End-to-end congestion control techniques for Router

B.Mahesh[1]
Department of CSE,
Intell Engineering College,
Anantapur, India.
mahesh.bhasutkar@gmail.com

M.Venkateswarlu[2]
Department of CSE,
Dr.K.V.S.R College of Engg. For women,
Kurnool, India.
venkateswarlu.maninti@gmail.com

M.Raghavendra[3]
Junior Engineer,
McLaren R&D,
Hyderabad, India.
raghu.knl@gmail.com

Abstract— **END-TO-END packet delay is one of the canonical metrics in Internet Protocol (IP) networks and is important both from the network operator and application performance points of view. The motivation for the present work is a detailed know-ledge and understanding of such "through-router" delays. A thorough examination of delay leads inevitably to deeper quest-ions about congestion and router queuing dynamics in general. Although there have been many studies examining delay statistics and congestion measured at the edges of the network, very few have been able to report with any degree of authority on what actually occurs at switching elements. In existing system the single-hop packet delay measured and analyzed through operational routers in a backbone IP network. However since the router had only one input and one output link, which were of the same speed, the internal queuing was extremely limited. In this paper work with a data set recording all IP packets traversing a Tier-1 access router. All input and output links were monitored, allowing a complete picture of congestion and in particular router delays to be obtained. This paper provides a comprehensive examination of these issues from the understanding of origins and measurement.**

Keywords- congestion, delay, busy period, IP router, packet delay analysis, input-queueing, scheduling

I. INTRODUCTION

End-to-End packet delay is an important metric to measure in networks, both from the network operator and application performance points of view. An important component of this delay is the time for packets to traverse the different forwarding elements along the path. This is particularly important for network providers, who may have Service Level Agreements (SLAs) specifying allowable values of delay statistics across the domains they control. A fundamental building block of the path delay experienced by packets in Internet Protocol (IP) networks is the delay incurred when passing through a single IP router. Examining such 'through-router' delays is the main topic of this paper.

The first aim of this paper is a simple one, to exploit this unique data set by reporting in detail on the magnitudes, and also the temporal structure, of delays on high capacity links with nontrivial congestion. Our second aim is to use the completeness of the data as a tool to investigate how packet delays occur inside the router. Packet delays and congestion are fundamentally linked, as the former occur precisely because periods of temporary resource starvation, or *microcongestion episodes*, are dealt with via buffering. Our third contribution is an investigation of the origins of such episodes, driven by the question, "What is the dominant mechanism responsible for delays?". We use a powerful methodology of virtual or semi experiments, that exploits both the availability of the detailed packet data, and the fidelity of the router model.

The paper is organized as follows. The router measurements are presented in Section II, and analyzed in Section III, where the methodology and sources of error are described in detail. In Section IV we analyze the origins of microcongestion episodes.

II. FULL ROUTER MONITORING

In this section, we describe the hardware involved in the passive measurements, present the router monitoring set-up, and detail how packets from different traces are matched.

A. *Router Architecture:*

Our router is of the store & forward type, and implements Virtual Output Queues (VOQ).. The router is composed of a switching fabric controlled by a centralized scheduler, and interfaces or linecards. Each linecard controls two links: one input and one output. When packet arrives at the input link of a linecard, its destination address is looked up in the forwarding table. This does not occur however until the packet completely leaves the input link and fully arrives in the linecard's memory. The packet is stored in the appropriate queue of the input interface where it is decomposed into fixed length cells. When the packet reaches the head of line it is transmitted through the switching fabric cell by cell to its output interface, and reassembled before being handed to the output link scheduler, i.e. the 'forward' part of store & forward. The packet might then experience queuing before being serialised without interruption onto the output link. In queuing terminology it is 'served' at a rate equal to the bandwidth of the output link capacity.

