Passive Inference of Path Correlation*

Lili Wang, James N. Griffioen, Kenneth L. Calvert, Sherlia Shi
Laboratory for Advanced Networking
University of Kentucky
{lwang0,griff,calvert,sherlia}@netlab.uky.edu

ABSTRACT

Overlays have been proposed as a means to improve application performance in many areas, including multimedia streaming and content distribution. Some overlays use parallel transmission to increase aggregate throughput or use backup paths to improve reliability. For such applications, an important consideration is whether the "virtual links" at the overlay level (i.e. paths between overlay nodes) share links in the underlying network. In particular, choosing parallel or backup paths without any information about path correlation can reduce the effectiveness of the overlay.

In this paper we show how to use passive measurement of TCP throughput to provide information about path correlation, for use in overlay routing decisions. Our methods have the advantage that they send no probe traffic to collect path information. We present results of experimental evaluation in both controlled testbed (Emulab) and real wide area network (Planetlab). Our results demonstrate that the methods together work well across a wide range of operating conditions.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Measurement

Keywords

Path correlation

1. INTRODUCTION

Recently application designers have turned to overlay networks as a way to obtain new services in the Internet. Because overlays are constructed from a set of cooperating end systems, new services needed by emerging applications can be deployed without requiring any changes to the Internet's existing network layer. As such, overlays are particularly appealing for multimedia applications that often want to multicast the data (e.g., application level multicast services), transcode the data enroute, survive network failures or route changes, etc.

Multimedia applications are also unique in that application quality —video quality, picture quality, sound quality, etc.—correlates directly with the capacity ("bandwidth") available for transmission. In other words, increasing the bandwidth available to the application results in the ability to offer a higher-quality multimedia experience. By providing greater application control of routing, overlay networks have the potential to support multipath approaches that enable applications to achieve increased bandwidth by sending data across multiple paths simultaneously, resulting in superior sound/video quality to the end user.

Such an approach involves the creation of end-to-end "paths" by concatenating connections between overlay nodes. Each such connection, or *virtual link*, corresponds to an end-to-end path at the IP layer. For (overlay) applications that wish to use a multipath routing approach to increase bandwidth, an important issue is avoidance of shared bottlenecks: If two virtual links in the overlay network (i.e., IP-level paths between overlay nodes) share a link in the underlying network, and that link is the bottleneck of both, any application-level flows that traverse those two virtual links will end up competing with one another. Ideally, the application would be able to query the network to determine if two paths share a bottleneck link. Unfortunately, the Internet Protocol (IP) does not provide any simple way to obtain this information. Various heuristics have been developed to identify the bottleneck bandwidth or guess at the level of "shared congestion" [9]. Many of these approaches are based on active measurement techniques, in which the application and/or overlay nodes must inject additional packets into the network to determine the level of (shared) congestion [14].

In this paper, we investigate approaches based on the passive measurement of TCP throughput to determine the level of shared congestion. Our approach does not send any probe traffic, but rather passively gathers information, gradually learning of congested links and sharing that information with other nodes in the system. It is intended for use with long-lived, high-bandwidth applications that are adaptive enough to use TCP. We present results from experiments run over Emulab and Planetlab, showing that our correlation technique is able to detect shared congestion (e.g., paths with a shared bottleneck link) with a high degree of accuracy. Similarly, it can tell when paths are independent with a high degree of accuracy. These methods are being applied in an overlay service we are developing for immersive environments [15], which achieves high bandwidth by aggregating the throughput of multiple overlay paths.

*This work supported in part by NSF Grants EIA-0101242 and ANI-0121438.

NOSSDAV'04, June 16–18, 2004, Cork, Ireland.

