Scalable Clustering of Internet Paths by Shared Congestion

Min Sik Kim*, Taekhyun Kim†, Yong-June Shin‡, Simon S. Lam§, and Edward J. Powers†
*School of Electrical Engineering and Computer Science, Washington State University
†Department of Electrical and Computer Engineering, The University of Texas at Austin
‡Department of Electrical Engineering, University of South Carolina
§Department of Computer Sciences, The University of Texas at Austin

***Abstract*— Internet paths sharing the same bottleneck can be identified using several shared congestion detection techniques. However, all of these techniques have been designed to detect shared congestion between a pair of paths. To cluster N paths by shared congestion, a straightforward approach of using pairwise tests would require $O(N^2)$ time complexity. In this paper, we present a scalable approach to cluster Internet paths based on DCW (Delay Correlation with Wavelet denoising) which does not require a common end point between paths. We present a function to map each path's measurement data into a point in a multidimensional space such that points are close to each other if and only if the corresponding paths share congestion. Because points in the space are indexed using a tree-like structure, the computational complexity of clustering N paths can be reduced to $O(N \log N)$. The indexing overhead can be further improved by reducing dimensionality of the space through wavelet transform. Computation cost is kept low by reusing for dimensionality reduction the same wavelet coefficients obtained in DCW. Our approach is evaluated by simulations and found to be effective for a large N. The tradeoff between dimensionality and clustering accuracy is shown empirically.**

I. Introduction

Information on network congestion is critical to network resource management. In particular, identifying paths sharing the same bottleneck enables sharing control information by multiple flows, and accordingly efficient and fair allocation of resources among them [1], [2]. For example, Congestion Manager [3] examines all flows of the host where it resides, and clusters them into flow aggregates, each of which consists of flows sharing the same bottleneck. By performing congestion control over flow aggregates, rather than separately over individual flows, Congestion Manager was shown to improve both efficiency and fairness of bandwidth sharing among flows.

Clustering paths sharing the same bottleneck also benefits overlay networks. Overlay networks have proliferated as an approach to circumvent limitations of the Internet and provide additional features. An overlay network consists of a number of participating end hosts and selected connections between pairs of such hosts. Because routing in an overlay network is performed through these connections, their selection is critical to the overlay network's performance. However, most overlay networks are unaware of the underlying network topology and they use simple heuristics to choose connections. As a result, some physical links may be shared by many connections, and these links become bottlenecks in bandwidth-demanding overlay applications, such as, end system multicast [4], file download from multiple servers, and overlay QoS (Quality of Service) routing. Such bottlenecks in an overlay network are avoidable if the overlay performs measurements, clusters connections that share the same bottleneck link, and replaces a subset of connections in each cluster with other connections not sharing the cluster's bottleneck.

Paths sharing the same bottleneck can be identified using shared congestion detection techniques [5]–[8]. However, all of these techniques, except DCW (Delay Correlation with Wavelet denoising) [8], require that paths share a common end point to be effective. This requirement limits their application to overlay networks, which need to cluster paths with different sources and different destinations. Moreover, all the techniques including DCW have been designed to detect shared congestion between two paths only. To cluster N paths, the straightforward approach of using pairwise tests would require $O(N^2)$ time complexity. There are other approaches proposed to reduce time complexity by performing per-cluster tests instead of per-path tests [6], [9]. In these approaches, one representative per cluster is maintained, and shared congestion detection is performed between a new path and each cluster representative to determine which cluster the new path should belong to. However, for reasons discussed in Section II, these approaches are not applicable to large-scale overlay networks.

In this paper, we present a scalable approach to cluster paths by shared congestion based on DCW [8], which does not require a common end point between paths. In our approach, measurement data are stored into a multidimensional space, where each data set collected from a network path is represented as a point. The most important characteristic of this space is that points are located closely if and only if their corresponding network paths are sharing congestion. Due to this characteristic, finding all paths sharing congestion with a given path can be replaced with neighbor search in the space. Because points in the space are indexed using a tree-like structure, adding paths and searching neighbors takes sublinear time. As a result, the computational complexity of clustering N paths can be improved to $O(N \log N)$. The indexing overhead can be further improved by reducing the dimensionality of the space through wavelet transform. Computation time is kept low because we can use the same wavelet transform for both wavelet denoising and dimensionality reduction. The

1-4244-0222-0/06/$20.00 (c)2006 IEEE

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the Proceedings IEEE Infocom.

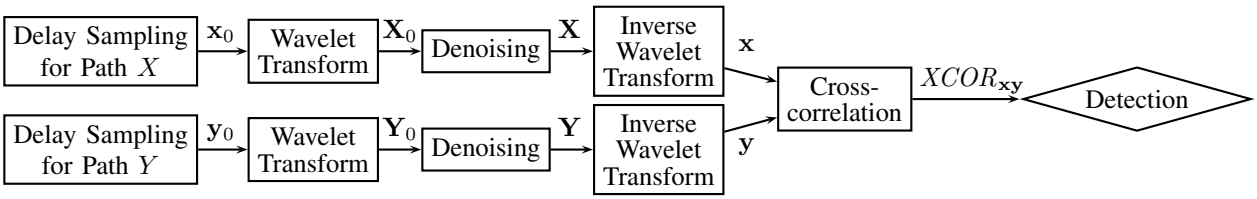


Fig. 1. Shared congestion detection procedure of DCW

tradeoff between dimensionality and clustering accuracy is investigated.

The remainder of this paper is organized as follows. Section II summarizes other approaches for path clustering. Section III presents the theory of our clustering approach including the data structure we use to store measurement data sets for paths. Section IV presents the basic implementation steps and our path clustering algorithm. A performance evaluation of our approach is presented in Section V, and we conclude in Section VI.

II. Related Work on Path Clustering

Among studies on identifying bottlenecks, FlowMate [9] and the entropy-based approach [6] have objectives that are most like our objective in this paper.

FlowMate is based on the technique proposed by Rubenstein et al. [7] for shared congestion detection. Given two paths, a sequence of delay samples is obtained for each path. If correlation between successive packets in the first sequence is higher than correlation between the two sequences, it is inferred that the two paths are sharing a congested point. When clustering paths, FlowMate maintains a "representative" path in each cluster, and applies the shared congestion detection technique to a new path and the representative of each cluster, instead of every path in the cluster, to reduce computational complexity.

The entropy-based approach was designed to cluster flows from a large number of sources to a common destination. Thus the paths used by the flows form a tree rooted at the destination. It is assumed that each path contains exactly one bottleneck. For each path, inter-packet arrival times are measured at the destination. For each path, it calculates the average entropy for every cluster assuming the path is in that cluster. Then the path is moved to the cluster with the minimum average entropy.

