An Information Theoretic Approach for Shared Bottleneck Inference Based on End-to-end Measurements

MIT/LCS/TM-604

By

Dina Katabi, Issam Bazzi, Xiaowei Yang

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Laboratory for Computer Science

545 Technology Square

Cambridge, Massachusetts 02139

March 2000

An Information Theoretic Approach for Shared Bottleneck Inference Based on End-to-end Measurements

Dina Katabi, Issam Bazzi, Xiaowei Yang

Laboratory for Computer Science,
Massachusetts Institute of Technology,
{dina, issam, yxw}@lcs.mit.edu

Abstract

Recent years have marked a growing interest in studying Internet path characteristics. However, most of the currently available tools to an end system to perform such measurements are slow inaccurate and generate an excessive amount of probing traffic. This paper introduces entropy as a novel and efficient metric for discovering Internet path characteristics based on data collected by an end system. In particular, the paper presents an entropy-based technique that enables an end system to cluster flows it receives according to their shared bottleneck. Our mechanism relies solely on information extracted from the packets' inter-arrivals at the receiver. It does not generate any probing traffic and can use data extracted from both TCP and UDP flows. Moreover, it requires only a small number of packets from each flow, which makes it useful for short-lived flows. We report the results of running the algorithm on simulated data and Internet traffic.

1 Introduction

Discovering the characteristics of Internet paths is one of the most challenging research areas. It can be done either by internal nodes or by end systems. In the first approach, routers use network management packages to collect statistics and report on links' performance. Because the routers have direct access to link-level information, this approach gives a more accurate estimate of the performance. However, there are many cases where this approach is infeasible or undesirable. For example, the internal routers might be inaccessible, unequipped with the necessary tools, or too busy to collect measurements. Moreover, as the Multiprotocol Label Switching (mpls) and similar techniques become widely used, the network layer information will be available only at the edges of a network cloud, and internal nodes will be unable to map the link level performance to the performance seen by different network flows.

The second approach for discovering Internet path characteristics relies on end systems examining traffic or probing the network to estimate its performance. It is desirable because it does not assume any help from the routers, yet it is intrinsically difficult because the information available to an end system is significantly noisy. Currently, the space of measurements that can be performed by and end system is limited, and usually the quality of these measurements is low and the process of collecting the data is expensive. For example, pathchar and cprobe are useful tools for discovering the bandwidth available along a path. However, to perform their task they consume a large amount of network resources. In particular, pathchar generates at least 10 Kbytes of probe traffic per hop and cprobe generates 5 Kbytes of probe traffic per hop [14]. The accuracy of these tools is acceptable for low bandwidth links (less than 10Mb/s) yet it becomes extremely low for high bandwidth links [21]. The Packet Bunch Mode (PBM) estimates the raw bottleneck bandwidth of a connection by looking for modalities in the timing structures of groups of back-to-back packets. Although more robust than pathchar, it requires information from both the sender and receiver sides [10]. Traceroute is a widely used tool for learning the intermediate routers and the

latency along a path yet it requires the intermediate routers to reply to ICMP echo messages, a feature that might not be available [16].

In contrast to the aforementioned tools, which rely on timing information, the authors in [19] propose the use of multicast loss-correlation to infer the loss rates over individual links along a path. Their simulation shows that the estimator tracks the changes in the loss rate. However, because the proposed approach multicasts probe packets in the network and requires the existence of a multicast service, its applicability for the current Internet is limited. The authors of [12] use loss-correlation among multicast receivers to infer the logical shape of the multicast tree. Their approach does not inject probe traffic in the network; however, its reliance on loss information limits its use to considerably long multicast sessions.

Although the above work is similar to ours in that is attempts to learn about an Internet path characteristics based on data collected at an end system, it does not address directly the problem of identifying flows that share the same bottleneck. Recently, a paper by Rubenstein et al attempts to detect whether two flows share the same bottleneck. In comparison, their work is more robust against a heavy cross-traffic and works at the sender and the receiver sides. However, it generates probing traffic and does not easily generalize to more than two flows [22].

In this paper we introduce new techniques for end-to-end Internet measurements that can capture the higher order statistics of the data and learn fairly complex information from a small data set. In particular, we present an entropy-minimization technique that enables an end system to cluster flows it receives according to their shared bottleneck. Our approach relies on the observation that the correct clustering minimizes the entropy of the inter-packets spacing seen by the receiver. It is passive (does not inject any traffic into the network,) fast and requires only few packets from each flow.[1] Moreover, it uses only the information available at the IP level; thus, it works with any type of traffic (TCP, UDP...etc). Although our design is developed in the context of a particular problem, we believe that data analysis based on entropy manipulation is a useful tool for reasoning and understanding network data.

We envision two applications for our clustering technique. First, our technique can guide network service providers (ISPs) in designing their topology and deciding on their routes. Designing the network topology for an ISP is a significantly difficult task. The optimal design (the one that minimizes the cost and maximizes the benefits) is usually extremely hard to compute and most ISP's rely on heuristics to find a reasonably good design. In this context, the exit border routers at the periphery of the provider's network dumps the source destination pair and the arrival time of each IP packet. By running our technique on the data dumped by each exit border router alone, then on the combination of all the logs, the provider learns the number of bottlenecks in her network, their level of congestion, and the share of each exit border router of each bottleneck. By combining this information with the routing information the provider draws a logical map of her network that shows the entrance and exit border routers and the bottlenecks. This map can indicate possible changes in the provider's external routing policy that produce a better traffic distribution in the network. In addition the map can indicate where to add new links to reduce congestion. The details of this application are left for future work.

Second, our mechanism enables a gateway at the entrance of an organization's network to cluster the flows the organization receives according to their shared bottleneck. The organization can benefit from this information to do the following:

- By clustering flows that share the same bottleneck, the organization can discover the loss rate it experiences at each bottleneck and, when possible, buy capacity at the bottleneck where the traffic is facing continuous congestion.
- In case the organization has bought some bandwidth at the bottleneck, then discovering the flows that are using it allows the organization to differentially allocate the bandwidth to these flows according to its own policy. In other words, the gateway can control the feedback to the senders to encourage the

[1] For the topology in Figure 7 we needed around 20 packets per source. This topology is significantly more complex than the topologies simulated by other papers addressing the same topic [22,19].

senders with the higher priority to send more while force the lower priority senders to reduce their rates. In the case of long TCP sessions, the gateway can control the feedback by delaying acknowledgments sent to the low priority senders in comparison with those sent to the high priority senders.

- This information allows estimating the bottleneck's bandwidth and the organization's share of it. The already available tools for this task send probe packets, which overload the network unnecessarily.