2. RELATED WORK

Multipath routing techniques have been studied in other contexts, each addressing slightly different issues. One of the best-known multipath overlay approaches is RON [4], a fault-tolerant routing service that can quickly re-route packets via alternate overlay nodes in response to failures. RON is based on active probes that continuously monitor the fully-connected overlay's virtual edges. Because RON's goal is resilience and fault-tolerance rather than optimized throughput, accurately measuring bandwidth and identifying shared bottlenecks is not one of RON's objectives. Moreover, the active probing approach introduces scalability problems that limit the size of the overlay that can be supported. Work by Kommareddy *et al.* [13], describes a multipath approach based on the ability to access AS-level information. Their more recent work [11] presents a more general, measurement-based approach that attempts to balance the load to meet certain requirements; identifying shared bottlenecks is not a goal.

In the context of a different problem, some applications have used multiple paths from multiple replicated sources to increase throughput or improve reliability. Apostolopoulos *et al.* [5] suggests using multiple description coding and path diversity to improve loss resilience. The client parallelly retrieves the multiple complementary descriptions through different paths. Byers *et al.* [8] proposes using erasure codes downloading data in parallel from multiple mirror sites to speed up the data transfer. Their methods works well when multiple paths do not share loss.

Several approaches have been proposed to integrate congestion control across multiple concurrent flows. Balakrishnan *et al.* in [6] suggests modifying TCP to share TCP control block state to improve performance. In [7], Balakrishnan *et al.* present an architecture to manage congestion across an ensemble of unicast flows. The congestion manager described in the paper maintains network congestion information and schedules data transmission. Unlike our goal, the goal of these works is to enable different flows to cooperatively share the available path bandwidth.

Various researchers have proposed techniques to detect shared congestion. Harfoush *et al.* [12] uses a packet-pair (active) approach designed for paths originating at the same site. Rubenstein *et al.* proposed an approach that measures loss/delay characteristics using Poisson probes and then computes the autocorrelation and cross-correlation. Like Harfoush, they assume either the senders or the receivers are co-located. A recent paper by Cui *et al.* [9] presents an active probe monitoring approach to capture loss and delay characteristics that are then used to estimated the level of shared congestion. Although they indicate how the loss monitoring might be done passively, the primary focus is again on active probing to identify loss. They also make assumptions about the way packets are lost at routers and the effectiveness of their technique when RED [10] is enabled is thus unclear.

In the following, we present two passive monitoring approaches that do not inject probes to identify shared bottlenecks, nor do they make any assumptions about the loss models at network routers. Consequently, they scale better than active approaches and can be used in a wide range of network environments under a range of network conditions.

3. CAPACITY CONSERVATION (ALG 1)

Our first algorithm is based on the idea that flows traversing the same bottleneck link share the link's capacity. If one flow decreases its usage of the bottleneck, the capacity available to other flows on the bottleneck link will increase proportionally; similarly, if a flow increases its usage, others should observe a decrease.

In theory, one should be able to determine if flows share a bottleneck link simply by modulating one of the flows' bandwidth usage and observing the throughput of the other flows. If their usage changes proportionally, one might conclude they share the same bottleneck. However, nodes in an overlay network can only observe overlay flows, not all flows sharing the bottleneck. We call the flows that can be observed by the overlay network *foreground flows*. All other (unobservable) flows that share the bottleneck link are called *background flows*. If the capacity consumed by background flows is relatively constant, the conservation principle can be applied to the foreground flows alone. However, if the background capacity changes frequently, it may affect the overlay's ability to detect proportional changes in the foreground flows.

3.1 The Algorithm

Given two (or more) flows, the conservation algorithm intentionally reduces the rate of one of the flows by a known amount for a fixed time period (i.e., we turn the flow off) and then later returns the flow to its normal rate (i.e., we turn the flow back on). During these time periods the algorithm looks for proportional changes in the bandwidth of the other flow(s). If a change is observed that is consistent with the other flow's modulation, we assume the flows share a bottleneck; otherwise we assume they take independent paths.