Both approaches [6], [9] are inappropriate for large overlay networks for the following reasons. First, while overlay networks consist of a large number of paths with different sources and destinations, these approaches can only cluster paths that share a common end point, FlowMate at the source and the entropy-based approach at the destination. Moreover, the latter requires the amount of cross traffic to be low. More specifically, for the entropy-based approach to be robust, more than 20% of the traffic at the bottleneck should arrive at the common destination.

Second, *suppose* the N paths to be clustered share a common end point. The worst-case computational complexity of these two approaches is still $O(N^2)$ because both of them use a clustering algorithm similar to K-Means clustering [10] with a low complexity only when the number of clusters is small. In a large-scale overlay network, however, there exist many independent paths (each of which is a cluster) in addition to multi-path clusters. Therefore, the number of clusters is likely to be large.

III. Our Approach

In our approach to cluster paths, we use DCW [8] to detect shared congestion. In DCW, a sequence of one-way delay samples, called a *delay sequence*, is measured for each path. The DCW procedure for detecting shared congestion between two paths is shown in Figure 1.

As shown in Figure 1, the measured delay sequences, denoted by $\mathbf{x}_0$ and $\mathbf{y}_0$[1], are denoised using wavelet transform. Let the denoised delay sequences be $\mathbf{x}$ and $\mathbf{y}$. Then the cross-correlation coefficient $XCOR_{\mathbf{xy}}$ between them is computed. DCW decides that the two paths share congestion if $XCOR_{\mathbf{xy}}$ is larger than a specified threshold value, $XCOR_{\text{threshold}}$.

A major disadvantage of DCW when applied to a large number of paths is that the cross-correlation coefficient must be computed for every pair of paths, which does not scale well. To avoid pairwise computation, we make use of a data structure, where delay sequences are stored in such a way that given a path, all other paths sharing congestion with the path are found and retrieved easily. For this purpose, we use a multidimensional space.

Suppose that delay samples were collected from three different paths: X, Y, and Z. Then we denoise them to obtain $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$, respectively. According to the DCW procedure in Figure 1, we should compute $XCOR_{\mathbf{xy}}$, $XCOR_{\mathbf{yz}}$, and $XCOR_{\mathbf{zx}}$. For better scalability, however, we instead map each denoised delay sequence to a point in a multidimensional space. A critical condition that the multidimensional space must satisfy is that points corresponding to strongly-correlated sequences should be located closely. For example, as shown in Figure 2, if $\mathbf{x}$ and $\mathbf{y}$ are strongly-correlated (because X and Y share congestion) and $\mathbf{z}$ is not, $\mathbf{x}$ and $\mathbf{y}$ should be mapped into points close to each other while $\mathbf{z}$ should be mapped to a point far from them. Then, in this space, all sequences

[1]We use a lower-case bold letter to represent a delay sequence, and an upper-case bold letter to represent a sequence of wavelet coefficients.

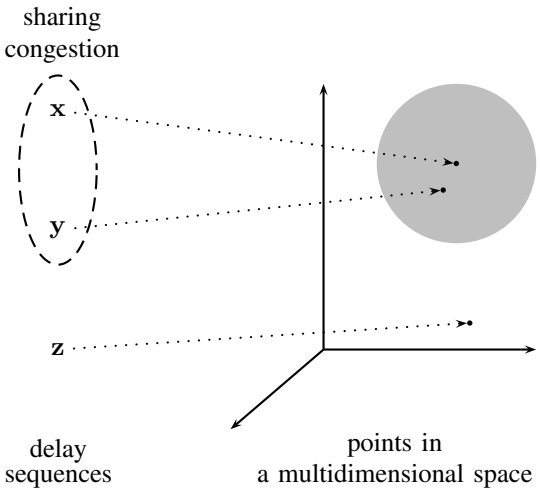


Fig. 2. Mapping delay sequences into a multidimensional space

that have strong correlation with a given sequence (in other words, all paths that share congestion with a given path) can be identified by searching neighbors of the point corresponding to the given sequence (or path). More specifically, we need a mapping such that the distance between two points in the multidimensional space is a monotonically decreasing function of the cross-correlation coefficient between the denoised delay sequences mapped to those points. With such a mapping, all the points within the radius corresponding to $XCOR_{\text{threshold}}$ in the multidimensional space must represent delay sequences of the paths sharing congestion.

Challenges of this approach are to find a multidimensional space with the desired property and to support efficient insertion and neighbor search operations in that space.

A. Mapping delay sequences into a multidimensional space

Given two paths, X and Y, let their delay sequences after denoising be the following:

$$\begin{aligned}\mathbf{x} &= (x_1, x_2, \ldots, x_m)\\ \mathbf{y} &= (y_1, y_2, \ldots, y_m)\end{aligned}$$

Then the cross-correlation coefficient between them is computed as

$$XCOR_{\mathbf{xy}} = \frac{\sum_{i=1}^{m}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{m}(x_i-\bar{x})^2\sum_{i=1}^{m}(y_i-\bar{y})^2}} \quad (1)$$

where $\bar{x}$ and $\bar{y}$ are the mean values of the elements of $\mathbf{x}$ and $\mathbf{y}$, respectively. The goal is to map the delay sequences $\mathbf{x}$ and $\mathbf{y}$ into two points $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$ in an m-dimensional Euclidean space[2] so that the distance between $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$ is a monotonically decreasing function of $XCOR_{\mathbf{xy}}$. This is achieved with the following mapping.

$$\tilde{\mathbf{x}} = \frac{(x_1-\bar{x}, x_2-\bar{x}, \ldots, x_m-\bar{x})}{\sqrt{\sum_{i=1}^{m}(x_i-\bar{x})^2}} \quad (2)$$

[2]The Euclidean space is necessary because multidimensional indexing schemes we will discuss in Section III-B require such a space.

$$\tilde{\mathbf{y}} = \frac{(y_1-\bar{y}, y_2-\bar{y}, \ldots, y_m-\bar{y})}{\sqrt{\sum_{i=1}^{m}(y_i-\bar{y})^2}} \quad (3)$$

Let $\tilde{\mathbf{x}} = (\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_m)$ and $\tilde{\mathbf{y}} = (\tilde{y}_1, \tilde{y}_2, \ldots, \tilde{y}_m)$. Then, by Eq. 2 and 3, the distance $D_{\tilde{\mathbf{x}}\tilde{\mathbf{y}}}$ between $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$ is derived as follows.

$$\begin{aligned}D_{\tilde{\mathbf{x}}\tilde{\mathbf{y}}} &= \sqrt{\sum_{i=1}^{m}(\tilde{x}_i-\tilde{y}_i)^2}\\ &= \sqrt{\sum_{i=1}^{m}\tilde{x}_i^2 - 2\sum_{i=1}^{m}\tilde{x}_i\tilde{y}_i + \sum_{i=1}^{m}\tilde{y}_i^2}\\ &= \sqrt{1 - \frac{2\sum_{i=1}^{m}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{m}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{m}(y_i-\bar{y})^2}} + 1}\end{aligned}$$

This is simplified using Eq. 1 to be

$$D_{\tilde{\mathbf{x}}\tilde{\mathbf{y}}} = \sqrt{2\,(1 - XCOR_{\mathbf{xy}})}\,. \quad (4)$$

Therefore, given two delay sequences $\mathbf{x}$ and $\mathbf{y}$, the distance between their mappings $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$ is a monotonically decreasing function of the cross-correlation coefficient between $\mathbf{x}$ and $\mathbf{y}$.

