FlowMate: Scalable On-line Flow Clustering

Ossama Younis and Sonia Fahmy

***Abstract*— We design and implement an efficient on-line approach, *FlowMate*, for clustering flows (connections) emanating from a busy server, according to shared bottlenecks. Clusters can be periodically input to load balancing, congestion coordination, aggregation, admission control, or pricing modules. *FlowMate* uses in-band (passive) end-to-end delay measurements to infer shared bottlenecks. Delay information is piggybacked on feedback from the receivers, or, if impossible, TCP or application round trip time estimates are used. We simulate *FlowMate* and examine the effects of network load, traffic burstiness, network buffer sizes, and packet drop policies on clustering correctness, evaluated via a novel accuracy metric. We find that coordinated congestion management techniques are more fair when integrated with *FlowMate*. We also implement *FlowMate* in the Linux kernel v2.4.17 and evaluate its performance on the Emulab testbed, using both synthetic and tcplib-generated traffic. Our results demonstrate that clustering of medium to long-lived flows is accurate, even with bursty background traffic. Finally, we validate our results on the Internet Planetlab testbed.**

***Index Terms*— network monitoring, network tomography, TCP, shared bottleneck inference, coordinated congestion management, load balancing**

I. Introduction

Network monitoring is critical to react appropriately to network conditions, as well as to predict future network behavior. Monitoring results can be used to make decisions regarding network provisioning, traffic engineering, fault tolerance, pricing, security, and QoS support. Network monitoring, however, poses many practical challenges, most notably the high probing and logging overhead, lack of a centralized authority, non-cooperative ISPs, privacy concerns, and difficulty in capturing the highly dynamic nature of the network. One way of overcoming the non-cooperative ISP problem is by exclusively using end-to-end measurements. The *inference* of *internal* network characteristics via end-to-end measurements is commonly referred to as *network tomography*. Two types of information can be inferred: (1) static information, such as link capacities or buffer sizes, and (2) dynamic information which depends on current network state, such as available bandwidth, delays, link losses, and shared bottlenecks.

Network *probing* uses network traffic to collect measurements, which can be input to inference mechanisms. Probing can be classified as active or passive. Active probing entails sending control (out-of-band) traffic along selected network paths, while passive measurements use actual (in-band) network traffic. Active probing is more flexible since it gives control over packet timing, packet sizes, and packet distribution in the network. Injecting new traffic, however, may alter the network state by increasing load, and may consume a significant portion of the sender resources (e.g., operating system processes or connections). An example active probing application is *traceroute*, which returns the path to a specified destination. Other active probing applications include *ping*, *periscope* [1], and *MGEN* [2]. Passive measurements are less flexible since they use uncontrolled actual traffic. Passive measurements, however, do not increase network load or consume resources. A simple example of passive probing is the round trip time (RTT) estimated by TCP connections. Passive measurement tools include *tcpdump* and Cisco IOS *Netflow*.

Ossama Younis and Sonia Fahmy are with the Department of Computer Sciences, Purdue University, USA

In this paper, we use passive delay measurements for on-line, sender-side, partitioning of flows (transport-level or application-level connections) into clusters of flows that share common bottlenecks. The problem is formulated as: given a set of flows $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$, we design a mapping protocol $\mathcal{P}$ that periodically maps each flow f_j, $1 \leq j \leq n$, to exactly one cluster c_i, $1 \leq i \leq k$, $k \leq n$, such that $\forall i$, all flows $\in c_i$, share a common bottleneck. Our approach, which we call *FlowMate*, can be integrated with many applications (as discussed in Section II).

Since TCP flows comprise the majority (80% or more) of Internet traffic, *FlowMate* currently clusters TCP flows. *FlowMate*, however, can be easily generalized to any flow for which delay information can be obtained. Accurate clustering requires a time scale larger than the life-time of short-lived TCP connections (e.g., small HTTP/1.0 transfers) to converge. Long-lived TCP connections (such as file downloads) still comprise the dominant traffic *load* on the Internet. Mean connection life-times are also increasing with the growing popularity of peer-to-peer applications, such as KaZaA [3] and Gnutella [4]. This is because peer-to-peer media file transfers typically involve tens or hundreds of mega bytes. At a sender, clustering such medium to long-lived connections (called elephants in the literature [5]) can increase network-adaptivity, responsiveness, and fairness among flows or hosts. We integrate our algorithm with a simple coordinated congestion management strategy to demonstrate the improved fairness.

FlowMate has the following features that distinguish it from other approaches in the literature: (1) no generation or transmission of out-of-band probes, (2) on-line re-clustering based on the latest measurements, (3) completely end-to-end: sender side only, or with timetamping support at receivers, and (4) low overhead and high scalability to large numbers of flows.

The remainder of this paper is organized as follows. Section II gives example *FlowMate* applications. Section III discusses related work. Section IV describes the *FlowMate* design in detail. Section V defines our proposed accuracy metric. Section VI studies the performance of *FlowMate* in a number of simulation configurations with FTP, Telnet, and

HTTP traffic. Section VII discusses an alternative clustering approach. Section VIII illustrates the performance of *FlowMate* integrated with coordinated congestion management. Section IX describes the implementation of *FlowMate* in the Linux kernel, and presents our experimental results on Emulab and on Planetlab. Finally, Section X summarizes our conclusions and discusses future work.

II. *FlowMate* Applications

Many applications can utilize *FlowMate*. In this section, we discuss three examples.

A. Overlay Networks

Overlay networks among hosts provide easily deployable solutions for the problems of group communication (multicasting), optimized inter-domain routing, and content sharing and distribution. Overlay network construction and adaptation can be performed more intelligently if network state is considered during neighbor assignment. In current peer-to-peer systems, such as Gnutella [4], a new user selects its neighbors from a randomly selected set of peers in the overlay structure. In recent proposals such as [6], a new user A picks half of its neighbors from the its closest "bin" of users (having the smallest RTT from A), and the other half is randomly selected from the entire set of users. Such techniques can be improved by considering *bottlenecks*. For example, prior to neighbor selection, user A can probe paths to other peers in the proposed set of neighbors, and identify if these paths share bottlenecks. User A then selects neighbors whose flows do not share bottlenecks.

B. Load Balancing

Load balancing decisions can be based upon network conditions, such as bandwidth availability, delays, and shared bottlenecks. For example, splitting content among cache servers can be performed according to bandwidth between a primary server and the cache servers. A primary server may choose to assign highly dynamic web content to the closest caches. In addition, a cache server that determines that flows destined to a set of frequent clients typically share common bottleneck(s) may notify the primary server. The primary server can then decide to replicate the cached content at another cache server to alleviate this bottleneck.

C. Coordinated Congestion Management

Current host congestion control mechanisms regulate the sending rate of each individual flow according to network conditions assessed by that particular flow. Recent research has shown that coordinating congestion control decisions among certain flows at a busy host (e.g., ftp/Web server) can increase the collective performance of the flows [7], [8]. An important problem in addressing coordinated congestion management is the composition of clusters, in order to perform congestion management decisions on a per-cluster basis. In current coordinated congestion management approaches [9], [10], [11], [12], flows between the same hosts (or same LANs) are assigned to the same cluster. This strategy assumes that those flows will likely share the same bottlenecks along their paths. This, however, may not necessarily be true, due to network address translation (NAT), quality of service (e.g., using several queues at certain router ports), load balancing schemes, and dispersity routing [13]. In these cases, flows destined to the same host or LAN may be routed on different paths with different bottlenecks, and, consequently, should not be coordinated. More importantly, extending coordination benefits to flows that share the same bottlenecks (but are *not* destined to the same host) can significantly enhance performance.

III. Related Work

Congestion coordination has been proposed and studied in [13], [9], [7], [10], [8]. The congestion manager (CM) [7] provides a general framework for applications to coordinate congestion management decisions among flows between the same hosts. Similar approaches also follow the same-host paradigm, including TCP-Int [9], Ensemble-TCP [10], [12], and TCP Fast Start [11]. Padmanabhan [8] studies the benefits of performing coordinated congestion control, and identifies topology discovery, delay and/or loss correlation, and enhanced notification as means of detecting shared bottlenecks among flows.

Recently, a number of studies have investigated network tomography (the inference of internal network characteristics using end-to-end measurements) [14], [15], [16], [17], [18]. Katabi et al [17] use an entropy function to compute correlation among a set of flows *at the receiver*. This technique uses passive measurements, but clustering correctness degrades with heavy cross-traffic. More recent measurement results using Renyi (as opposed to Shannon) entropy demonstrate more robust clustering [19]. Rubenstein et al [18] propose novel loss and delay correlation tests among a *pair* of flows to determine shared bottlenecks. Poisson probes are injected to collect loss or delay information. We adopt Rubenstein's delay correlation test, but address the challenges of its on-line application for multiple flows at a busy server, using passive measurements. Harfoush et al [16] use Bayesian probing instead of Markovian probing to infer shared losses. As with Rubenstein's approach, this approach uses active probing. In contrast, Padmanabhan et al. [20] use passive loss measurements for Bayesian inference of lossy links in the Internet.

IV. *FlowMate* Design

This section describes *FlowMate* and analyzes its complexity. Please refer to [21] for a detailed description of algorithms and data structures.