Our algorithm records the TCP throughput of every flow at ten second intervals. Given the throughput record and information about when flows were modulated on/off, the algorithm can compute the observed change in throughput. If the modulated flow stops transmitting at time τ, we compute the change in throughput (slope) for each flow i at time τ as $s_i(\tau) = (r_i(\tau) - r_i(\tau+2))/2$, where $r_i(\tau)$ is the throughput of flow i at time τ. Assuming fair sharing of the bottleneck by n flows (i.e. each gets $1/n$ of the link capacity), the slope should be negative and proportional to n. Since we don't know n, we simply check whether the slope is negative or not for each on-to-off transition point. Similarly, if the modulated flow is restarted at time τ, we compute $s_i(\tau) = (r_i(\tau+2) - r_i(\tau))/2$, again looking for negative slopes. If there is a high enough percentage of slopes that are negative, we consider that the two flows are correlated.

Clearly this approach is sensitive to several factors including the number of flows sharing the bottleneck. As n increases, each foreground flows' share of the bottleneck capacity decreases making it more difficult to distinguish the effects of modulation from "noise" changes in throughput (i.e., normal changes resulting from the steady-state behavior of TCP congestion avoidance). A second problem with this method is that background flows may also transition between on and off states; these transitions may contribute to (or against) the changes observed in the foreground flows.

3.2 Experimental Results

We conducted experiments using the University of Kentucky Emulab facility [3, 1] to investigate the effectiveness of "conservation of bandwidth" as a basis for determining path correlation. Figure 1 shows the topology and parameters used in the experiments. Nodes S1 and S2 are the sources of the foreground flows, and D is the destination. N1 and N2 both send background traffic to D. Each flow is an on/off data source, with mean durations of on and off periods $1/\mu$ and $1/\lambda$, respectively. (That is, $1/\mu_b$ is the mean "on time" for background flows, while $1/\lambda_b$ is the average interarrival time, with $1/\mu_f$ and $1/\lambda_f$ denoting the corresponding means for the foreground flows.) All interarrival times were exponentially distributed; background service times were chosen according to a truncated Pareto distribution (shape=1.25, truncated to 1000 sec-

n_b	$\frac{1}{\mu_f}$	$\frac{1}{\lambda_f}$	$\frac{1}{\mu_b}$	$\frac{1}{\lambda_b}$	Exp. 1		Exp. 2	
					#	% corr	#	% corr
4	500	100	20	20	34	0.7059	26	0.8846
4	500	100	100	20	27	0.5185	28	0.8571
4	200	100	20	100	42	0.8810	44	0.7727
4	200	100	200	100	45	0.8444	40	0.6750
4	20	100	20	100	57	0.6140	56	0.7143
4	20	100	200	100	69	0.6522	68	0.6176
20	500	100	20	20	32	0.5312	24	0.4167
20	500	100	100	20	20	0.4500	30	0.5667
20	200	100	20	100	35	0.6286	44	0.6591
20	200	100	200	100	42	0.4286	41	0.7561
20	20	100	20	100	48	0.6875	56	0.7143
20	20	100	200	100	77	0.5325	92	0.6304

Table 1: Parameters and results for Exp 1 and Exp 2.

onds). To investigate sensitivity to n, the number of background flows on each path was either 2 or 10 for each background source (so n_b, the total number of background flows was either 4 or 20).

Table 1 lists the parameters used in each of the tests along with the correlation results for each test. We simulated a wide range of traffic scenarios including mixes of "long" and "short" foreground and background flows, as well as higher and lower "utilizations" of the foreground flows. For each set of parameters we ran the experiment for 7200 seconds of simulation time.

In the first experiment (Exp 1), the two foreground paths had different round-trip times, but shared a bottleneck link that strongly constrained their throughput. In the second experiment (Exp 2), the R3–R4 link had capacity 20 Mb/s, while the links adjacent to the flow sources were set to 10 Mb/s, and all other links to 100 Mb/s.

