A Passive Approach for Detecting Shared Bottlenecks

Dina Katabi, Issam Bazzi, Xiaowei Yang

MIT Laboratory for Computer Science

Abstract **-** There is a growing interest in discovering Internet path characteristics using end-to-end measurements. However, the current mechanisms for performing this task either send probe traffic, or require the sender to cooperate by time stamping the packets or sending them back-to-back. Furthermore, most of these techniques require the packets to carry sequence numbers to detect losses, and a few of them assume the existence of multicast.

This paper introduces a completely passive approach for learning Internet path characteristics. In particular, we show that by noting the time difference between consecutive packets, a passive observer can cluster the flows into groups, such that all the flows in one group share the same bottleneck. Our approach relies on the observation that the correct clustering minimizes the entropy of the inter-packet spacing seen by the observer. It does not inject any probe traffic into the network, does not require any cooperation from the senders, and works with any type of traffic whether it is TCP, UDP, or even multicast.

1 Introduction

The Internet literature contains a large body of research that addresses the problem of learning Internet path characteristics using end-to-end measurements. The vast majority of this research requires the end system conducting the measurements to send probe traffic along the path whose characteristics it wants to learn [3,4,10,19,21]. The few previous mechanisms that do not generate probe traffic are not completely passive because they require the sender to cooperate by time stamping the packets [10,22] or sending them back-to-back [12,23]. Furthermore, most of the proposals for inferring path characteristics from end-to-end measurements are not applicable to standard UDP traffic because they assume a packet loss is detectable [12,19,22,23], and some of them work only with multicast traffic [19].

This paper examines the amount of information a completely passive observer can learn about the characteristics of the upstream paths. We use the term "passive observer" to refer to a receiver or an intermediate node that attempts to learn the characteristics of the upstream paths without sending any probe traffic and without requiring any cooperation from the senders. A passive observer makes no assumptions about the content of the packets or the protocol the end systems use. In particular, a passive observer should not assume that the packets carry sequence numbers, and consequently it cannot rely on losses being detectable. Also, a passive observer should not assume that the upstream routers use a particular queuing discipline such as Drop-Tail or RED (Random Early Detection [6]). It should work with any queuing discipline and any transport protocol.

In this paper, we show that such a completely passive observer can cluster the flows into groups that share the bottleneck, and consequently, can learn substantial information about the topology and the congestion-state of the upstream network. Detecting shared bottlenecks using end-to-end measurements is useful for sharing congestion information [22,23], constructing the topology [12], and monitoring and debugging the network. The importance of studying the problem of *passive* detection of shared bottlenecks is twofold. First, a passive approach to detecting shared bottlenecks is highly desirable because it is resource efficient (i.e., it does not generate probe traffic) and it is extremely general (i.e., it makes no assumptions about the transport protocols or the queuing discipline). As such, it is beneficial to examine the accuracy of this approach and the situations in which it is applicable. Second, studying the capabilities of a completely passive observer helps discovering potential covert channels that allow a malicious downstream observer to collect substantial information about the topology and the congestion-state of the upstream network. Since a passive observer is hard to detect, the best way to throttle an attack by such an observer is to understand the capabilities of the passive observer and take preventive measures.

This paper introduces a class of techniques that allow a passive observer to use only the time difference between consecutive packets to cluster the flows into groups that share the bottleneck. Our approach relies on the observation that the correct clustering minimizes the entropy of the inter-packet spacing seen by the observer. It does not generate any probe traffic, works with TCP UDP and multicast flows, and does not assume any particular queuing discipline. Our simulations show that passive detection of shared bottlenecks using entropy-minimization is highly accurate when the observer is strategically located so that a sizeable fraction of the output traffic of the bottlenecks eventually traverses the monitored link. However, the passive approach becomes inaccurate when only a small fraction of the bottlenecked traffic traverses the monitored link, which makes it impractical as a means for an end receiver to discover flows crossing a common bottleneck and share their congestion information.

Also, by studying the capabilities of the passive observer, this paper reveals the existence of covert channels that a malicious strategically located observer can use to learn sensitive information about the topology of a private network. As an example, consider an army base located far from the main office and possibly on a foreign land. To contact its main office using the Internet, the base encrypts its packets including the TCP header and inserts random delays and dummy packets in each flow. This paper shows that these counter measures, though sufficient to throttle previous proposals for detecting shared bottlenecks [22,23], do not

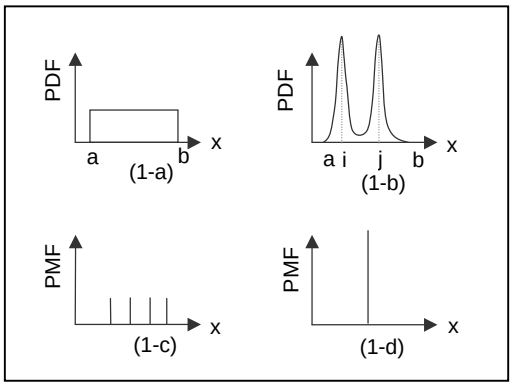


Figure 1: The entropy increases with the flatness of the probability distribution function. The continuous random variable whose PDF is shown in 1-a has higher entropy than the continuous random variable whose PDF is shown in 1-b. Similarly, the discrete random variable whose PMF is shown in 1-c has higher entropy than the discrete random variable whose PMF is shown in 1-d.

prevent a spying agent at the ISP from constructing useful information about the topology and the congestion state of the network at the base.