A. Basic Architecture

FlowMate is a module that can be invoked from various other modules to provide information about flows sharing common bottlenecks along their paths from a single sender to multiple receivers. One possible *FlowMate* organization is depicted in Fig. 1. The TCP implementation at the sender must be configured to timestamp packets before being sent. Usable samples are later selected at the "Sampler" when

timestamped acknowledgments (ACKs) are received, as described in Section IV-C. Sample delay lists are then provided to the "Flow Correlator" module, which performs clustering and sends the resultant clusters to other modules, e.g., load balancer. Another possible organization of *FlowMate* is at the application layer. We discuss the tradeoffs among both organizations in Section IX. In this work, we focus on the *FlowMate* organization depicted in Fig. 1.

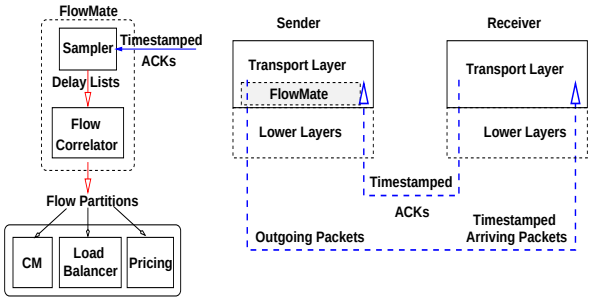


Fig. 1. One possible *FlowMate* organization

B. Correlation Test

The delay correlation test that we use in *FlowMate* was proposed in [18] to statistically identify shared bottlenecks using Poisson-distributed probe packets. We apply an analogous method on actual (non-Poisson-distributed) data packets. Pearson's correlation function [22] is applied to the delay samples as follows:

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 \sum_{i=1}^{n}(y_i - \overline{y})^2}} \quad (1)$$

where r_{xy} is the correlation coefficient (with range $[-1, 1]$) of the two sample sets x_i and y_i whose averages are $\overline{x}$ and $\overline{y}$ respectively, and n is the number of samples. By definition, the closer r_{xy} approaches $+1$ (-1), the more positively (negatively) linear the samples (x_i, y_i) are. A linear relationship between two variables means that their values fit a straight line on a scatter plot. If $r_{xy} \approx 0$, the samples show no linear relationship.

In [18], a *cross measure* is defined as the correlation coefficient of sample sets of two different variables, whereas an *auto measure* is defined as the correlation coefficient of two sample sets of the same variable. The correlation test among two flows is defined as follows [18]: (1) Compute the *cross measure*, M_x, between pairs of packets in two flows f_1 and f_2, spaced apart by time $t > 0$. (2) Compute the *auto measure*, M_a, between packets of the same flow, spaced apart by time $T > t$. (3) If $M_x > M_a$, then the flows share a common bottleneck; otherwise they do not. The intuition behind this test is that if two flows share a bottleneck, then the cross correlation coefficient should exceed the auto correlation coefficient, if the spacing between packets of different flows at the bottleneck *is smaller than* the spacing between packets of the same flow. More details on how we compute the correlation coefficients are given in Section IV-D.

C. Delay Computation

Delay correlation tests typically converge faster than loss correlation tests, and yield more accurate results. Delay correlation tests, however, impose the requirement of packet timestamping. For the delay correlation test to work best, the delays of packets on the forward path from sender to receiver must be collected at the sender. One method for collecting delay information is to utilize standard timestamping mechanisms presented in [23]. These mechanisms use the "Options" field in the TCP header [24] to include the time a packet is sent by the sender, and the time an ACK is sent by the receiver. The TCP timestamping option is currently supported in TCP implementations in most operating systems, including FreeBSD, Linux, and Windows (it is enabled by default in the latest Linux and Windows TCP implementations). The ACK send time gives an approximate indication of packet reception time. It is not entirely accurate, however, because the estimated delay is now dependent on the receiver load, scheduling mechanisms, and TCP implementation details. We experiment with this approach on Emulab in Section IX.

A second alternative is for the sender to use RTT samples (which TCP anyway computes for retransmission timeout computation purposes) [25], or throughput estimates [26]. The receiver need not use the TCP timestamping option field (or an equivalent application layer mechanism) in this case. Using RTT information instead of forward delay may, however, degrade the clustering accuracy when dynamic bottlenecks in the reverse direction alter the packet delay correlation properties. Furthermore, the load and capabilities at the receiver affect the RTT in the same manner they affect ACK send times (discussed above). We have repeated all our experiments in Section VI with RTT samples instead of one-way delays, and the reduction in accuracy values was about 5%. Section IX presents data from Emulab and Planetlab experiments using RTT samples.

A third alternative is to extend the timestamp field of the ACK to include the time at which the last packet being ACKed was received, as shown in Fig. 2. (Alternatively, this information can be added to the application layer payload in the reverse direction.) We use this extended timestamping approach in our simulation experiments in Section VI since it is the most accurate. Note that clock-skewness between the sender and receiver is not a problem, if it remains approximately constant throughout the flow duration (refer to [18] for more details).

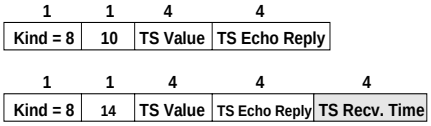


Fig. 2. Extending the TCP options field in ACK packets

D. In-Band Packet Sampling

We observe that the scalability of out-of-band delay correlation tests for flows at a busy server (as in [18]) is limited by the need to generate and transmit Poisson probes on all

flow paths. For example, a server with flows to one thousand destinations has to set up another one thousand active probe flows, which consumes a significant portion of server and network resources.

To avoid generating and injecting out-of-band control traffic in the network, we use selected data packets as samples. The sampling process proceeds as follows. Assume flow f_1 yields n_1 samples, and flow f_2 yields n_2 samples. Without loss of generality, assume that $n_1 \leq n_2$. Let $x_t(i)$ denote the timestamp (send time) of sample x_i from f_1, and $y_t(j)$ denote the timestamp of sample y_j from f_2, where $1 \leq i \leq n_1$, and $1 \leq j \leq n_2$. We merge the two sets $x_t(i)$ and $y_t(j)$ and compute the mean (for all packets of the two flows) spacing, t, between every two consecutive packets of f_1 and f_2. That is $t = \frac{\sum_{i \leq n_1, j \leq n_2} |x_t(i) - y_t(j)|}{n_{pairs}}$, where every sample $x_t(i)$ is paired with a peer sample $y_t(j)$ that minimizes $|x_t(i) - y_t(j)|$ for all j. After computing t, the auto correlation coefficient can be computed for any of the two flows. In this computation, we select samples from the flow sample set with packet spacing higher than t. Samples that are not used in the auto correlation test (due to packet spacing violation) are marked and are not used in cross correlation computation (for each particular test). This is the primary restriction on the correctness of the correlation tests (as explained in [18]), and not how probes are distributed.

To validate our results, we repeated our experiments with the following simple sampling approach. We selected data packets that are closest to Poisson probe send times (at a rate of 10 Poisson samples per second), and then applied the spacing restriction discussed above (Poisson probes are used in [18]). Our results were not significantly different from the general case without Poisson sampling (see [21] for more details). Therefore, in Section VI, we only use the inter-packet spacing restriction.

E. Triggering Clustering

It is important to trigger clustering only when sufficient usable samples are available. Since each flow has its own congestion window according to its start time and encountered losses, some flows may have only transmitted a few packets and thus have very few samples. Assume that the last clustering process was triggered at time t. We trigger the next clustering process at time $t + d$, where d is a period during which all flows have received at least a minimum of M delay values. Assuming a minimum of k usable samples are required for correlation testing, the threshold M is selected to be at least twice the value of k. We have experimentally determined that $k \geq 10$ is typically adequate. With low background traffic load, at least 20 samples are required for accurate results, because more samples are needed to capture the delay pattern. The value of k is also dependent on how packets of various flows are interleaved. With little interleaving, more samples are required for accurate clustering, as discussed in Section VI-B.5. If a time d_{max} elapses before the threshold M is reached for all flows, clustering is anyway triggered. In this case, we only consider flows with sufficient samples in the clustering process. Flows which are not considered for clustering are not grouped with any other flow (or with each other). To prevent frequent clustering and its associated overhead, clustering is not invoked before a period d_{min} elapses since the last clustering process.

F. The Clustering Process

Many clustering algorithms have been proposed in the literature, especially in the context of data mining and pattern recognition [27]. Since our objective is to obtain reasonably accurate clusters with the least overhead, we employ a very simple clustering mechanism. The clustering process takes as input a set of flows (with sufficient samples) to be clustered. We designate one flow in every cluster we form as the cluster "representative." A flow is only compared to the cluster representative in order to determine whether it should belong to the same cluster. This ensures that all flows that are clustered together are highly correlated with the same representative flow. *FlowMate* selects the first flow in a cluster to be its representative. Switching the cluster representative dynamically is currently under study (a simple approach is evaluated in Section VII). A flow is compared to *all* cluster representatives to determine if it should join an existing cluster, or form (and represent) a new cluster.

Consider, however, the case when a new flow *f* is highly correlated with more than one cluster representative. *FlowMate* takes the following conservative approach in this case. The cross correlation coefficients in all successful tests of a flow *f* are compared, and *f* joins the cluster whose representative yielded the highest cross correlation coefficient. This is because a flow typically exhibits the highest correlation with the correct cluster representative. Fig. 3 provides a summarized pseudo-code of *FlowMate* (for detailed pseudo-code, refer to [21]). *FlowMate* also includes the following (optional) test (function *re-organize()* in the pseudo-code): whenever a new cluster is created, all flows in other clusters, except for the representatives, are compared to the new cluster representative to determine if they have a higher correlation with the newly created cluster. This technique increases accuracy in cases when flow delay patterns are similar.