The results for Exp 1 and Exp 2 are shown in Table 1. The column titled "#" shows the number of on-off (or off-on) transitions recorded in the experiment; the column titled "% corr" indicates the percentage of those transitions in which the slope of the non-transitioning flow had the appropriate sign. The main conclusion to be drawn from these results is that with a small number of background flows, the fraction of transitions with slopes of opposite signs was almost always over 60%; however, with more background traffic the fraction was much closer to 50%. In other words, this method is better suited for situations with a modest number of background flows.

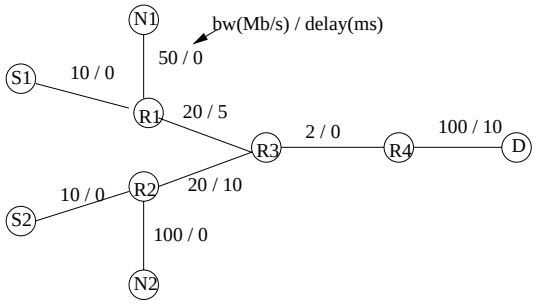

Figure 1: Topology with link parameters for Exp 1.

4. CORRELATED VARIATION (ALG 2)

Although the previous algorithm works well when the background traffic in constant, it is less effective when the background traffic changes frequently, or when the total number of flows sharing the bottleneck is large. Although there are situations in which one may assume the background traffic is stable, there are also many situations where the level of background traffic is constantly changing.

Our second algorithm avoids these problems by taking a kind of opposite approach, which is based on the observation that if two flows share a bottleneck link, their behavior should be similar. That is, any changes in background traffic should cause the foreground flows to react in similar ways—they should see the same congestion and adapt their windows in similar ways.

4.1 The Algorithm

The basic idea is that random background traffic will cause random changes to the foreground flows. Our *correlated variation* algorithm treats TCP throughput as a random variable and infers correlation between two flows by calculating the *correlation coefficient* of the flow throughputs observed at approximately the same times.

The correlation coefficient between random variables X and Y is defined in the normal way to be

$$\rho = Cov(X,Y)/\sqrt{Var(X)Var(Y)}$$

For this method, we record the data throughput *only while the application is sending data* to rule out the possibility that unavailability of application data limits TCP throughput. We then use the formula above to calculate the correlation coefficient between the two TCP flows.

4.2 Performance

To investigate the effectiveness of this method, we conducted experiments using both Emulab and PlanetLab. In all these experiments, the foreground flows transmitted continuously.

Emulab Experiments

To evaluate the performance of *correlated variation* in a controlled environment, we again used the University of Kentucky Emulab Facility [3, 1] which allows us to construct different topologies and test our methods under diverse conditions. Our goal was to see whether two TCP flows would behave in similar ways if they shared a common bottleneck link (but nothing else). Similarly, we wanted to verify that two independent TCP flows would *not* exhibit good correlation.

For each experiment we present both the topology and results in a single figure; foreground flows are represented by solid lines, and background flows are represented by dashed lines. The interarrival time of background flows is exponentially distributed and the distribution of their lifetime is a Pareto distribution (shape=1.25). We record TCP throughput of each foreground flow at one second intervals and use the average value over a window of 10 intervals as samples for computing the coefficient of variation. Each experiment runs for 1000 seconds. Each row of each table shows the mean interarrival and service times for the background flows.

The first experiment (Figure 2) involves a single congested link shared by the two foreground flows. From the table we see that the two flows show strong correlation. Moreover, the effectiveness of the algorithm does not decrease when background traffic increases or is bursty. To evaluate the effect RTT has on correlation, we intentionally used flows with very different RTTs (25ms vs 50ms). It is well-known that TCP flows with significantly different RTTs will to react differently to congestion (e.g., one will react more aggressively than another) which could lead to low correlation scores. However, despite the large difference in round trip times, the flows in Figure 2 exhibit remarkably good correlation.