The paths sharing congestion (or having the cross-correlation coefficient greater than $XCOR_{\text{threshold}}$) with a given path can be found by searching for neighbors of the path within the following radius.

$$D_{\text{threshold}} = \sqrt{2\,(1 - XCOR_{\text{threshold}})} \quad (5)$$

The impact of this radius on clustering accuracy is investigated in Section V-A.

B. Choice of an indexing scheme

By mapping delay sequences into a multidimensional space, pairwise computation of cross-correlation coefficients becomes unnecessary; inserting delay sequences into the multidimensional space and searching for neighbors within a radius replace the pairwise computations. This means that the complexity of those two operations, insertion and neighbor search, is critical to the overall performance. In this section, we introduce an index structure to facilitate them.

It is known that a well-designed multidimensional indexing scheme can insert N points in $O(N \log N)$ time and perform neighbor search within a sphere in $O(\log N)$ time [11]. Many indexing schemes have been proposed to store and manage multidimensional data in the Euclidean space, including the R-tree [12], R+-tree [13], R*-tree [14], SS-tree [15], and SR-tree [16]. As their names suggest, they are all based on a tree-like index structure with a similar insertion algorithm. However, each has a different search performance mainly because they employ different bounding shapes, which encompass the data in a subtree. The R-tree and its successors use rectangles as bounding shapes, and the SS-tree uses bounding spheres instead. The SR-tree integrates bounding rectangles and spheres to enhance the performance of neighbor search, especially for high-dimensional data. Since the SR-tree outperforms other schemes in neighbor search [16], it is used

as the multidimensional indexing structure in the experiments presented in this paper.

Note that the clustering algorithm we propose does not depend on a specific indexing scheme; any multidimensional indexing scheme that efficiently performs insertion and neighbor search can be used.

C. Dimensionality reduction

To achieve high accuracy in detecting shared congestion, delay samples need to be collected for more than 10 seconds at a sampling rate of 10 Hz [8]. This means that the number of elements in a delay sequence is over 100, and so is the dimensionality of the multidimensional space. However, such high dimensionality increases the overhead of path clustering based on the multidimensional space, because the performance of multidimensional indexing deteriorates as the dimensionality of the data sets increases [16].

In our mapping between delay sequences and points in the multidimensional space, each delay sample corresponds to one coordinate of a point. This means that reducing dimensionality is equivalent to discarding delay samples, which immediately results in lower accuracy. Since all delay samples are considered to be "equally important," discarding any of them is equally harmful to accuracy. However, wavelet coefficients can break this symmetry. Using wavelet coefficients instead of time series enables efficient proximity search with lower dimensionality than that of using all delay samples. This is possible by utilizing wavelet coefficients at only large scales which bear information for slow-varying pattern of delay sequences [17].

Let the discrete wavelet transform[3] of $\tilde{\mathbf{x}}$ be

$$\tilde{\mathbf{X}} = (\tilde{X}_1, \tilde{X}_2, \ldots, \tilde{X}_m) = DWT(\tilde{\mathbf{x}}). \tag{6}$$

One problem in mapping $\tilde{\mathbf{X}}$ instead of $\tilde{\mathbf{x}}$ to a multidimensional space is that the relationship between the distance in the multidimensional space and the corresponding cross-correlation coefficient shown in Eq. 4 may not hold any more. Fortunately, *if we choose DWT in Eq. 6 to be an orthonormal wavelet transform, the Euclidean distance between two time series is equal to the distance between their wavelet coefficients* [18].

Figure 3 shows the energy (i.e. information) distribution of delay time series obtained from a congested link using ns-2 [19]. Most of the energy is concentrated on large-scale wavelet coefficients and any remaining energy is distributed sparsely over small-scale wavelet coefficients. Therefore, using wavelet coefficients only at large scales can achieve performance comparable to using all coefficients, while effectively reducing dimensionality. We will show empirically in Section V-B how many dimensions are needed to achieve good performance.

D. Reusing results of wavelet denoising

DCW uses discrete wavelet transform based on the Daubechies wavelet [8], which is orthonormal [20]. Therefore,

[3]Depending on the wavelet transform, the number of wavelet coefficients may be slightly different from the number of elements in $\tilde{\mathbf{x}}$.

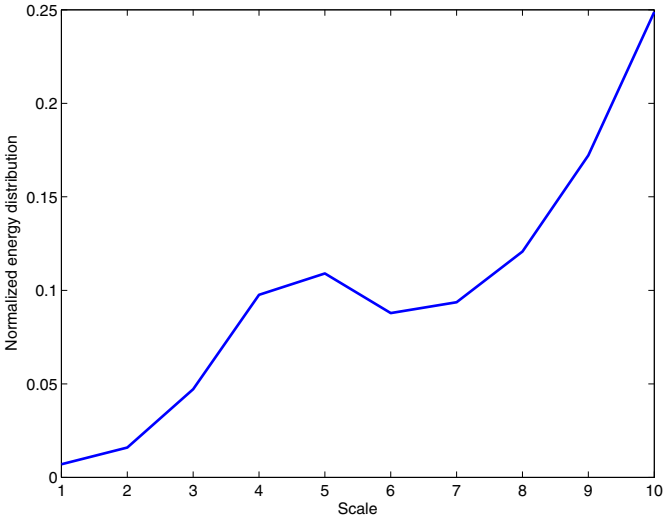


Fig. 3. Energy distribution of wavelet coefficients

we may use in $DWT(\tilde{\mathbf{x}})$ the same discrete wavelet transform that is used for denoising in Figure 1 to keep the computation cost low. In fact, $\tilde{\mathbf{X}} = DWT(\tilde{\mathbf{x}})$ can be obtained directly from $\mathbf{X} = DWT(\mathbf{x})$ without any need to compute $\tilde{\mathbf{x}}$.