The rest of the paper is organized as follows. The next section provides some necessary definitions. In Section 3, we present a high level description of our approach and some of the relevant issues. Sections 4 and 5 describe the data generation process and the data analysis. Guided by the insight gained from the data analysis, we present our clustering techniques in Section 6. Section 7 presents the results of testing the algorithms on the simulated data and section 8 describes the results of running the algorithms on Internet traffic.

2 Definitions

Before describing our methodology we provide the following definitions:

- Entropy: the concept of entropy is used as a measure of the uncertainty in a random variable. The entropy $H(x)$ of a discrete random variable x, with probability mass function $p(x)$ is defined by the following expression.

$$H(x) = -\sum_{x} p(x)\log_2 p(x)$$

- Flow: we define a flow to be the set of packets from the same source.[2] We assume that all packets in a flow follow the same route and consequently share the same bottleneck. This is a fair assumption given that the time scale for route changes is significantly larger than the lifetime of a flow [10].
- Cluster: we define a cluster to be a set of flows. For example, the cluster $\{S_i, S_j\}$ contains the packets from sources i and j ordered by their arrival time at the receiver.
- The correct cluster is a set of flows that share the same bottleneck.
- An incorrect cluster is any set of flows that is not a correct cluster.
- Inter-packet spacing: if we sort all the packets in a cluster according to their arrival times, then the inter-packet spacing is the difference in time between two consecutive packets. Note that inter-packet spacing is defined per cluster and not per source, so it doesn't matter whether the two packets under consideration belong to the same flow or not, as long as they belong to the same cluster. Our technique uses inter-packet spacing as the random variable whose entropy should be minimized.

3 Methodology

A bottleneck is the dominant congested router along a path. If we assume that the queue at the bottleneck router is almost never empty, then the output link is always used to the maximum capacity. Consequently, packets leave the bottleneck equally spaced and the time difference between the first bits of 2 consecutive packets is equal to the transmission time of 1 packet (packet_size / bottleneck_bandwidth.) Subsequent

[2] This is almost equivalent to defining a flow as the packets with the same source-destination pair because we only deal with flows received by our end system.

routers do not affect this spacing much since their queues are empty[3] for most of the time (please refer to Figure 1).

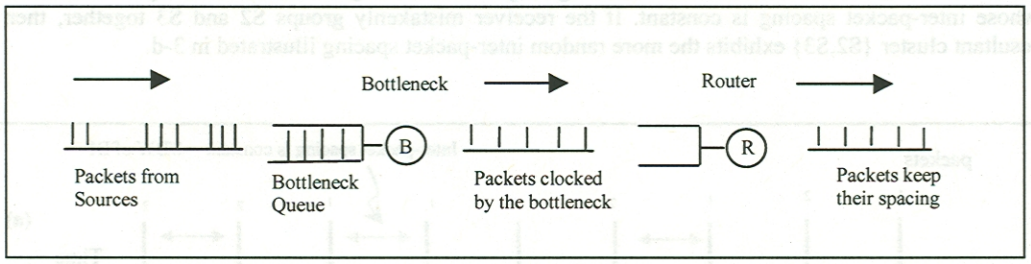


Figure1: No matter what is the original inter-packets spacing, packets leave a bottleneck equally spaced, and subsequent routers (to a high degree) preserve this spacing.

Before the packets reach the receiver they get mixed with traffic coming along other paths, and the uniform spacing is lost. The receiver sees random spacing and can't tell which packets were queued in the same queue and crossed the same bottleneck. Thus, to cluster the flows that share the same bottleneck the receiver has to recover the uniform spacing out of the noisy random spacing it sees. Assuming the receiver knows an upper bound on the number of bottlenecks, we present a mechanism that finds the correct clustering by minimizing the entropy of the inter-packet spacing of the clusters.

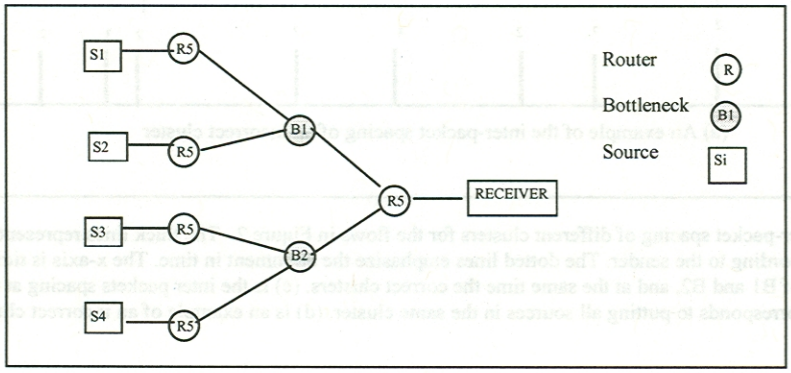


Figure 2: A simple clustering example; though the receiver receives all traffic over the same link, it wants to cluster S1 and S2 together, and S3 and S4 together.

We describe the intuition behind our method using the simple network in Figure 2. In this scenario, 4 sources send to the same receiver. S1 and S2 are behind the same bottleneck B1 and their total rate is larger than B1's output link capacity. S3 and S4 share the bottleneck B2 and their total rate exceeds B2's output-link capacity. The receiver receives all packets on the same link. Figure 3 illustrates the inter-packet spacing at different points in our simple topology. Figures 3-a and 3-b show the packet spacing at the

[3] Otherwise the subsequent router becomes the bottleneck causing TCP to reduce its window and alleviate the congestion at the old bottleneck.

output of B1 and B2 respectively. In addition, they represent the inter-packet spacing for the correct clusters ({S1,S2}, {S3,S4}). Figure 3-c shows the inter-packet spacing at the receiver, which is the overlaying of the output of B1 and that of B2. Note that 3-c does not show the nice constant spacing observed in 3-a and 3-b. If the receiver does a good job in clustering the flows it ends up with two clusters whose inter-packet spacing is constant. If the receiver mistakenly groups S2 and S3 together, then the resultant cluster {S2,S3} exhibits the more random inter-packet spacing illustrated in 3-d.