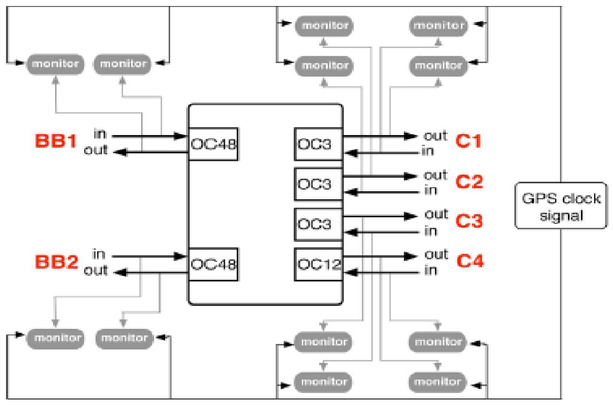


Figure 1. Experimental setup: gateway router with 12 synchronized DAG cards

B. *Packet Matching:*

We match identical packets coming in and out of the router by using a hash table. The hash function is based on the CRC algorithm and uses the IP source and destination addresses, the IP header identification number, and in most cases the full 24 byte IP header data part. In fact when a packet size is less than 44 bytes, the DAG card uses a padding technique to extend the record length to 64 bytes. Since different models of DAG cards use different padding content, the padded bytes are not included in the hash function. Our matching algorithm uses a sliding window over all the synchronized traces in parallel to match packets hashing to the same key. When two packets from two different links are matched, a record of the input and output timestamps as well as the 44 byte PoS payload is produced. Sometimes two packets from the same link hash to the same key because they are identical: these packets are duplicate packets generated by the physical layer. They can create ambiguities in the matching process and are therefore discarded, however their frequency is monitored.

Assume that the matching algorithm has determined that the m^{th} packet of output link Λ_j corresponds to the n^{th} packet of input link λ_i . This can be formalized by a *matching function M,* obeying

$$M(\Lambda_{j\,,}\, m)=(\lambda_i,n). \qquad (1)$$

In the remainder of the paper we focus on C2-out, an OC-3 link, because it is the most highly utilized. Its traffic is overwhelming due to the two OC-48 backbone links BB1-in and BB2-in carrying 47.00% and 52.89% of its traffic respectively (almost equal due to the Equal Cost Multi Path policy deployed in the network when packets may follow more than one path to the same destination). This is clearly seen in the packet time series of Fig. 2 across the full 13 hours. The bytes times series has an almost identical shape.

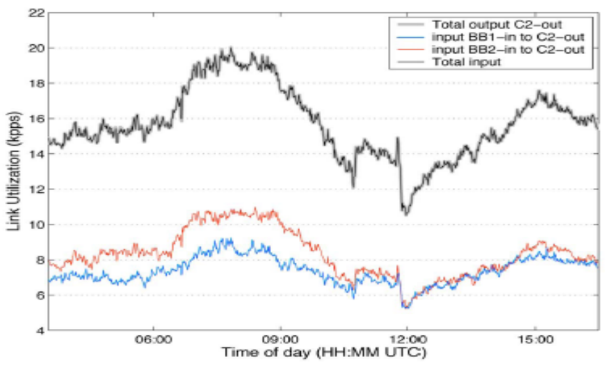


Figure 2. Utilization for link C2-out in kilo packet per second(kpps)

III. PRELIMINARY DELAY ANALYSIS

In this section we analyze the data obtained from the packet matching procedure.

A. *System Definition*

The DAG timestamps an IP packet on the incoming interface at time , and on the outgoing interface at time $t(\lambda_i,n)$, and on the outgoing interface at time $t(\Lambda_{j\,,}\, m)$. As the DAG cards are physically close to the router, one might think to define the through-router delay as $t(\Lambda_{j\,,}\, m)$ - $t(\lambda_i,n)$. However, this ignores the bit position at which DAG timestamps are made. We now establish the precise relationships between the DAG timestamps and the time instants $T(\lambda_i,n)$ of arrival and $T(\Lambda_{j\,,}\, m)$ of departure of a given packet to the system as just defined. Let θ_i and θ_j be the corresponding link bandwidths in bits per second. We denote by H the function giving the depth of bytes into the IP packet where the DAG timestamps it. H is a function of the link speed. For a given link $\lambda_{i,}$ H is defined as

