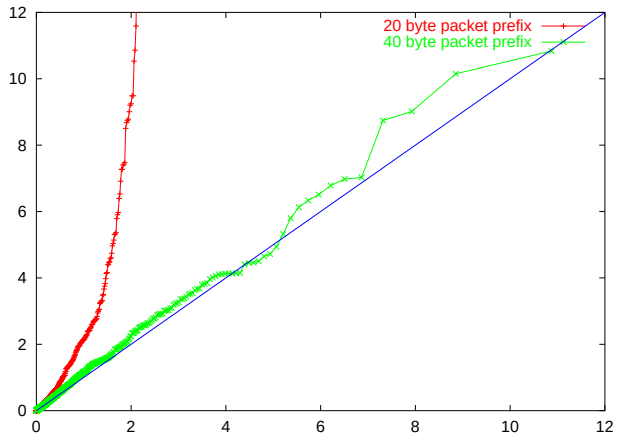

Figure 6: HASH-SAMPLED ADDRESS BITS DISTRIBUTIONS. Quantile-quantile plot of address bit chi-square values vs. chi-squared distribution with 1 degree of freedom; for various traces, primes A, thinning factors r/A; see text. Close agreement for 40 byte packet prefixes; marked disagreement for 20 byte packet prefixes (i.e. no payload included for sampling hash)

Bitwise Address Distributions. Let x_k denote the k^{th} packet in a stream, and $x_k(\ell)$ its ℓ^{th} bit. For each bit position ℓ we construct the 2-by-2 contingency table in which m_{ij} is the number of packets k for which the sample hash $h(\phi(x_k)) = i$ and the ℓ^{th} bit is $x_k(\ell) = j$. We calculated the corresponding chi-squared statistic T for each address bit, using each of two traces, three distinct primes $A = 1013, 10037$ and 16979 and thinning factors r/A between approximately $10^{-1/2}$ and 10^{-4}, all hashing on a 40 byte packet prefix. According to the null hypothesis, each such T should follow a chi-square distribution with 1 degree of freedom. We summarize these statistics in Figure 6 through a quantile-quantile plot of the T values against a this chi-square distribution. This shows close agreement; the plot is similar to that obtained using randomly generated statistics from the expected distribution. For comparison we also show quantiles obtained with a 20 byte packet prefix, i.e., using only the invariant header for sample hashing. In this case there is poor agreement, with many high T values, presumably due to the increased frequency of collisions.

Temporal Sampling Distributions. For a trace of a single ftp session between two hosts, we check that the packet

[4]Chi-squared and related statistics are evaluated as discrepancy metrics for sampled network traffic in [8, 19]; the latter paper discusses optimization of bin sizes for ordinal data such as inter-event times.

[5]See §4.3 in [22] for treatment of small expected occupations

sample process is consistent with that of independent sampling at the average sampling rate. We allocate packets into one of two bins, according to whether the succeeding packet in the session is sampled or not. This results in a 2-by-2 contingency table in which m_{ij} is the number of packets k for which the sample hash $h(\phi(x_k)) = i$ while that of its successor is $h(\phi(x_{k+1})) = j$. According to the null hypothesis, the statistic T follows a chi-squared distribution with 1 degree of freedom. We performed a number of experiments using $A = 2377$, thinning factors between $10^{-1/2}$ and 10^{-4}, and packet prefixes of 50 bytes or larger. In each experiment we were able to accept the hypothesis at the 95% confidence level.

3.4 Optimal Sampling

We next discuss the choice of the number of samples n and the number of bits m per sample. For convenience, we let $M = 2^m$ denote the alphabet size of the identification hash.

Based on the discussion in Section 2.2, if two different trajectories happen to use the same label, then they may or may not be ambiguous. The probability that we get an unambiguous sample of a trajectory depends on the statistical properties of all the other trajectories that might interfere. This is difficult to analyze. However, we are able to obtain a lower bound on the number of unambiguous labels. For this purpose we assume that the label subgraph is ambiguous whenever there is a label collision. In other words, we disregard the cases discussed in Section 2.2, where several trajectories with the same label can be ambiguous.

