On Efficient On-line Grouping of Flows with Shared Bottlenecks at Loaded Servers

Ossama Younis and Sonia Fahmy*
Department of Computer Sciences, Purdue University, W. Lafayette, IN 47907–1398, USA
e-mail: {oyounis,fahmy}@cs.purdue.edu

Technical Report CSD-02-018

Abstract

We design an efficient on-line approach, FlowMate, *for partitioning flows at a busy server into flow groups that share bottlenecks. These groups are periodically input to congestion coordination, aggregation, load balancing, admission control, or pricing modules.* FlowMate *uses in-band packet delay measurements to the receivers to determine shared bottlenecks among flows. Packet delay information is piggybacked on returning feedback, or, if impossible, flow (e.g., TCP) round trip time estimates are used. We simulate* FlowMate *to examine the effect of network load, traffic burstiness, network buffer sizes, and packet drop policies on partitioning correctness. Our results demonstrate accurate partitioning of medium to long-lived flows even under heavy load and self-similar background traffic. Experiments with HTTP/1.1 flows demonstrate difficulties in partitioning bursty foreground traffic. We also study fairness of coordinated congestion management when integrated with* FlowMate.

beginkeywords network measurement, network tomography, TCP, shared bottleneck identification, coordinated congestion control, load balancing endkeywords

1 Introduction

Current end system congestion control mechanisms regulate the sending rate of each individual connection (flow) according to network conditions assessed by that particular connection. Recent research has shown that coordinating congestion control decisions among certain flows at a busy end system (e.g., ftp/Web server) can increase the collective performance of the flows [3, 19]. An important problem in addressing coordinated congestion management is the partitioning of flows from a single sender to multiple receivers into groups, in order to perform congestion management decisions on a per-group basis. Figure 1 depicts a multiple receiver topology (referred to as Inverted-Y in [22]). In current coordinated congestion management approaches [2, 9, 18, 23], flows between the same hosts (or same LANs) are grouped together. 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 (NATs), quality of service (e.g., using several queues at certain router ports), load balancing schemes, and dispersity routing [1]. 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 grouped and coordinated. Moreover, extending coordination benefits to flows that share the same bottlenecks, but are *not* destined to the same host, can significantly enhance performance.

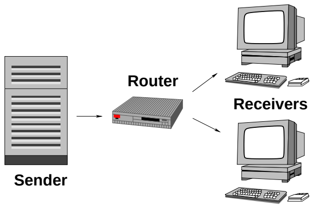


Figure 1. Logical inverted-Y topology

In this paper, we examine on-line partitioning of flows at a sender into groups with shared bottlenecks, without introducing out-of-band traffic. The problem can be stated as follows: given a set of flows $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$, we design a mapping protocol $\mathcal{P}$ that maps each flow f_j to exactly one group g_i, $1 \leq i \leq n$, such that $\forall i$, all flows $f_j \in g_i$ share a common bottleneck. Our approach, which we call *FlowMate*, can be integrated with any congestion coordination scheme that coordinates decisions within each group g_i,

*This research has been sponsored in part by the Purdue Research Foundation, and the Schlumberger Foundation technical merit award.

such as the Congestion Manager [3] or TCP-Int [2]– the partitioning and coordination schemes are completely orthogonal. *FlowMate* outputs flow groups that can also be input to load balancing, admission control, and pricing modules.

We use the packet delay correlation test proposed in [22] to periodically determine shared bottlenecks and partition flows. Delay correlation tests usually converge faster than loss correlation, and yield more accurate results. Delay correlation tests, however, impose the requirement of timestamping packets. We extend the techniques for timestamping packets in [13] for this purpose. The TCP timestamping option is currently supported in TCP implementations in most operating systems, such as FreeBSD, Linux, and Windows (it is currently enabled by default in the Windows 2000 TCP implementation). We use TCP *round trip time (RTT)* estimates (which TCP maintains for timeout computation purposes), however, if timestamping is not possible.

Since TCP flows comprise the majority (80% or more) of traffic in the Internet, we experiment with TCP flows, although our algorithm can be generalized to any flow for which delay information can be obtained (e.g., feedback information gathered by RTCP). Partitioning requires a time scale larger than the life-time of very short TCP connections (e.g., small HTTP/1.0 transfers) to converge. Long TCP connections (such as file downloads) still comprise a dominant traffic *load* on the Internet (the heavy-tail portion of the distribution). At the server, partitioning such medium to long-lived connections (called elephants in the literature, e.g., [15]), and coordinating congestion decisions within a group, increases responsiveness and fairness among all flows originating from the same server. We integrate our algorithm with a coordinated congestion management strategy and illustrate the improved fairness.

FlowMate has the following features that distinguish it from other approaches in the literature: (1) no generation nor transmission of out-of-band probing traffic, (2) on-line adaptivity to flow and network dynamics during flow lifetimes, (3) completely end-to-end: sender side only, or with timetamping support at receivers, and (4) low overhead and complexity.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 explains our design in detail. Section 4 analyzes the performance of *FlowMate* in a number of network configurations with HTTP, FTP and Telnet traffic, and demonstrates the effect of network parameters on a correctness index metric, which we define. Section 5 studies the performance of *FlowMate* integrated with coordinated congestion management. Finally, section 6 summarizes our conclusions and discusses future work.

2 Related Work

Coordination among flows has been proposed and studied in [2, 3, 9, 19]. The congestion manager (CM) [3] provides a general framework for applications to coordinate congestion management decisions among flows between the same end systems. TCP-Int [2] uses one congestion window for all concurrent TCP connections between the same end systems. Ensemble-TCP [9, 23] also groups flows between the same end systems, and caches information about the measured state of the network to expedite the start-up of new connections. TCP Fast Start [18] additionally marks these extra packets sent by new connections (more than the usual slow start permits) with a drop-preference flag.