Let $\overline{\mathbf{X}} = DWT(x_1 - \bar{x}, x_2 - \bar{x}, \ldots, x_m - \bar{x})$. Then

$$\begin{aligned}
\tilde{\mathbf{X}} &= DWT(\tilde{\mathbf{x}}) && (7)\\
&= DWT\left(\frac{(x_1 - \bar{x}, x_2 - \bar{x}, \ldots, x_m - \bar{x})}{\sqrt{\sum_{i=1}^{m}(x_i - \bar{x})^2}}\right) && (8)\\
&= \frac{DWT(x_1 - \bar{x}, x_2 - \bar{x}, \ldots, x_m - \bar{x})}{\sqrt{\sum_{i=1}^{m}(x_i - \bar{x})^2}} && (9)\\
&= \frac{\overline{\mathbf{X}}}{\|\overline{\mathbf{X}}\|}. && (10)
\end{aligned}$$

Since the discrete wavelet transform is a linear operation,

$$\begin{aligned}
\overline{\mathbf{X}} &= DWT(x_1 - \bar{x}, x_2 - \bar{x}, \ldots, x_m - \bar{x})\\
&= DWT(x_1, x_2, \ldots, x_m) - DWT(\bar{x}, \bar{x}, \ldots, \bar{x})\\
&= \mathbf{X} - \bar{x}DWT(\mathbf{I})
\end{aligned}$$

where $\mathbf{I} = (1, 1, \ldots, 1)$.

For indexing with wavelet coefficients at the K largest scales, we use only their corresponding coefficients from the above calculation. Thus, the final sequence to be stored in the multidimensional space is

$$\tilde{\mathbf{X}}' = (\tilde{X}_1, \tilde{X}_2, \ldots, \tilde{X}_k) \tag{11}$$

where k is the number of wavelet coefficients corresponding to the K largest scales.

IV. Basic Implementation Steps

An actual implementation of path clustering consists of the following steps:

1) Select network paths to measure delay.

2) Measure delay samples to get $\mathbf{x}_0$ for each path.
3) Process $\mathbf{x}_0$ to obtain a wavelet coefficient vector with reduced dimensionality, $\tilde{\mathbf{X}}'$.
4) Collect $\tilde{\mathbf{X}}'$ for each selected path.
5) Cluster paths.

The first and fourth steps are application-dependent. For example, in the case of overlay multicast, delay is measured at every congested edge of a multicast tree, and each internal node of the tree collects data from its child nodes.

In this section, we will only describe the application-independent steps, i.e., what a node measuring delay should do (the second and third steps), and how a node collecting data performs clustering (the last step).

A. Measuring and processing delay samples

Either a source or destination of a path measures one-way delay with sampling frequency of 10 Hz as recommended by DCW [8]. Delay samples ($\mathbf{x}_0$ in Figure 1) are collected for 12.8 seconds to make the number of samples a power of 2 for calculation convenience. Then $\mathbf{x}_0$ is converted into $\tilde{\mathbf{X}}'$ by (i) using the wavelet transform, (ii) performing denoising, and (iii) applying Eq. 10–11. Only $\tilde{\mathbf{X}}'$ for the path is submitted to the node that clusters paths.

B. Path clustering

In general, a clustering problem is NP-hard [21]. However, since we know $D_{\mathrm{threshold}}$, the maximum radius of a cluster defined in Eq. 5, we can design a simple and efficient algorithm for path clustering. The pseudo code is presented in Figure 4.

```
PATH-CLUSTERING(P)
 1 ▷ P is a set of X̃′ for all paths.
 2 S ← ∅
 3 for each p ∈ P
 4     s ← NEAREST-NEIGHBOR-IN-SPHERE(S, p)
 5     if s = nil
 6         INSERT(S, p)
 7         C_p ← {p}
 8         P ← P − {p}
 9 for each p ∈ P
10     s ← NEAREST-NEIGHBOR-IN-SPHERE(S, p)
11     C_s ← C_s ∪ {p}
12 return {C_s | s ∈ S}
```

Fig. 4. Clustering algorithm

The algorithm begins with two sets: P, a set of $\tilde{\mathbf{X}}'$ for all paths, and S, initially empty and implemented with a multidimensional space indexed as described in Section III. For notational simplicity, we use p to denote a member of P. We assume that the multidimensional indexing scheme being used (SR-tree [16] in our experiments) supports two operations: INSERT(S, p) which adds a point p to the space S, and NEAREST-NEIGHBOR-IN-SPHERE(S, p) which searches in S for the nearest neighbor of p among points in the sphere centered at p with radius $D_{\mathrm{threshold}}$. The latter returns one of them if there are multiple nearest neighbors, and **nil** if there is no neighbor in the sphere.

For each member p in P, the algorithm tests if any point stored in the multidimensional space is closer than the threshold from p. If none, the path represented by p does not share congestion with any of the paths corresponding to the previously inserted points, and thus it is added to S to create a new cluster. If there is a point closer than $D_{\mathrm{threshold}}$, ignore p because p should belong to an existing cluster. In this way, after the first loop (Lines 3–8), S contains a set of points such that every point in P shares congestion with at lease one point in S while the points in S do not share congestion with one another. Each point inserted into S represents the center of a cluster. For each cluster, a set (C_p in Line 7) containing its center point is created to store points belonging to the cluster.

The second loop (Lines 9–11) identifies the members of each cluster. For each member p in P, the cluster of the closest center s in S is selected, and p is added to the selected cluster, C_s.

Finally, a set of all clusters is returned (Line 12).

For performance reason, our implementation always keeps the entire SR-tree in memory, although the original proposal for the SR-tree assumes that the tree is maintained on disk.

Note that the algorithm selects only one cluster for each path, whereas a path may belong to multiple clusters. If finding all clusters is more desirable, Lines 10 and 11 should be modified so that p is added to every cluster of which the center is in the sphere.

V. PERFORMANCE EVALUATION

In evaluating the proposed path clustering approach, we focus on the performance of clustering and various tradeoffs with different parameter values. Because the shared congestion detection technique (DCW) itself was extensively evaluated [8], we do not repeat it in this paper.

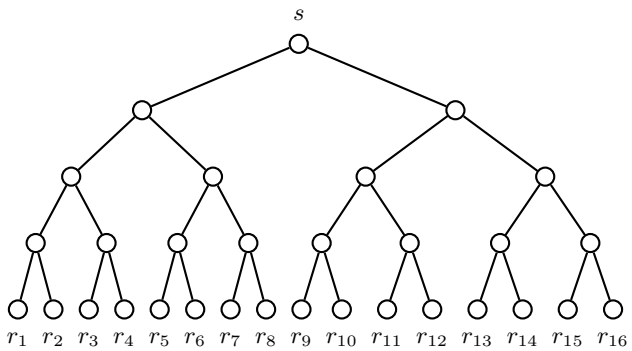


Fig. 5. Network topology

We analyze the performance of the proposed approach using simulation data from ns-2 with the topology shown in Figure 5. The bandwidth of each link is 1.5 Mbps. To create background traffic, a different amount of short-lived TCP traffic is added to each link. TCP flows are created by ns-2's web traffic generator.

One-way delay samples are measured on 16 paths, from node s to node r_i for $1 \leq i \leq 16$. Along the path from node s to each r_i, at most one link is selected as a congested link, which is used by a large number of web sessions simultaneously, resulting in a loss rate between 5 and 10%. The number of web sessions is chosen uniformly between 180 and 250. The other links have less than 70 web sessions and no packet is lost. Every experiment was repeated 500 times to get an average.