Note that the cross and auto correlation measures and the delay statistics are maintained and continuously updated for every pair of flows that have been tested. When clustering is triggered, new samples update the mean and variance of flow delays, and consequently, the corresponding cross and auto measures. These statistics are maintained in the flow data structures throughout a flow life-time. To illustrate this, recall equation (1) presented in Section IV-B (and expanded here):

$$r_{xy} = \frac{\sum_{i=1}^{n} (x_i y_i - \overline{x} y_i - \overline{y} x_i + \overline{xy})}{\sqrt{\sum_{i=1}^{n} (x_i^2 - 2\overline{x} x_i + \overline{x}^2) \sum_{i=1}^{n} (y_i^2 - 2\overline{y} y_i + \overline{y}^2)}}$$
$$= \frac{\sum_{i=1}^{n} x_i y_i - n\overline{xy}}{\sqrt{(\sum_{i=1}^{n} x_i^2 - n\overline{x}^2)(\sum_{i=1}^{n} y_i^2 - n\overline{y}^2)}}$$

New samples for a pair of flows are used to update $\sum_{i=1}^{n} x_i y_i$, $\sum_{i=1}^{n} x_i^2$, $\sum_{i=1}^{n} y_i^2$, $\overline{x}$, $\overline{y}$, and n in this equation. When clustering is triggered, the correlation coefficients are

Fig. 3. *FlowMate* pseudo-code

```
1. clusterList ← NULL
2. numClusters ← 0
3. FOR i ← 1 TO numFlows
4.     IF (f_i.numSamples < sampleThreshold)
5.         CONTINUE        // ignore f_i
6.     ELSE IF (numClusters = 0)
7.         Create cluster C_1
8.         C_1.representative = f_i
9.         clusterList.append(C_1)
10.        numClusters++
11.    ELSE
12.        highestCoeff ← small magic number
13.        selectedCluster ← NULL
14.        FOR j ← 1 TO numClusters
15.            result ← test(f_i, C_j.representative)
16.            IF (result = YES)
17.                update selectedCluster
18.                update highestCoeff
19.        END FOR
20.        IF (selectedCluster ≠ NULL)
21.            selectedCluster.append(f_i)
22.        ELSE
23.            numClusters++
24.            Create cluster C_numClusters
25.            C_numClusters.representative = f_i
26.            clusterList.append(C_numClusters)
27.            re-organize(clusterList)        // optional
28. END FOR
```

computed and stored along with the above terms, while the delay samples themselves are discarded. This approach is stable and scales well, but does not adapt to rapidly changing bottlenecks. This is because old delay samples still impact the delay statistics and correlation coefficients. An alternative approach would be to maintain a set of recent delay samples and re-compute the mean and variance of only these samples. Older delay samples would be periodically discarded or given less weight (effectively employing a sliding window of samples). This technique is more adaptive to rapidly changing bottlenecks, at the expense of lower stability, and slightly higher storage and re-computation overhead. Selecting which of these two techniques to employ must be based upon flow life-times, and how they compare to the constancy of Internet path properties [28]. We use the first technique in our experiments, since we do not experiment with extremely long-lived flows or with highly dynamic bottlenecks (except for the Internet experiments in Section IX-B).

Table I gives an example of clustering five flows, where the expected output is three clusters. In the example, f_4 passes the correlation test with both f_1 and f_2. It is clustered with f_2 because their cross correlation coefficient is the highest. Observe that although f_3 and f_4 are already clustered before f_5 is introduced, they are compared with f_5 after a new cluster is created to check whether a re-organization is required.

TABLE I
EXAMPLE OF CLUSTERING 5 FLOWS WITH 3 CORRECT CLUSTERS
$\{\{f_1,f_3\},\{f_2,f_4\},\{f_5\}\}$

Flow	Rep.	Test	Clusters
f_1	-	-	$\{f_1\}$
f_2	f_1	No	$\{f_1\},\{f_2\}$
f_3	f_1	Yes	
	f_2	No	$\{f_1,f_3\},\{f_2\}$
f_4	f_1	Yes	
	f_2	Yes (larger)	$\{f_1,f_3\},\{f_2,f_4\}$
f_5	f_1	No	
	f_2	No	$\{f_1,f_3\},\{f_2,f_4\},\{f_5\}$
f_3	f_5	No	$\{f_1,f_3\},\{f_2,f_4\},\{f_5\}$
f_4	f_5	No	$\{f_1,f_3\},\{f_2,f_4\},\{f_5\}$

G. Time Complexity

In this section, we show that *FlowMate* complexity is low, and thus *FlowMate* can be applied on-line.

Lemma 1: Assume that N flows are being clustered, S_p is the average cluster size, and P is the number of generated clusters (on the average, P is N/S_p). *FlowMate* time complexity is $O(NP)$.

Proof. FlowMate computations are divided into two main components: (1) sample selection, and (2) correlation tests. Using appropriate bounds in the triggering condition limits the number of delay values being processed for each flow. Computing the coefficients depends on the number of selected samples, which is less than the number of received delay values. Each flow is tested against all cluster representatives, which upper bounds the number of correlation tests by NP. Hence, the complexity depends on the number of tests multiplied by the number of operations required to compute the coefficients (which is upper bound by a constant). The asymptotic time complexity of *FlowMate* is therefore $O(NP)$. □

Observe that *FlowMate* does not require any pre-computations to estimate the appropriate number of clusters. Observe also that flows with insufficient samples are excluded from clustering, which may further reduce complexity. Clusters are created and re-adjusted as more flows are incorporated in the clustering process. The complexity is typically lower than comparing every pair of flows which is $O(N^2)$. *FlowMate* clustering is a lower-cost approximation of the K-Means clustering technique [29]. *FlowMate* overhead is lowest if only a few large clusters are formed (due to the representative-based testing approach). The worst case occurs if all flows do not share any common bottlenecks and each forms a separate cluster, which should not occur often. This is due to the locality of server requests, as well as Internet topology characteristics (power-law and small-world properties).

V. ACCURACY METRIC

Clustering inaccuracies are introduced by either (1) erroneously including a flow in a cluster where it does not belong (this includes merging two or more clusters), or (2) splitting a cluster into two or more sub-clusters. We use the term "false sharing" (fs) to denote (1) above, i.e., erroneous inclusion of

a flow with a cluster it does not share bottlenecks with (we borrow this term from [13]).

Measuring the accuracy of the output clusters in a unified manner is challenging due to the possibility of simultaneous occurrence of the two error types. The two error types have different impact on the performance of applications using *FlowMate*. For example, consider a coordinated congestion management application applied to eight flows, as shown in Fig. 4. Assume the correct clustering is two clusters with four flows each. Assume that in one instance, output clusters are four instead of two. Although coordination cannot be fully exploited in this case, all flows clustered together indeed share bottlenecks. Therefore, consequent coordination decisions are not erroneous. In the second instance, the output is two clusters with flows not sharing bottlenecks. In this case, when one flow, say f_1 experiences packet losses, the consequent decisions taken by the congestion coordination application may incorrectly affect other flows, such as f_7 (which may unnecessarily enter the slow start phase). This difference between the two cases requires the metric to account for different error types with different weights according to their severity (unlike the metric proposed in [17] which treats all error types equally). We believe that false sharing is more severe than cluster splits for most applications. Thus, the second instance in our example is considered less desirable than the first one. Our accuracy index (AI) described below reflects this requirement.

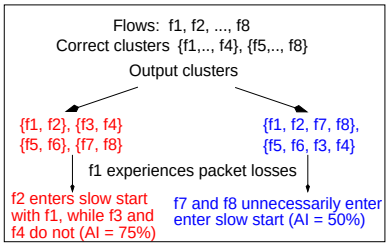


Fig. 4. Example of AI computation

Let N denote the total number of flows, P_c denote the set of correct clusters, P_o denote the set of clusters in the *FlowMate* output, n_{fs} denote the number of flows erroneously included in a resulting cluster, and s_j denote the number of sub-clusters of a correct cluster $\in P_c$ that was split into s_j sub-clusters in P_o. The cluster accuracy index (AI) is computed as follows:

$$Accuracy\ Index\ (AI) = 1 - \frac{\sum_{i=1}^{|P_o|}(n_{fs})_i}{N} - \frac{\sum_{j=1}^{|P_c|}(s_j - 1)}{N} \quad (2)$$

where $(n_{fs})_i$ of a cluster $p_i \in P_o$ is computed as follows: Map p_i to a corresponding cluster $p_c \in P_c$, such that $|p_i \cap p_c|$ is maximized. The total number of flows in set f such that $(f \in p_i) \wedge (f \notin p_c)$ is the number of flows erroneously included in a cluster $(n_{fs})_i$ (i.e., number of false shared flows).

Observe that there is no case in which *all* flows are erroneously clustered. Therefore, the accuracy index varies between a fraction (above 0) and 1. For a fixed number of flows, as the number of correct clusters $|P_c|$ increases (decreases), the average number of flows per cluster decreases (increases). Therefore, the merge effect is, on the average, diminished (exacerbated). The split effect is more uniform. Our interpretation of accuracy considers a cluster split into two clusters to be of equal severity (thus prompting an equal deduction) to false sharing of one flow (while incorrect merging of two clusters entails a penalty for each flow that was incorrectly merged with the larger set). For example, if the correct clusters are {1,2,3}, and {4,5}, and a merge occurred (i.e., {1,2,3,4,5} was output), then n_{fs} is equal to 2 (size of set {4,5}). This is because cluster splits have fewer undesirable effects than false sharing and merging.