In the second experiment, background traffic is sent along the link between R0 and R1, and between R1 and R2. As shown in

$1/\lambda$	$1/\mu$	correlated
10	100	0.92
10	200	0.76
20	100	0.93
20	200	0.93
20	300	0.97
20	400	0.95

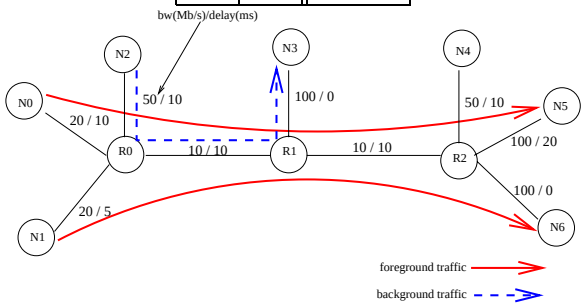

Figure 2: Single Congested Link

$1/\lambda_1$	$1/\mu_1$	$1/\lambda_2$	$1/\mu_2$	correlated
10	100	10	100	0.85
10	200	10	200	0.97
10	200	20	100	0.93

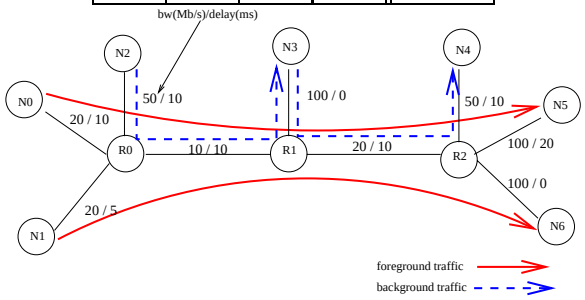

Figure 3: Two Congested Links

B_1	B_2	$1/\lambda_1$	$1/\mu_1$	$1/\lambda_2$	$1/\mu_2$	correlated
10	100	10	100	10	100	0.91
10	50	10	100	10	100	0.84
10	30	10	100	10	100	0.81
10	100	10	200	10	200	0.85
10	100	10	100	20	100	0.81

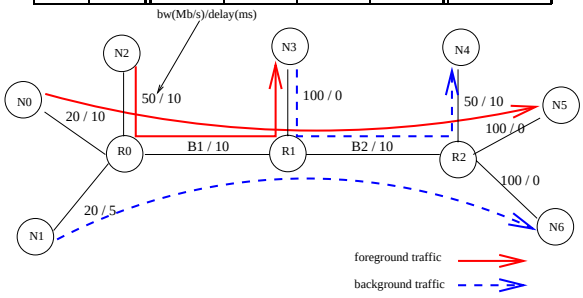

Figure 4: Shared Link Followed by Nonshared Link

B_1	B_2	$1/\lambda_1$	$1/\mu_1$	$1/\lambda_2$	$1/\mu_2$	correlated
10	100	10	100	10	100	- 0.11
10	50	10	100	10	100	- 0.04
10	30	10	100	10	100	- 0.05
100	10	10	100	10	100	0.85
50	10	10	100	10	100	0.81
30	10	10	100	10	100	0.81

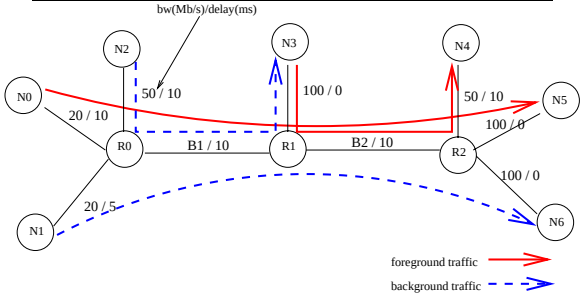

Figure 5: Nonshared Link followed by Shared Link

Figure 3, the two foreground flows share *two* congested links. Despite the "double bottleneck", our correlation-based algorithm is able to determine that the two paths are highly correlated.