Given N paths, we use the following as performance metrics for accuracy.

- **False positive rate** The number of path pairs such that the two paths in a pair do not share congestion with each other but belong to the same cluster, divided by *the number of path pairs that do not share congestion.*
- **False negative rate** The number of path pairs such that the two paths in a pair share congestion with each other but belong to different clusters, *divided by the number of path pairs that actually share congestion.*
- **Clustering accuracy** The fraction of path pairs that are neither false positives nor false negatives among all $(N(N-1)/2)$ pairs of paths.

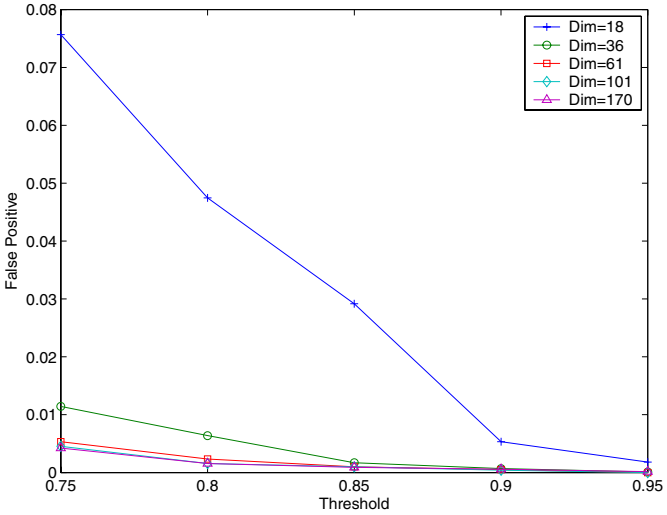
In this section, we use these metrics to study the impact of the threshold on neighbor search and the required dimensionality to maintain a reasonable accuracy. We also investigate the scalability of our clustering approach by comparing against clustering with pairwise operations.

A. Shared congestion threshold

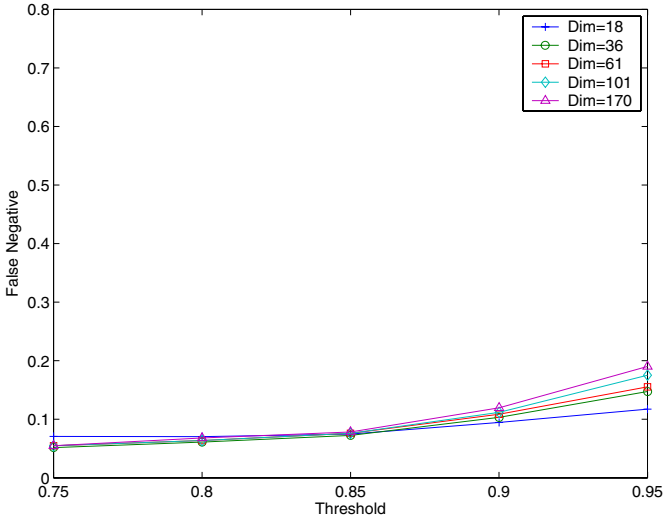
The cross-correlation coefficient threshold ($XCOR_{\text{threshold}}$) affects both false positive and false negative rates directly, because the threshold determines the radius of neighbor search in clustering. A smaller radius (larger threshold) means more clusters with finer granularity, and accordingly it is less likely to get false positives. This observation is demonstrated in Figure 6(a), which shows the false positive rate versus threshold for a range of dimensionality between 18 and 170. The false positive rate decreases as the threshold increases for every dimensionality. It is especially prominent with the lowest dimensionality, 18.

Similarly, in Figure 6(b), the false negative rate increases as the threshold increases because a smaller radius leads to more clusters than needed. Therefore, there is clearly a tradeoff to be made between the false positive and false negative rates. The clustering accuracy is also affected by the threshold as shown in Figure 6(c). Depending on the dimensionality, a threshold between 0.75 and 0.9 maximizes the clustering accuracy.

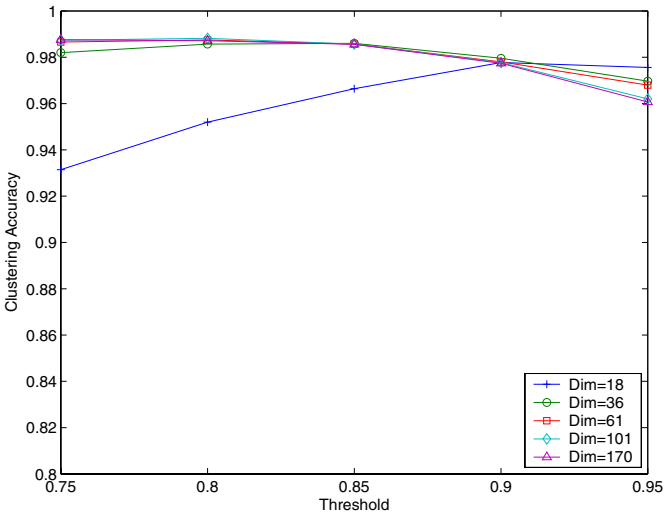
Results in Figure 6 were obtained on the topology where all paths share the source node s. In large-scale overlay networks, however, most paths have different sources. Therefore, when those sources send probe packets to measure one-way delay, packets from different sources arrive at the point of shared congestion at different times. This time difference is what we call *synchronization offset* [8]. Since the synchronization offset also affects clustering accuracy, the interaction with the shared congestion threshold in clustering needs to be investigated.