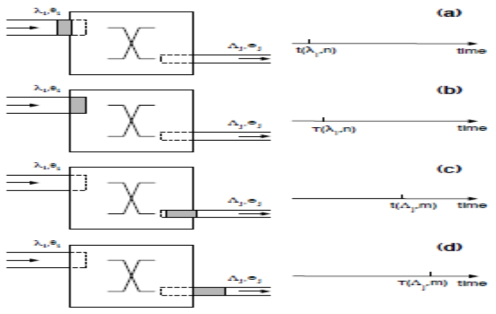


Figure 3. Four snapshots of a packet crossing the router.

$$\begin{aligned} H(\lambda_i) &= 4 \text{ if } \lambda_i \text{ is an OC-48 link,} \\ &= b \text{ if } \lambda_i \text{ is an OC-3 or OC-12 link,} \end{aligned}$$

Where we take b to be a uniformly distributed integer between 0 and $\min(l_n,53)$ to account for the ATM based discretisation. We can now derive the desired system arrival and departure event times as

$$T(\lambda_i,n) = t(\lambda_i,n)+8(l_n - H(\lambda_i))/ \theta_i$$

$T(\Lambda_j, m) = t(\Lambda_j, m) + 8(l_m - H(\lambda_j)) / \theta_j$ (2)

The through-system delay experienced by packet m on link Λ_j is defined as

$d_{\lambda i, \Lambda j}(m) = T(\Lambda_j, m) - T(\lambda_i, n)$ (3)

B. Delay Statistics:

Figure 4 shows the minimum, mean and maximum delay experienced by packets going from input link BB1-in to output link C2-out over consecutive 1 minute intervals. In any router architecture it is likely that many components of delay will be proportional to packet size. To investigate this here we compute the 'excess' minimum delay experienced by packet of different sizes. Formally, for every packet size L we compute

$\Delta_{\lambda i, \Lambda j}(L) = \min\{ d_{\lambda i, \Lambda j}(m) - 8l_m / \theta_j \mid l_m = L_m$ (4)

Figure 5 shows the values of $\Delta_{\lambda i, \Lambda j}(L)$ for packets going from BB1-in to C2-out. The IP packet sizes observed varied between 28 and 1500 bytes.

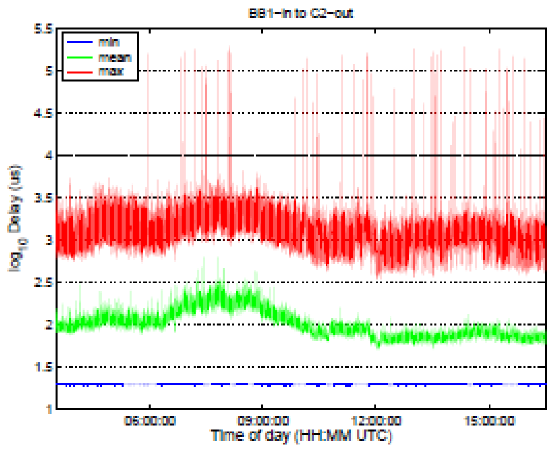


Figure 4. Packet delays from BB1-in to C2-out. Those>10 ms are due to option packets.

IV. MICROCONGESTION TECHNIQUES

We are now in a position to exploit the completeness of the data set to look inside the system. This enables us to find a physically meaningful model which can be used both to understand and predict the end-to-end system delay very accurately. Congestion appears in a network when compute nodes impose a load close to that the network is able to cope with. As new packets fill up the router buffers, the router will not only stop accepting additional traffic, but will also delay any in-transit traffic. Although congestion may emerge in localized areas, if injection pressure is not reduced it may quickly spread through the whole network. Some networks are able to operate at maximum capacity (in saturation) while exhibiting only minor levels of congestion. However large networks with multiple injection ports show signs of throughput degradation at loads beyond their saturation points.

A. Experiments using burst-synchronized traffic:

With this traffic, each node tries to inject a burst of b packets as fast as the network is able to accept them; then the node stops. Nodes will start injecting another burst only when all the packets of the previous one have been consumed. The figure of merit in these experiments is the time it takes to consume a burst, with and without using congestion control mechanisms. As we stated before, results have been normalized: value "1" represents the time of the Base case (without congestion control).