Padmanabhan [19] 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 bottleneck links among flows.

Recently, a number of studies have investigated the inference of internal network characteristics using end-to-end measurements (sometimes referred to as “network tomography”), by applying innovative statistical techniques [5, 8, 12, 16, 22]. Katabi et al [16] use an entropy fuction to compute correlation among flows at then receiver. This technique does not require probe traffic and proposes general flow partitioning algorithms, but partitioning correctness degrades under heavy cross-traffic. More recent measurement results using Renyi (as opposed to Shannon) entropy demonstrate more robust partitioning [17]. Rubenstein et al [22] propose novel loss and delay correlation tests among flows to determine shared bottlenecks. They inject Poisson probes to collect loss or delay information. They do not use in-band measurements or present a general partitioning algorithm. Moreover, they do not discuss maintaining information from multiple receivers. We adopt their delay-correlation test, but address the additional issues required for its on-line application for multiple flows at a busy server. Delphi [21] sends probes at the sender to collect delay information from receivers organized in a multifractal wavelet model to infer the amount of cross traffic at certain bottlenecks. The accuracy of this approach depends on the utilization levels at bottlenecks. The higher the utilization, the more accurate the computed estimates. Harfoush et al [12] use Bayesian probing instead of Markovian probing to infer shared losses. Their approach is more effective with active probing, rather than in-band measurements.

To avoid problems with collecting delay information and clock synchronization, correlation among TCP round trip time (RTT) estimates (e.g., [6]) or throughput estimates at the sender (e.g., [24]) may substitute one-way delays. Although using these metrics eliminates any changes (e.g., timestamping support) to the receiver, the delay on the forward path cannot be isolated from that of the reverse path and the delays at the receivers themselves, as discussed in the next section.

3 *FlowMate* Design

This section describes our system and analyzes its complexity. Figure 17 contains pseudocode for FlowMate.

3.1 Basic Architecture

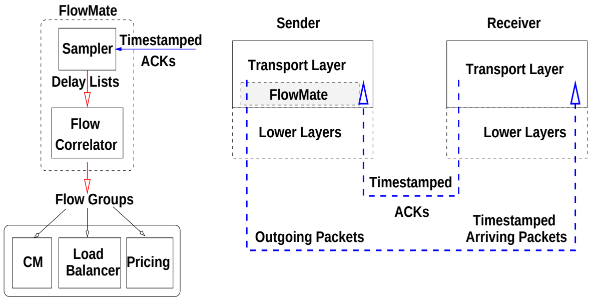


Figure 2. *FlowMate* **organization**

The *FlowMate* module can be invoked to provide information about groups of flows sharing common bottlenecks along their paths from a sender to various receivers. The *FlowMate* organization is depicted in figure 2. Basic modifications to the TCP implementation are required at the sender side to provide delay samples for correlation. Packets are timestamped before being sent. Usable samples are later selected at the "Sampler" when timestamped ACKs are received, as described in section 3.3. Sample delay lists are then provided to the "Flow Correlator" module, which performs partitioning and sends the resulting groups to other modules, e.g., load balancer.

3.2 Correlation Tests

The delay correlation test that we use in *FlowMate* was proposed in [22] to statistically identify shared bottlenecks using Poisson-distributed probe packets. We apply an analogous method on actual data packet delays. Pearson's correlation function [22, 26] is used on the delay samples as follows:

$$r_{xy} = \frac{\Sigma(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\Sigma(x_i - \overline{x})^2 \Sigma(y_i - \overline{y})^2}}$$

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. The closer r_{xy} approaches $+1$ (-1), the more positively (negatively) linear the samples (x_i, y_i) are. If $r_{xy} = 0$, the samples show no linear relationship.

The correlation test among two flows is defined as follows [22]: (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 within a 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 after the bottleneck is smaller than the spacing between packets within the same flow.

3.3 Delay Computation

The delays of packets on the forward path from sender to receiver should be collected at the sender. If timestamping ACKs is not possible, RTT samples (which TCP anyway computes for retransmission timeout calculation purposes) are used instead. The receiver need not handle the TCP timestamping option field (or an equivalent application layer mechanism) in this case: the receiver is entirely *FlowMate*-unaware. Using RTT information instead of forward delay may, however, degrade the partitioning accuracy when bottlenecks in the reverse direction alter the packet delay correlation properties. Furthermore, delayed acknowledgments (and even the operating system and scheduling at the receiver) affect the RTT. We have repeated all our experiments in section 4 with RTT samples instead of one-way delays, and the reduction in accuracy values was less than 5%. This performance degradation is primarily due to the interference of reverse path bottleneck dynamics with delay correlation values.

Standard timestamping mechanisms presented in [13] use the Options field in the TCP header [20] to include the time a packet is sent by the sender, and the time an ACK is sent by the receiver, as shown in figure 3. We extend this field (KIND=8) to also include the time at which the packet was received. (Alternatively, this information can be added to the application layer payload if the receiver does not support this extension.) Note that clock-skewness between the sender and receiver clocks is not a problem if it is approximately constant throughout the flow duration.