TABLE II

EXAMPLE OF COMPUTING THE ACCURACY INDEX (AI) FOR 10 FLOWS WITH 2 CORRECT CLUSTERS (P_c = {1,..,5},{6,..,10})

Output Clusters P_o	AI	Interpretation
All split: {1}, {2},$\cdots$,{10}	0.2	1 correct flow per cluster
All merged: {1,2,$\cdots$,10}	0.5	only 1 correct cluster
Splits: {1,2,3}, {4,5}, {6,7,8}, {9,10}	0.8	2 errors (splits)
More splits: {1,2}, {3,4}, {5}, {6}, {7,8}, {9,10}	0.6	4 errors (splits)
False sharing: {1,$\cdots$,7}, {8,9,10}	0.8	2 errors (flows 6 and 7)
More false sharing: {1,$\cdots$,9}, {10}	0.6	4 errors (flows 5 to 9)
Combined errors: {1,2,3}, {4,5,6,7}, {8,9,10}	0.7	3 errors (1 split + 2 false sharing)
Combined errors: {1,2}, {3,4}, {5,6}, {7,8}, {9,10}	0.6	4 errors (3 splits + 1 false sharing)

TABLE III

AVERAGE ACCURACY INDEX OVER ALL PERMISSIBLE CLUSTERS WHEN THERE ARE 3 CORRECT CLUSTERS ($|P_c| = 3$)

# of flows	8	9	10	11	12	13	14
Average AI	0.55	0.41	0.38	0.5	0.36	0.47	0.35

Consider an example of clustering six flows where the correct clusters are {1,2,3} and {4,5,6}. If the clusters output by *FlowMate* are {1,2}, {3,4,5}, and {6}, then the accuracy index is computed as: $1 - \frac{1}{6} - \frac{(2-1)}{6} = 0.67$. In this case, one sixth is deducted for flow 3, which was incorrectly clustered, and another one sixth is deducted for the split of cluster {4,5,6} into clusters {4,5} and {6}. Note that a single flow is penalized only once, either for being clustered incorrectly (false shared), or for not being merged with its correct cluster. This is why the accuracy index is 50% in the right-hand size of Fig. 4. Table II gives additional examples. Observe that, on the average, a random clustering will likely result in an erroneous number of clusters, in addition to false sharing per cluster, yielding values typically less than 50% for the accuracy (depending on the number of flows and number of correct clusters $|P_c|$). Table III gives the average AI for all permissible clusters of a number of flows for a case with 3 correct clusters. Results are congruent with our argument on average accuracy of random clustering.

Fig. 5 further validates our argument by considering random cluster assignments in different cases. We restrict the maximum cluster size (which we refer to as "cluster limit") to a

TABLE IV
SIMULATION PARAMETERS

TCP flows	12–48, infinite FTP flows, Telnet flows, or HTTP/1.1 flows
Cross traffic	24 flows, CBR (256 kbps each)
Background traffic	to all receivers (256 kbps Pareto/traces)
Reverse traffic	64 kbps average rate for each (from receivers to sender)
Buffer size	250 packets (except in one experiment)
Drop policy	Drop-Tail (RED in one experiment)

certain ratio of the total number of flows. For each cluster limit, a random correct assignment P_c is selected. Another random assignment of flows is selected as the output P_o, and the AI is computed. This process is repeated 1000 times and the average AI is reported. The figure illustrates that the average accuracy for random clustering assignments highly depends on the cluster limit, more than on the total number of flows. This is intuitive, since a larger number of correct clusters yields more false sharing errors than cluster splits.

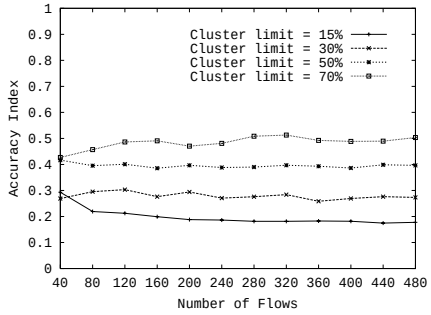


Fig. 5. Accuracy of random cluster assignments with respect to different cluster size limits

VI. SIMULATION EXPERIMENTS

We have implemented *FlowMate* in the ns-2 network simulator [30]. In this section, we investigate *FlowMate* robustness with different background traffic models and traces, and with various foreground (to be clustered) traffic types, including FTP, Telnet and HTTP. We also study the effect of router buffer sizes, router drop policies, and *FlowMate* parameters. More results can be found in [21].

Table IV summarizes the simulation parameters. Two topologies (one somewhat symmetric and one asymmetric) are used in the experiments. In the first topology (Fig. 6), a single source establishes a number of concurrent TCP connections with receivers on three different branches. The upper two branch links are bottlenecks with bandwidths 1.5 Mbps and 3 Mbps, respectively. The third branch link has a bandwidth of 10 Mbps, but is congested by a number of cross CBR flows. All other links have a capacity of 10 Mbps. A number of multiplexed Pareto flows (originating at the same source) are generated as background traffic. A number of other multiplexed Pareto flows are generated by the receivers in the reverse direction.

Fig. 7 depicts the second simulation topology, where the upper two branch links have limited bandwidth, while the

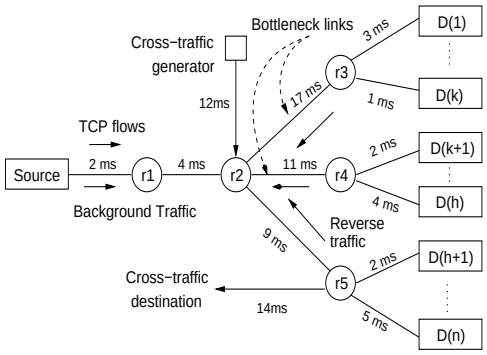


Fig. 6. Simple simulation configuration

link on the third branch is congested by high background traffic load. Background traffic is injected using a real traffic trace (the "Star Wars" movie [31]). One "Star Wars" flow is transmitted on each of the three main branches starting from router $r2$ to a randomly selected receiver on each branch, so as not to create a bottleneck on the main shared path. In both topologies, three clusters of flows comprise the expected clustering: one cluster for each one of the three branches. Simulation time is 60 seconds. This allows the effect of the transients to be visible.

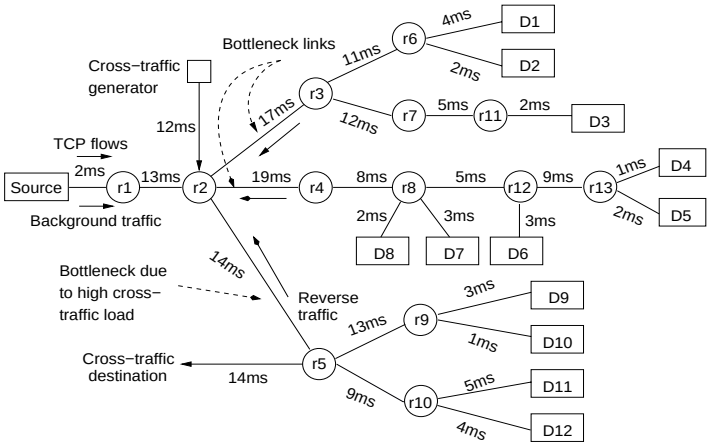
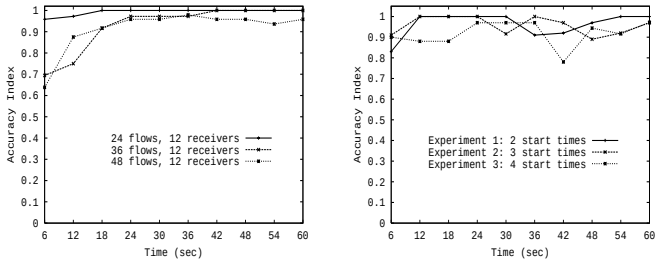


Fig. 7. More complex simulation configuration

A. FlowMate *Accuracy*

In this section, we discuss the results of experiments on the topology depicted in Fig. 6. In our first experiment, we compute the accuracy index when clustering 24, 36, or 48 TCP flows. To interpret the results easily, we trigger clustering at fixed intervals of d_{max}, and do *not* trigger it earlier, even if sufficient samples are received before d_{max}. All the other triggering rules apply (not before d_{min}, and flows with insufficient samples are discarded). The value used for d_{max} is 6 seconds. Triggering clustering according to the number of samples (as proposed in Section IV-E) may improve system performance if it occurs between d_{min} and d_{max}. Note that we compute the accuracy index by comparing against a *static* correct clustering, even though the background traffic variations entail a dynamic clustering goal. We select this more

conservative approach for ease of accuracy index computation, and to show the worst case index value.



(a) Performance with different loads

(b) Performance with staggered start times

Fig. 8. Accuracy index of *FlowMate* for the simple simulation configuration

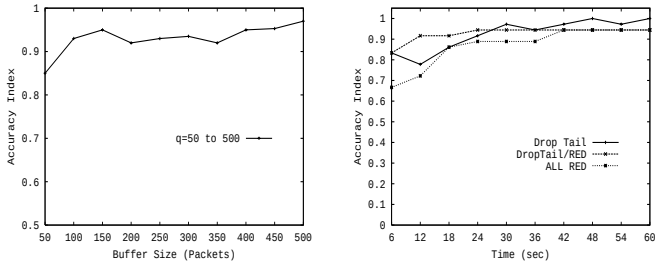
Fig. 8(a) illustrates that in steady state, performance is reasonable (average index $>$ 90%). During the initial transient period, which includes the first one or two clustering invocations, sample delay patterns are not unique for each cluster of flows, so accuracy is lower. After the transient period, accuracy is higher. Observed inaccuracies are mostly due to a few cluster splits. Flows used in this experiment start 10 to 50 ms apart. We also perform experiments with more staggered start times with 36 TCP flows and 12 receivers. In the first experiment, half of the flows begin at time zero (using a 40 ms mean interval between flow start times), and the remaining 18 start around 30 seconds later. In a second experiment, one third of the flows start near time zero, another third after approximately 18 seconds, and the last third after approximately 36 seconds. Finally, we conduct a third experiment where flows are divided into 4 sets, starting at times near 0, 18, 30, and 48 seconds. The performance results are depicted in Fig. 8(b). A large number of flows starting during the same period causes an abrupt degradation in accuracy, unlike the case where flows are added gradually. The performance is still reasonably good in steady state, and if a dynamic accuracy metric (that considers transient bottlenecks) is used, the accuracy index increases.