(a) False positive rate

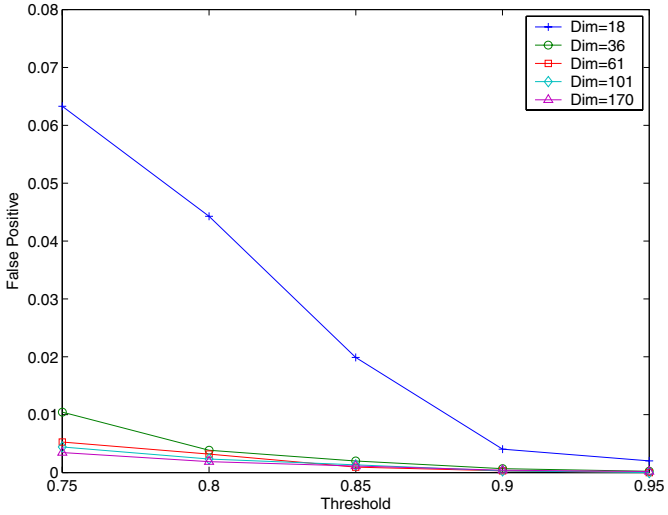


(b) False negative rate

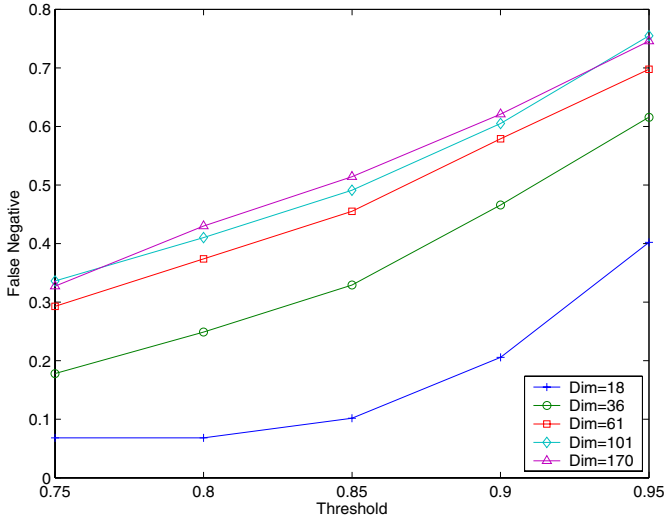


(c) Clustering accuracy

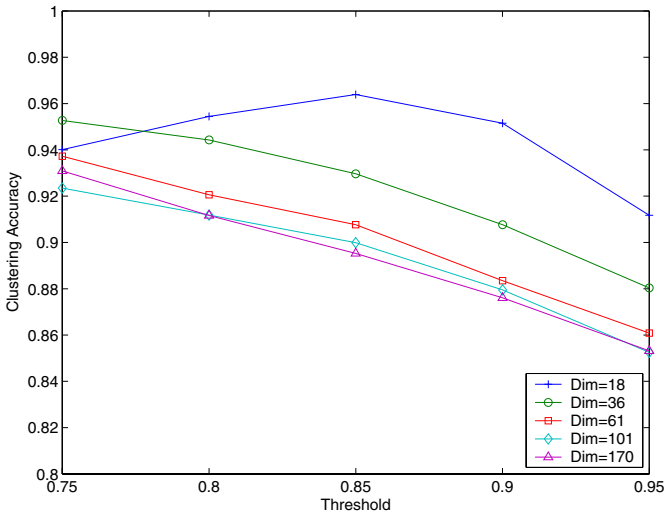
Fig. 6. Impact of threshold



(a) False positive rate



(b) False negative rate



(c) Clustering accuracy

Fig. 7. Impact of threshold with synchronization offset

In Figure 7, we repeat the experiments used in Figure 6, with non-zero synchronization offsets between paths. To simulate paths with different sources, we added a random offset to each delay sequence obtained on the topology in Figure 5. The offset was chosen uniformly between 0 and 600 ms.

The false positive rate is little affected by the synchronization offset, while the false negative rate is increased noticeably. As a result, the overall clustering accuracy gets lower, especially with high dimensionality. Nevertheless, with a low threshold value, we can still achieve the accuracy higher than 90%. We will show more results for different synchronization offsets when we discuss dimensionality in Section V-B.

From the results presented in this section, the shared congestion threshold of 0.8 seems to be a good choice. However, note that false positives are tolerable for some applications, but they may be completely intolerable for others [22]. So are false negatives. Therefore, an appropriate choice of threshold will vary from application to application.

B. Dimensionality

Dimensionality is another important parameter that affects performance. Because using fewer dimensions means that ignored dimensions cannot contribute to separating paths any more, the false positive rate increases as dimensionality decreases, while the false negative rate decreases.

Figure 6 shows that, with a low threshold (below 0.85), the decrease in the false positive rate as dimensionality increases is larger than the increase in the false negative rate. Therefore, the overall clustering accuracy is usually better with higher dimensionality as shown in Figure 8. With a high threshold (above 0.9), however, it is the opposite; the clustering accuracy gets worse with more dimensions. The reason is as follows.

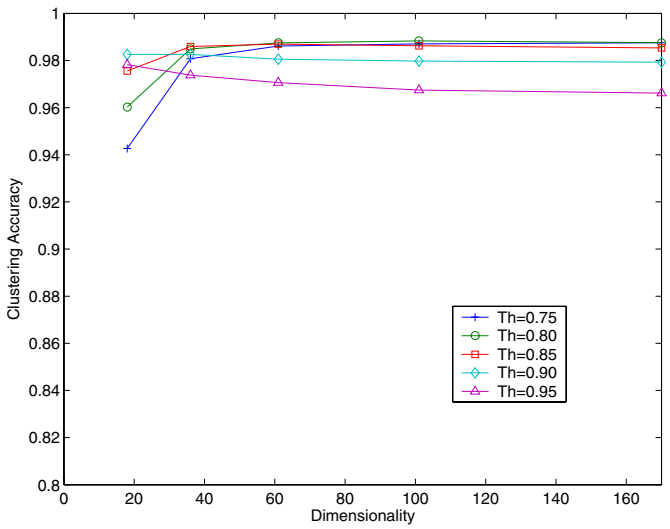


Fig. 8. Tradeoff between accuracy and dimensionality

More dimensions often contribute to separating paths. However, if a threshold is high (meaning a small radius), the false positive rate is negligible, which means that the separations

caused by additional dimensions are more likely to become false negatives than to correct false positives.