In the third experiment, the foreground flows share one link (from R0 to R1), and both flows see cross traffic on that link. However, one of the flows also sees cross traffic on the nonshared link between R1 and R2. As the table in Figure 4 shows, our method still correctly classifies the two correlated paths. From the first three rows of the table in Figure 4, we can see that the correlation is still greater than 0.80, even when the capacity of link between R1 and R2 is reduced.

Figure 5 shows our fourth experiment, which is similar to the previous one in that the two foreground flows see the same cross traffic on one link, but one flow sees different cross traffic on the other link. Whether the flows share a bottleneck thus depends on the capacities B1 and B2. The first three rows of the table show performance when R0 to R1, which only one foreground flow traverses, is the bottleneck link for that flow. In this case the flow from N0 to N5 (correctly) does not correlate with the flow from N3 to N4, despite the fact that they share the R1–R2 link. Even when the link from R1 to R2 becomes more congested, the correlation estimate is still low, because R0 to R1 remains the primary bottleneck. However, when the R1 to R2 link becomes the primary bottleneck, the two foreground flows exhibit strong correlation (see rows 4-6 in the table). Moreover, as the R0-R1 capacity is reduced, the method still works well.

Figure 6 shows our fifth experiment. The two target paths reside on separate networks and thus do not share any link. However, the links in both of the topologies have the same parameter settings and thus should behave similarly. However, because the background traffic is not the same in the two topologies, the foreground flows do not exhibit correlated behavior. From the table in Figure 6, our method correctly detects that the two paths do not share any link.

Planetlab Experiments

To evaluate how well the correlated variation algorithm works in a real overlay network, we performed experiments using 220 nodes from the *Planetlab* overlay network [2]. The 220 nodes are sited at locations across the world and the virtual links connecting them experience real Internet background traffic. Again we tested situations where (a) the two flows share a bottleneck, and (b) the two flows do not share a bottleneck link.

To ensure the two flows in case (a) shared a bottleneck link, we selected the two source nodes from the same planetlab site, and we selected the two destination nodes from the same planetlab site. Consequently, packets from both flows followed the same path across the Internet to the destination. The planetlab sites used

$1/\mu_1$	$1/\lambda_1$	$1/\mu_2$	$1/\lambda_2$	correlated
10	200	5	10	0.06
10	100	10	100	0.06
20	100	10	200	0.06
50	100	10	300	0.10

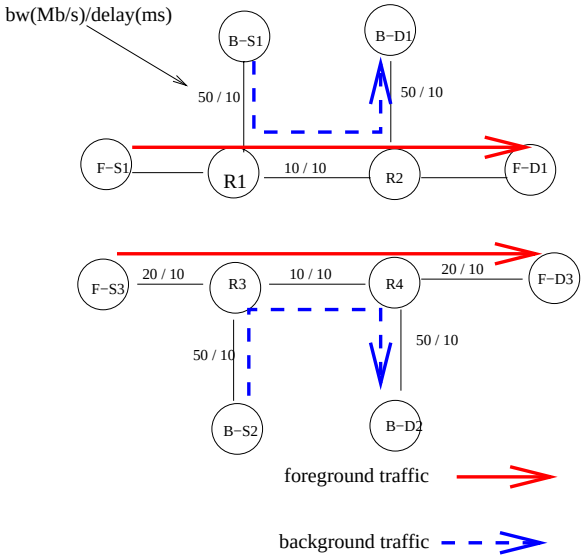

Figure 6: No shared bottleneck link

for source and destination locations were varied throughout the test to capture performance under a range of bottlenecks conditions.

To achieve independent paths for the test cases in (b), we randomly selected nodes from different planetlab sites making it far less likely that the flows shared a bottleneck link. Simple traceroute analysis of the paths selected seemed to indicate that the selected paths did not share a link, let alone a bottleneck link.