We have found that varying the maximum correlation interval duration d_{max} does not have a profound impact on *FlowMate* results. Results for d_{max} values between 2 and 10 seconds follow almost the same pattern as the results with 6 seconds given in this paper (refer to [21]). Below 2 seconds, samples are few, and many flows are discarded from the clustering process. When the correlation period is too long (above 10 seconds), accuracy is not significantly enhanced. We have also observed that during underloaded transient periods, the frequency of false sharing is typically higher than that of cluster splits. This is why the AI is lower during these periods than during more loaded steady states, when errors are mostly due to cluster splits. This behavior was observed throughout all simulation experiments in this section.

B. Impact of Network Conditions

The performance of *FlowMate* is affected by network conditions. Router buffer size is an important network parameter since the delay correlation test performs better in networks with large buffer sizes [18]. The packet drop policy and foreground and background traffic patterns may also impact the results. We demonstrate the effect of these parameters on the topology shown in Fig. 7. The "Star Wars" trace is used as a source of self-similar background traffic, except when varying background traffic load. In background traffic experiments, a number of Pareto sources are multiplexed, in order to easily experiment with different background rates and on/off periods. 24–36 TCP flows are used as foreground traffic, evenly divided among all 12 receivers, and, as before, the correct clustering is three clusters– one for each main branch.

1) Buffer Size: Although the delay correlation is more clearly manifested in bottlenecked routers with long queues, varying buffer sizes from 50 to 500 packets does not result in significant performance variation in steady state, as illustrated in Fig. 9. Variation in performance is more pronounced during transient periods, which is expected any time a large number of flows start at the sender simultaneously. We believe that having routers with larger buffers typically enhances performance, though.



(a) Average accuracy index (averaged over the simulation time) with different router buffer sizes

(b) Effect of packet drop policy on accuracy

Fig. 9. Impact of buffer sizes and drop policy

2) Packet Drop Policy: The prevailing drop policy in today's Internet routers is Drop-Tail. We use this policy in all our experiments, except in this experiment, where we use Random Early Detection (RED). Fig. 9(b) shows the resulting accuracy index in three cases. One case uses the Drop-Tail policy for all router queues, another case uses both Drop-Tail and RED queues, and the last case uses only RED in all queues. Results show that using RED for all queues reduces the accuracy. This agrees with the results presented in [16] about Markovian probing performance with the RED queuing discipline. The reason for RED interference is that random packet drop alters samples and introduces noise to the correlation process. Variations among different flow delay patterns are also reduced by RED, which complicates the process of determining the best cluster for a certain flow. This is also consistent with the results presented in [18]. However, even with the use of policies other than DropTail in a subset of the routers on a path, *FlowMate* still performs reasonably well.

3) Background Traffic Load: We study the performance of *FlowMate* in our two configurations (Fig. 6 and Fig. 7) with

different background traffic loads. To generate background traffic with various loads, we multiplex a number of Pareto sources, each with average rate of 400 kbps. The Pareto sources are synchronized to start at the same time (1 second before foreground traffic starts). The load values shown on the x-axis in Fig. 10(a) are computed according to the first branch, which has the least physical bandwidth. Load is slightly lower on other branches. Simulation results show that *FlowMate* is robust with heavy background traffic. We also conducted another experiment in which the ratio of the on/off periods of the Pareto sources is varied to demonstrate the effect of different burst sizes. The results illustrate that performance is consistent, which indicates that different on/off period ratios have a relatively minor effect on the clustering accuracy.

4) Foreground Traffic Load: In our experiments thus far, we have used FTP applications as our foreground traffic sources. In this experiment, we demonstrate the effect of higher burstiness in foreground traffic, and determine the number of samples required for correct clustering. We use Telnet traffic with bursty packet inter-arrivals, and control the packet mean inter-arrival time, t. As shown in Fig. 10(b), a large value of t reduces the number of samples available for correlation, and consequently reduces accuracy. For $t = 100$ ms, the figure depicts significant performance degradation, since very few samples are used in the correlation tests. In most of the cases where we observed cluster splits, the number of available samples was less than 10 per flow. Degraded performance continues throughout the simulation period. We conclude that large average packet inter-arrival times limit *FlowMate* effectiveness, since the reduced number of samples either disables the clustering entirely or adversely impacts the results. This does not a pose a serious problem, however, since the applications discussed in Section II are not applicable to very low-rate flows.

5) HTTP Traffic: Problems arise when HTTP traffic is considered. First, most HTTP connections are short-lived [5]. This implies that a connection may very well terminate before clustering is triggered, even for a small d_{min} value. Second, since HTTP packets are sent in short bursts, and since we only select samples whose inter-packet spacing exceeds the inter-flow packet spacing, we may have no available samples during many intervals. The above two problems are exacerbated by the delayed ACKs option, which delays receiver ACKs in order to piggyback them on any available data in the reverse direction. In [32], the use of parallel HTTP connections was measured over a year on a server running FreeBSD. Results show that a client typically does not use more than four parallel HTTP connections with the server simultaneously.

Fortunately, these problems are mitigated by HTTP/1.1 with persistent or pipelined connections [33]. The HTTP/1.1 specification entails that connections not be terminated after each request/response, as in the case of HTTP/1.0. A connection remains alive to be used for other requests, and only times out if it stays idle for a specified interval of time. Although this resolves the short connection problem, burstiness remains an important concern. A study presented in [34] advises against using parallel persistent connections between a server and a client

TABLE V
HTTP SIMULATION PARAMETERS

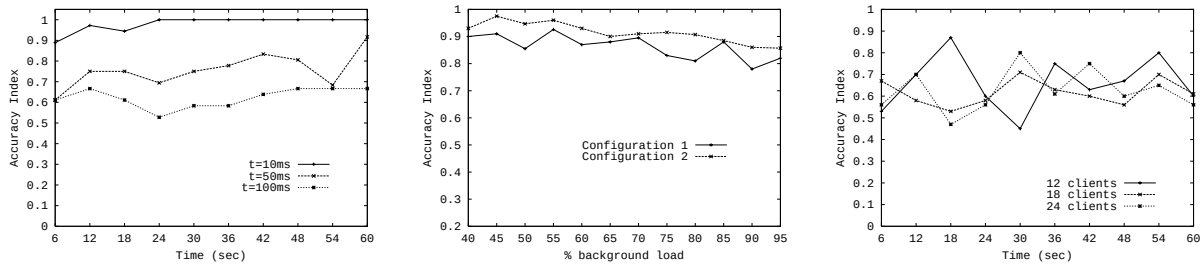
Number of web clients	12, 18, and 24
Number of sessions/client	20
Mean number of pages/session	50
Mean inter-page interval	10 ms
Mean page size	12 kB
Mean number of embedded objects/page	2
Mean object size	120 kB

FlowMate was applied to HTTP/1.1 traffic on the configuration in Fig. 6 (results for the other configuration are similar [21]). We used the SURGE model [35] for web workload traffic generation. This model is implemented in "nsweb" [36]. Table V summarizes the HTTP/1.1 parameters used in our experiments. SURGE parameters are chosen as in [35], while other parameters used in the experiments are similar to those in [36]. Fig. 10(c) depicts the performance of *FlowMate* using different numbers of web clients with 12 receivers. Performance is similar with different numbers of clients. We compute accuracy by comparing against predefined correct clusters throughout the simulation, and do not account for the fact that bursty HTTP connections may have samples with totally disjoint sets of send times. Therefore, *FlowMate correctly* reports that there is no linear correlation (i.e., the flows are not sharing a bottleneck at the same time). The reported AI, however, is too conservative in this case.

We conclude that clustering HTTP flows significantly depends on two factors: connection life-time and traffic burstiness. While it is still possible for *FlowMate* to perform reasonably well with some burstiness, connection life-time is crucial in determining if clustering is applicable. When clustering is triggered, short-lived flows have either already terminated and their information has been deleted, or they do not exceed the minimum threshold of samples required to be considered in the correlation process. As discussed above for the case of Telnet, this does not a pose a serious problem since the applications discussed in Section II are not applicable to very short-lived flows, or flows which are temporarily dormant.

VII. MAXIMUM DISTANCE CLUSTERING

In this section, we propose an alternative clustering approach for *FlowMate*. In the standard *FlowMate* clustering (discussed in Section IV-F), flows are only tested against cluster representatives. This limits the number of correlation tests required, rendering on-line clustering feasible. This simplicity, of course, comes at the expense of reduced accuracy. To mitigate this problem, we investigate an approach which we call *maximum distance clustering*. In this approach, two representative flows for each cluster (that has two or more flows) are designated. The representatives are the two flows in this cluster that have the lowest cross correlation coefficient. A new flow is tested against both representatives. If the correlation test passes with both representatives, we must determine whether the new flow should replace one of the two representatives in its role as cluster representative. The new flow becomes a representative if its cross correlation



(a) With different background average load

(b) With bursty Telnet traffic

(c) With HTTP/1.1 traffic

Fig. 10. Accuracy with different traffic loads and types

coefficient with one of the flows is lower than the cross correlation coefficient among the two representatives.