1	1	4	4
Kind = 8	10	TS Value	TS Echo Reply

1	1	4	4	4
Kind = 8	14	TS Value	TS Echo Reply	TS Recv. Time

Figure 3. Extending the timestamped ACK options field

3.4 In-Band Packet Sampling

The scalability of out-of-band delay correlation tests to flows at a busy server is limited due to the need for generating and transmitting Poisson probes on all flow paths [22]. To avoid injecting out-of-band control traffic in the network, we use selected data packets as samples. The sampling proceeds as follows. For the two flows being tested, we merge the two sets of sample delays according to their packet send times. We compute the average spacing between every two consecutive packets, t. To compute the autocorrelation of one of the two flows, samples are selected from its sample set with packet spacing higher than t. This is the main restriction on the correctness of the correlation tests (as explained in [22]), and not how probes are distributed. To verify this, we repeated our experiments with the simple sampling approach illustrated in figure 4. 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. Our results were not significantly different in both cases. Therefore, in section 4, we only use the inter-packet spacing restriction.

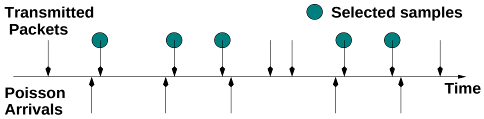


Figure 4. Poisson-like sampling

3.5 Data Structures

FlowMate uses three main data structures to maintain information about flows and groups, as shown in figure 5:

- The *Flow Info* maintains information related to a flow with flow ID *fid* and destined to host address *dst*. Two lists are maintained for each flow: (1) *SampleList* maintains sample delay values gathered during the current interval of time. This list is reset after partitioning is performed for the current interval; and (2) *CorrList* is used to maintain correlation history of one flow with other group flows (used in future partitioning for better accuracy). After performing correlation tests for two flows, this information is stored in the history list of the flow with smaller *fid*.

- The *Flow Table* maintains *Flow Info* of all flows. The table is hashed based on the last 8 bits of the destination address to speed up searches [14]. (Moreover, flows to the same destination IP address can be found in the same "bucket." This facilitates applying simple partitioning mechanisms that assume no NAT or dispersity routing [1] for certain flows.)

- The *Group List* is the final output of each partitioning process. This list is reset before re-partitioning. Each "group" contains a list of highly correlated flows. Due to the locality patterns of flows and the power-law properties of Internet topology, the number of groups is usually limited during any given interval, consequently aiding in performing better coordinated congestion management/load balancing/pricing decisions.

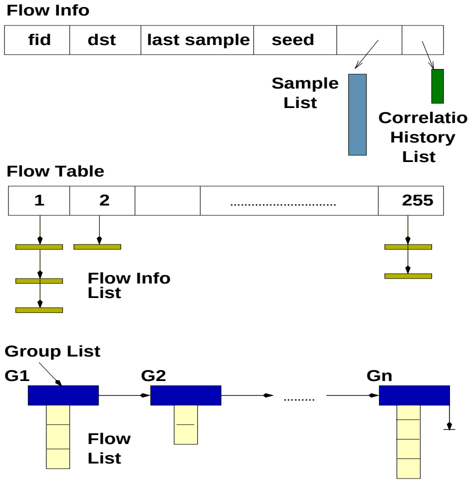


Figure 5. Data structures used in *FlowMate*

3.6 Triggering Partitioning

It is important to trigger partitioning only when sufficient samples are usable. This cannot be easily achieved for all flows, however, since each flow has its own congestion window according to its start time and encountered losses. Assume that the last partitioning was invoked at time t. We next trigger partitioning 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 usually adequate. Under low background load, at least 20 samples are required for accurate results. The value of k is also dependent on how packets of various flows are interleaved. With little interleaving, more samples are required, as discussed in section 4.3.5. If a time d_{max} elapses before the threshold M is satisfied for all flows, partitioning is automatically triggered. In this case, we only consider flows with sufficient samples. To prevent frequent triggering, partitioning is not invoked before a period d_{min} elapses since the last partitioning.

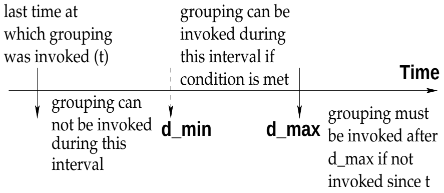

last time at which grouping was invoked (t)
grouping can be invoked during this interval if condition is met
Time
grouping can not be invoked during this interval
d_min
d_max
grouping must be invoked after d_max if not invoked since t

Figure 6. Summary of triggering conditions

3.7 Partitioning

FlowMate partitioning starts with empty group lists and a set of target flows (with sufficient samples) to be grouped. We designate a "representative" flow in every group. A new flow is only compared to the group representative to determine whether it should join the group. This ensures that all flows that are grouped together are highly correlated with the same representative flow. *FlowMate* selects the first flow in a group to be its representative. Selecting and switching the group representative dynamically is currently under study. A new flow is compared to *all* group representatives to determine if it should join an existing group or create a new group. Consider, however, the case when a new flow is highly correlated with more than one group representative. *FlowMate* follows a conservative rule: *no grouping* is better than *incorrect grouping*. The cross correlation coefficients of the new flow in all successful tests are compared, and the flow joins the group with highest cross coefficient. This is because a flow typically exhibits the highest correlation with the correct group. Optionally, whenever a new group is created, all flows in other groups, except for the representatives, may be compared to the new group representative to determine if they have a higher correlation with the newly created group. This technique increases accuracy in cases where flow delay patterns are similar. Note that the cross and auto measures and their delay statistics are maintained and continuously updated for every pair of flows that have been tested. When partitioning is triggered, new samples update the mean and variance of flow delay samples, and consequently, the corresponding cross and auto measures.