For each pair of selected paths, we sent a 30 second TCP flow along each path, monitored the throughput. The interval between path tests was exponential distribution with mean value of 150 seconds. A total of 32 pairs of independent paths and 40 pairs of shared paths were tested. Testing of all path pairs constituted a round, and 20 rounds of tests were conducted to get a good cross section of performance over time. Results shown are for rounds that occurred between 9am and 5pm EST (i.e., during normal work hours). At other times when there was little background traffic, the correlation tended to be low. (Note that in the absence of a bottleneck the lack of correlation leads to the correct conclusion.)

Figure 7 shows the Planetlab experiment results, in the form of the cumulative distribution of the correlation values for the two different types of paths. The results show that there is a stronger correlation among the shared paths than the independent paths. It also shows that setting the correlation cutoff for deciding whether two flows are dependent or independent can be done easily and robustly. Note the plateau in both lines between 0.2 and 0.4, implying that almost any cutoff in that range would yield the same results. The vertical line shows a cutoff at 0.28; using that cutoff, the algorithm would consider pairs with a correlation factor less than 0.28 to be independent, and pairs with a correlation factor greater than 0.28 to be dependent (i.e., correlated). At a 0.28 cutoff, the number of incorrectly classified path pairs is less than 20% for both the independent and dependent pairs.

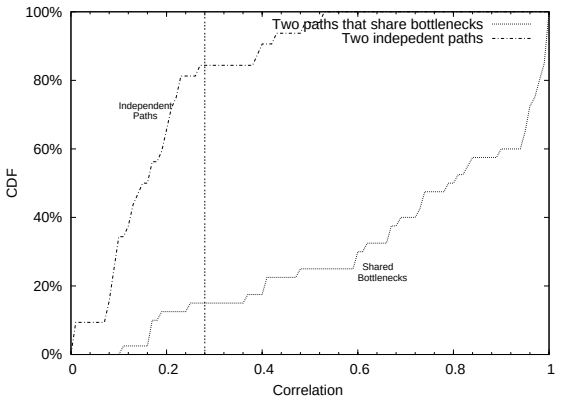

Figure 7: Planetlab Experiment

5. COMPUTING THE CORRELATION

The foregoing sections describe a method of collecting data at endpoints, to be used in determining correlation of virtual links. A key step in this process is synchronizing and matching up the measurements taken on each of the virtual links for computation of correlation coefficients. Because there will be times when no data is being transferred across a virtual link, these periods should be removed from the correlation computation. We assume that end systems know when data is being sent (at full TCP bandwidth) and not being sent, to within the granularity of the measurement periods. Thus, each endpoint has a list of "on" periods for the links on which it is a source, including the start and end times of the period, and the periodic throughput measurements during each "on" period. For both methods discussed in this paper, the correlation computation involves selecting the relevant overlapping portions of the "on" periods of the two links, and then performing a computation on those portions of the on intervals. Note that each node maintains a time window of "on" periods for each virtual link—say, M minutes' worth. If on average there are m "on" periods per recording interval M, then it takes $O(m)$ time to compute the correlation between two virtual links. With N overlay nodes there are $N^2 - N$ virtual links that must be compared to each other. To check correlation for all pairs of virtual links thus requires $O(mN^4)$ time. Moreover, each node needs to ship its data to all other nodes so they are able to perform the computation. Clearly, a brute force approach will not scale. Even with the use of readily available topology data (e.g., BGP information) to rule out some of the comparisons, the computational overhead can still be quite high.

To reduce network load and the time complexity of the correlation computation, we take a lazy-evaluation approach. Periodically, nodes exchange their available bandwidth numbers (an N^2 matrix of available bandwidth passively observed by the N overlay nodes [15]). Using this information, candidate paths through the overlay are selected [15]. Only after the paths have been selected is the path correlation computation performed, by having the source contact each node along the selected paths to request throughput measurements for the appropriate link. As a result, the correlation measurement lists are only transferred for links on which correlation is a concern—not on all links. This reduces network traffic overhead, and also restricts the correlation computation to only those links in use. Finally, as nodes discover path correlation information, they record it for future use in selecting paths. (Of course, any such information needs to be aged and eventually discarded, so that routing and topology changes can be caught.)