The paper is organized as follows. The next section provides some useful definitions. Section 3 describes our methodology. Section 4 presents our entropy-minimization techniques for clustering flows into groups that share the bottleneck. Performance is discussed in Sections 5. Section 6 discusses related work and Section 7 presents our conclusion.

2 Definitions

We start by providing the following useful 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, which has a probability mass function $p(x)$, is defined by the following expression.

$$H(x) = -\sum_{x} p(x)\log_2 p(x) \qquad (1)$$

As a result of the definition, the flatter the probability distribution of the random variable, the higher its entropy will be. For example, the distribution in Figure 1-a has higher entropy than the distribution in Figure 1-b because there is more uncertainty in the value of the random variable. Said differently, by looking at the distribution in 1-a, one cannot make a good guess of the value of the random variable since it is equally likely to take any value in the interval $[a,b]$. However, by looking at the distribution in Figure 1-b, one can say with high confidence that the random variable is either i or j. Similarly, the distribution in 1-c has higher entropy than the distribution in 1-d.

Observer: For the purpose of this paper the terms: "observer", "receiver", and "intermediate receiver" are equivalent.

Flow: We define a flow as the set of packets from the same source.[1] We assume that all packets in a flow follow the same upstream route and consequently share the same bottleneck. This is a reasonable assumption given that the time scale over which Internet routes change is significantly larger than the lifetime of a flow [10].

Cluster: We define a cluster as a set of flows. For example, the cluster $\{S_i, S_j\}$ contains the packets sent by sources i and j ordered according to their arrival time at the observer. We use the term "correct cluster" for a set of flows that share the same bottleneck, and the term "incorrect cluster" for any set of flows that is not a correct cluster.

Inter-packet spacing: If one orders all the packets in a cluster according to their arrival time at the observer, then the inter-packet spacing is the time difference between the arrivals of two consecutive packets divided by the size of the first packet in the pair. Note that inter-packet spacing is defined per cluster; thus, it does not matter whether the two packets 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.

Bottleneck: We say that a link is a bottleneck over the interval T, if its input traffic is continuously larger than its capacity except for short intervals compared to the size of the interval T. Sometimes we use the word "bottleneck" for the router attached to the bottleneck link or for the output queue of the bottleneck link.

3 Approach

Given that the input traffic at a bottleneck is larger than its capacity, the bottleneck link is busy processing packets most of the time. Consequently, packets leave the bottleneck equally spaced and the inter-packet spacing between two

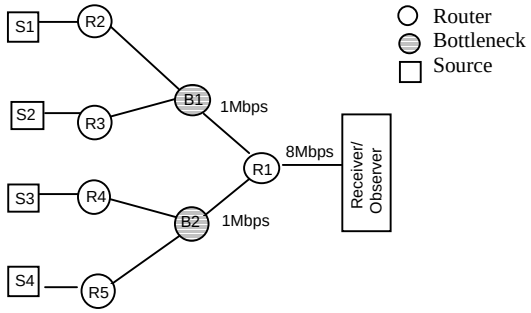


Figure 2: A simple clustering example; the observer is collocated with the receiver. It receives all flows over the same link, but it wants to cluster S1 and S2 together, and S3 and S4 together.

[1] It is possible to define a flow as the set of packets that have the same source-destination pair. However, such a definition ignores that flows sharing the sender and observed by the same observer necessarily share the upstream path.