3.8 Time Complexity

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 to process for each flow. Computing the coefficients depends on the number of selected samples, which is less than the number of delay values received. Assume that N flows are currently being partitioned; G is the number of generated groups; and S_g is the average group size. *FlowMate* time complexity is $O(NG)$, where G is approximately N/S_g. This is better than comparing every pair of flows which is $O(N^2)$. Therefore, *FlowMate* partitioning is a lower-cost approximation of the K-Means clustering technique [7]. In addition, flows with insufficient samples are excluded from partitioning, which may further reduce complexity. *FlowMate* overhead is lowest if only a few large groups are formed. Large groups do not require as many correlation tests among individual flows (due to the representative-based approach). The worst case occurs if all flows do not share any common bottlenecks and each is grouped separately, which would not occur often. This is due to the locality of server requests, as well as Internet topology power-law characteristics.

Table 1. Simulation parameters

TCP flows	12–48; infinite FTP flows; Telnet flows; 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)
Queue size	250 packets (except in one experiment)
Drop policy	Drop-Tail (RED in one experiment)

4 Performance Analysis

We have implemented *FlowMate* in the ns-2 network simulator [25]. In this section, we conduct several experiments to evaluate its performance. We investigate *FlowMate* robustness under heavy background traffic using Pareto sources or self-similar traces, and with various foreground traffic models, including FTP, Telnet and HTTP. We also study the effect of physical bandwidth constraints, buffer sizes, drop policies and *FlowMate* parameters. Table 1 summarizes the simulation parameters. Two topologies (one symmetric and one asymmetric) are used in the experiments. In the first topology (figure 7), a single source has 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. Other multiplexed Pareto flows are generated by the receivers in the reverse direction.

Figure 8 depicts the second simulation topology, where the upper two branch links have limited bandwidth, while the link on the third branch is congested by high background traffic load. This topology is not as symmetric as

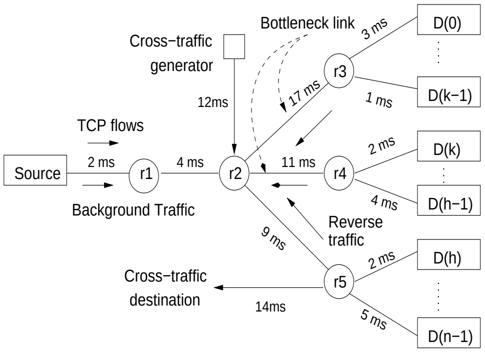


Figure 7. Simple simulation configuration

the first one. Background traffic is injected using a real traffic trace (the "Star Wars" movie [11]). One "Star Wars" flow is transmitted on each of the three main branches starting from router $r2$ to a randomly-chosen receiver on each branch, so as not to create a bottleneck on the main shared path. In both topologies, three groups of flows comprise the expected partitioning: one group for each one of the three branches.

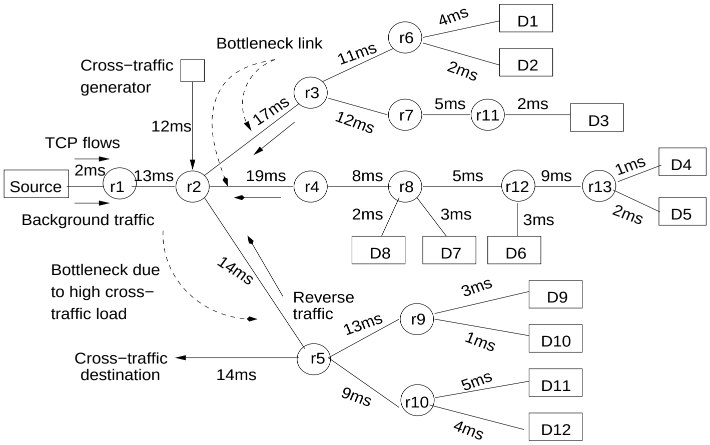


Figure 8. More complex simulation configuration

4.1 Group Accuracy Metric

Partitioning inaccuracies are introduced by either erroneous grouping of flows (including merging two or more groups) or splitting a group into two or more subgroups. We use the term "false sharing" (fs) to denote erroneous grouping of a flow with a group it does not share bottlenecks with. Let N denote the total number of flows; G_c denote the set of correct groups; G_r denote the set of resulting groups; n_{fs} denote the number of flows grouped erroneously in a resulting group; and s_j denote the number of subgroups of a correct group $\in G_c$ that was split into s_j subgroups in G_r. The group accuracy index (AI) is computed as follows:

$$Accuracy\ Index\ (AI) = 1 - \frac{\sum_{i=1}^{|G_r|}(n_{fs})_i}{N} - \frac{\sum_{j=1}^{|G_c|}(s_j - 1)}{N}$$

where $(n_{fs})_i$ of a group $g_i \in G_r$ is computed as follows: Map g_i to a corresponding group $g_c \in G_c$, such that $|g_i \cap g_c|$ is maximized. The total number of flows f such that $f \in g_i \wedge f \notin g_c$ is the number of flows grouped erroneously $(n_{fs})_i$.

For example, consider 6 flows with correct groups {1,2,3} and {4,5,6}. If the groups produced 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 grouped, and another one sixth is deducted for the split of group {4,5,6} into groups {4,5} and {6}. Note that a single flow is penalized only once, either for being grouped incorrectly, or for not being grouped (merged). Table 2 gives some additional examples. There is no case in which *all* flows are erroneously grouped. Therefore, the accuracy index varies between a fraction (above 0) and 1. For a fixed number of flows, as the number of correct groups increases (decreases), the average number of flows per group decreases (increases). Therefore, the merge effect is, on the average, diminished (exacerbated). The split effect is constant, since it only depends on the number of flows. Our interpretation of accuracy considers a group split into two or more groups to be of equal severity (thus prompting an equal deduction) to incorrect grouping of one flow (while incorrect merging of two groups entails a penalty for each flow that was incorrectly merged with the larger set). This may be too strict, since group splits often occur during transient periods. In addition, group splits have fewer undesirable effects than false sharing. A group split simply does not exploit the full benefits of coordination among the group, but the consequent decisions (congestion control, load balancing, or pricing) are not incorrect. This is in contrast to false sharing which may, for example, cause a flow to enter the slow start phase if other members of that group are bottlenecked. We are currently investigating the effectiveness of our accuracy index metric more carefully.