We obviously face two conflicting goals for the choice of n and m. On the one hand, the reliability of traffic estimates increases with the number of unambiguous samples we can collect. On the other hand, we have to limit the total amount of measurement traffic between the routers in the domain and the collection system. Note that the amount of traffic incurred over a measurement period is given by nm bits, because an m-bit label is transmitted to the collection system for each of the n samples (ignoring packet headers for the measurement packets and other overhead).

We therefore formulate the following simple optimization problem: we want to maximize the expected number of unique (unambiguous) samples, subject to the constraint that the total measurement traffic nm must not exceed a predefined constant c. We assume that each sample independently takes one of the M label values with uniform probability $p = 1/M$. The marginal distribution of the number of samples taking a given label is binomial $B(n, p)$. Hence the probability that the label is generated exactly once in the domain with the measurement period is

$$p_u = np(1-p)^{n-1}. \tag{8}$$

Let Z_i be the random variable that takes the value 1 if label i is taken by exactly 1 sample, and 0 otherwise. The mean number of unique samples is then

$$A(n,m) = \mathsf{E}[\sum_{i=1}^{M} Z_i] = \sum_{i=1}^{M} \mathsf{E}[Z_i] = Mp_u = n(1-p)^{n-1} \tag{9}$$

where E denotes the expected value under the assumed uniform label distribution. For fixed n, $A(n,m)$ is obviously maximized for $m = c/n$, and we therefore maximize

$$A(n) = n(1-2^{-c/n})^{n-1} \tag{10}$$

Solving $A'(n) = 0$ yields the maximizing n^*, where $A'(n)$ is the derivative of $A(n)$,

$$A'(n) = (1-2^{-c/n})\left[1 - 2^{-c/n}\left(1 + \frac{n-1}{n} c\log(2)\right)\right] = 0.$$

We find [6]

$$n^* = \frac{M^*}{\log(M^*)}, \qquad M^* = c\log(2) \tag{11}$$

Finally, we compute the sample collision probability at the optimal operating point.

$$p_{coll} = 1 - \frac{A(n^*)}{n^*} = 1 - (1-2^{-c/n^*})^{n^*-1} \approx 1 - e^{-\frac{1}{m^*\log(2)}} \tag{12}$$

Figure 7 illustrates how $n = n^*$ maximizes $A(n)$: for $n < n^*$, collisions are very rare - we waste label bits for too few samples; for $n > n^*$, collisions are too frequent - we waste samples through collisions because label identifiers are too short. Note that the optimal M^* can obviously not be achieved exactly. In practice, we choose the largest integer $B \leq M^*$ satisfying the conditions put forth in Section 3.1.

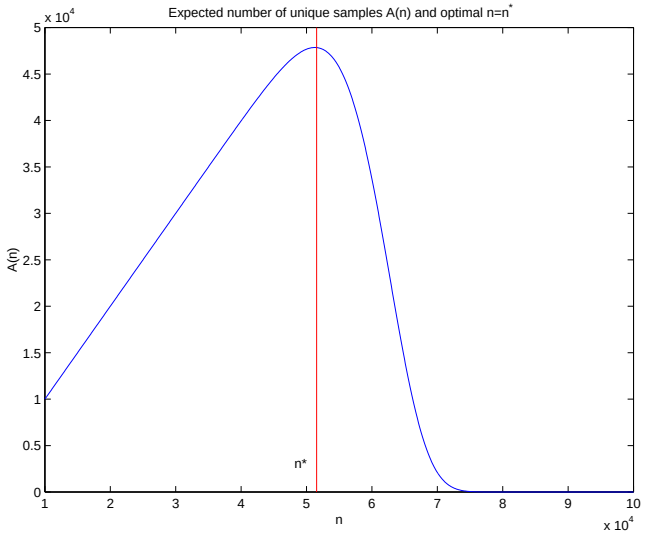

Figure 7: THE EXPECTED NUMBER OF UNIQUE SAMPLES $A(n)$ AS A FUNCTION OF n, FOR $c = 10^6$ BIT. The optimal number of samples n^* is approximately $5.15 \cdot 10^4$, with m^*=19.4 bit per label. The collision probability p_{coll} is approximately 0.072, i.e., 7.2% of the samples transmitted to the collection system have to be discarded.