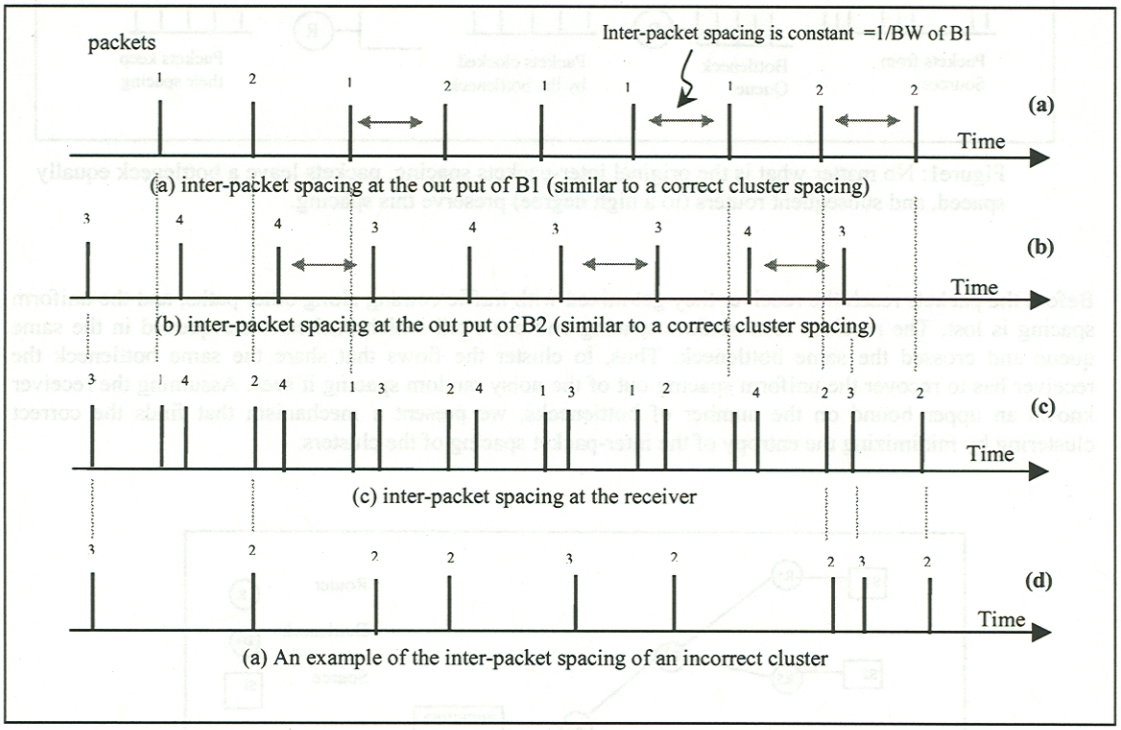


Figure 3: The Inter-packet spacing of different clusters for the flows in Figure 2. The thick lines represent packets, they are numbered according to the sender. The dotted lines emphasize the alignment in time. The x-axis is time. (a) and (b) are the outputs of B1 and B2, and at the same time the correct clusters. (c) is the inter packets spacing at the receiver, which corresponds to putting all sources in the same cluster. (d) is an example of an incorrect cluster.

Figure 4 shows the probability mass function (PMF) for the inter-packet spacing of the correct cluster in Figure 3-a {S1, S2}, and the incorrect cluster in Figure 3-d {S2, S3}. Because the correct cluster is similar to the output of the bottleneck B1, it has a uniform spacing. Consequently its PMF, which is illustrated in 4-a, shows one long pulse at a spacing that is equal to the time necessary to transmit a packet over the bottleneck link. On the other hand, clustering S2 and S3 together results in a random inter-packet spacing whose PMF, which is illustrated in 4-b, shows many small pulses at random locations. Definitely, the PMF in Figure 4-b has a higher entropy than the one in Figure 4-a. Thus, for this simple case the receiver can try all the possible clusters, and choose the solution that minimizes the total entropy (which is in this case {{S1, S2}, {S3, S4}}).

Finally, note that this approach does not use any assumptions about the exact bottlenecks' bandwidth nor about their queuing disciplines. Thus, it is effective even when the output links of the different bottlenecks have exactly the same bandwidth. (For example, B1 and B2 in Figure 2 have the same bandwidth.)

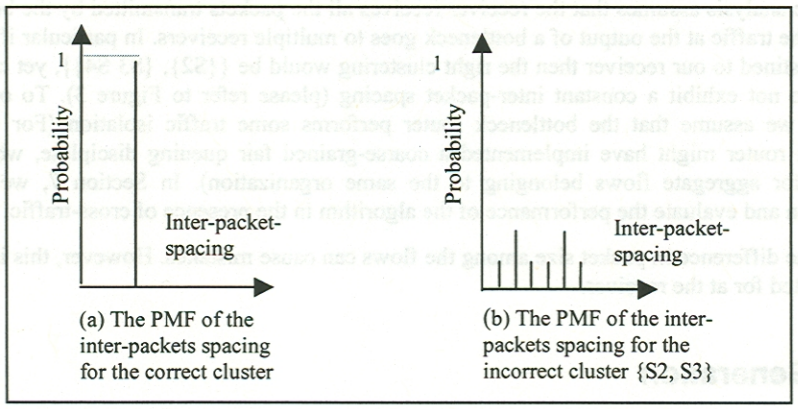


Figure 4: A comparison between the PMF of the inter-packet-spacing of a correct cluster {S2, S1} (shown in 3-a) and that for an incorrect cluster {S2, S3} (shown in 3-d)

In fact, the network given in Figure 2 is very simple and ignores many real life situations that render the problem significantly harder:

- In Figure 2, we assumed that the sources send enough to keep the bottleneck always busy. In practice, the size a bottleneck's queue fluctuates between zero and the maximum queue size. Thus, for most of the time the queue is not empty and the bottleneck shapes the traffic by clocking it as fast as its output link permits. Occasionally, the bottleneck output queue is empty, in which case the inter-packet spacing at its output corresponds to that of its input, which is usually the sources' traffic pattern. Since most sources are bursty TCPs, the PMF of the inter-packet spacing of the correct cluster, instead of being one long pulse as in Figure 4-a, exhibits many smaller pulses that follow a first big one. This effect is illustrated in Figure 5.
- Moreover, subsequent routers do not always have empty queues, which means that they occasionally re-space the packets. Because these routers usually have more bandwidth than the bottleneck, they compress packets that get queued at them. Thus, they cause the PMF of the correct cluster to show small pulses that precede the long pulse, which represents the bottleneck spacing. Figure 5 shows the PMF of the correct cluster {S2, S3} when S2, S3 are TCP sources and router R5 occasionally queues the packets.

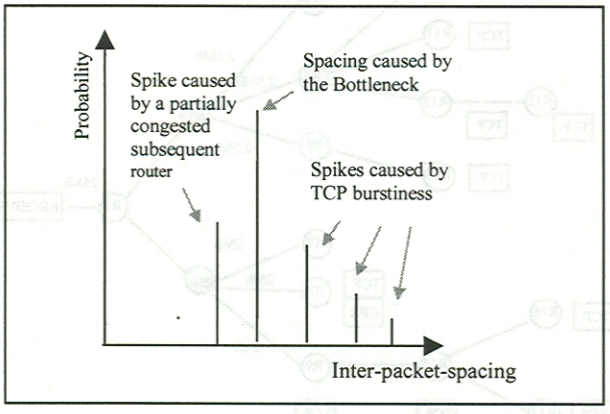


Figure 5: The PMF of inter-packet-spacing of a correct cluster with TCP traffic and a partially congested subsequent router.