4.2 *FlowMate* Accuracy

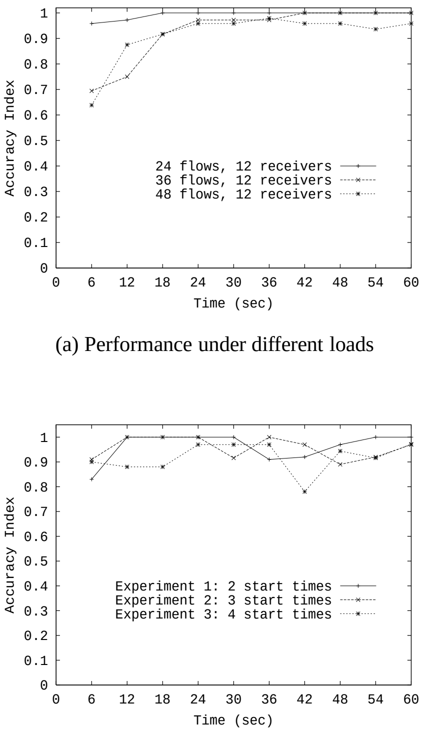
In this section, we discuss the results of experiments on the topology in figure 7. In our first experiment, we compute the accuracy index with different numbers of flows. Figure 9(a) shows the performance using 24, 36, and 48 TCP flows as foreground traffic. To interpret the results more easily, we trigger partitioning at fixed intervals and

Table 2. Computing the accuracy index (AI) for 10 flows with 2 optimal groups ({1,..,5},{6,..,10})

Output Groups	AI	Interpretation
All split: {1}, {2},· · ·,{10}	0.2	1 correct flow per group
All merged: {1,2,· · ·,10}	0.5	only 1 correct group
Splitting: {1,2,3}, {4,5}, {6,7,8}, {9,10}	0.8	2 errors (splits)
More splitting: {1,2}, {3,4}, {5}, {6}, {7,8}, {9,10}	0.6	4 errors (splits)
Some false sharing: {1,· · ·,7}, {8,9,10}	0.8	2 errors (flows 6 and 7)
More false sharing: {1,· · ·,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)

do *not* trigger it early if sufficient samples are received before d_{max}. The value used for d_{max} is 6 seconds. Therefore, the results of the first partitioning can be seen at time 6 seconds, the second at time 12, and so on. Triggering partitioning according to the number of samples (as proposed in section 3.6) may improve system performance (e.g., congestion control or load balancing) if it occurs between d_{min} and d_{max}. The main effect of only triggering at d_{max} intervals on *FlowMate* accuracy computation is to alter the number of flows considered for partitioning (according to their number of samples). Note that we compute the accuracy index by comparing against a static correct partitioning, even though the background traffic variations entail a dynamic partitioning goal. We select this more conservative approach for ease of accuracy index computation, and to show the worst case index value.

We observe that in steady state, performance is reasonable (average index $>$ 90%). During the initial transient period, which includes the first one or two partitioning invocations, sample delay patterns are not unique for each group of flows, so accuracy is lower. After the transient, accuracy is higher: observed inaccuracies are mostly due to a few group splits. Flows used in this experiment start at 10 to 50 ms intervals 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 flows), 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 groups, each starting at times near 0, 18, 30, and 48 seconds. The performance results are depicted in figure 9(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 the steady state, and if a dynamic accuracy metric (that considers transient bottlenecks) is used, the accuracy index increases.



(a) Performance under different loads

(b) Performance with staggered start times

Figure 9. Accuracy index with *FlowMate*

4.3 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 [22]. The packet drop, policy and traffic patterns may also impact the results. We demonstrate the effect of these parameters on the topology shown in figure 8. The effect of varying the maximum correlation interval duration d_{max} does not have a profound impact on the 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 section as shown in figure 10. As mentioned above, the “Star Wars” trace is used as a source of self-similar background traffic, except when varying background traffic load, when 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 partitioning is three groups– one for each main branch. Simulation time is 60 seconds. This allows the effect of the transients to be visible, even in experiments where the average accuracy is computed over the simulation period.

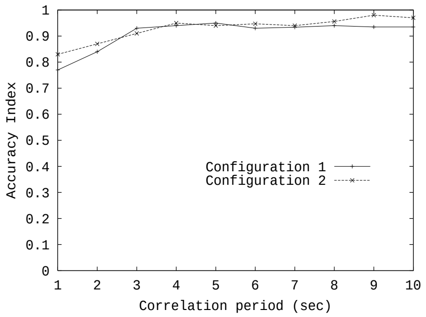
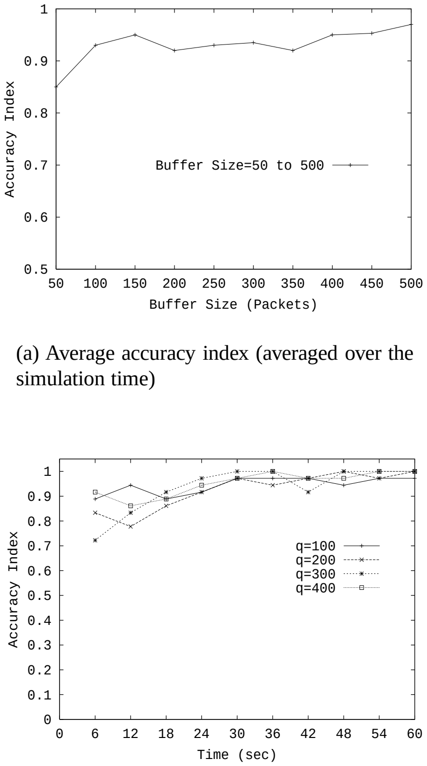


Figure 10. Effect of the maximum correlation period



(a) Average accuracy index (averaged over the simulation time)

(b) Transient and steady state performance

Figure 11. Effect of router buffer size

4.3.1 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 figure 11-(a). Detailed results for specific buffer sizes are shown in figure 11-(b). Variation in performance is more pronounced during the transient period, which is expected any time a large number of connections start at the sender simultaneously. We believe that having routers with larger buffers usually enhances performance.