Let us look at a specific example that illustrates how m and n would be chosen in practice. Assume that the measurement domain consists of 100 OC-192 links (10 Gbps each). Suppose the measurement system can handle 10Mbps of in-

[6] For the trivial solution $n = c$, $A(n) < 0$.

coming label traffic for the entire domain[7]. Furthermore, we choose a measurement epoch to be $T = 10$ seconds; this is a conservative upper bound on the lifetime of a packet traversing the domain. For simplicity, we assume that all packets are 1500 bytes long.

The bound on the total amount of measurement traffic is $c = T \times 10 =$ 1e8 bits. The number of samples we should collect over the measurement period is $n^* = 3.84e6$, or about 3840 samples per link per second. A fully loaded OC-192 link can carry about 833k 1500-byte packets per second. Therefore, we would configure the sampling hash in this domain so that the sampling probability for a packet would be approximately $3840/8.33e5 \approx 1/217$. The labels would be $m^* = \log_2(M^*) \approx 26$ bit long. The actual number of samples n will obviously depend on how heavily each link is loaded. The main point of the above analysis is to allocate enough bits m to labels such that under peak load, the collision probability does not become too frequent. Note that if the average packet size is less than 1500 bytes, we simply have to reduce the sampling probability accordingly (e.g., by reducing r). However, the number of samples n^* and the label size m^* are not affected, as they depend only on c.

4. Traffic Measurement

In this section, we use trajectory sampling for a simple measurement task. The goal of this experiment is to illustrate how estimators can be constructed based on the sampled labels received from the measurement domain. We study the following simple scenario. Assume that a service provider wants to determine what fraction of packets on a certain backbone link belongs to a certain customer. To estimate this fraction, the service provider can use the labels collected from the backbone link under study and from the access link(s) where the customer connects to the network.

For the purposes of experimentation, we adapt the packet trace used in the previous section to the present context as follows. All packets with a certain source prefix are designated as originating from the customer, while the remaining packets form the other traffic on the backbone link; see Figure 8.

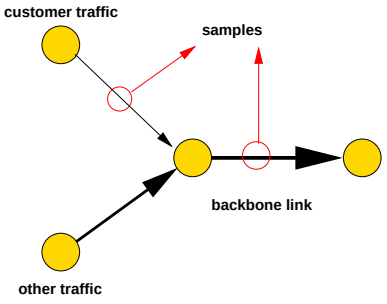

Figure 8: MEASUREMENT EXPERIMENT. A simple experiment where labels from two links are compared to estimate what fraction of traffic on the backbone link comes from the customer access link.

[7]We do not discuss distributed implementations of the measurement collection system in this paper, but the potential of distributed measurement processing to increase the amount of measurement traffic is obvious.

For the sake of exposition, assume that we sample packets and collect labels only from the customer access point, and from the backbone link. We then proceed as follows: any label that appears more than once on the backbone link is discarded, because this can only be due to a collision. Among the remaining unique labels, we determine which labels are only observed on the backbone link, and which labels are observed on both links. This allows us to obtain an estimate for the fraction of customer traffic on the backbone link, given by

$$\hat{\mu} = \frac{n_{c,b}}{n_b}, \tag{13}$$

where $n_{c,b}$ is the number of unique labels observed on both the customer access link and on the backbone link, while n_b is the total number of unique labels observed on the backbone link[8].

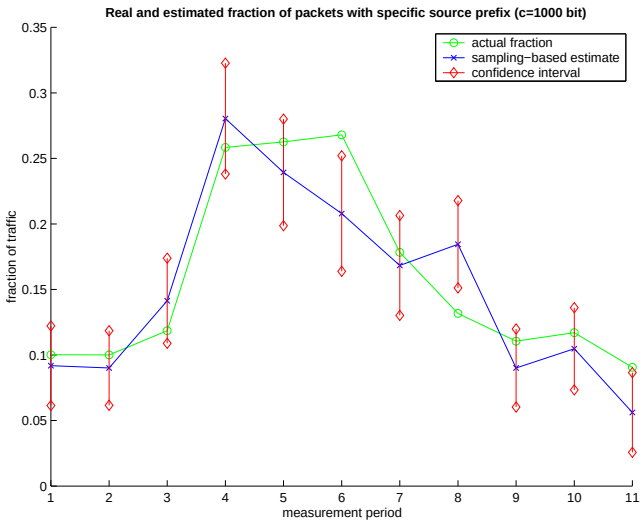

Figure 9: REAL AND ESTIMATED FRACTION OF CUSTOMER TRAFFIC. For $c = 1000$ bit for this link ($M^* = 693.1$, $B = 691$, $n^* = 106$).