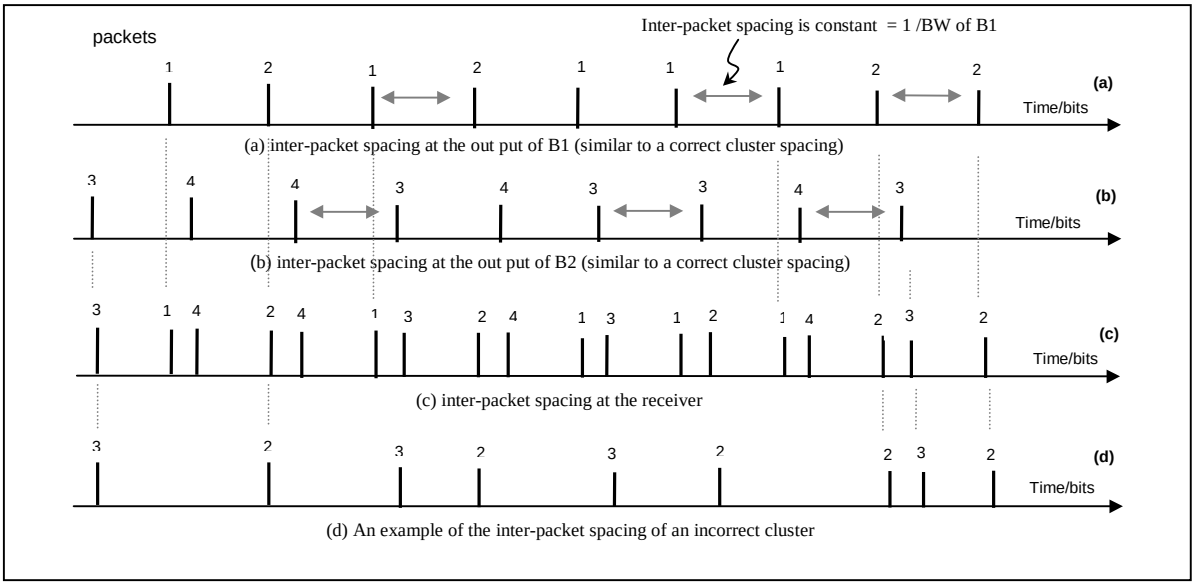


Figure 3: The Inter-packet spacing of various clusters of the flows in Figure 2. The thick lines represent packets, they are numbered according to the sender. The dotted lines emphasize the alignment in time (normalized time). The x-axis is time. (a) and (b) are the outputs of B1 and B2, and the correct clusters. (c) is the inter-packet spacing at the observer, which corresponds to putting all sources in the same cluster. (d) is an example of an incorrect cluster. (Plots are simplified by assuming the queue at router R1 is empty.)

consecutive packets is the inverse of the bottleneck bandwidth.[2] Subsequent non-bottleneck routers, which frequently have empty queues, preserve some of this spacing. Before the packets reach the observer, they get mixed with traffic that did not cross the same bottleneck, which destroys the constant spacing. The observer sees random spacing and cannot tell which packets crossed the same bottleneck. Thus, to discover the flows that share the bottleneck, the observer has to recover the constant inter-packet spacing from the noisy random spacing it observes. Assuming the observer 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.

We describe the intuition behind our approach using the simple network in Figure 2. In this scenario, four sources send to the same receiver. S1 and S2 are behind the same bottleneck B1, and their total sending rate is larger than the capacity of B1. S3 and S4 share the bottleneck B2 and their total rate exceeds its capacity. In this experiment, the receiver plays the role of the passive observer. It receives packets form all four sources on the same link yet wants to group the sources that share the same bottleneck together.

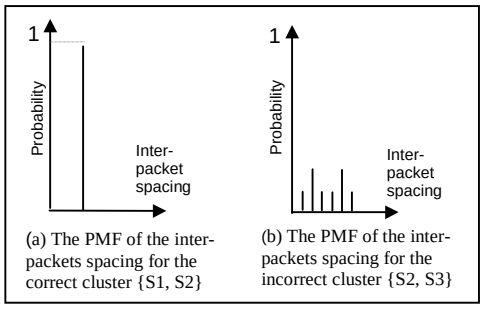
Figure 3 shows the inter-packet spacing at different points in our simple topology. Figures 3-a and 3-b show the inter-packet spacing at the output of B1 and B2 respectively. In addition, they represent the inter-packet spacing of the correct clusters ({S1,S2} and {S3,S4}). Figure 3-c shows the inter-packet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

Figure 4 compares the probability mass function (PMF) of the inter-packet spacing of the correct cluster in 3-a (i.e., {S1, S2}) to the PMF of the incorrect cluster in 3-d (i.e., {S2, S3}).[3] Because the correct cluster is similar to the output of the bottleneck B1, it shows a constant spacing. Consequently, its PMF, illustrated in 4-a, shows one long spike at the inter-packet spacing that corresponds to the inverse of the bottleneck capacity. On the other hand, clustering S2 and S3 together results in a random inter-packet spacing whose PMF, illustrated in 4-b, shows many small spikes at random locations. Definitely, the PMF in Figure 4-b has higher entropy than the one in 4-a. Thus, one can find the correct clusters by looking for the clustering that has the minimum entropy.

The simple scenario in Figure 3 is useful for explaining the intuition behind our entropy minimization approach but it does not reveal the full complexity of the problem. Passive

[2] Recall that the inter-packet spacing is the time difference between consecutive packets normalized by the size of the first packet in the pair.

[3] The graphs in Figure 4 are a simplified version of the graphs that we would have obtained if we simulated the scenario in Figure 2 and plotted the simulated data. The simplification approximates the distribution and does not change the characteristics of the graphs.