4.3.2 Packet Drop Policy

The most common drop policy used in routers is Drop-Tail. We use this policy in all our experiments, except in this experiment, where we use Random Early Detection (RED). Figure 12 shows the resulting accuracy index in three cases. One case uses the Drop-Tail policy for all queues, another case uses some Drop-Tail and some 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 [12] 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 group for a certain flow. This is consistent with the results presented in [22]. The Drop-Tail policy currently prevails in Internet routers, however, and even with the use of other policies in *some* routers on a path, *FlowMate* still performs reasonably well.

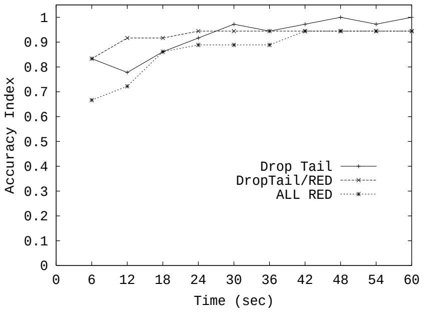


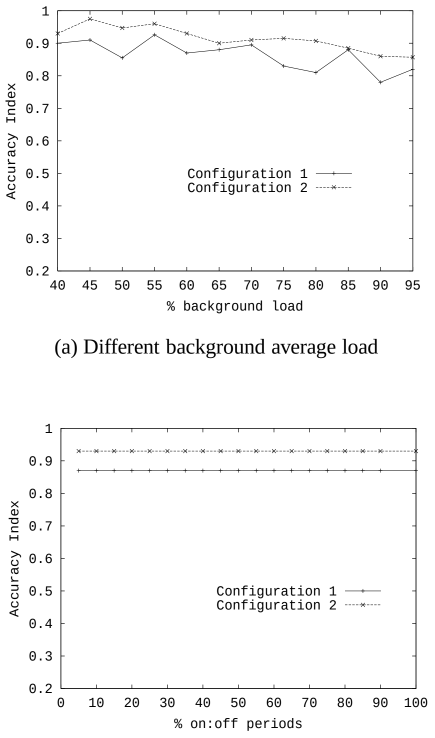
Figure 12. Effect of packet drop policy

4.3.3 Background Traffic Load

We study the performance of *FlowMate* in our two configurations (figure 7 and figure 8) under different background traffic 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 figure 13-(a) are computed according to the first branch which has the least physical bandwidth; load is slightly lower on other branches. Results show that *FlowMate* is robust under 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, depicted in figure 13-(b) illustrated that performance is consistent, which indicates that different on/off period ratios have a relatively minor effect on the partitioning accuracy. It is worth noting that performance on the more complex configuration is superior to the simpler one. This can be attributed to its asymmetric nature.

4.3.4 Foreground Traffic Load

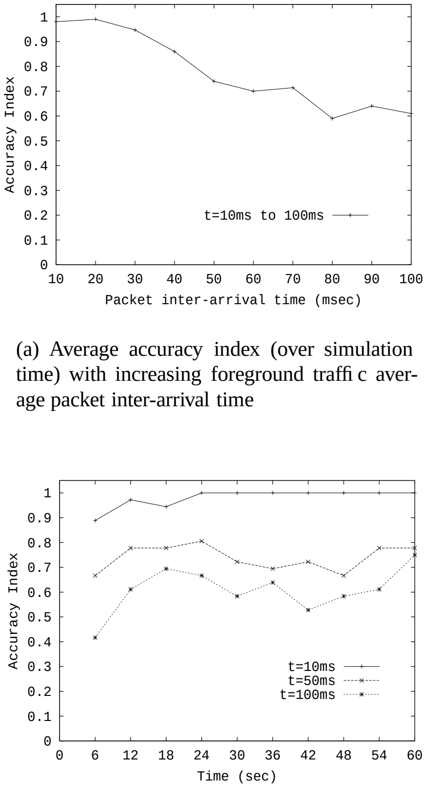
In our experiments thus far, we have used FTP applications as our foreground traffic sources. In this experiment, we demonstrate the effect of higher burstiness in foreground traffic, and determine the number of samples required for correct results. We use Telnet traffic with bursty packet inter-arrivals, and control the packet inter-arrival mean t. Figure 14-(a) shows results with different inter-packet arrival periods. As shown in figure 14-(b), a large t value 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 saw group 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 partitioning entirely or impacts the results.