Figures 9 and 10 compare the estimated and the actual fraction of traffic on the backbone link, for ten consecutive measurement periods. For simplicity, we have defined a measurement period as a sequence of 10^5 consecutive packets in the trace, rather than as a time interval. The graph also shows confidence intervals around the estimated values. The confidence intervals are obtained as follows. We compute the standard deviation of the estimator $\hat{\mu}$ assuming that each packet gets sampled independently and with equal probability. If this were true, then the probability that a sampled packet belongs to the customer is μ. The standard deviation of the estimator $\hat{\mu}$ is then[9]

$$\sigma = \sqrt{\frac{\mu(1-\mu)}{n_b}} \tag{14}$$

The confidence interval we plot is $[\hat{\mu} - \sigma, \hat{\mu} + \sigma]$, i.e., one

[8]Note that $n_b < n$ because of collisions; $\mathsf{E}[n_b] = A(n)$, defined in Section 3.4.

[9]The variance of a Bernoulli random variable with mean p is $p(1-p)$.

standard deviation around the estimated value.

Note that the amount of measurement traffic per measurement period from the backbone link ($c = nm$) is quite small (1000 bits in Fig. 9 and 10kb in Fig. 10). The confidence interval is reduced as the amount of measurement traffic increases.

A statistical estimator such as the one considered here relies on an underlying random sampling process. The size of the confidence interval is then a consequence of the central limit theorem for independent random variables. However, trajectory sampling is based on a *deterministic* sampling process, and the sampling decision for a packet is a function of this packet's content. Nevertheless, we observe in this experiment that the true value of the estimated quantity lies within or very close to the confidence interval without exception. This is despite the fact that there is strong correlation between the packet content (because the customer packets all have the same source prefix) and the events we are counting (packet belongs to customer). This correlation does not translate into a biased sampling process here. This demonstrates that good hash functions can sufficiently "randomize" sampling decisions such that the set of sampled packets (and their labels) are representative of the entire traffic for the purpose of statistical estimation.

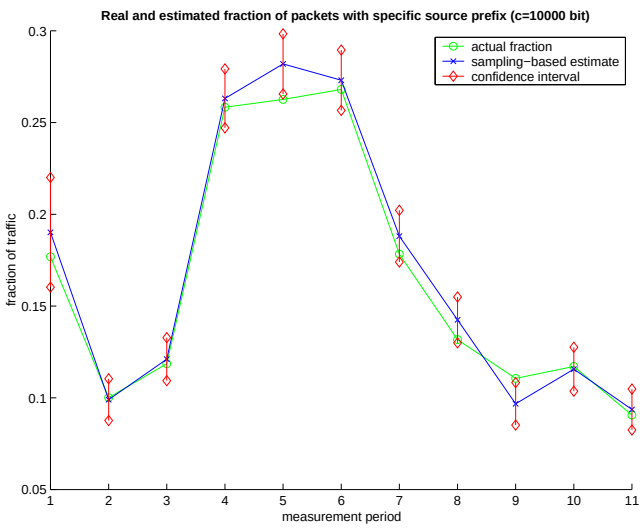

Figure 10: Real and estimated fraction of customer traffic. For $c = 10$ kbit for this link ($M^* = 6931.5$, $B = 6917$, $n^* = 782$).

5. Discussion

5.1 Implementation Issues

We argue that the implementation cost for trajectory sampling is quite acceptable even for the highest interface speeds available today. Trajectory sampling requires a device for each interface capable of (a) computing the sampling hash and making a sampling decision, and (b) computing the identification hash for the sampled packets.

The computational cost is obviously dominated by the operations that have to be executed for each packet that goes through this interface (as opposed to operations only on sampled packets). In our conceptual description of the sampling process, we have viewed computation of the sampling and the identification hash as sequential. The identification hash would only be computed if the packet is to be sampled, otherwise the packet is discarded. However, from an implementation point of view, this is undesirable, as it would require buffering each packet until the sampling hash is computed.