(a) The PMF of the inter-packets spacing for the correct cluster {S1, S2}

(b) The PMF of the inter-packets spacing for the incorrect cluster {S2, S3}

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

detection of shared bottlenecks is a significantly difficult problem for the following reasons. First, the dynamics of TCP and the occasional queuing at routers downstream a bottleneck both add randomness to the inter-packet spacing of the correct clusters. For example, when a relatively small number of TCPs share a Drop-Tail bottleneck, the bottleneck link might cycle between periods of severe congestion followed by a huge number of drops followed by periods of underutilization. During the periods of underutilization, the bottleneck does not clock the packets and the inter-packet spacing at its output is not constant. However, these periods of underutilization are unlikely when the number of flows is large. Furthermore, their duration at a bottleneck is relatively short compared to the duration of the periods in which the bottleneck clocks the packets. Similarly, routers downstream from a bottleneck occasionally build queues without being bottlenecks. Hence they might re-space the packets (i.e., clock them according to their capacity). If the queue at such a downstream router is almost never empty then the entropy-based mechanism will identify the router as a shared bottleneck. (In fact, identifying this router as a bottleneck is consistent with our definition of a bottleneck but it might not be relevant for sharing congestion information given that there is an upstream bottleneck that is dropping the packets). However, if the queues between the bottleneck and the observer are occasionally empty, then the entropy minimization techniques can potentially find the correct clustering. For example, in Figure 2, if router R1 continuously has a non-empty queue then the entropy is minimized by clustering S1, S2, S3, and S4 together. However, if the queue at router R1 is occasionally empty then the cluster {S1,S2,S3,S4} will show one big spike that corresponds to the inter-packet spacing of R1, and a number of random spikes.[4] On the other hand, the cluster {S1,S2} (and similarly the cluster {S3,S4}) will show only two spikes: one that corresponds to the inter-packet spacing of B1; and one that corresponds to the inter-packet spacing of R1. Thus, if the queue at R1 is sometimes empty then the entropy is minimized by splitting {S1,S2,S3,S4} into two clusters {S1,S2} and {S3, S4}.

[4] The reader should not confuse this situation with that in Figure 3-c. Figure 3-c assumes that R1 has always an empty queue.

Another issue that complicates the passive detection of shared bottlenecks is that, in most cases, only a small fraction of the bottleneck output traffic ends up traversing the link monitored by the observer. For example, in Figure 2, if the packets sent by S1 do not cross the link monitored by the observer then the correct clustering is {{S2}, {S3 S4}}. In this case, though the cluster {S2} does not exhibit a constant inter-packet spacing, the observer is likely to figure out the correct clustering. In particular, although the cluster {S2} has high entropy, any attempt to put S2 in the same cluster with S3 or S4 (or to put S3 and S4 in different clusters) is likely to further increase the entropy (i.e., the randomness) of the clustering. In Section 5.3, we plot the accuracy of the entropy-minimization techniques as a function of the fraction of traffic that traverses the link monitored by the observer.

The third difficulty in clustering flows that share the bottleneck arises for the fact that the clustering problem scales badly with size of its input. (It is an NP-complete problem [24].) This is a characteristic of the clustering problem even when clustering is done based on loss or delay correlation as in previous non-passive approaches [22,23]. For example, in the simple topology of Figure 2, finding the clustering that minimizes the entropy can be done by examining all the possible clusters and choosing the solution that minimizes the total entropy. Such a brute force approach grows exponentially with the size of the problem, which makes it infeasible for large number of sources or complex topologies. In Section 4, we develop practical and efficient techniques for finding the clustering that minimizes the entropy.

Finally, we note that the approach described in this section does not make any assumptions about the bandwidth of the bottlenecks nor about their queuing disciplines. Thus, it works with Drop-Tail, RED, and other queuing discipline.[5] Also, this approach works when the different bottlenecks have exactly the same capacity. This is because the entropy does not depend on the exact values the random variable takes, which can be easily noticed from Equation 1, where the computation of $H(x)$ does not involve the value of x. Thus, the entropy of the inter-packet spacing is independent of the value of the inter-packet spacing. Hence, it is independent of the capacity of the bottleneck links.

4 Passive Techniques for Clustering Flows that Share the Bottleneck

To develop a clustering technique based on entropy-minimization, two design issues need to be resolved. The first issue is the computational complexity of the clustering problem. More specifically, the brute force approach, which looks at all possible clustering of the flows to find the clustering that minimizes the entropy, is exponential in the number of flows and bottlenecks and does not scale to a reasonable number of flows or complex topologies. To reduce the computational complexity we use iterative procedures, which start with an initial random clustering and iterate by

[5] It works with all work conserving queuing disciplines.

moving a source from one cluster to another to obtain an incremental reduction in the entropy.

The second issue is choosing the function that should be minimized. Equation 1 shows how to compute the entropy of the inter-packet spacing of a cluster. However, it does not indicate how to combine the entropies of the various 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 levels of accuracy, as it is shown in Section 5.3.