(a) Different background average load

(b) Different ratios of on-off periods

Figure 13. Performance under different background load and burstiness



(a) Average accuracy index (over simulation time) with increasing foreground traffi c average packet inter-arrival time

(b) Transient and steady state performance

Figure 14. Performance degradation with bursty Telnet traffic

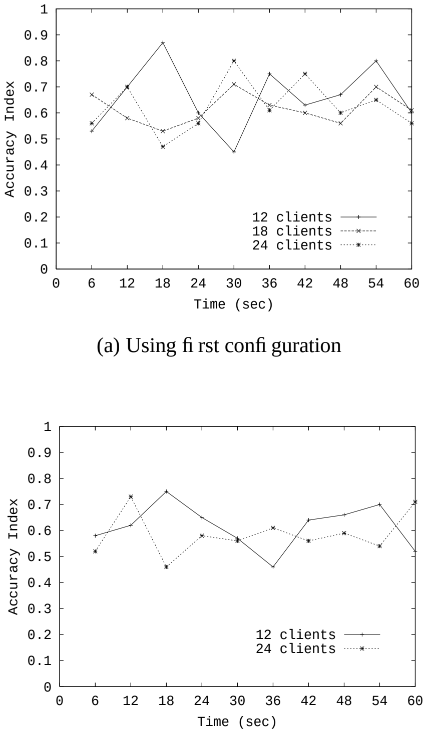
Table 3. HTTP simulation parameters

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

4.3.5 HTTP Traffic

Many problems arise when HTTP traffic is considered. First, most HTTP connections are short-lived [15]. This implies that a connection may very well terminate before partitioning 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, then 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. Fortunately, these problems are somewhat mitigated by HTTP/1.1 with persistent or pipelined connections [10]. The HTTP/1.1 specification entails that connections are not 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.

FlowMate was applied to HTTP/1.1 traffic on the two configurations in figure 7 and figure 8. We used the SURGE model [4] for web workload traffic generation. This model is implemented in “nsweb” [27]. Table 3 summarizes the HTTP/1.1 parameters used in our experiments. SURGE parameters are chosen as in [4], while other parameters used in the experiments are similar to those in [27]. Figure 15-(a) depicts the performance of *FlowMate* using different numbers of web clients on the first configuration with 12 receivers (figure 15-(b) shows results for the second configuration). Performance is similar with different numbers of clients. We have found that accuracy is actually higher than what is computed by our accuracy metric. This is because the metric compares against static groups throughout the simulation, and does not capture scenarios where two flows have samples with totally disjoint sets of send times. In such cases, correlation fails (correctly), and *FlowMate* avoids false sharing. We conclude that partitioning HTTP flows significantly depends on two main factors, namely, connection lifetime and traffic burstiness. While it is still possible for *FlowMate* to perform reasonably well under some burstiness, connection life-time is crucial in determining if partitioning is applicable. When partitioning 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.



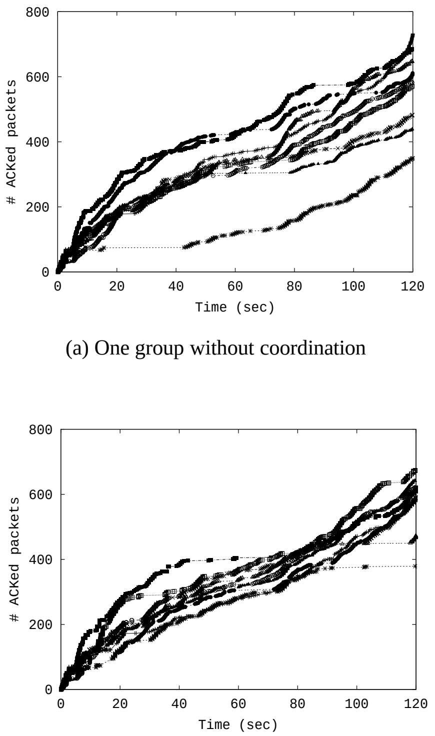
(a) Using first configuration

(b) Using second configuration

Figure 15. Using *FlowMate* with HTTP/1.1

5 Application of *FlowMate* to Coordinated Congestion Management

In this section, we demonstrate one application that may benefit from *FlowMate*, namely, coordinated congestion management. As previously stated, groups of flows are provided as the input to any coordinated congestion management scheme, such as CM. We implement a simple coordination mechanism that works as follows. Each flow maintains its own congestion window. When loss is detected by any member of a group, all group member windows are reduced to react to incipient congestion. All group members increase their windows after *three* consecutive window increases within the group. Thus, flows react more conservatively to detected available bandwidth. Experiments are conducted using the configuration in figure 8. Figures 16(a) and (b) show the number of ACKed packets during a simulation period of 120 seconds for one of the resulting groups, without and with *FlowMate* and simple coordination. Figure 16(b) illustrates that the flow throughput values are more similar and consequently fairness among flows sharing a common bottleneck is better with *FlowMate*. We believe that using flow groups generated by *FlowMate* in schemes such as [2, 3, 9, 18, 23] will extend the benefits of these congestion coordination schemes to flows with different destinations but common bottlenecks. Moreover, *FlowMate* will also false sharing of state among flows with different bottlenecks.



(a) One group without coordination

(b) One group with coordination using *FlowMate*

Figure 16. Using *FlowMate* for congestion coordination

6 Conclusions and Future Work

In this paper, we have presented *FlowMate*, an algorithm that exploits end-to-end packet delays to periodically partition flows originating at a busy sever into groups, based upon whether they share bottlenecks. *FlowMate* does not require generation and transmission of probe traffic for collecting delay information. Although using out of band probes introduces little load (usually about 5% of the total load), the overhead of generating probe flows is proportional to the number of flows to be grouped. Moreover, a flow and its corresponding probe flow may not follow the same path, and may, consequently, face different bottlenecks. This emphasizes the need for a scheme to dynamically group flows based on in-band measurements.

