Evaluation of Congestion Control Mechanisms using Realistic Loads

F.J. Ridruejo[1], J. Navaridas[1], J. Miguel-Alonso[1], Cruz Izu[2]

Abstract—**Large networks with multiple injection sources will exhibit significant throughput degradation unless congestion control mechanisms are included in the router design. A proper evaluation of these mechanisms for a given interconnection network needs to consider the level of synchronization amongst the computing nodes.**

This paper explores this issue by evaluating two local congestion control mechanisms in three different ways. We start using burst-synchronized synthetic traffic patterns, which give us a first estimation of the potential gains of the congestion control proposals. Additional experiments using traffic from traces taken from the NPB confirm the preliminary findings: loosely coupled applications reduce their execution times when congestion control is applied, but tightly coupled application may marginally increase their execution times. Results obtained in an execution-driven simulation environment are even more optimistic: keeping congestion under control substantially accelerates the execution of MPI applications.

Keywords—**Interconnection networks, congestion control, trace-driven simulation, execution-driven simulation.**

I. Introduction

THE interconnection network is a key element of any parallel computing platform, more so when executing communication-intensive applications. Network performance has been evaluated by the following methods: network simulation using synthetic loads [2][3][4][7][14][16][19], analytical models [17], and full system simulation [13][14][20].

Traditionally, network simulators measure network performance under a well-known range of synthetic traffic patterns such as random (or uniform), hot-spot and permutations. These patterns model worst-case scenarios of little locality and unbalanced usage of network resources [5]. Analytical models have been proposed for simple networks but they rely on unrealistic assumptions, such as the traffic has to be uniform and the nodes have infinite injection and delivery queues, so they have limited use. Both synthetic and analytical models assume that nodes generate traffic independently of each other, so that they ignore the different levels of coupling and synchronization that exist in most parallel applications. This was of little concern up to now, as they seemed to be reasonable indicators of network performance for a range of parallel benchmarks [14][17]. A full system simulation, in which network traffic is provided by an execution-driven simulator such as RSIM [13] or Simics [20], provides the more meaningful results but limits the evaluation to small networks of up to a hundred nodes.

Ideally, an interconnection network should perform well at any load, and be able to sustain its peak throughput for loads well over its capacity. However, if congestion spreads through the whole network, its performance can drop significantly. Congestion is a well-known problem in standard computer networks [9], but has not been considered a critical issue in small and medium direct networks, which do not exhibit throughput degradation at loads beyond saturation. Many network proposals had single injection queues and, when the network is small, the head-of-line blocking at injection is enough to throttle the network and control congestion [8]. This is not the case for large networks with multiple injection sources such as IBM's BG/L [3], which are prone to suffer from congestion at heavy loads. Congestion control techniques based on restricting the injection of new packets can be applied, in order to accelerate the delivery of in-transit packets. This delivery should free network resources, then allowing new packets to enter.

New congestion control techniques have been proposed and evaluated for wormhole [2][19] and virtual cut-through networks [10][18]. For the evaluation, all but [10] used standard, synthetic traffic patterns, which consider each node as an independent traffic source. Note that most parallel applications will apply some level of self-throttling as nodes synchronize and may stop sending new messages as they wait for messages delayed by congestion. Thus, it is critical to test any congestion control technique under loads that reflect the synchronization and coupling among application processes.

The main purpose of this study is to explore whether we can evaluate congestion control techniques for interconnection networks using carefully chosen synthetic traffic. Although the use of real application loads from execution-driven simulators would provide the most meaningful conclusions, it limits our study to small networks of up to a hundred nodes. Synthetic traffic that models the self-throttled nature of parallel applications would provide less precise results but allows for evaluations of large systems with thousands of nodes. As congestion becomes an issue when the network has large rings [8], it is critical to evaluate congestion control techniques on systems as large as possible.

In [7] we proposed burst-synchronized synthetic traffic to evaluate a range of congestion control techniques. In this paper we will compare results obtained that way with additional ones obtained when applying real work-loads from the NAS Parallel Benchmarks (NPB). If the synthetic loads reflect the synchronized nature of parallel applications, both results should lead to the same conclusions. Here we will focus on two promising congestion control techniques: IPR (In-transit Priority

1. Departamento de Arquitectura y Tecnología de Computadores, U. del País Vasco UPV/EHU. E-mail: franciscojavier.ridruejo@ehu.es
2. School of Computer Science. U. of Adelaide. Australia.
Trabajo hecho con financiación del Ministerio de Educación y Ciencia, TIN2004-07440-C02-02. J. Navaridas financiado por una beca predoctoral de la UPV/EHU.

Restriction) and LBR (Local Buffer Restriction). Both are easy to implement in virtual cut-through networks, because they use local information to regulate the admission of new traffic.