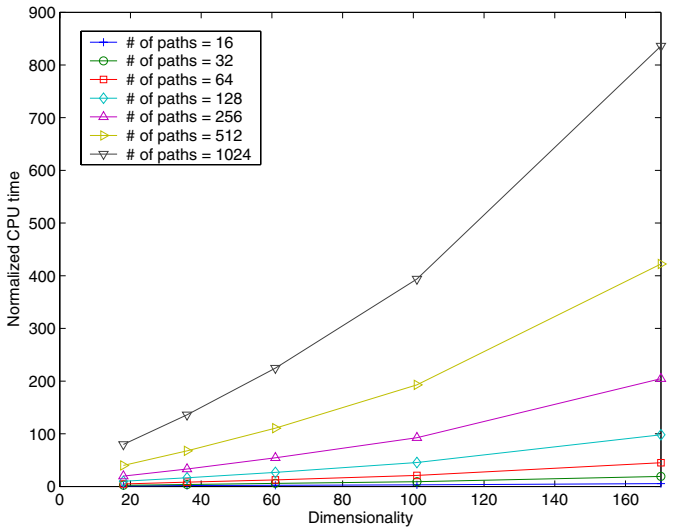


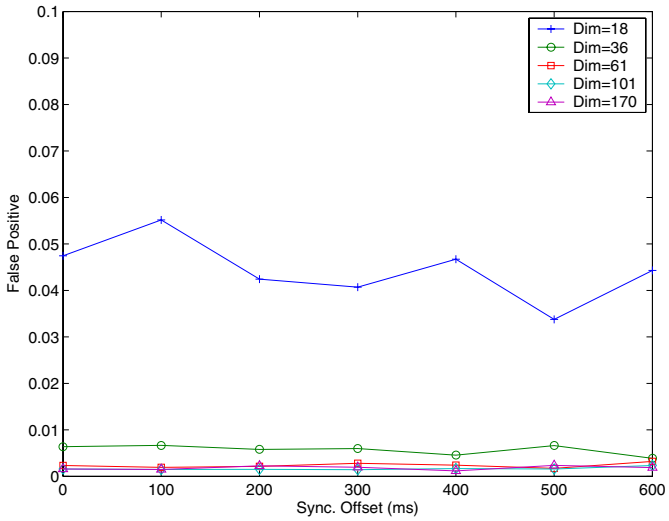
Fig. 9. Overhead of high dimensionality

Even with a low threshold, increasing dimensionality is not the best choice. The overhead of maintaining the index structure must be taken into account; it is well-known that high dimensionality often incurs significant overhead in multidimensional indexing. To demonstrate this, we plot in Figure 9 the CPU time required for clustering as a function of dimensionality. Since the actual CPU time depends on many factors, we normalize it so that the CPU time of the fastest case (16 paths with 18 dimensions) is equal to one unit of time. With a C++ implementation on a Pentium 2GHz machine, one unit of time is about 5 milliseconds.

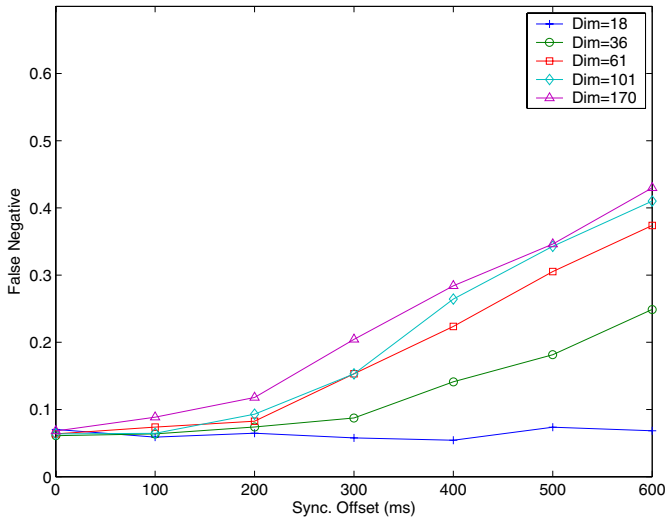
Figure 9 shows that the CPU time increases rapidly as the number of dimensions increases, especially with a large number of paths. Hence, the dimensionality should be kept minimal as long as the false positive and negative rates are acceptable. Considering the results in Figures 6 and 8, we believe that 36 dimensions are more than sufficient in most applications, and that 18 dimensions are reasonable if used with a high threshold.

The impact of synchronization offset for different dimensionalities is shown in Figure 10. As in the experiments for Figure 7, a random offset selected uniformly between 0 and the maximum offset was added to each delay sequence. The maximum offset varied from 0 to 600 ms. The threshold of 0.8 was used in all experiments.

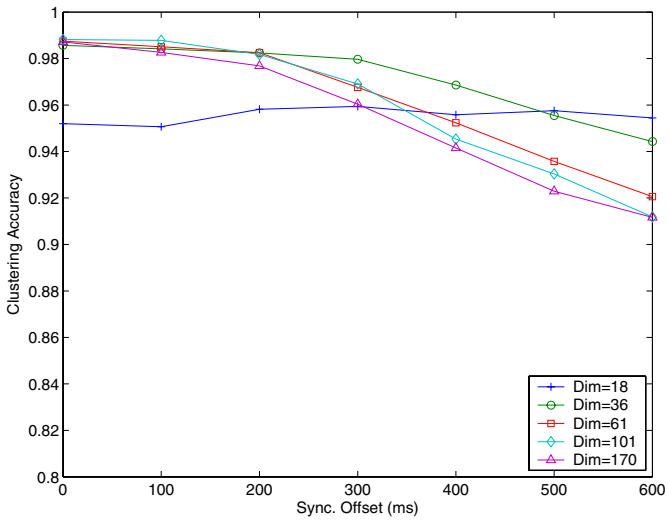
The change in the false positive rate is negligible; regardless of dimensionality, the false positive rate remains stable. In contrast, the false negative rate increases rapidly as the synchronization offset increases. The increase is more prominent with higher dimensionality, because high dimensions store information on small-scale delay variations, which are more sensitive to a small synchronization offset in calculating cross-correlation than large-scale variations. However, a reasonable degree of accuracy is still achievable even with the maximum



(a) False positive rate



(b) False negative rate



(c) Clustering accuracy

Fig. 10. Impact of synchronization offset

synchronization offset of 600 ms, by having low dimensionality at the cost of slightly more false positives.

As in selecting the threshold, the dimensionality needs to be tuned for each application, depending on how tolerable the application is to false positives and false negatives. In the case when high accuracy is required, a hybrid approach combining multidimensional indexing and pairwise tests may be used. For example, two threshold values may be chosen, one below which most points share congestion and therefore are in the same cluster with very few false positives, and the other above which most points do not belong to this cluster. Then for points between two thresholds, cross-correlation values with a representative point in the cluster are calculated to find those points that should be members of the cluster.

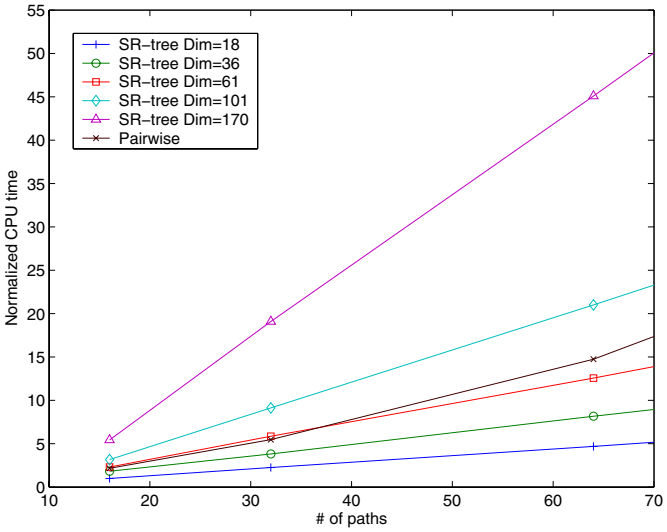
C. Scalability

The main goal of the clustering approach proposed in this paper is to achieve better scalability than the use of pairwise comparisons. While the multidimensional indexing improves the theoretical bound on time complexity, it would be more interesting to study when and how much the proposed approach outperforms the use of pairwise comparisons.