- The above analysis assumes that the receiver receives all the packets transmitted by the bottlenecks. In real life, the traffic at the output of a bottleneck goes to multiple receivers. In particular if S1's packets are not destined to our receiver then the right clustering would be {{S2}, {S3 S4}}, yet cluster {S2}'s packets do not exhibit a constant inter-packet spacing (please refer to Figure 3). To overcome this difficulty we assume that the bottleneck router performs some traffic isolation (For example, the bottleneck router might have implemented a coarse-grained fair queuing discipline, which provides isolation for aggregate flows belonging to the same organization). In Section 7, we discard this assumption and evaluate the performance of the algorithm in the presence of cross-traffic.
- Finally, the difference in packet size among the flows can cause mistakes. However, this issue is easily compensated for at the receiver.

4 Data Generation

We generate our data using the VINT-ns-2 network simulator, which has extensive capabilities to simulate different topologies and different traffic patterns. Running the algorithms in a simulated environment enables us to control the receiver's share of the bottlenecks' bandwidth. In addition it facilitates the performance evaluation. This task becomes significantly difficult when we run the algorithm on Internet data because the available tools such as pathchar and traceroute do not allow us to find the correct clustering to compare it with the output of our algorithm. We start with the assumption that the receiver's share of the bottlenecks' bandwidth is not affected by cross-traffic. This simulates either an environment where the bottleneck router implements a coarse-grained fair queuing, or an environment where the receiver has bought some bandwidth at the bottleneck.

We generate data for the following environments:

- 2-bottlenecks, 16 sources, receiver share of the bottleneck is constant

We evaluate our method on the topology given in Figure 6. The network has 2 bottlenecks B1 and B2, 14 TCP sources with different round trip times and 2 CBRs (Constant Bit Rate source). We generate multiple data sets by changing the sources start times and the bottlenecks' bandwidth. By choosing each time a different permutation for the sources' start times, the arrival times for each flow and the interleaving of packets from different flows all differ from one trace to another. Since these are the only characteristics we use in our techniques, the traces generated as described result in different data points. Other characteristics of the environment are given in Table 1.

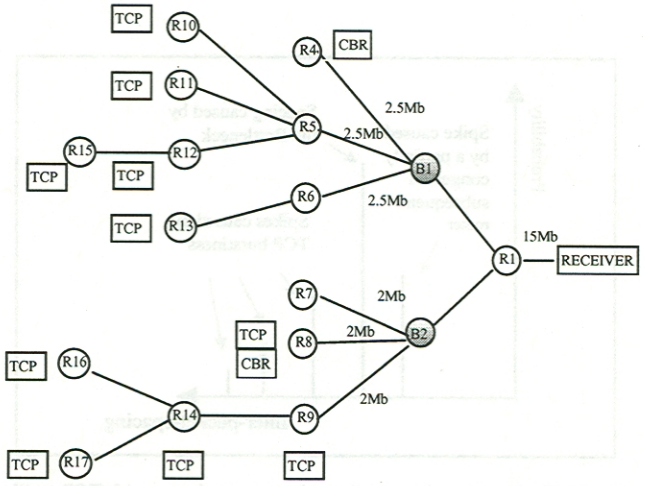


Figure 6: 2-bottlenecks with traffic isolation
There are 2 Bottlenecks B1 and B2 and 14 TCP sources and 2 CBR sources.

B1 output-bandwidth	1 Mb, 1.5Mb
B2 output bandwidth	0.5 Mb, 1 Mb, 1.5 Mb
CBR rate	20 Kb
TCP rate	Has always data to send
Queue Type	RED, Drop Tail

Table 1: Characteristics of the simulation environment
Different values in the same row are used in different runs.