In order to achieve these goals, we use a simulation-based workbench that consists of a network simulator with three alternative mechanisms to provide its workload: synthetic load, traces taken from previous runs of an application, or via integration with a full-system simulator in which applications actually execute on (simulated) nodes that generate and consume packets.

The remaining of this paper is organized as follows. Section II discusses related work. Congestion control techniques are described in Section III. The experimental, simulation-based workbench is described in Section IV. Section V compares and contrasts the results of the different experiments. Finally, Section VI summarizes the findings of this work.

II. Motivation and related work

In this section we review a range of synthetic traffic models used to evaluate network performance at heavy loads. Most IN (Interconnection Network) studies relied solely on synthetic traffic patterns, which include uniform traffic, hot spot traffic and permutations [5]. The figures of merit are latency at low loads and peak network throughput. This model appeared to provide meaningful result to compare alternative designs under a range of load conditions. Most studies normalized the applied load to the network bisection limit, so that the networks were not tested for loads beyond their theoretical capacity. As most network proposals sustained their maximum throughput after saturation, the evaluation of the network at heavy loads was not considered of interest.

However, network congestion appears around saturation loads and congestion control mechanisms must be tested at loads beyond saturation [2][10][18] in order to see if they prevent burst traffic from degrading network performance. Our first evaluation of congestion control techniques led to conflicting results as throughput was reduced while channel utilization increased [7]. Further examination revealed the cause of this unusual behaviour: network unfairness caused significant variation in node injection rate. Although this throughput unfairness was unexpected, it is obvious from that work that any network whose injection compete in a fair basis with in-transit traffic will reach a situation of variable injection rates at saturation: nodes in less busy areas will get a larger fraction of node bandwidth that those in heavily congested areas. This means that average throughput as reported in [2][18] is not representative of application loads, because “fast” injecting nodes will eventually wait for the slow ones to catch up. Consequently, we need to evaluate network performance at heavy loads using traffic patterns that reflect the level of synchronization amongst computing nodes [4]. This is critical when using congestion control techniques, as the level of self-throttling present in the parallel application may be sufficient to reduce congestion to manageable levels.

The need to characterize network workload and produce better synthetic models was identified long time ago [4]. Instead of developing new synthetic loads, some IN studies combine the standard evaluation with real workload evaluation [10]. Other studies use synthetic loads that mimic the bursty nature of network traffic [16][18], which extends standard packet generation, which followed a Poisson or Bernoulli distribution, with a sequence of ON/OFF states, so that packets are generated only during the ON state.

Most routing proposals were evaluated under static conditions: the same pattern and load applies over the time each experiment is run, and only the steady-state performance was studied. However, the network performance is characterized by both a steady-state and a transient state, as real application loads will continuously change in volume and traffic distribution. Recent studies evaluated this behaviour by plotting throughput over time for non-static traffic patterns: in [2][18] the traffic pattern is uniform but the load alternates between low and high phases, in [19] each high phase uses a different traffic pattern. The figure of merit is total execution time, which is a better indication of the network ability to cope with communication intensive phases. Note that a computing node located in a less busy area is able to advance to the next high phase load ahead of the rest, an unlikely scenario in a parallel application. This can be avoided by making the “low” phases long enough for all nodes to accept their backlog loads.

Although these newer synthetic patterns reflect the temporal variations that occur in application loads, they fail to model the synchronization and coupling amongst application nodes. The burst-synchronized traffic deals with this issue by modelling the barrier synchronization primitives used in many parallel applications. This pattern has been sparingly used in studies focused on injection issues: the NIFDY interface [4] and on Channel Queue routing [17] which decides on injection whether to use minimal or non-minimal path based on network congestion.

Lastly, all these synthetic traffic patterns are claimed to represent real workload but there is no study confirming or denying this fact. Our work tries to fill this gap by comparing the insights obtained from synthetic loads with those from real workloads.

III. Congestion control techniques

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 [8].

As congestion appears when the network is overloaded, congestion control techniques deal with it by limiting packet injection as soon as the network exhibits signs of being congested. They differ in the way congestion is diagnosed. Global methods estimate network congestion by examining the status of the whole network (for example, the number of packets held in the routers, as in [18]); thus a mechanism is needed to gather and distribute that information. Local methods are simpler because each node restricts its own injection based on its own congestion level. In [10] a thorough evaluation of multiple congestion control methods was performed. In this work, we focus only on two local methods that showed good performance with minimal implementation costs, IPR and LBR.

In-transit Priority Restriction (IPR) can be defined as follows: For a given fraction P of cycles, priority is given to in-transit traffic, meaning that, in those cycles, injection of a new packet is only allowed if it does not compete with packets already in the network. P may vary from 0 (no restriction) to 1 (absolute priority to in-transit traffic). This is the method applied in IBM's BG/L torus network, in which P may take any value, although published evaluations [3] have been carried out with P=1.