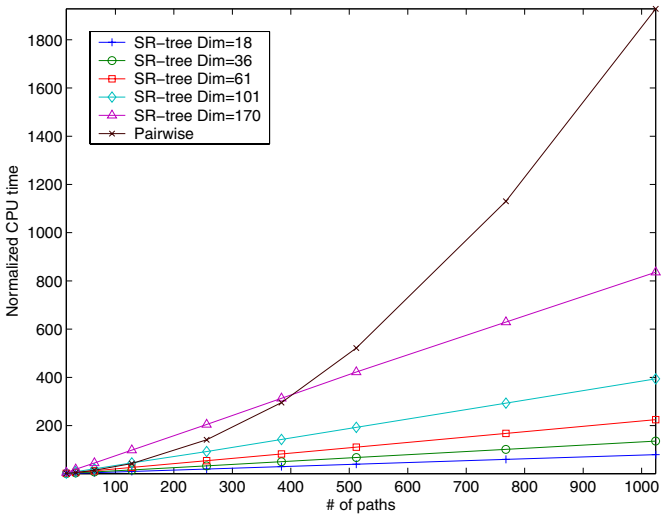
In Figure 11, we compare the proposed multidimensional indexing approach against the pairwise approach. Both approaches use DCW as a shared congestion detection technique. We plot the CPU time required for clustering versus the number of paths for different dimensionalities. Two different scales, less than 70 paths in Figure 11(a) and up to 1024 paths in Figure 11(b), are considered. The CPU time is normalized so that the case with 18 dimensions and 16 paths takes 1 unit of time.

The comparison for a small number of paths presented in Figure 11(a) shows the overhead caused by multidimensional indexing. The multidimensional indexing approach takes nontrivial time to maintain a complicated data structure. Therefore, the pairwise approach is faster if 61 or more dimensions are used to cluster less than about 30 paths. However, due to its better time complexity, our approach exhibits better performance when the number of paths gets larger. Notice that the difference in slope between curves due to different time complexity.

This better CPU time performance is clearer when the curves are extended in Figure 11(b). Because of its $O(N^2)$ complexity, the pairwise approach curve diverges from the other curves as the number of paths increases. The CPU time increase with dimensionality is significant, and low dimensionalities incur a fairly small overhead. Since the difference between 18 and 36 dimensions in terms of accuracy is rather large as observed in Figure 6, 36 dimensions would be a reasonable choice in practice.



(a) Small-scale clustering



(b) Large-scale clustering

Fig. 11. Clustering time

VI. Conclusion and Future Work

For large-scale distributed systems such as overlay networks, it is crucial to identify bottlenecks in the network so as to allocate network resources efficiently. However, previously proposed techniques to detect network bottlenecks shared by multiple paths do not scale well because they handle only two paths at a time. We proposed a scalable approach to cluster paths sharing congestion by employing multidimensional indexing and wavelet transform. It outperforms previous approaches when dealing with more than tens of paths. The granularity of clustering is controllable by adjusting the neighbor search radius. We also investigated tradeoffs between time-space complexity and accuracy with different dimensionalities.

Our future work involves applying the proposed clustering approach to large-scale overlay networks. Because a full implementation of path clustering in the context of an overlay network requires application-specific knowledge, we presented application-independent steps only in this paper. We will first extend them to overlay multicast, for which application-specific steps were already investigated and presented [4]. Further extension to a general overlay topology is also planned.

As in the case of overlay multicast, other overlay networks will benefit from scalable path clustering by using the clustering information to improve its topology and, in turn, overall throughput.

REFERENCES

[1] J. Touch, "TCP control block interdependence," IETF RFC 2140, Apr. 1997.

[2] L. Eggert, J. Heidemann, and J. Touch, "Effects of Ensemble-TCP," *Computer Communication Review*, vol. 30, no. 1, Jan. 2000.

[3] H. Balakrishnan, H. Rahul, and S. Seshan, "An integrated congestion management architecture for Internet hosts," in *Proceedings of ACM SIGCOMM '99*, Sept. 1999.

[4] M. S. Kim, Y. Li, and S. S. Lam, "Eliminating bottlenecks in overlay multicast," in *Proceedings of IFIP Networking 2005*, May 2005.

[5] K. Harfoush, A. Bestavros, and J. Byers, "Robust identification of shared losses using end-to-end unicast probe," in *Proceedings of the 8th IEEE International Conference on Network Protocols*, Nov. 2000.

[6] D. Katabi, I. Bazzi, and X. Yang, "A passive approach for detecting shared bottlenecks," in *Proceedings of the 10th IEEE International Conference on Computer Communications and Networks*, Oct. 2001.

[7] D. Rubenstein, J. Kurose, and D. Towsley, "Detecting shared congestion of flows via end-to-end measurement," *IEEE/ACM Transactions on Networking*, vol. 10, no. 3, pp. 381–395, June 2002.

[8] M. S. Kim, T. Kim, Y. Shin, S. S. Lam, and E. J. Powers, "A wavelet-based approach to detect shared congestion," in *Proceedings of ACM SIGCOMM 2004*, Aug. 2004.

[9] O. Younis and S. Fahmy, "Flowmate: Scalable on-line flow clustering," *IEEE/ACM Transactions on Networking*, vol. 13, no. 2, pp. 288–301, Apr. 2005.

[10] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*. University of California Press, 1967, pp. 281–297.

[11] S. Arya, T. Malamatos, and D. M. Mount, "Space-time tradeoffs for approximate spherical range counting," in *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, Jan. 2005, pp. 535–544.

[12] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proceedings of ACM SIGMOD '84*, 1984, pp. 47–57.

[13] T. K. Sellis, N. Roussopoulos, and C. Faloutsos, "The R+-tree: A dynamic index for multi-dimensional objects," in *Proceedings of the 13th International Conference on Very Large Data Bases*, 1987, pp. 507–518.

[14] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles," in *Proceedings of ACM SIGMOD '90*, 1990, pp. 322–331.

[15] D. A. White and R. Jain, "Similarity indexing with the SS-tree," in *Proceedings of the 12th International Conference on Data Engineering*. IEEE Computer Society, 1996, pp. 516–523.

[16] N. Katayama and S. Satoh, "The SR-tree: An index structure for high-dimensional nearest neighbor queries," in *Proceedings of ACM SIGMOD 1997*, May 1997.

[17] I. Popivanov and R. J. Miller, "Similarity search over time-series data using wavelets," in *Proceedings of the 18th International Conference on Data Engineering*, Feb. 2002.

[18] S. Mallat, *A Wavelet Tour of Signal Processing*, 2nd ed. Academic Press, 1999.

[19] K. Fall and K. Varadhan, Eds., *The ns Manual*. The VINT Project, 2005.

[20] I. Daubechies, "The wavelet transform, time-frequency localization and signal analysis," *IEEE Transactions on Information Theory*, vol. 36, no. 5, pp. 961–1005, Sept. 1990.

[21] P. Hansen and B. Jaumard, "Cluster analysis and mathematical programming," *Mathematical Programming*, vol. 79, no. 1-3, pp. 191–215, 1997.

[22] A. Akella, S. Seshan, and H. Balakrishnan, "The impact of false sharing on shared congestion management," in *Proceedings of the 11th IEEE International Conference on Network Protocols*, Nov. 2003.

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the Proceedings IEEE Infocom.