For example, let f_1 and f_2 be the two representatives of a cluster p_i with a cross correlation coefficient of 0.7. Consider a new flow f_3 which passes the correlation test with both representatives, with cross correlation coefficients 0.8 and 0.6, respectively. The cluster representatives now become f_2 and f_3. The time complexity of this clustering approach is the same as the standard approach (see Section IV-G).

We compare the performance of the two clustering approaches with different foreground traffic types: FTP, Telnet, and HTTP/1.1. We use the configuration depicted in Fig. 7, with the same parameters we used in evaluating the standard approach. For Telnet traffic, we observe that the standard (simple) clustering works well until the packet mean inter-arrival time exceeds 40 ms. The degradation in performance for mean inter-arrival time of 100 ms is depicted in Fig. 11. The figure shows that *Maximum Distance Clustering* exhibits higher accuracy. For FTP and HTTP/1.1 traffic, the results with the two clustering approaches are similar. This is not surprising, since the burstiness and connection life-time concerns with HTTP traffic (and the abundance of FTP traffic) impact *FlowMate* accuracy more than the clustering mechanism.

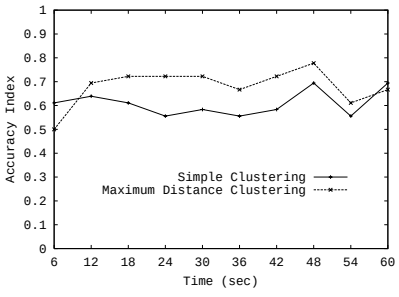
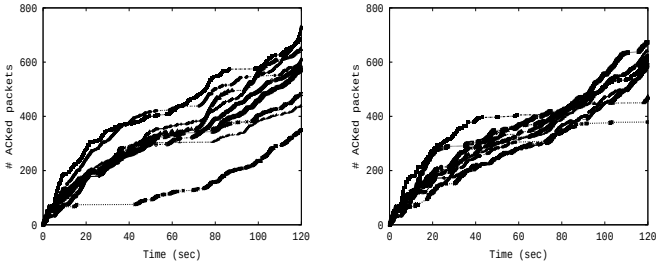


Fig. 11. Accuracy of the two clustering approaches with Telnet traffic and inter-packet mean inter-arrival time = 100 ms

VIII. Coordinated Congestion Management

In this section, we demonstrate one *FlowMate* application, namely, coordinated congestion management. As discussed in Section II, clusters of flows can be provided as input to any coordinated congestion management scheme, such as the congestion manager (CM) [7]. We implement a simple coordination mechanism that operates as follows. Each flow maintains its own congestion window. When loss is detected by any member of a cluster, all cluster member windows are reduced to react to incipient congestion. On the other hand, all cluster members increase their windows when there are no packet losses for *three* consecutive window transmissions for any member in the cluster. Thus, flows react more conservatively to detected available bandwidth. Simulation experiments are conducted using the configuration in Fig. 7. Figs 12(a) and (b) show the number of ACKed packets during a simulation period of 120 seconds for flows in one of the resulting clusters, without and with *FlowMate* and simple coordination. Fig. 12(b) illustrates that the flow throughput values are more similar and consequently fairness among flows sharing a common bottleneck increases with *FlowMate*. We believe that using flow clusters generated by *FlowMate* in schemes such as [9], [7], [10], [11], [12] will extend the benefits of these congestion coordination schemes to flows with different destinations but common bottlenecks. Moreover, *FlowMate* will also prevent false sharing of state among flows with different bottlenecks.



(a) Flows in one cluster without coordination

(b) Flows in one cluster with coordination using *FlowMate*

Fig. 12. Using *FlowMate* with congestion coordination

IX. Implementation and Experiments

We have implemented *FlowMate* in the Linux kernel (v2.4.17) [37]. Timestamping is enabled by default in this kernel implementation, which facilitates delay collection. However, if timestamping is not enabled, RTT values can be

used instead of forward delays. Implementing *FlowMate* in the kernel space (as opposed to user space) simplifies its operation for two reasons. First, it provides easy access to flow information and accurate delay samples, since no boundaries between kernel and user spaces are crossed. Second, it aids in timely provisioning of *FlowMate* output, especially to other kernel-level modules that may need this information.

As illustrated in Fig. 1, *FlowMate* is typically implemented as a separate module at the transport layer. In order to keep the receiver *FlowMate*-unaware, we use the ACK timestamps to represent the packet reception time at destinations. Although this reduces clustering accuracy to some extent (as discussed in Section IV-C), traffic burstiness and background load remain the dominant factors in end-to-end delay variation, and hence this technique is more accurate than RTT samples.

FlowMate requires little interaction with the transport layer. Fig. 13 provides a closer view of *FlowMate*'s interaction with other modules, where the solid arrows represent control flow (function calls), and the dotted arrows represent data flow. The transport layer (TCP) notifies the *Sampler* of arriving flow information (delay samples), and the *Flow Correlator* outputs the computed flow clusters. Notification is accomplished via a simple function call from the main TCP module (to *gather_samples()*) when an ACK packet is received. The parameters to this function are: (1) *sock_id*, which represents the flow identifier, (2) ACK send time (*tsval*), which *approximately* represents the packet reception time at its destination, (3) Timestamp echo reply (*tsecr*), which represents the timestamp of the generated packet (at the sender), and (4) Round trip delay (*rtt*), which can be used instead of forward delay, in case of disabled timestamping.

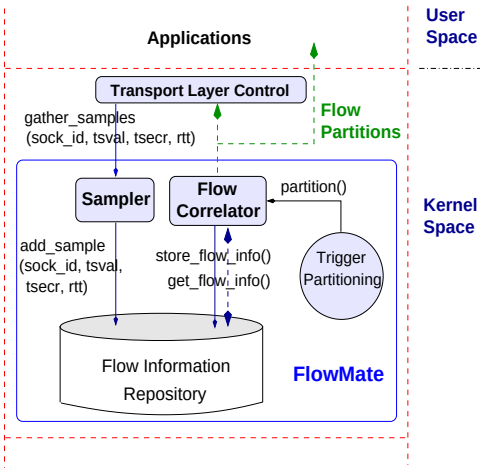


Fig. 13. Control and data flow among *FlowMate* modules and transport layer modules

The forward delay value is computed by subtracting *tsecr* from *tsval* at the *Sampler* (Fig. 13). *Sampler* stores sample delays into the *Flow Information Repository* during the sampling phase. When clustering is triggered, sample delay lists and the correlation history are input to the *Flow Correlator* module, and a new cluster list is constructed. The delay lists used in computations are discarded, while correlation history statistics are sent back to the repository for future use (as explained in Section IV-F). The output (flow clusters) can be provided to any module at the transport layer, such as coordinated congestion management. This output can also be used by higher layers, such as overlay networking modules, through an appropriate API.

Our implementation only adds about 1800 lines of code to the TCP/IP code (about 1500 in C files and 300 for header files). Only two function calls are added to the original TCP code– one for creation of the *Correlator*, and the other for collecting samples. To compute the approximate memory requirements for data structures, we consider the *Flow Information Repository* which contains sample lists and correlation history. A sample requires 24 Bytes. With appropriate bounds on the maximum number of samples per correlation interval (e.g., 100), a sample list for one flow requires about 2.4 kBytes. A *CorrSet*, which contains cross and auto correlation coefficients of two flows, requires 116 Bytes. The number of stored *CorrSet* items in the correlation history lists of flows depends on the number of performed correlation tests, which is $O(NP)$ (product of number of flows and number of clusters), as discussed in Section IV-G. Therefore, the memory required for the *Flow Information Repository* for 100 flows and 10 clusters $\approx 2.4 \times 1024 \times 100 + 116 \times 100 \times 10 \approx 360$ kBytes. This is an extremely small percentage of the memory available on today's servers.

A. Emulab Experiments

We use *EmuLab* (part of *NetBed*) [38]– an emulation testbed at the university of Utah. The testbed provides the capability of remotely configuring machines and links, using *ns*-like scripts and a web interface. Each machine can run either a default operating system (Redhat Linux or FreeBSD) or a customized operating system version. The server (source machine) in our experiments runs our modified Linux kernel that includes *FlowMate*. Other machines (routers and receivers) in our experiments run standard Linux.

We generate foreground traffic (TCP) and background traffic (UDP) using two traffic generation tools and real file transfers. One tool that generates TCP and UDP traffic is *TG* (Traffic Generator) [39]. TG provides a simple syntax for writing scripts that run on servers and receivers to open/close ports, set the mode of operation (server/receiver), start/stop traffic, and configure traffic type and rate. In order to generate FTP traffic, we used Netspec [40]. Netspec can generate emulated TCP and UDP traffic with specific support for popular applications, such as FTP, Telnet, CBR/VBR streams, and HTTP traffic.

To evaluate the clustering accuracy, we simulate the three-branch topology illustrated in Fig. 14. Two bottlenecks lie on the upper two branches (bottleneck link capacities are 0.5 Mbps and 1.5 Mbps), while the lowest branch has no bottlenecks. Bandwidth on the shared link is high (100 Mbps) to avoid bottlenecks near the source. Bandwidth is 10 Mbps on all other links. Propagation delay is indicated on each link in the figure.