An alternative implementation illustrated in Figure 11 computes both the sampling hash and the identification hash for both packets concurrently and on the fly as the bits come in. The hash functions discussed in Section 3.1 allow such an implementation. This removes the burden of having to make a separate copy of the packet for the purpose of computing the identification hash. The processor computes both hashes, and simply writes the identification hash g into the label store if the sampling hash h is equal to one. The label store accumulates packet labels until it reaches a predefined size, then sends the labels to the measurement system as a single IP packet[10].

As an example, a state-of-the-art off-the-shelf digital signal processor can process up to about 600M 32-bit multiply-and-accumulate (MAC) operations per second. This corresponds to a raw data rate of 20 Gbps. Also, raw memory I/O bandwidth can be up to 256 bit per memory cycle, which corresponds to 77 Gbps at 300MHz clock speed. In comparison, an OC-192 interface (the fastest commercially available SONET interface) carries 10Gbps.

While these arguments are based on peak processor performance, which typically cannot be sustained for various reasons (such as pipeline stalls in the processor), these numbers do illustrate that the computational requirements necessary for trajectory sampling are within reach of current commodity processors. It is also interesting to note that the price of such a processor is roughly two orders of magnitude lower than that of an OC-192 interface card. Adding logic for trajectory sampling to high-speed interfaces would therefore be comparatively cheap. Also note that to add measurement support to interface cards is in line with the trend over the last few years to move processing power and functionality from the router core to the interfaces.

We expect the relative cost of the sampling logic with respect to the interface hardware per se to evolve in our favor. In fact, it appears that processor performance increases slightly faster (doubling every 18 months according to Moore's law) than maximum trunk speed (doubling every 21 months) [21]. If these trends persist, then the cost of incorporating trajectory sampling into the next generations of high-speed interfaces can be expected to be negligible.

The link sampling device also requires a simple management interface to enable/disable packet sampling, to tell the device where to send measurement traffic, and to set the parameters of the hash functions. A simple SNMP MIB, indexed by the IP address of the interface, could fulfill this function.

[10]This should be done reliably (e.g., using TCP) in order to avoid loss of samples during congestion, and therefore possible bias in traffic estimators.

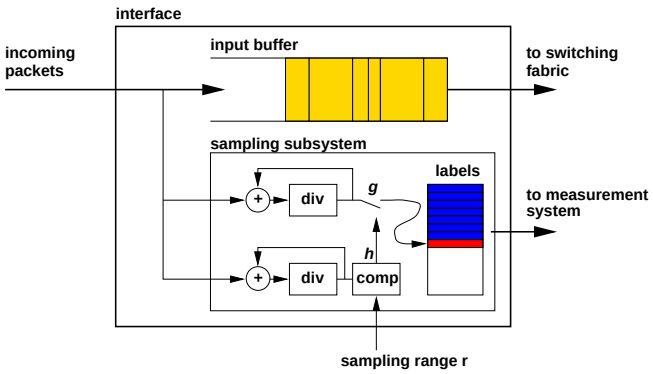

Figure 11: IMPLEMENTATION. A possible implementation of trajectory sampling computes both the sampling and the identification hash concurrently and on the fly. This removes the need to make a separate copy of each packet. The computation of the two hashes, defined in (6), can be implemented with the elementary multiply-and-add (resp. divide-and-add) function supported in off-the-shelf DSPs. A small buffer stores labels before they are copied into an IP packet and sent to the collection system. Some additional logic would be necessary on some nodes (probably on slower ingress nodes) to extract other fields of interest from a packet, e.g., length, and source and destination addresses.

5.2 Comparison with other Approaches

We next discuss several common measurement approaches for IP networks and put them into perspective in light of the points we made in the introduction. There are two general classes of measurement approaches. *Aggregation-based approaches* are deterministic functions of the observed data. They usually compute the sum or the maximum of some metric over the dataset (e.g., the sum of packets traversing a link during an interval, or the maximum end-to-end round-trip delay for a set of packets). *Sampling-based approaches* extract a random subset of all of the possible observations. This sample subset is supposed to be representative of the whole. The law of large numbers asserts that reliable estimators of desired metrics can be constructed from these samples. The first two methods we discuss, links measurements and flow aggregation, are aggregation-based. The third method, end-to-end probing, are sampling-based.