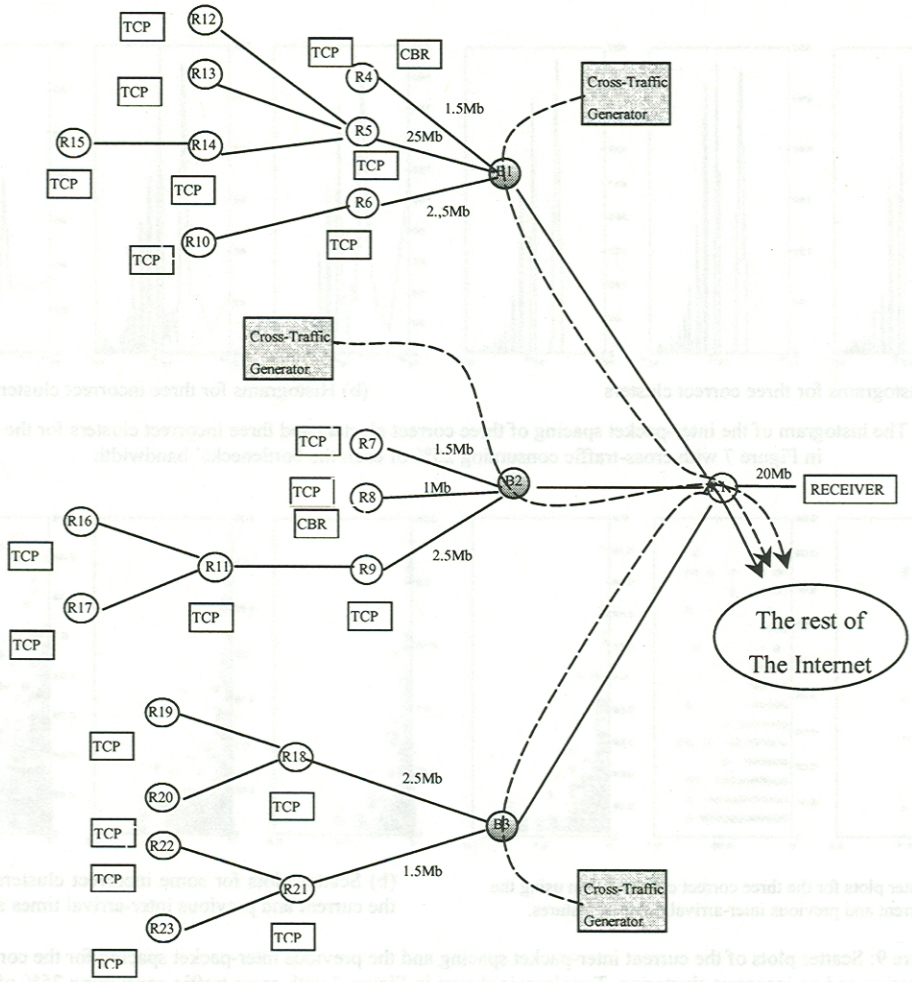


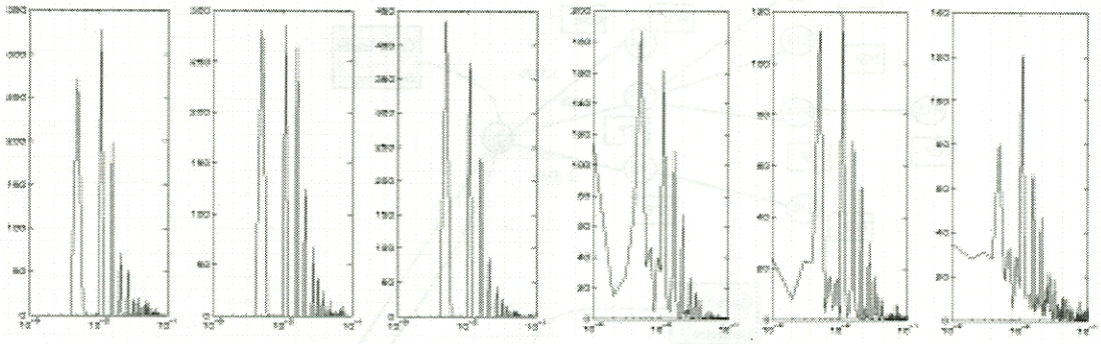
Figure7: 3-bottlenecks topology with cross-traffic
There are 3 bottlenecks B1 B2 and B3, 20 TCP sources and 2 CBR sources sending to the receiver. Sources are sharing the bottlenecks with the cross-traffic.

- 3-bottlenecks, 22 sources, with cross-traffic

We also test our method on the topology in Figure 7. We experiment with different dropping mechanisms such as DropTail and RED (Random Early Detection [6]). In these simulations the received flows are no longer isolated from the cross-traffic with which they share the bottleneck. Our cross-traffic generator is a combination of 20 Pareto sources with an on-off period that takes value in the range [10 msec, 1 sec]. Table 1 states the other characteristics of the simulation environment.

5 Data Analysis

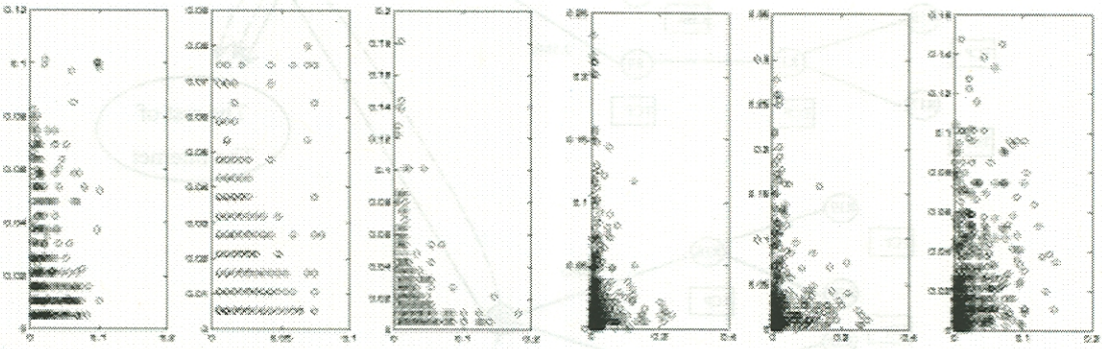
Our traces are arrays of 2 columns. Each row represents a packet. The first column is the time at which the packet is received. The second column is the source of the packet. This data is very simple to obtain in a real network setting. It requires only that the receiver log the arrival time and the source IP-address of each packet. We call each of these arrays a data set.



(a) Histograms for three correct clusters

(b) Histograms for three incorrect clusters

Figure 8: The histogram of the inter-packet spacing of three correct clusters and three incorrect clusters for the topology in Figure 7 with cross-traffic consuming 25% of each the bottlenecks' bandwidth



(a) Scatter plots for the three correct clusters when using the current and previous inter-arrival times as features.

(b) Scatter plots for some incorrect clusters when using the current and previous inter-arrival times as features.

Figure 9: Scatter plots of the current inter-packet spacing and the previous inter-packet spacing for the correct clustering and an incorrect clustering. Topology is shown in Figure 7 with cross-traffic consuming 25% of the bottlenecks' bandwidth

After generating the data, we examine the PMF of the inter-packet spacing for some of the correct clusters and the incorrect ones. The data generated by the simulator support our intuition described in Section 3.

Figures 8-a and 8-b show the histogram[4] of inter-packet spacing of the correct clustering and an incorrect clustering respectively. The data is for the topology in Figure 7, which exhibits three bottlenecks with cross-traffic. We notice that the PMFs (the histograms after normalization) of the 3 correct clusters, which are shown in Figure 8-a, are less random (have less entropy) than the PMFs of the 3 incorrect clusters illustrated in Figure 8-b. In particular, we notice the existence of a large pulse followed by smaller ones. We attribute the first pulse to the bottleneck clocking while the other ones are due to TCP burstiness.

In Figures 9-a and 9-b, we look at two features for each packet: the current inter-packet spacing and the previous one. The figure shows scatter plots for the correct clustering and an incorrect clustering of the sources in Figure 7. Adding the previous inter-packet spacing captures any dependency or correlation between successive data packets. We notice in 9-a that the inter-packet spacing jumps between a set of different values according to the size of the cross-traffic burst that is queued between 2 packets received by our receiver.

Both Figure 8 and Figure 9 indicate that the PMF of the inter-packet spacing of the correct clustering has more structure than that of an incorrect clustering. Thus, a clustering technique based on entropy-minimization is plausible.

6 Clustering Techniques

In this section, we describe a set of entropy-based techniques for clustering flows according to their common bottleneck. We start by motivating the need for iterative clustering techniques, then we describe a variety of iterative clustering techniques in their convergence properties. The techniques discussed below are written in MATLAB, which provides some mathematical functions useful for our task. Their performance is discussed in Section 7.