We generate TCP traffic from the source (server) to all destinations D_i, where $1 \leq i \leq 8$. Background traffic consists of exponential UDP flows from the source to all destinations.

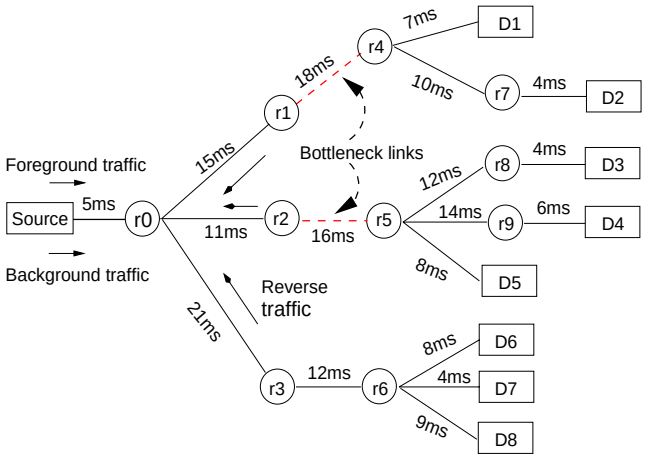


Fig. 14. Configuration used in Emulab experiments

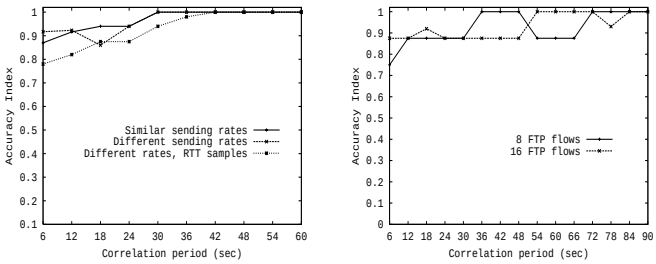
Their mean packet inter-arrival times are in the range of 30−40 ms. All background flows start 1−2 seconds earlier than the foreground traffic start time (t_0). UDP traffic in the reverse direction (from receivers to source) is also generated to interfere with returning ACKs. We generate one exponential UDP flow on each branch with packet inter-arrival times in the range of 10−20 ms. In our experiments, the correct clusters are: $\{F_1, F_2\}$, $\{F_3, F_4, F_5\}$, where F_j is the set of flows destined to receiver D_j, $1 \leq j \leq 5$. We expect each of the remaining three flows to be clustered separately, since they do not share bottlenecks with each other or with other flows.

The first experiment assesses the *FlowMate* clustering accuracy for 16 TCP foreground flows (2 flows per destination), in the presence of 8 background UDP flows (1 flow per destination), and 3 UDP flows in the reverse direction. All flows are generated using TG with exponential TCP flows as foreground traffic, each starting within a random $[0..150]$ ms interval from the initial start time (t_0). Other parameters are similar to those in simulations: Drop-Tail policy for buffer management and buffer sizes of 150 packets.

Fig. 15(a) shows the accuracy index results for very similar source sending rates (the solid line) and for another experiment with different source sending rates (the dashed line). For similar sending rates, each flow has a mean packet inter-arrival time in the range of 10−12 ms. For different sending rates, each flow has a mean packet inter-arrival time in the range of 10−25 ms, i.e., some flows may have double the sending rate of others. The results show accurate clustering.

We also compare performance when RTT samples are used, as opposed to using forward delays (Section IV-C discusses the pros and cons of both alternatives). We use different source rates in this experiment. Fig. 15(a) shows that the accuracy with RTTs (dotted line) is slightly lower than that with forward delays (dashed line) for different rates. The performance remains reasonably good, which implies that using RTT samples is still practical. Almost all the errors that occurred in this experiment were due to cluster splits.

We conducted another experiment using Netspec to cluster FTP flows. UDP flows are used as background traffic. Fig. 15(b) shows that clustering accuracy is high for 8 and for 16 foreground flows. Again, most of the errors observed were due to cluster splits. We observed similar results when we conducted 8 real file transfers, one per receiver. The results show that *FlowMate* is robust with different numbers of flows.



(a) With TG exponential foreground traffic

(b) With Netspec FTP foreground traffic

Fig. 15. Accuracy of *FlowMate* in Emulab Experiments

In addition, we experimented with different values for the timer granularity (default granularity is 10 ms), and found that they have little impact. We used a timer granularity of 1 ms in our experiments (the finest supported granularity by this kernel is 1/1024 second).

B. Internet Experiments

We have investigated the performance of *FlowMate* on the Internet. We use one machine at Purdue University (PU) as the source node and nodes from Planetlab testbed [41] as the destination nodes. To increase the likelihood of bottlenecks, we use hosts in different continents. Fig. 16 depicts our experimental topology.

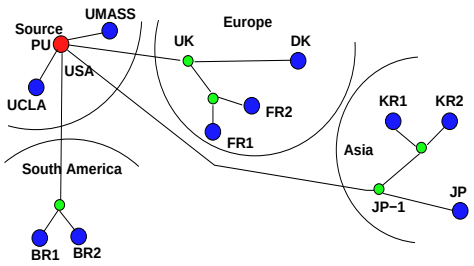


Fig. 16. Internet experimental topology

We transfer a file of size 30 MB to each destination in the graph simultaneously using the *sftp* utility. Our experimental configuration consists of the following hosts: (1) two hosts, UMASS and UCLA, at the University of Massachusetts and the University of California at Los Angeles, respectively. These hosts do not share most of the path to the PU node, (2) two hosts, BR1 and BR2, in Brazil (South America) that share the entire path from our source machine, (3) three hosts in Europe that share the path until a node in the United Kingdom. This is verified using the *traceroute* utility. Two of the European nodes are in France (Eurecom, FR1, and Inria, FR2), and the third is in Denmark (DK), and (4) three hosts in Asia, two of which are co-located in Korea (KR1 and KR2), and the third is in Japan. The Korean nodes share a gateway

TABLE VI
AVAILABLE BANDWIDTH AND TIME TAKEN TO TRANSFER A 30 MB FILE FROM THE PU NODE TO DIFFERENT HOSTS

Host	Measurement time (a.m.)	Transfer interval (sec)	Available bandwidth (Mbps)
UMASS	2:30/10:30	40/45	> 50 / > 0
UCLA	2:30/10:30	55/60	> 30 / > 0
BR1	2:30/10:30	185/635	8.7 / 0.4
BR2	2:30/10:30	190/630	8.5 / (0.45-0.6)
FR1	2:30/10:30	150/155	2.5 / (2.3-3.3)
FR2	2:30/10:30	100/170	(6.1-6.7) / (0-29.8)
DK	2:30/10:30	145/140	> 4.32 / (0-70)
KR1	2:30/10:30	325/220	6.9 / > 0
KR2	2:30/10:30	300/195	> 0 / > 0
JP	2:30/10:30	385/485	39 / > 0

TABLE VII
FLOWMATE INTERNET EXPERIMENTAL RESULTS

Host1	Host2	Time (a.m.)	% shared time
UMASS	UCLA	2:30/10:30.	8.3% / 16.6%
BR1	BR2	2:30/10:30	26% / 87%
KR1	KR2	2:30/10:30	96% / 52.5%
	JP	2:30/10:30	0% / 0%
KR2	JP	2:30/10:30	0% / 27.5%
FR1	FR2	2:30/10:30	81.8% / 60.2%
	DK	2:30/10:30	61.5% / 22%
FR2	DK	2:30/10:30	77.3% / 33.3%

in Japan with the Japanese node. Prior to performing the file transfers, we expected flows sharing most of the path to be clustered together most of the time if their bandwidth in the non-shared part is not severely limited.

We carried out experiments at two different times on 11/6/2003 in order to experiment with loads at different time zones. We report the results at 2:30 a.m. (EST), which corresponds to morning time in Europe and afternoon time in Asia, and at 10:30 a.m., which corresponds to afternoon time in Europe and night time in Asia. We use the *Pathload* utility [42] to estimate the available bandwidth prior to file transfers. The file average transfer time and available bandwidth for each destination host are given in Table VI. We compute *FlowMate* clusters every five seconds using RTT samples and give the results in Table VII.

As expected, {BR1, BR2} and {KR1, KR2} are clustered during most of the transfer interval, especially when the available bandwidth is low. Nodes BR1 and KR1 appear to be more loaded than their peers BR2 and KR2, respectively: the RTT values reported by their flows show higher variance. The Eurecom (FR1) and INRIA (FR2) flows are clustered together during the morning more than in the afternoon, when bottlenecks are more likely. Flows traveling to {FR1, FR2} and DK are frequently clustered, which implies that the path up to the UK common node on their path sometimes contains a shared bottleneck. Flows traveling to the two nodes in the US were not clustered most of the time, which is due to the abundant bandwidth on the only two hops they share.

The only unexpected result in Table VII is that the JP flow is not clustered with either KR1 or KR2. Using ping, we found that the RTT values are significantly different between the JP packets and the KR1 (and KR2) packets, and the variance in the RTT values of packets going to JP is much lower than that for either KR1 or KR2. Using traceroute, we found that paths to JP and {KR1, KR2} are shared until a gateway JP-1, which is very close to JP. Pathload also indicated that the bandwidth from PU to JP is usually much higher than that to {KR1, KR2}. Therefore, we deduced that a bottleneck may occur only on the path from JP-1 to {KR1, KR2}, which explains the results of JP and {KR1, KR2} in Table VII.