Link measurements (aggregation-based, direct). In this approach, aggregate traffic statistics are measured on a per-link basis, and are reported periodically (e.g., every five minutes). Metrics typically include the number of bytes and packets transferred and dropped within a reporting period. Some of these statistics are defined as part of the SNMP (Simple Network Management Protocol) MIBs (Management Information Base) [24].

The limitation of this approach is that some information is lost in the aggregation; therefore, it does not allow to classify the traffic (e.g., by protocol type, source or destination address etc.). More importantly, it is not possible in general to infer spatial traffic flow, i.e., to infer what path(s) the traffic follows between an ingress and an egress point. As such, this approach is better suited to detect potential problems, manifesting itself through link congestion, than to actually analyze the problem and modify routing information to remedy it.

Flow aggregation (aggregation-based, indirect). In this approach, one or several routers within the domain collect per-flow measurements. A flow comprises a sequence of packets with some common fields in their packet header and which are grouped in time [9]. The router has to maintain a cache of active flows[11]. A flow record may include specification of the source and destination IP address and port number, flow start-time, duration, the number of bytes and packets, amongst others.

One disadvantage of flow aggregation is that the amount of measurement data can be considerable; the traffic generated can impose a significant additional load on the network. This is especially true in the presence of large numbers of short flows, such as `http-get` requests. Also, the measurement traffic is hard to predict. It depends heavily on the way the router identifies individual flows, which in turn depends on various control parameters (such as the degree of aggregation of source and destination addresses), the traffic mix (protocols), and the cache size. A further complication may arise if traffic measurements are to be used for real-time control functions. Since a flow record is usually generated only upon a flow's completion, this implies that an on-line statistic may miss a long-lived flow that has not yet terminated.

A full path matrix over the domain can be obtained if flow aggregation measurements are available at each ingress point *and* if we know how the traffic is routed through the domain. While this is currently the only approach we are aware of to obtain a full traffic matrix in IP networks, it has several drawbacks:

- *emulation of routing protocols:* even for non-adaptive routing, we have to rely on emulation of the routing protocol to correctly map the ingress traffic measurements onto the network topology; this requires full knowledge of the details of the routing protocol as well as its configuration.
- *no verification:* as mentioned before, one important role of traffic measurement is in the verification and troubleshooting of routing protocols and policies; obviously, routing emulation precludes detecting problems in the *actual* routing, e.g., due to protocol bugs.
- *dynamic and adaptive routing:* dynamic routing (routing around failed links) or adaptive routing (load balancing across multiple links/paths) further complicates emulation, because precise link state information would have to be available at each time (note that widely used routing protocols such as OSPF have some provisions to balance load among several shortest paths in a pseudo-random fashion; this would be impossible to emulate exactly).

Active end-to-end probes (sampling-based, indirect). In this approach, hosts (endpoints) connected to the network send probe packets to one or several other hosts to estimate path metrics, such as the packet loss rate and the round-trip delay [5, 1, 2]. In a variation of this approach, hosts

[11] For some router models, flow caches already exist to speed up route and access control list (ACL) lookup.

do not actually generate probe packets, but they collect and exchange measurements of the traffic of a multicast session (e.g., RTCP [15]).

This approach gives direct measurements of *end-to-end path* characteristics, such as round-trip delay and packet loss rate; per-link characteristics have to be inferred. This approach can be viewed as an alternative way to obtain per-link aggregate measurements. Its advantage is that it does not require any measurement support from the network. It has the same disadvantages as the "link measurement" approach.

5.3 Related Work

Sampling has been proposed as a method to measure the end-to-end performance of individual flows in connection-oriented networks [10, 27]. ATM cells are sampled at the ingress and egress points of a virtual circuit in order to measure QoS metrics such as the end-to-end delay and the loss rate. To compute these metrics, cells at the ingress and egress points have to be matched. The authors propose to label cells using a hash function over the header and payload.

While this use of identification hashing is similar to ours, there are some fundamental differences between this method and trajectory sampling. The focus of end-to-end hash-based sampling is on determining the QoS of a single connection, rather than obtaining a statistically representative sample of the entire path matrix over a domain. Therefore, these methods do not require a pseudo-random sampling hash function to determine which packets to sample. The goal is simply to select a subset of cells for which the end-to-end performance is measured. In fact, in [10], it is suggested to use simple bit pattern matching in the cell content to sample packets; this would not be an acceptable sampling hash function.