6.1 Design Issues

In order to develop an entropy-minimization-clustering algorithm, two main design issues need to be resolved. The first issue is the computational complexity of the problem. In other words, the brute force approach, which looks at all possible clustering of the sources (flows), to find the clustering that minimizes the entropy, is exponential in the number of flows. Consequently, it does not scale to a reasonable number of flows. For example, a brute force approach to clustering the flows in the small topology of Figure 6 would require 2^{16} iterations (the number of bottlenecks to the number of souces). In order to reduce the computational complexity, we use iterative procedures that start with some initial random clustering and then move the sources among clusters to obtain an incremental reduction in the entropy.

The second issue is choosing the exact formula of the function to be minimized. Equation 1 tells us how to compute the entropy of the packet inter-spacing for a cluster. However, it does not indicate how to combine the entropies of the different clusters into a quantity that we can minimize. We call the quantity we want to minimize the 'Cost Function'. Different choices of the cost function exhibit different performance and robustness, as it is shown in Section 6.2 and Section 7.2.

6.2 Iterative Clustering Techniques

Iterative clustering procedures start with an initial random clustering and iterate according to some rule to cause an incremental reduction in a cost function.

There are two general classes of clustering techniques. The first class looks at the distance (dissimilarity) between pairs of the clustered entities to find the clustering that minimizes the average distance between any two entities in the same cluster. The second class of clustering techniques looks directly at the distance between a clustered entity and an entire cluster (usually the center of the cluster) and iterates by reassigning

[4] A histogram is a non-normalized PMF.

an entity to the closest cluster. The first algorithm described in the next section belongs to the pairwise class, whereas, the algorithms in subsequent sections directly compare a flow to a cluster to decide whether the flow belong to the cluster or not.

All of the techniques described are iterative. Thus, it is important to emphasize the time it takes a technique to stabilize and compare it to the time necessary to do a full search of the solution space (the brute force approach). The algorithms differ in the iterating rule they use and the cost function they attempt to minimize.

6.2.1 Iterative Pairwise Entropy-Based Clustering

In this technique, we look at the similarity between *pairs* of flows to find the clustering that maximizes the average similarity between any two flows in the same cluster. Thus, our cost function is

$$Cost = \frac{1}{N} \sum_{i=1}^{i=N_f} \frac{1}{N_f - 1} \sum_{all\ j \neq i} H_{ij},$$

N_f is the total number of flows, N is the number of clusters, and H_{ij} is the entropy of the inter-packet spacing of the cluster $\{S_i, S_j\}$.

The following pseudocode details the approach:

1. Pick a random clustering for initialization
2. On each iteration:
3. Pick a source S_i (Either at random or round-robin)
4. For each source S_j different from S_i:
5. Find the entropy H_{ij} of the cluster $\{S_i, S_j\}$ resulting from
6. clustering the packets of two sources together.
7. Move S_i to the cluster with the minimum average cross
8. entropy.
9. Repeat until no sources change their cluster.

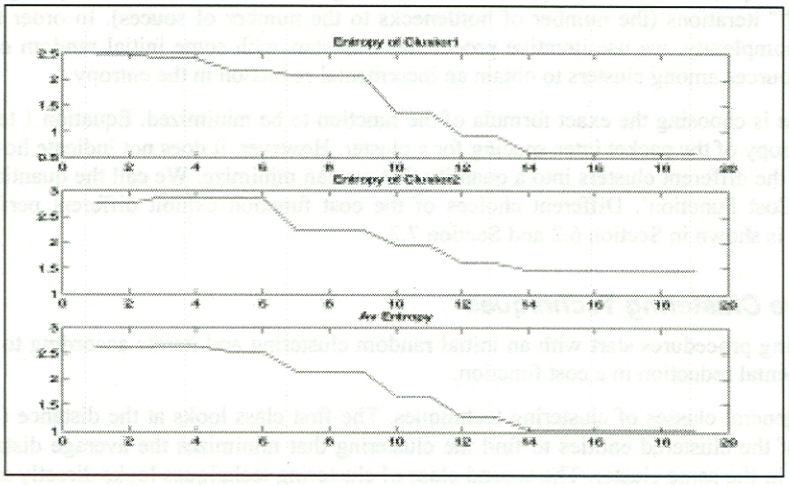


Figure 10: Convergence properties of the iterative pairwise entropy-based technique
A typical run of the technique on the topology in Figure 6. The x-axis is the number of iterations and the y-axis is the entropy.

Figure 10 illustrates the convergence properties of the iterative pairwise entropy-based technique. It shows a typical run of the algorithm on the 2-bottleneck topology in Figure 6. The x-axis is the number of iterations while the y-axis is the entropy. The figure shows that although the space of possible solutions is 2^{16} (the topology has 16 sources and 2 bottlenecks) the pairwise algorithm takes only 14 iterations to stabilize.

6.2.2 Iterative Entropy-Based KMeans Clustering

In contrast to the pairwise technique, which looks at the pairwise entropy, the clustering techniques described in this section compute the cost function by directly comparing a flow to an entire cluster. Since this concept is similar to that used by the well-known KMeans clustering technique[5] when distance is measured using entropy, we call the set of procedures described in this section the Entropy-Based KMeans Clustering techniques.

An important design issue is to come up with a robust cost function. One possible approach is to minimize the average entropy of the clusters as in the above section. Unfortunately, using the average entropy has a tendency towards putting all flows in the same cluster. For example, assume that the maximum number of bottlenecks is 3 and that the correct clustering results in 3 clusters (3 bottlenecks) whose entropies are 2, 2, and 3, and that putting all of the flows in one cluster results in a cluster whose entropy is 6 and two empty clusters with zero entropy. Thus, although the incorrect cluster has higher entropy than any of the correct clusters the average entropy might be minimized by generating many empty clusters. We can counter this effect by weighting the entropy of each cluster by a factor that reflects the number of sources in the cluster. We experiment with the following cost functions.

1. The Cluster-Weighted KMeans Technique

The cost function is a weighted average of the entropies of the clusters, where the weighting factor is the number of sources in the cluster.

$$Cost = \frac{1}{N} \sum_{c=1}^{c=N} N_c \; H_c$$

N_c is the number of sources in cluster c, H_c is the entropy of cluster c, N is the number of clusters.

2. The Sample-Weighted KMeans Technique

The cost function is a weighted average of the entropies of each cluster, where the weighting factor is the number of packets in the cluster.

$$Cost = \frac{1}{N} \sum_{c=1}^{c=N} NS_c \; H_c$$

NS_c is the number of samples (packets) in cluster c, H_c is the entropy of cluster c, N is the number of clusters.