We also find that false sharing only occurs in our measurements in a few cases with similarities in delay patterns for non-congested flows. However, no two flows were incorrectly clustered for more than three consecutive clustering intervals. Cluster splits are more prevalent in the results than false sharing. Both false sharing and cluster splits may not necessarily be erroneous, however, since Internet bottlenecks change dynamically, and the flows in our experiments share a few hops before they split.

X. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented *FlowMate*, a system that exploits end-to-end packet delay information to periodically cluster flows originating at a busy server, based upon whether they share bottlenecks. *FlowMate* does not require generation or transmission of out-of-band probe traffic for collecting delay information. *FlowMate* is likely to produce multi-member clusters at a busy server, due to the locality of requests, and the power-law and small-world characteristics of Internet topology. Therefore, *FlowMate* complexity, which depends on the number of clusters, is reasonable. *FlowMate* accuracy was observed to be high in various configurations with different propagation delays, bottlenecks, buffer sizes, and drop policies. The primary factor that degrades performance is the burstiness of the flows being clustered themselves, as seen in our HTTP/1.1 and Telnet results. Background traffic load and burstiness do not have a detrimental effect, since we consider the history of correlation statistics. Fairness of coordinated congestion control is observed to significantly increase with *FlowMate*.

We have implemented *FlowMate* in the Linux kernel v2.4.17, and experimented with it using an emulation testbed and on the Internet. Results show high accuracy for both synthetic and tcplib-generated traffic, even with *FlowMate*-unaware receivers. In our future work, we plan to integrate *FlowMate* into overlay network middleware.

ACKNOWLEDGMENTS

We would like to thank Jay Lepreau and the entire Emulab/Netbed team at the University of Utah, Venkat Padmanabhan, the anonymous reviewers, Dan Rubenstein (Columbia University), and Anja Feldmann and Jorg Wallerich (University of Munich) for their help. This research has been sponsored in part by NSF grant ANI-0238294 (CAREER), and the Schlumberger Foundation.

REFERENCES

[1] K. Harfoush, A. Bestavros, and J. Byers, "PERISCOPE: An active measurement API," Proceedings of PAM, March 2002.

[2] MGEN, "http://manimac.itd.nrl.navy.mil/MGEN/," 2003.

[3] KaZaA, "www.kazaa.com," 2003.

[4] Gnutella, "www.gnutella.com," 2003.

[5] S. Jin, L. Guo, I. Matta, and A. Bestavros, "The War Between Mice and Elephants," in *Proceedings of IEEE ICNP*, November 2001.

[6] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-Aware Overlay Construction and Server Selection," in *Proceedings of the IEEE INFOCOM*, New York, 2002.

[7] H. Balakrishnan and S. Seshan, "An Integrated Congestion Management Architecture for Internet Hosts," in *RFC 3124 and MIT technical report MIT/LCS/TR-771*, 2001, also appears in ACM SIGCOMM 1999.

[8] V. N. Padmanabhan, "Coordinated Congestion Management and Bandwidth Sharing for Heterogeneous Data Streams," in *Proceedings of NOSSDAV*, 1999.

[9] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, M. Stemm, and R. H. Katz, "TCP Behaviour of a Busy Web Server: Analysis and Improvements," in *Proceedings of IEEE INFOCOM*, March/April 1998.

[10] L. Eggret, J. Heidemann, and J. Touch, "Effects of Ensemble-TCP," in *ACM Computer Communication Review*, January 2000.

[11] V. Padmanabhan and R. Katz, "TCP Fast Start: A Technique For Speeding up Web Transfers," in *IEEE GLOBECOM 98 Internet Mini-Conference*, November 1998.

[12] J. Touch, "TCP Control Block Interdependence," RFC 2140, April 1997.

[13] S. A. Akella, S. Seshan, and H. Balakrishnan, "The Impact of False Sharing on Shared Congestion Management," in *The Eleventh IEEE International Conference on Network Protocols (ICNP)*, November 2003, also CMU-CS-01-135.

[14] R. Caceres, N. Duffield, J. Horowitz, D. Towsley, and T. Bu, "Multicast-based Inference of Network-internal Characteristics: Accuracy of Packet Loss Estimation," in *Proceedings of the IEEE INFOCOM*, New York, March 1999, http://www.ieee-infocom.org/1999/papers/03a_04.pdf.

[15] N. Duffield, F. L. Presti, V. Paxson, and D. Towsley, "Inferring Link Loss Using Striped Unicast Probes," in *Proceedings of the IEEE INFOCOM*, Anchorage, Alaska, April 2001, http://www.ieee-infocom.org/2001/papers/687.pdf.

[16] K. Harfoush, A. Bestavros, and J. Byers, "Robust identification of shared losses using end-to-end unicast probes," in *Proceedings of the IEEE International Conference on Network Protocols (ICNP)*, October 2000.

[17] D. Katabi, E. Bazzi, and X. Yang, "A Passive Approach for Detecting Shared Bottlenecks," in *Proceedings of IEEE ICCCN*, October 2001.

[18] D. Rubenstein, J. F. Kurose, and D. F. Towsley, "Detecting Shared Congestion of Flows via End-to-end Measurement," in *Proceedings of ACM SIGMETRICS (Measurement and Modeling of Computer Systems)*, 2000, pp. 145–155, extended version to appear in IEEE/ACM Transactions on Networking.

[19] D. Katabi and C. Blake, "Inferring Congestion Sharing and Path Characteristics for Packet Interarrival times," MIT-LCS-TR-828, December 2001.

[20] V. Padmanabhan, L. Qiu, and H. J. Wang, "Passive Network Tomography Using Bayesian Inference," in *Internet Measurements Workshop*, Marseille, France, November 2002.

[21] O. Younis and S. Fahmy, "On Efficient On-line Grouping of Flows with Shared Bottlenecks at Loaded Servers," Purdue University/Technical Report CSD-TR-02-018, August 2002.

[22] H. Wadsworth, Ed., *Handbook of Statistical Methods for Engineers and Scientists*, 2nd ed. McGraw-Hill, 1998.

[23] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," RFC 1323, May 1992.

[24] J. Postel, "Transmission Control Protocol," RFC 793, September 1981.

[25] H. Chang, R. Gopalakrishna, and V. Prabhakar, "Intelligent Grouping of TCP Flows for Coordinated Congestion Management," Purdue University/Technical Report CSD-TR-01-017, 2001.

[26] T. Tuan and K. Park, "Multiple Time Scale Congestion Control for Self-Similar Network Traffic," *Performance Evaluation*, vol. 36, pp. 359–386, 1999.

[27] P. Berkhin, "Survey of Clustering Data Mining Techniques," Accrue Software, 2002.

[28] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker, "On the Constancy of Internet Path Properties," Internet Measurements Workshop (IMW'01), 2001.

[29] R. O. Duda, P. E. Hart, and D. G. Stork, Eds., *Pattern Classification and Scene Analysis, Part 1: Pattern Classification*, 2nd ed. John Wiley, 2001.

[30] UCB/LBNL/VINT groups, "UCB/LBNL/VINT Network Simulator," http://www.isi.edu/nsnam/ns/, May 2001.

[31] M. W. Garrett and W. Willinger, "Analysis, Modeling and Generation of Self-Similar VBR Video Traffic," in *Proceedings of ACM SIGCOMM Conference*, London, UK, 31st - 2nd 1994, pp. 269–280.

[32] M. Allman, "A Web Server's View of the Transport Layer," *ACM SIGCOMM Computer Communications Review (CCR)*, vol. 30, no. 5, pp. 10–20, June 2000.

[33] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616, June 1999.

[34] Z. Liu, N. Niclausse, and C. Jalpa-Villaneuva, "Traffic Model and Performance Evaluation of Web Servers," *Performance Evaluation*, vol. 46, no. 2, pp. 77–100, 2001.

[35] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," in *Proceedings of ACM SIGMETRICS*, July 1998.

[36] J. Wallerich, "Design and implementation of WWW workload generator for the ns-2 network simulator," August 2001, http://www.net.uni-sb.de/~jw/nsweb (also on ns-2 web page).

[37] L. Kernel, "http://www.kernel.org," 2002.

[38] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An Integrated Experimental Environment for Distributed Systems and Networks," in *Proc. of the Fifth Symposium on Operating Systems Design and Implementation.* Boston, MA: USENIX Association, Dec. 2002, pp. 255–270.

[39] T. T. Generator, "http://www.postel.org/tg/," SRI International and USC/ISI Center for Experimental Networking, January 2002.

[40] N. A. tool for Network Experimentation and Measurement, "http://www.ittc.ku.edu," Information and Telecommunication Community Center, University of Kansas, April 1999.

[41] Planetlab, "http://www.planet-lab.org," 2003.

[42] P. 1.1.0, "http://www.cc.gatech.edu/fac/Constantinos.Dovrolis/pathload.html," April 2003.

Ossama Younis (S '02 / ACM S '02) received his B.S. and M.S. degrees from the Computer Science Department, Alexandria University, Egypt, in 1995 and 1999, respectively. Since 2000, he has been pursuing his Ph.D. degree at the Department of Computer Sciences, Purdue University. His current research interests include sensor networks, Internet routing and tomography, and network security. His email address is: oyounis@cs.purdue.edu

Sonia Fahmy (S '96–A '00–M'03 / ACM '94) received her PhD degree at the Ohio State University in August 1999. Since then, she has been an assistant professor at the Computer Sciences department at Purdue University. Her research interests are in the design and evaluation of network architectures and protocols. She is currently investigating Internet tomography, overlay networks, network security, and wireless sensor networks. Please see http://www.cs.purdue.edu/homes/fahmy/ for more information. Her email address is: fahmy@cs.purdue.edu