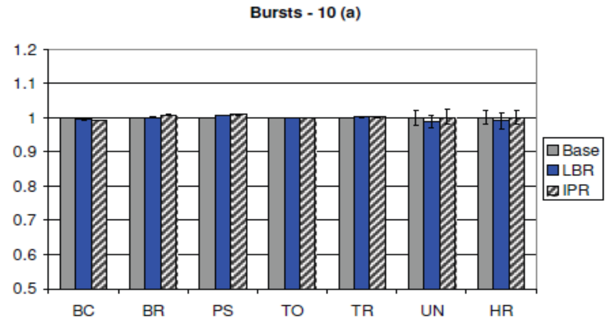
We have considered the following values of b:

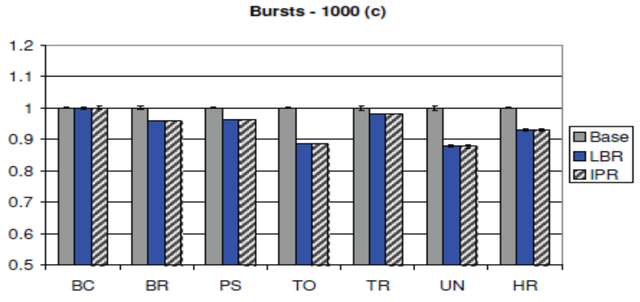
10 – to emulate very tightly-coupled applications.

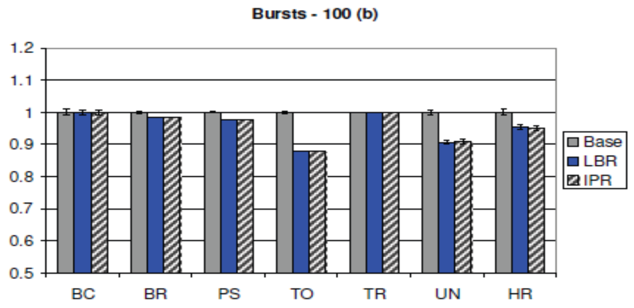
100 – to emulate applications that synchronize often.

1000 – to emulate loosely-coupled applications.

10000 – to emulate applications that seldom synchronize, thus being able to saturate the network for long periods of time. In the case of very tightly-coupled applications, b=10, the utilization of IPR or LBR is ineffective or even harmful. This is because network congestion is sporadic and localized, but it never last long enough to degrade performance. Restrictions in the injection of new packets may result in unnecessary delays, increasing the overall application execution time. Fortunately, when this happens the increment is only around 1%. In general, performance figures using congestion control techniques improve with growing values of b. This is particularly true for TR, but applicable to all patterns. Results for b=1000 are much more favourable for IPR and LBR than those with b=100 Regarding the choice of congestion control technique, IPR versus LBR, their performance figures for these experiments are very similar, being LBR slightly superior on average







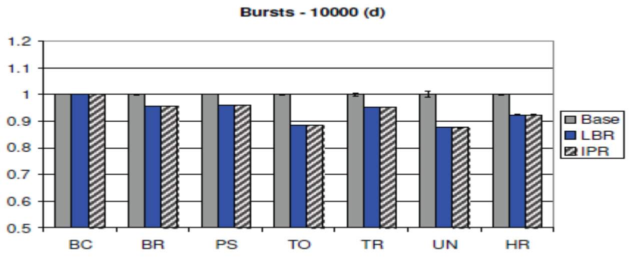


Figure 5. Performance for burst-synchronized traffic, for burst sizes 10, 100, 1000 and 10000. Normalized times to consume a burst, averaged from 10 runs.