The clustering procedure (regardless of which of the cost functions is used) starts with some initial clustering for the sources (a guess). On each iteration, it picks a source in a round robin fashion and tries to

[5] KMeans is a well-known clustering Technique [2] that works as follows. Let' s assume that we have a set of points in a plane and we want to cluster them in K classes. We begin by assigning the points at random to K sets and computing the x and y coordinates of the mean point in each set. Next, each point is re-assigned to a new set according to which is the nearest mean vector. The means of the sets are then recomputed. The procedure is repeated until there is no further change. Although, entropy does not have all of the properties of a distance (does not satisfy the triangle's inequality) the similarity between our technique and the KMeans approach suggests the naming.

change its cluster to obtain a lower cost of the clustering. The procedure can be best described with the following pseudocode:

1. Pick a random clustering for initialization
2. On each iteration:
3. Pick a source S_i (round-robin)
4. Remove Si from its cluster
5. For each cluster C_j:
6. Add S_i to C_i and compute the cost of the clustering.
7. Move S_i to the cluster that results in a minimum cost.
8. Repeat until no sources change their cluster.

Figure 11 illustrates the convergence properties of the iterative cluster-weighted KMeans technique. It shows a typical run of the algorithm on the 3-bottleneck topology in Figure 7. The x-axis is the number of iterations while the y-axis is the entropy. The figure shows that although the space of possible solutions is 3^{22} (the topology has 22 sources and 3 bottlenecks) the algorithm takes only 30 iterations to stabilize. The sample-based KMeans technique has similar convergence properties.

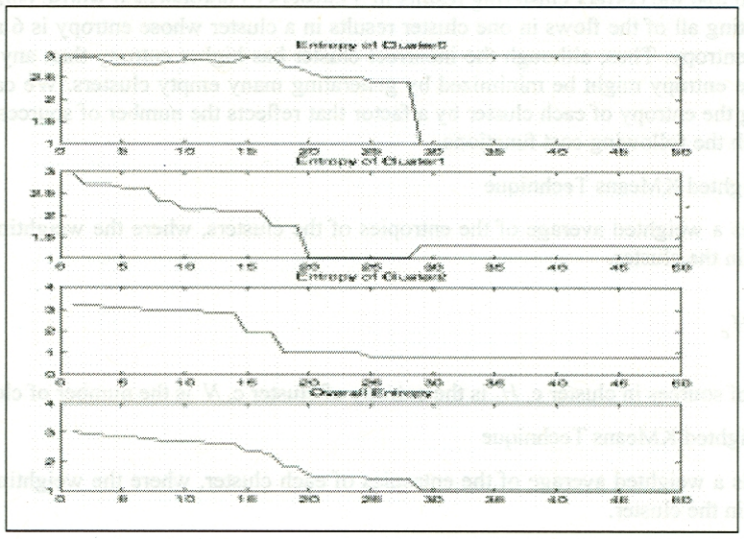


Figure 11: Convergence properties of the iterative KMeans entropy-based techniques
A typical run of the cluster-weighted technique on the topology in Figure 7. The x-axis is the number of iterations and the y-axis is the entropy. The sample-based technique exhibits similar convergence properties.

6.3 Using 2 Dimensional Feature

The above techniques use one feature that is the inter-packet spacing in a cluster. However, Figure 9, which shows scatter plots of the current inter-packet spacing and the previous inter-packet spacing, indicates that using a two dimensional feature might increase the performance. Modifying the techniques to use a 2D feature is fairly easy. Instead of computing the entropy of the PMF of the random variable representing the inter-packet-spacing, we compute the entropy of the PMF of a random vector whose first component is the current inter-packet-spacing and the second component is the previous inter-packet spacing. This approach allows the clustering technique to use the time correlation in the data.

7 Performance

Our clustering techniques found the scenario in Figure 6 trivial (2 bottlenecks, 16 sources, and no cross-traffic). For example, the 2D KMeans Clustering technique showed 100% accuracy over all of the 15 random runs we tried. Thus, in this section we only report the results of running the algorithms on the scenario in Figure 7, which contains 3 bottlenecks, 22 sources, and cross-traffic. We change the cross-traffic's average share of the bottlenecks' bandwidth from 0% to 85%. For each different cross-traffic's rate, we generate 5 data sets by changing the permutation of the source's start times and the bottlenecks' bandwidth. We run all the techniques presented in section 6.2 on all the data sets and report the results.

7.1 Error Computation

We measure performance by counting the number of sources that are correctly classified. For example if the correct clusters are C1={S1, S2, S3, S4} and C2={S5, S6, S7, S8} and the hypothesis we get from an approach is C1={S1, S2, S3, S4, S7} and C1={S5, S6, S8}, then the performance is 87.5% (7 out of 8).

7.2 Results

Table 2 summarizes the results of the four techniques described in Section 6.2 as a function of the percentage of traffic going to the receiver. Figure 12 shows the corresponding plots for the performance in Table 2.

The performance is best with the 2D KMeans approach under all traffic patterns. The key observation is that the performance is significantly high (90% to 99%) when the receiver's share of the bottlenecks' bandwidth is large, yet it degrades significantly as the fraction of the bottleneck's traffic going to the receiver decreases to less than 15%. This happens because the cross-traffic plays the role of noise for our purpose. As the cross-traffic's share of the bandwidth increases the received data becomes more and more immersed in noise.

Another important observation is that the 2D KMeans technique outperforms the other techniques. This means that the temporal correlation between the current inter-packet spacing and the previous one holds useful information for the purpose of separating flows sharing the same bottleneck. It also indicates that using a 3 Dimensional or a 4 Dimensional feature that captures more of the temporal correlation might increase the performance further and render the technique more robust in the face of a heavy cross-traffic.

We also note that our technique requires a small number of packets (around 20 packets per flow). Thus, the fact that current Internet transfers have a small mean duration should not affect the performance significantly.

Technique \ % of bottleneck's traffic delivered to the receiver	100%	60%	45%	30%	20%	15%
2D Cluster-Weighted KMeans	98.5%	98.4%	97.2%	90.1%	89.3%	81.1%
Sample-Weighted KM	94.7%	94.7%	88.8%	84.7%	79.0%	61.0%
Cluster-Weighted KM	93.6%	92.5%	89.6%	88.3%	79.7%	63.5%
Iterative Pairwise	81.0%	81.5%	80.2%	74.8%	53.1%	43.2%

Table 2: Performance results (accuracy of each technique)