4.1 Iterative Clustering Techniques

Below, we describe a set of iterative techniques that minimize different entropy-based cost functions to cluster flows according to their common bottleneck. In all of these techniques we assume that the observer knows an upper bound on the number of bottlenecks. If the real number of bottlenecks is smaller than the upper bound the clustering technique generates some empty clusters.

4.1.1 Iterative Pair-wise Entropy-Based Clustering

This technique starts by guessing a random clustering. Every iteration, it picks a random flow and computes the entropy of the inter-packet spacing resulting from combing the packets of this flow with the packets of another flow according to their arrival time at the observer. We call this entropy "the pair-wise entropy." The clustering procedure then averages the pair-wise entropy over all the flows in each possible cluster and assigns the flow to the cluster with which it shares the lowest average pair-wise entropy.

The following pseudo code details the technique:

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 pair-wise entropy H_{ij} of $\{S_i, S_j\}$ resulting from combining the packets of the two sources together*
6. *For each Cluster C_k compute the average H_{ij} over all $S_j \in C_k$*
7. *Move S_i to the cluster C_k with the minimum average pair-wise entropy.*
8. *Repeat until no source changes its cluster*

4.1.2 Iterative Entropy-Based KMeans Clustering

In contrast to the pair-wise technique, which considers in each step two flows, the clustering techniques described in this section compare a flow and an entire cluster. Since this concept is similar to that used by the well-known KMeans clustering technique,[6] we call the set of procedures described in this section the Entropy-Based KMeans clustering techniques.

Each of these techniques starts by guessing some initial clustering. Every iteration, the clustering procedure picks a random flow, changes its cluster, and computes the entropy of the resulting clusters. Then it evaluates a cost function that is a weighted average of the entropy of the clusters. If the new assignment has reduced the cost function, the flow is kept in the assigned cluster. Otherwise, it is returned to its previous cluster.

The different techniques in this section differ by the weight they assign to the entropy of a particular cluster. The following example illustrates the reasons why the cost function should be a weighted average rather than the average entropy of the clusters. Assume that the maximum number of bottlenecks is 3. It is possible 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 weighing the entropy of each cluster by a factor that reflects the number of sources or the number of packets in the cluster. We define the following cost functions.

The Cluster-Weighted KMeans Technique: The cost function is a weighted average of the entropies of the clusters, where the weighing factor is the number of sources in the cluster.

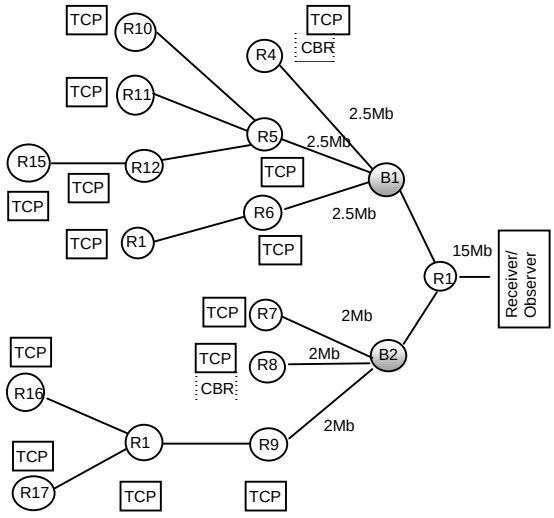


Figure 5: 2-bottlenecks, 14 TCP sources, and 2 CBR sources. All sources are sending to the receiver (the observer). The capacities of the bottlenecks are in Table 1.

[6] 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. We iterate by assigning each point to the class whose mean is nearest to the point under consideration and re-computing the mean vectors. The procedure is repeated until no point changes its class. Although, "entropy" does not have all of the properties of a distance measure (does not satisfy the triangle's inequality) the similarity between our technique and the KMeans approach suggests the naming.