B. Experiments using application traces:

We claimed in the previous section that burstsynchronized traffic sources emulate actual traffic in a better way than non-synchronized, independent traffic sources. In order to support this claim, we need to consider application workloads such a set of the wellknown NAS Parallel Benchmarks (NPB, see [12]). We have carried out a set of experiments using traces from six of the eight benchmarks of the NPB 2.4 suite. EP has not been included because it does not make intensive use of the network. FT could not be run because it requires a Fortran 90 compiler, which was not available in our setup. To obtain these traces, we used a modified version of MPICH [11]. The standard MPICH would register in those trace files all MPI operations—which means that a global operation (such as MPI_Broadcast) is logged as such. However, our modified MPICH also registers all the point-to-point operations involved in global operations. After a filtering process, we obtain trace files than only include point-to-point interchanges of messages, which are adequate to feed a FSIN simulation. Notice that the time-stamps in each trace file depend not only on the characteristics of the interconnection network, but also on those of the processors involved in the execution of the benchmarks. We want to force the network to be the bottle-neck, so we provide workload to the simulator as fast as we can, regardless of the timestamps found in the traces. However, in order to preserve the causal relationship among messages, we maintain the order shown between arrivals and new injections: when the trace file indicates that a message was received at a given node, injection of later packets from that node is suspended until that message arrives. We compiled and run the "A" class of the benchmarks, for 64 processors (A.64). This setup generates traces of long messages that cause saturation peaks. Once the traces are available, we measure with FSIN the number of network cycles that, in the simulated network, are necessary to complete all the interchanges of messages stored in the trace files. Results (normalized) are presented in Fig. 6. For five of the six benchmarks (BT, IS, LU, MG, SP), results using traces confirm those of the previous section for large burst sizes: congestion control mechanisms are beneficial (improvements range from 6% for MG to 21% for LU). In the other case (CG) the utilization of a congestion control mechanism is slightly counterproductive, with a performance drop around 2%. Note that these results match with those obtained under burst-synchronized traffic for small b (see again Fig. 2). Differences between IPR and LBR are negligible, except in LU where the LBR's improvement over IPR is near the 4%.

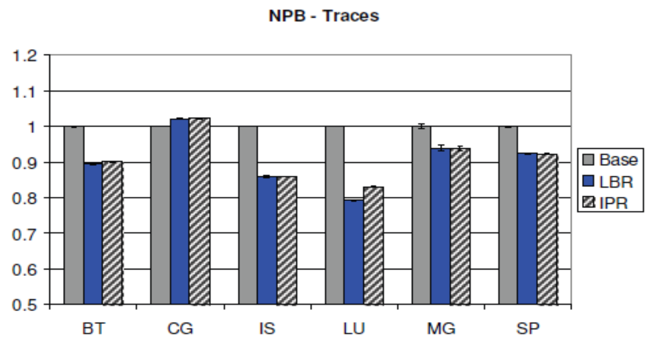


Figure 6. Normlized results with traces of the NPB (A.64). Averages of 10 runs, and 99% confidence intervals.

C. Experiments with an execution-driven simulation

One of the limitations of our trace-based simulation environment is that the mechanism implemented to enforce causality relations may be excessively restrictive. It may happen that the presence of a Receive message before a Send message in an application trace does not mean that there is a real dependency between those messages, and that the Send could be processed ahead of the Receive without disturbing application semantics. An execution-driven simulation would not impose such an order in the injection and consumption of messages, but will only enforce the traffic dependences inherent to the application. Thus, they will allow us to know whether the restrictions introduced in the trace based simulation are too strong or not. We have run the same collection of benchmarks in a Simics based workbench in which the network is simulated by FSIN, so that applications do actually run on simulated processors and provide/receive load to/from the network. As three of the benchmarks (BT,LU, SP) take a long time to run, because they repeat 200, 250 and 400 iterations respectively (that are identical in terms of communications), we have reduced them to run just 20, 25 and 20 of these iterations. This trims down experimentation time without affecting the quality of the results. Fig. 7 summarizes the obtained results. We measure the number of FSIN cycles used by the application since the moment the parallel section starts (MPI_Init) until it ends (MPI_Finalize). Again we have normalized the results to the Base case. Results show two applications, BT and SP that do not benefit so much from congestion control, even when the trace-based simulation predicted the opposite. The reason is easy to explain: these applications are CPU intensive; they do not force a high utilization of the network. So, congestion does not appear, or concentrate in very short periods of time. In these cases, congestion control does not show its potential. Regarding the remaining ones, we see that congestion control generates clear benefits: a 26-38% reduction in execution time. This is much better than predicted. In fact, we predicted no gain for CG and still can observe it. To explain this behaviour, we need to take a closer look at how simulation is being carried out. As we stated before, each simulated node is a full PC with a network adapter that looks like an Ethernet card. The MPICH implementation uses this card via the p4 mechanism: MPI on top of TCP/IP on top of Ethernet. TCP includes an end-to-end congestion-control mechanism that is very sensitive to network delays, and to variations on this delay (jitter). When the network is saturated (actually, when delays become longer than expected), TCP reduces traffic injection drastically, Applying the standard control algorithm, situation that does not allow for the full utilization of this resource. Congestion control inside the network prioritizes intransit traffic, and this reduces network delay and jitter. This proves to be an effective tactic, because keeps TCP working normally. What we observe in the experiments is not only the direct effect of congestion control accelerating applications, but also the indirect effect of IPR or LBR preventing TCP misbehaviour.