FlowMate will likely produce multi-member groups at a busy server, due to the locality of requests and Internet topology characteristics. Therefore, *FlowMate* complexity, which depends on the number of groups, is reasonable. *FlowMate* accuracy is high in various configurations with different propagation delays, bottlenecks, buffer sizes, and drop policies. The main factor that degrades performance is the burstiness of the flows being partitioned themselves, as seen in our HTTP/1.1 and Telnet results. Background traffic load and burstiness do not have a detrimental effect, due to our design which considers the history of correlation statistics.

We have implemented *FlowMate* in the Linux kernel v2.4.17. We plan to measure the benefits of *FlowMate* with coordination schemes in wide area experiments. UDP flows may also be considered by measuring delays at the application layer. For example, RTP flows can be grouped with TCP flows or with each other (at large time scales) and controlled according to multimedia application requirements. Finally, we will integrate *FlowMate* into other components in addition to congestion management– specifically load balancing in overlay networks.

Acknowledgments

We would like to thank Dan Rubenstein (Columbia University) for providing us with his correlation computation code; Anja Feldmann and Jorg Wallerich (University of Munich) for answering our “nsweb” questions; and the anonymous reviewers for their valuable comments.

References

[1] S. A. Akella, S. Seshan, and H. Balakrishnan. The Impact of False Sharing on Shared Congestion Management. CMU-CS-01-135, June 2001.

[2] 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.

[3] 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.

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

[5] R. Caceres, N. Duffield, J. Horowitz, D. Towsley, and T. Bu. Multicast-based Inference of Network-internal Characteristics: Accuracy of Packet Loss Estimation. In *Pro-*

ceedings of the IEEE INFOCOM, New York, March 1999. http://www.ieee-infocom.org/1999/papers/03a_04.pdf.

[6] 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.

[7] R. O. Duda, P. E. Hart, and D. G. Stork, editors. *Pattern Classification and Scene Analysis, Part 1: Pattern Classification*. John Wiley, second edition, 2001.

[8] 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.

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

[10] 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.

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

[12] K. Harfoush, A. Bestavros, and J. Byers. Measuring Bottleneck Bandwidth of Targeted Path Segments. BUCS-2001-016, July 2001.

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

[14] R. Jain. A Comparison of Hashing Schemes for Address Lookup in Computer Networks. *IEEE Transactions on Communications*, 40(3):1570–1573, 1992.

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

[16] D. Katabi, E. Bazzi, and X. Yang. A Passive Approach for Detecting Shared Bottlenecks. In *Proceedings of IEEE ICCN*, October 2001.

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

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

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

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

[21] V. Rebeiro, M. Coates, R. Riedi, S. Sarvotham, B. Hendricks, and R. Baraniuk. Multifractal Cross-Traffic Estimation. In *Proceedings of ITC Specialist Seminar on IP Traffic Measurement, Modeling, and Management*, September 2000.

[22] 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)*, pages 145–155, 2000. Extended version to appear in IEEE/ACM Transactions on Networking.

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

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

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

[26] H. Wadsworth, editor. *Handbook of Statistical Methods for Engineers and Scientists*. McGraw-Hill, second edition, 1998.

[27] 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).

```
PROCEDURE TriggerPartitioning(condition)
 IF (condition is met)
   THEN return triggered
   ELSE return not triggered
END PROC


PROCEDURE SelectSamples(Flow f1, Flow f2)
   Flow f3 ← NULL
   Flow f4 ← NULL
   f3 ← merge samples from f1 and f2
   AvgDist ← average time between consecutive packets in f3
   f4 ← f2 packets with inter-packet distance $>$ AvgDist
   return f3 and f4
END PROC


PROCEDURE Regroup(GroupsList)
   $// optional\ procedure$
   $n$ ← numGroups
   FOR k ← 1 TO $n - 1$ DO
   BEGIN
      for each flow $f_i$ of $G_k$
       ($f_a$, $f_b$) ← SelectSamples($f_i$, $G_n$.representative)
       result = Test($f_a$, $f_b$)
       check (result.CrossCoeff $>$ original coeff with $G_k$)
       IF (TRUE) THEN BEGIN
         remove $f_i$ from $G_k$
         add $f_i$ to $G_n$
       END
   END
END PROC

FlowMate MAIN PROC
 Initialize:
    Groups List ← NULL
    Flows Table ← NULL
 Start:
    FOR i ← 1 TO numFlows DO
       SampleList($f_i$) ← NULL
       collect delay information from received ACKs
       store delay information in Flow Table
       check TriggerPartitioning(condition)
       If (triggered) THEN
       BEGIN
         Partition()
         generate GroupList
         Goto Start
       END
ENDPROC
```

```
PROCEDURE Partition()
   FOR i ← 1 TO numFlows DO
   BEGIN
      IF (GroupsList = NULL) THEN BEGIN
       $G_1$ ← create new group for $f_i$
       $G_1$.representative ← $f_i$
      END
      ELSE BEGIN
       MaxCoeff ← NEGATIVE_VALUE
       ChosenGroup ← NULL
       FOR k ← 1 TO numGroups DO
       BEGIN
         $f_k$ ← $G_k$.representative
         ($f_a$, $f_b$) ← SelectSamples($f_i$, $f_k$)
         result ← Test($f_a$, $f_b$)
         j ← min(i, k)
         $f_j$.CorrList.add(result)
         IF (result.success) and (result $>$ MaxCoeff)
         THEN BEGIN
            ChosenGroup ← $G_k$
            Update MaxCoeff
         END
       END
       IF (ChosenGroup = NULL) THEN BEGIN
         numGroups ← numGroups + 1
         k ← numGroups
         $G_k$ ← create a new group for $f_i$
         $G_k$.representative ← $f_i$
         Regroup(GroupsList)
       END
      END
   END
END PROC
```

Figure 17. The *FlowMate* algorithm