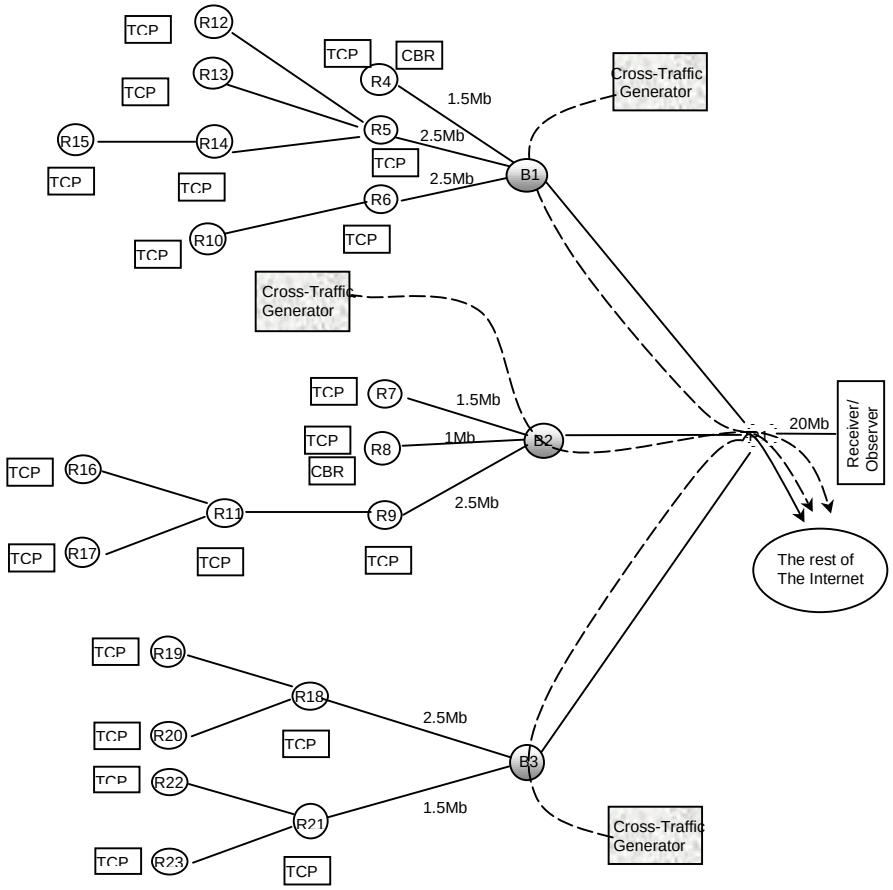


Figure 6: 3-bottlenecks topology with cross-traffic. There are 3 bottlenecks B1 B2 and B3, 20 TCP sources and 2 CBR sources sending to the receiver (the observer). Sources are sharing the bottlenecks with the cross-traffic. The capacities of the bottlenecks are in the range [1Mbits, 1.5Mbits]

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

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

The Sample-Weighted KMeans Technique: The cost function is a weighted average of the entropies of the clusters, where the weighing factor is the number of packets (i.e., samples) in the cluster.

$$Cost = \frac{1}{N}\sum_{c=1}^{N} P_c \, H_c$$

P_c is the number of packets in cluster c, H_c is the entropy of cluster c, N is the number of clusters.

The KMeans clustering procedure (regardless of which of the cost functions is used) can be best described with the following pseudo code:

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

4.2 Temporal Dependency

The above techniques ignore any temporal dependency that might exist between two consecutive data points. One can improve the techniques by taking into consideration that when a packet is queued at a bottleneck the next packet traversing the same bottleneck is likely to be queued too. Modifying the techniques to incorporate this temporal dependency 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. Since the space of the data (i.e., the random vector) becomes 2-dimensional, we call this approach the 2D-KMeans technique.

5 Performance

We evaluate the above clustering techniques using simulation. We generate packet traces using the VINT-ns-2 network simulator, which has extensive capabilities to simulate

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.

different network topologies and different traffic patterns. We evaluate the clustering techniques by running them on the packet traces resulting from the simulations. For convenience, the clustering techniques are implemented in MATLAB, which provides some mathematical functions useful for this task.

5.1 Simulated Environments

We evaluate our method using the following simulation environments.

2-bottlenecks, 16 sources, no cross traffic: The network is illustrated in Figure 5. It 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 start times of the various sources and the capacity of the bottlenecks. Every run, we choose a different permutation of the sources' start times. As a result, the packet arrival times and the interleaving of the packets from different flows 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.

3-bottlenecks, 22 sources, with cross-traffic: We also test our method on the topology in Figure 6. We experiment with both Drop-Tail and RED queues. Also, we simulate scenarios in which the observer monitors only a small fraction of the output traffic of the bottleneck. Thus, in these simulations, there is cross traffic that traverses the bottlenecks but does not traverse the link monitored by the observer. Our cross-traffic generator is a combination of 20 on-off sources with the on and off periods are taken from a Pareto distribution with an average burst time randomly chosen from the interval [30msec, 1sec]. The other characteristics of the simulation environment are similar to those stated in Table 1.

5.2 Convergence Characteristics of Iterative Techniques

In this section, we examine how successful the iterative techniques are in reducing the computational complexity of the clustering problem.

Figure 7 illustrates the convergence characteristics of the iterative pair-wise entropy-based technique. It shows a typical run on the 2-bottleneck topology in Figure 5. The *x*-axis is the number of iterations whereas the *y*-axis is the entropy. The figure shows that although the space of possible solutions is of the order of 2^{16} (the topology has 16 sources and 2 bottlenecks), the pair-wise algorithm takes only 14 iterations to stabilize.

Figure 8 illustrates the convergence characteristics of the iterative cluster-weighted KMeans technique. It shows a typical run of the algorithm on the 3-bottleneck topology in Figure 6. The *x*-axis is the number of iterations while the *y*-axis is the entropy. The figure shows that the technique stabilizes in 30 iterations; though, the space of the possible solutions is of the order of 3^{22} (the topology has 22 sources and 3 bottlenecks). The sample-based KMeans technique has similar convergence time.