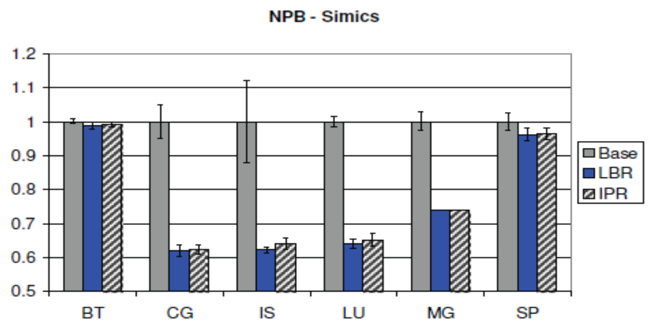


Figure 7. Normalized results of the execution of the NPB (A.64) in a Simics+FSIN environment. Averages of 10 runs, and 99% confidence intervals.

V. CONCLSION AND FUTURE WORK

In this paper we have evaluated two congestion control techniques (In-transit Priority Restriction and Local Buffer Restriction) under three traffic models: burst synchrnozied traffic, traces and real workloads generated by applications as they run.

In the experiments we have used a Tier-1 access router. The evaluation of large interconnection networks requires the utilization of synthetic traffic patterns, because other alternatives are too expensive. However, in order to obtain accurate predictions, those synthetic patterns must reflect the behavior of actual traffic; in particular, they must reflect how communicating process synchronize.

REFERENCES

[1] K. Papagiannaki, S. Moon, C. Fraleigh, “Analysis of measured single-hop delay from an operational backbone network,” in *Proc. IEEE INFOCOM*, New York, Jun. 2002, pp. 535–544.

[2] N. Hohn, D. Veitch, K. Papagiannaki, and C. Diot, “Bridging router performance and queueing theory,” in *Proc. ACM Sigmetrics*, New York, Jun. 12–16, 2004, pp. 355–366.

[3] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581, April 1999.

[4] W.J. Dally, B. Towles. “Principles and Practices of Interconnection Networks”. Morgan-Kaufmann, 2004.

[5] T. Callahan and S.C. Goldstein, “NIFDY: A Low Overhead, High Throughput Network Interface”, in Proc. 22nd Annual Int. Symp. on Computer Architecture (ISCA), June 2005, Santa Margherita Ligure, Italy

[6] Virtutech Inc. “Simics page”. Available (May 2006) at http://www.virtutech.se/products/

[7] C. Izu, J. Miguel-Alonso, J.A. Gregorio. “Evaluation of Interconnection Network Performance under Heavy Non-uniform Loads”. Lecture Notes in Computer Science, Volume 3719 / 2005 (Proc. ICA3PP 2005), pp. 396 - 405.

[8] R. Addie, P. Mannersalo, and I. Norros, “Performance formulae for queues with Gaussian input,” in *Proc. 16th Int. Teletraffic Congr.*, dinburgh, Scotland, U.K., Jun. 1999, pp. 1169–1178.

[9] S. Keshav and S. Rosen, “Issues and trends in router design,” *IEEE Commun. Mag.*, vol. 36, no. 5, pp. 144–151, May 1998.

[10] K. Papagiannaki, R. Cruz, and C. Diot, “Network performance monitoring at small timescales,” in *Proc. ACMInternet Measurement Conf.*,Miami, FL, 2003, pp. 295–300.