6. CONCLUSION

Recently, many applications have used path diversity to provide better performance. In this paper, we proposed two techniques to detect path correlation which helps such applications make informed decisions. One technique is to observe the throughput change (increase/decrease) of one flow when the other flow starts or finishes sending. The other technique is to apply statistic technique to infer path correlation by collecting TCP throughput samples at different times. The advantages of our methods are that no probing traffic is sent, and they can be easily applied to a wide range of topologies. Our techniques assume busy sources (i.e. source that transmit fairly constantly), but do not impose any other constraints. Our "Conservation of Capacity" method works best when background traffic is fairly constant, while our method based on correlated variation works best when background traffic is changing randomly. Thus, the two methods are complementary. In future work, in addition to further experiments, we will consider methods for dynamically adapting the method to existing conditions.

7. REFERENCES

[1] Netbed/Emulab, 2004. http://www.emulab.net.

[2] PlanetLab, 2004. http://www.planet-lab.org.

[3] The UK Emulab, 2004. http://www.uky.emulab.net.

[4] D. G. Andersen and H. Balakrishnan. Resilient Overlay Networks. In *Proceedings of the 18th ACM Symposium on Operating Principles (SOSP)*, Banff, Canada, October 2001.

[5] J. Apostolopoulos, T. Wong, W. tian Tan, and S. Wee. On Multiple Description Streaming with Content Delivery Networks. In *IEEE INFOCOM*, July 2002.

[6] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, M. Stemm, and R. H. Katz. TCP Behavior of a Busy Internet Server: Analysis and Improvements. In *Proceedings of INFOCOM 1998*, San Francisco, CA, March 1998.

[7] H. Balakrishnan, H. Rahul, and S. Seshan. An Integrated Congestion Management Architecture for Internet Hosts. In *Proceedings of the SIGCOMM 1999*, Cambridge, MA, September 1999.

[8] J. Byers, M. Luby, and M. Mitzenmacher. Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed Up Downloads. In *Proceedings of the IEEE INFOCOM Conference*, pages 275–283, March 1999.

[9] W. Cui, S. Machiraju, R. H. Katz, and I. Stoica. Estimating Shared Congestion Among Internet Paths. http://sahara.cs.berkeley.edu/jun2003-retreat/wdc_talk.pdf.

[10] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. In *IEEE/ACM Transactions on Networking*, August 1993.

[11] T. Guven, C. Kommareddy, R. J. La, M. Shayman, and B. Bhattacharjee. Measurement Based Optimal Multi-path Routing. In *Proceedings of the IEEE INFOCOM Conference*, March 2004.

[12] K. Harfosuh, J. Byers, and A. Bestavros. Robust Identification of Shared Losses Using End-to-end Unicast Probes. In *Proceedings of the ICNP 2000 Conference*, November 2000.

[13] C. Kommareddy, T. Guven, B. Bhattacharjee, R. La, and M. Shayman. Intradomain Overlay: Architecture and Applications. Technical Report UMIACS-TR 2003-70, July 2003. http://www.umiacs.umd.edu/res/proj/menter/umiacs_tr_2003_70.pdf.

[14] D. Rubenstein, J. Kurose, and D. Towsley. Detecting Shared Congestion of FLows Via End-to-End Measurement. In *IEEE/ACM Transactions on Networking*, June 2002.

[15] S. Shi, L. Wang, K. L. Calvert, and J. N. Griffioen. A Multi-path Routing Service for Immersive Environments. In *Proceedings of the CCGRID 2004 Conference: Grid and Advanced Networks Track*, April 2004.