5.3 Accuracy of the Iterative Techniques

Applied to the scenario in Figure 5 (2 bottlenecks, 16 sources, and no cross-traffic), the iterative clustering techniques

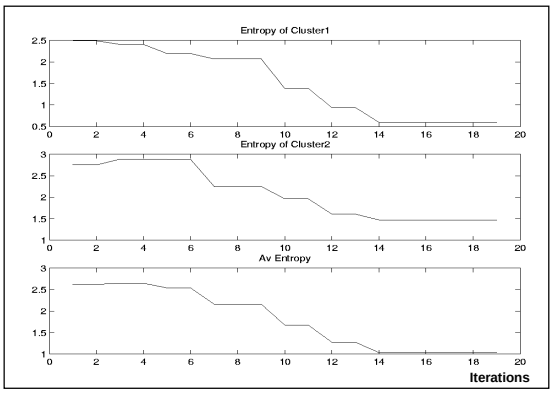


Figure 7: Convergence characteristics of the iterative pair-wise entropy-based technique. A typical run of the technique on the topology in Figure 5. The *x*-axis is the number of iterations and the *y*-axis is the entropy.

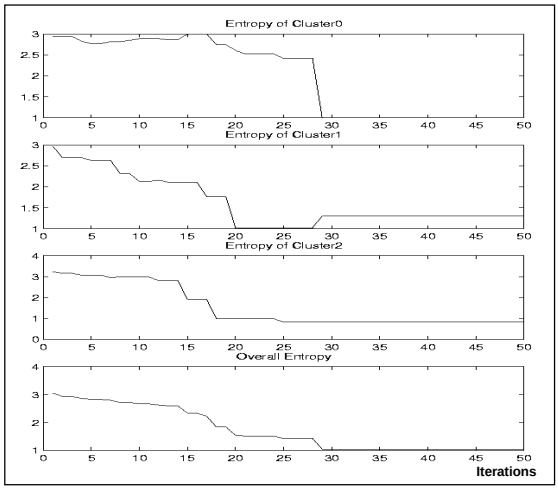


Figure 8: Convergence characteristics of the iterative KMeans entropy-based techniques. A typical run of the cluster-weighted technique on the topology in Figure 6. The *x*-axis is the number of iterations and the *y*-axis is the entropy. The sample-based technique exhibits similar convergence properties.

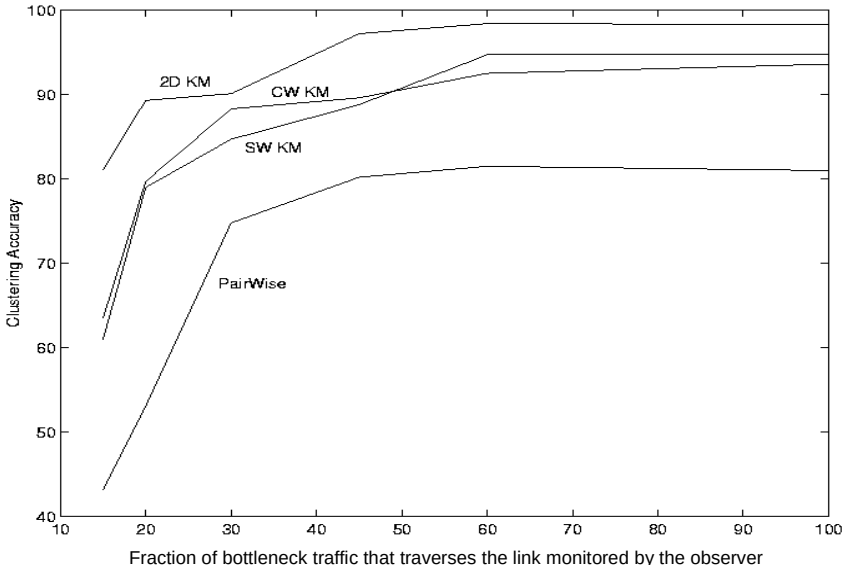


Figure 9: Accuracy of the clustering techniques as a function of the fraction of traffic monitored by the passive observer. Notations: PairWise: Iterative Pair-wise; CW: Cluster-Weighted KMeans; SW: Sample-Weighted KMeans, 2D KM: 2 Dimensional Cluster-Weighted KMeans.

showed a perfect accuracy. For example, the 2D-KMeans clustering technique showed 100% accuracy over all of the 15 random runs that we conducted. Thus, in this section we report only the results of running the algorithms on the scenario in Figure 6, which contains 3 bottlenecks, 22 sources, and cross-traffic. We control the fraction of the bottlenecked traffic that traverses the monitored link by changing the average rate of the cross-traffic generators. For each different cross-traffic rate, we generate 5 data sets by changing the sources' start times, the bandwidth of the bottlenecks, or the queuing discipline.