In contrast, trajectory sampling critically relies on a sampling hash function to select a statistically representative subset of packets *over all the flows traversing the network.* This is because there is a strong correlation between some fields in the packet (e.g., the destination address) and the path taken by the packet. The focus of trajectory sampling is to directly observe the entire traffic flowing through a domain, rather than a single flow at its endpoints, and to infer statistics on the spatial flow of this traffic.

5.4 Extensions and Other Applications

Distributed Denial-of-Service Attacks (DDoS). This type of attack floods a network or a host with bogus traffic with the intent of breaking down service to legitimate clients [7]. Attackers often use packet spoofing, i.e., using false source addresses, to evade detection and exacerbate the impact of the flood. Because of this, it is difficult to identify the real source(s) of the attacking traffic, because there is no a-posteriori information available to deduce where a packet entered the network and what path it followed. The method presented in this paper may help in the detection of such an attack, as sample trajectories provide the actual paths packets are taking to reach the targeted system despite the fake source address.

Filtering. There may be situations where it is desirable to apply trajectory sampling only to a subset of the traffic in a domain. For example, a network operator might want to examine only the traffic destined for a particular customer, or only the traffic of a certain service class. The amount of measurement traffic can be reduced in such a situation if only the traffic matching the desired criterion is sampled. This can be achieved by preceding the sampling device described in Section 5.1 with a configurable *packet filter*. The network operator could then configure the filters of all the interfaces in the network to sample only the desired subset of traffic. This could again be achieved through the sampling device's SNMP MIB.

Probe Packets. In a network domain which supports trajectory sampling, it is possible to probe end-to-end routes in a novel way. Assuming that the sampling and identification hash functions in the domain are known, it is possible to *construct packets that will be sampled* as they traverse the network. Suppose we wish to check the path of a packet with a given header between a specific ingress and egress node. We can then append a payload to this header that forces the sampling of this packet, by selecting the payload such that $h(\phi(x)) = 1$. The label for this packet can also be determined. This method could be used to verify specific routes for debugging or for monitoring purposes.

6. Conclusions and Further Work

In this paper we have proposed a method for the consistent sampling of packet trajectories in a network. Only a subset of packets are sampled, but if a packet is sampled at one link, it will be sampled on every other link it traverses. On traversing the network, each packets implicitly indicates whether or not it should be sampled through its invariant part, i.e. those bits that do not change from link to link. A hash of these bits it calculated at each router, and only those packets whose sampling hashes fall within a given range of values are selected. For selected packets, a different hash, the identification hash, is used to stamp an identity on the packet. This is communicated by the sampling router to the measurement systems. This enables post sampling analysis of distinct trajectories once the samples are reported. The method has a number of desirable properties:

- *Simple Processing:* the only per packet operations required are the division arithmetic on a small number of bytes in the packet header. No packet classification or memory lookups are used.
- *No Router State* is required in the per packet processing of the router: packets being processed individually. No caching is required in the measurement subsystem of the router, thus avoiding cache delay and possible biasing through the requirement of cache expiry policies. This does not exclude the possibility of having state in the *reporting* system in the router; it may be desirable to aggregate discrete reports to the measurement system rather than sending them individually.
- *Packets are directly observed:* the course of the packets through the network can be determined without a network model that specifies how they ought to be routed. This is important for debugging since routing may not easily specify current routing state of the system [14]. Moreover, configuration or other errors may cause actual routing behavior to deviate from that specified by the model.

In the future, we plan to investigate hash functions that satisfy stronger randomization properties [25]. We also propose to evaluate trajectory sampling in a network context. The aims are to understand trajectory reporting over a wide network, and to develop technique for systematic trajectory reconstruction, including resolution of ambiguities of the type discussed in Section 2.2. The approach combines routing information and traffic traces to make a network simulation that captures the topology and traffic patterns of real networks.

Acknowledgments

We thank Albert Greenberg, Gisli Hjálmtýsson, Colin Mallows, K. K. Ramakrishnan, Jennifer Rexford, and Jean Walrand for useful discussions and suggestions. We are indebted to Balachander Krishnamurthy for providing the software for BGP address prefixing.

7. REFERENCES

[1] G. Almes, S. Kalidindi, and M. Zekauskas. A One-Way Delay Metric for IPPM. *RFC 2679, available from http://www.ietf.org/rfc*, September 1999.