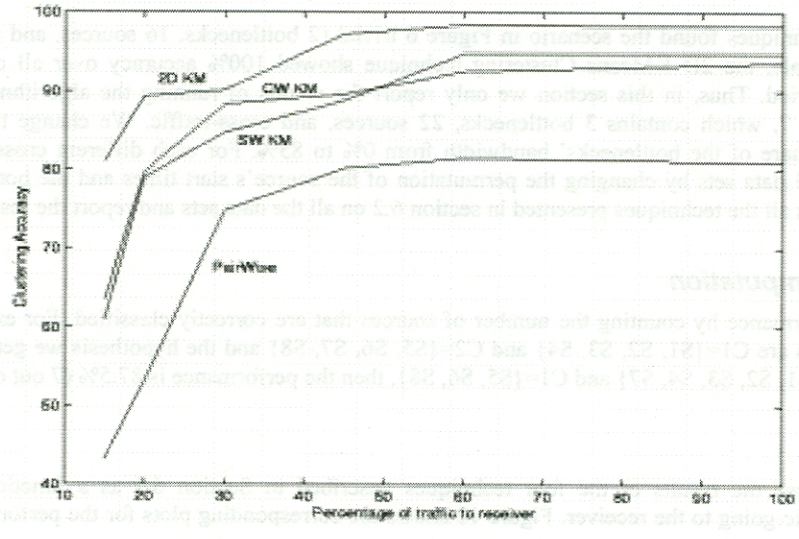


Figure 12: Performance Results
PairWise: Iterative PairWise; CW: Cluster-Weighted KMeans; SW: Sample-Weighted KMeans, 2D KM: 2Dimentional Cluster-Weighted KMeans

8 Testing on Internet Traffic

This part of our work is still under development. Nevertheless, the initial results, which we report in this section, are promising. Note that it is intrinsically hard to test the algorithm on Internet traces since our receiver is not guaranteed any significant share of the bottleneck's bandwidth. To collect our data, we ftp some files from a set of web servers and mark the arrival times using tcpdump [15]. We run our experiments at night in order to increase our chance in obtaining a considerable share of the bottleneck's bandwidth.

We have tested the algorithm on clustering flows sent by the servers in Table 3. Our choice of the servers was dictated by the desire to make the correct clustering intuitive such that it is easy to decide whether the algorithm did the right job or not. Our connection to the three Californian servers goes through vbns, while our connection to Harvard goes directly through media-one. Thus, with a high confidence, we know the correct clustering should separate the Californian servers from the ones at Harvard.

Table 4 gives the results of our experiment.

Mash.cs.berkeley.edu
Amber.berkeley.edu
Www2.stanford.edu
www.harvardfas.harvard.edu
Radcliffe.harvard.edu
Gseweb.harvard.edu
www.hno.harvard.edu
Jupiter.harvard.edu
Divweb.harvard.edu

Table 3: sites involved in first experiment.

Time	Accuracy
12am	1 misclassified source, 8 correct
2 am	1 misclassified source, 8 correct
4am	1 misclassified source, 8 correct
6am	2 or 3 misclassifications depending on initial guess

Table 4: Results of running the 2D KMeans technique the transfers from the sites in Table 3

9 Conclusion

We presented a set of entropy-based techniques for clustering flows that share the same bottleneck. In contrast to other approaches, these techniques are completely passive and do not generate any probe traffic. Moreover, they are effective with both TCP and UDP traffic. Finally, they can be used with short-lived flows since they require only a small number of packets.

The approach shows significantly high performance when the fraction of the bottleneck traffic delivered to the receiver is large, yet it degrades considerably as the receiver's share of the bottleneck bandwidth decreases below 15%. Thus, we think that in its current state our technique is a useful tool to guide ISPs in designing their networks, but its applicability to congestion control is confined to environments where the traffic enjoys some isolation. Nonetheless, the results of our simulation shows that it is likely that using a higher dimensional feature that captures more of the temporal correlation in the data might render the technique more robust in the face of a heavy cross-traffic.

10 References

1. A. J. Bell and T. J. Sejnowski, "An Information Maximization Approach to Blind Separation and Blind Deconvolution, *In Proc. of ICASSP'95*, 1995.
2. C. M. Bishop, " Neural Network for Pattern Recognition," Clarendon Press, Oxford, 1997.
3. J. Bolot, "End-to-end Packet Delay and Loss behavior in the Internet," *In Proc. of SIGCOMM'93*, Sep 1993.
4. R. Cater and M. Crovella, "Measuring Bottleneck link Speed in Packet-Switched Network," Technical Report TR-96-006, Boston University, Mar. 1996.
5. T. M. Cover, J. A. Thomas, "Elements of Information Theory," Wiley Series in Telecommunications, 1991.
6. S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," IEEE/ACM Transactions on Networking, V.1 N.4, August 1993.
7. V. Jacobson, "Congestion Avoidance and Control," *In Proc of SIGCOMM'88*, Aug. 1988.
8. S. Keshav, "Congestion Control in Computer Networks," Thesis Dissertation, Sep. 1991.
9. pathchar – A tool to infer characteristics of Internet paths. ftp://ee.lbl.gov/pathchar.tar.Z, 1997.
10. V. Paxson, "Measurements and Analysis of End-to-end Internet Dynamics," Thesis Dissertation, 1997.
11. V. Paxson et al., "An Architecture for Large Scale Internet Measurements," *IEEE Communications Magazine*, To appear 1998.
12. S. Ratnasamy and S. MacCanne, "Inference of Multicast Routing Trees and Bottleneck Bandwidths using End-to-end Measurements, *In Proc of INFOCOM'98*, Mar. 1998.

13. C. E. Shannon, "A Mathematical Theory of communication," Bell Sys. Tech Journal, 1948.

14. M. Stemm, R. Katz, and S. Seshan, "SPAND: Shared Passive Network performance Discovery," http://spand.cs.berkeley.edu.

15. tcpdump – the protocol packet capture and dumper program, http://ee.lbl.gov/tcpdump.tar.Z.

16. traceroute – a tool for printing the route packets take to a network host, http://ee.lbl.gov/traceroute.tar.Z.

17. P. Viola and W. M. Wells III, "Alignment by Maximization of Mutual Information," International Journal of Computer Vision , 1997

18. P. A. Viola , N. N. Schraudolph and T. J. Sejnowski," Empirical Entropy Manipulation for Real-World Problems, " Advances in Neural Information Processing , 1995.

19. R. Caceres, N. G. Duffield, J. Horowitz, D. Towsley, and T. Bu, "Multicast-Based Inference of Network-Internal Characteristics: Accuracy of Packet Loss Estimation," In Proc of INFOCOM'99, 1999.

20. M. Allman and V. Paxson, "On Estimating End-to-End Network Path Properties," *In the Proc. of SICGOMM'99*, 1999.

21. Allen Downey, "Using Pathchar to Estimate Link Characteristics," *In the Proc. of SICGOMM'99*, 1999.

22. D. Rubenstein, J. Kurose, and D. Towsley, "Detecting Shared Congestion of Flows Via End-to-End Measurements," *In Proc. of SIGMETRICS'00*, 2000.