We measure the performance by counting the number of correctly classified flows. For example let the correct clusters be C1={S1, S2, S3, S4, S5, S6, S7, S8, S9}, C2={S10, S11, S12, S13, S14, S15, S16}, and C3={S17, S18, S19, S20, S21, S22, S23, S24}. If the hypothesis we get from a clustering technique is C1={*S10*, S2, S3, S4, S5, S6, S7, S8, S9}, C2={*S1*, S11, S12, S13, S14, S15, S16}, and C3={S17, S18, S19, S20, S21, S22, S23, S24} then the performance is 91.7% (22 correct out of 24).

Table 2 shows the accuracy of the four techniques described in Section 4 as a function of the fraction of bottlenecked traffic that traverses the monitored link. Figure 9 shows a graphical representation of the data in Table 2.

The key observation is that the accuracy is significantly high (90% to 99%) when a large fraction of the bottlenecked traffic traverses the monitored link. Yet, it degrades significantly as the fraction of the bottleneck traffic that traverses the monitored link becomes less than 15%. This happens because the cross-traffic plays the role of noise for our purpose. As more of the bottleneck output traffic becomes cross-traffic, the observed data become increasingly immersed in noise. This observation means that passive detection of shared bottleneck using these techniques is not practical when the observer is an end receiver in the Internet. However, it is highly accurate when the passive observer is monitoring a major link that is traversed by a high fraction of the bottlenecked traffic.

Another important observation is that the 2D-KMeans technique outperforms the others. This means that the correlation between the current inter-packet spacing and the previous one holds useful information for the purpose of separating flows sharing the bottleneck. It also indicates that using a 3 dimensional or a 4 dimensional feature to capture more of the temporal dependency might increase the accuracy further and render the techniques more robust against heavy cross-traffic.

We also note that the clustering techniques require a relatively small number of packets. When there is no cross traffic and the topology is not highly complex, around 20 packets per flow are enough for correct clustering. As the cross traffic becomes heavy more packets per flow are needed. Yet, The KMeans techniques always use a relatively small number of packets (around 100 packets). The small number of packets is a consequence of the fact that the KMeans techniques do not compare only the packets of two flows, they rather compare the packets of one flow against all the packets in a cluster.

Technique \ fraction of bottlenecked traffic monitored by the observer	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 Pair-wise	81.0%	81.5%	80.2%	74.8%	53.1%	43.2%

Table 2: Accuracy of Entropy-Minimization Techniques.

6 Related Work

The traditional approach for learning Internet path characteristics relies on sending probe traffic. For example, pathchar and cprobe are useful tools for discovering the bandwidth available along a path. However, 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 significantly 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. It requires the intermediate routers to reply to ICMP echo messages, a feature that might not be available [16].

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, the proposed approach sends probe packets into the network and requires the existence of a multicast service.

The authors of [12] use loss correlation among the receivers in a multicast group to infer the logical shape of a multicast tree. Their approach does not inject probe traffic in the network; however, its reliance on loss information limits its use to significantly long multicast sessions.

Recently, there were two proposals for detecting whether a pair of flows shares the same bottleneck [22, 23]. Both proposals are non-passive as they generate probe traffic, require sender cooperation, and assume a particular queuing discipline. However, they are more robust against a heavy cross-traffic than the entropy-minimization approach. Also, these proposals do not generalize their techniques to more than two flows. Thus, the clustering problem that we address in this paper is intrinsically harder than the problem addressed by these proposals.

7 Conclusion

This paper shows that a strategically located passive observer can learn sensitive information about the congestion-state and the topology of the upstream networks. In particular, we present a set of entropy-minimization techniques that allow a completely passive observer to cluster flows into groups that share the bottleneck. The observer can perform this task without generating any probe traffic and without any cooperation from the senders. Moreover, the passive observer can cluster TCP, UDP, and multicast flows, and can perform its task regardless of the queuing discipline. The paper shows that the passive observer performs its task fairly accurately if it is strategically located so that a reasonable fraction of the traffic processed by each bottleneck eventually traverses the link monitored by the observer. However, the accuracy of such completely passive observer decreases when it observes only a little fraction of the bottlenecked traffic.

Passive techniques for discovering the characteristics of a network based on measurements conducted at its edges are highly desirable because they are extremely general and resource efficient. The techniques presented in this paper are a step toward a better understanding of the capabilities of a passive observer of the traffic.

8 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.

[23] K. Harfoush, A. Bestavros, and J. Byers, “Robust Identification of Shared Losses Using End-to-End Unicast Probe,” Technical Report, BUCS-2000-013, 2000.

[24] M. Gary, and D. Johnson, “Computers and Intractability -- a Guide to the Theory of NP-Completeness,” W.H. Freeman, 1979.

[25] Y. Freund and R. Schapire, “A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting*,” Journal of Computer and System Sciences*, 1997.