[2] G. Almes, S. Kalidindi, and M. Zekauskas. A One-Way Packet Loss Metric for IPPM. *RFC 2680, available from http://www.ietf.org/rfc*, September 1999.

[3] Daniel O. Awduche. MPLS and Traffic Engineering in IP Networks. *IEEE Communications Magazine*, December 1999.

[4] Y. Bernet, J. Binder, S. Blake, M. Carlson, B. E. Carpenter, S. Keshav, E. David, B. Ohlman, D. Verma, Z. Whang, and W. Weiss. A Framework for Differentiated Services. *Internet Draft (work in progress)*, February 1999.

[5] R. Cáceres, N. G. Duffield, J. Horowitz, and D. Towsley. Multicast-based inference of network-internal loss characteristics. *IEEE Transactions on Information Theory*, 45:2462–2480, 1999.

[6] R. Callon et al. A Framework for MPLS. *IETF Internet draft, work in progress*, September 1999.

[7] CERT/CC and FedCIRC. Advisory CA-2000-01 Denial-of-Service Developments. *Carnegie Mellon Software Engineering Institute*, January 2000.

[8] Kimberly C. Claffy, George C. Polyzos, and Hans-Werner Braun. Application of Sampling Methodologies to Network Traffic Characterization. *Computer Communication Review*, 23(4):194–203, October 1993. appeared in Proceedings ACM SIGCOMM'93, San Francisco, CA, September 13–17, 1993.

[9] Cisco Corp. Netflow Services and Applications (white paper). August 1999.

[10] I. Cozzani and S. Giordano. Traffic Sampling Methods for end-to-end QoS Evaluation in Large Heterogeneous Networks. *Computer Networks and ISDN Systems*, 30(16-18), September 1998.

[11] Anja Feldmann, Albert Greenberg, Carsten Lund, Nick Reingold, and Jennifer Rexford. NetScope: Traffic engineering for IP networks. *IEEE Network Magazine*, March 2000.

[12] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[13] S. Floyd and V. Jacobson. The synchronization of periodic routing messages. *IEEE/ACM Transactions on Networking*, 2(2):122–136, April 1994.

[14] Timothy G. Griffin and Gordon Wilfong. An Analysis of BGP Convergence Properties. *Computer Communication Review*, 29(4):277–288, October 1999. appeared in Proceedings ACM SIGCOMM'99, Cambridge, MA, August 30–September 3, 1999.

[15] R. Frederick H. Schulzrinne, S. Casner and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. *RFC 1889, available from: ftp://ftp.isi.edu/in-notes/rfc1889.txt*, January 1996.

[16] Van Jacobson, Craig Leres, and Steven McCanne. `tcpdump`. available at ftp://ftp.ee.lbl.gov, 1989.

[17] Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, 1998.

[18] C. Labovitz, G. R. Malan, and F. Jahanian. Internet Routing Instability. *IEEE/ACM Transactions on Networking*, 6(5):515–528, October 1998.

[19] Vern Paxson. Empirically-Derived Analytic Models of Wide-Area TCP Conections. *ACM/IEEE Transactions on Communications*, 2(4):8–17, August 1994.

[20] K. K. Ramakrishnan and S. Floyd. A Proposal to add Explicit Congestion Notification (ECN) to IP. *RFC 2481, available from: ftp://ftp.isi.edu/in-notes/rfc2481.txt*, January 1999.

[21] Lawrence G. Roberts. Beyond Moore's Law: Internet Growth Trends. *IEEE Computer*, January 2000.

[22] Lothar Sachs. *Applied Statistics*. Springer, 1984.

[23] W. Simpson. RFC 1853, IP in IP Tunneling. available at ftp://ftp.isi.edu/in-notes/rfc1853.txt, 1995.

[24] William Stallings. *SNMP, SNMP v2, SNMP v3, and RMON 1 and 2 (Third Edition)*. Addison-Wesley, Reading, Mass., 1999.

[25] Mikkel Thorup. Even strongly universal hashing is pretty fast. In *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 496–497, 2000.

[26] V. Paxson and S. Floyd. Why We Don't Know How To Simulate The Internet. In *Proceedings of the 1997 Winter Simulation Conference*, December 1997.

[27] Jean Walrand. Private communication. February 2000.