Regarding Local Buffer Restriction (LBR), most routers split each physical link into several virtual channels (see next section) in such a way that the combination of an escape sub-network with one or more adaptive sub-networks provides deadlock-free adaptive routing [6]. The LBR mechanism has been designed specifically for adaptive routers that rely on Bubble Flow Control [14] to avoid deadlock in the escape sub-network. A previous study showed the bubble restriction also provides congestion control for the escape sub-network [8]. LBR extends this mechanism to all new packets that enter the network. That is, a packet can only be injected into an adaptive virtual channel if such action leaves room for at least B packets in the transit buffer associated to that virtual channel. The parameter B indicates the number of buffers reserved for in-transit traffic. In other words, congestion is estimated by the current buffer occupancy.

The parameters (P for IPR and B for LBR) allow us to vary the degree of restriction. In this work, we have set these parameters to values that, in previous experiments, proved to perform well in most scenarios: P=1 (the maximum) and B=3 (inject in an adaptive channel only when its queue is empty or almost empty). Note that the adaptive bubble router in [14] corresponds to the "Base" case of B=0 and P=0.

IV. Experimental setup

Experiments have been performed using the in-house developed Interconnection Network Simulation and Evaluation Environment (INSEE for short) described in [15]. It consists of two main modules: an interconnection network simulator (FSIN), and a traffic-generation module (TrGen), which either provides traffic (synthetically or from traces) or interfaces with a Simics based full system simulation environment [20].

The execution-driven setup limits system size to a maximum of 64 nodes that we use in all the experiments. We have decided to use a 1D ring network topology because the effects of congestion are particularly visible in networks with long rings. In other words, an 8x8 torus network (8-node rings) exhibits minimal congestion, while a 64-node ring becomes heavily congested at high loads. So, although FSIN is able to model 1, 2 and 3D networks (with thousands of nodes), we focus on the router model depicted in Fig. 1.

Each network channel in the router is split into three virtual channels (VCs): an Escape channel (governed by the bubble routing rules [14]), and two adaptive channels. Note that a ring network has just one minimal path from source to destination, so packets cannot adapt. Thus, the only difference between the Escape VC and the other two is that accesses to the "adaptive" VCs are not restricted by the bubble rules. Each node is able to simultaneously consume several packets arriving to the reception port. There are two injection ports, and the interface should perform a pre-routing decision: packets moving towards the X+ axis are stored in the I+ injection port, and those towards X– go to the I– injection port. Transit and injection queues are able to store 4 packets of 16 phits each. Phit length is 4 bytes.

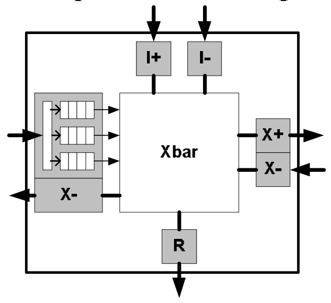

Fig. 1. Model of router simulated by FSIN, with a detailed view of the X+ input port showing the 3 virtual channels that share its link.

For the Simics-based setup, we simulate a cluster of 64 Intel Pentium 4 processors, running at 200 MHz (1 Simics cycle = 5 ns), with 64 MB of RAM. Nodes run a RedHat 7.3 operating system, and use the MPICH-over-p4 implementation of MPI, using a (simulated) Ethernet card that accesses the 64-node network.

The link bandwidth of FSIN routers is 32 bits/cycle, or 1280 Mbits/s. Interaction with FSIN has been adjusted to run 2 FSIN cycles per 10 Simics cycles (1 FSIN cycle = 25 ns). These values allow us to obtain realistic execution times while still arriving to situations in which network congestion appears.

All the experiments have been repeated 10 times, using different random seeds. We measure execution times in terms of FSIN cycles. As these times vary from pattern to pattern (or from benchmark to benchmark) we represent values normalized to the Base case (without congestion control). Reported values are averages of 10 simulation runs. We also plot the 99% confidence intervals.

V. Analysis of experiments

In this section we use INSEE to evaluate IPR and LBR using different classes of synthetic traffic, traces taken from the NPB, and an execution-driven environment based on Simics, running the same benchmarks.

Regarding synthetic traffic, in Subsection *A* we use these patterns: BC (bit complement), BR (bit reversal), PS (perfect shuffle), TR (transpose), TO (tornado), UN (uniform) and HR (hot-region, see [3]). The first five are permutations: all packets generated at a given node have the same destination. With UN and HR destinations are chosen randomly—in the case of HR, 25% of the traffic goes to the first 8 nodes (labelled 0 to 7), and the remaining traffic is uniform.

A. Experiments using burst-synchronized traffic

In [7] we argued that the utilization of workloads based on the assumption of independent traffic sources (widely used in many simulation-based performance studies) is not realistic, because it is not representative of the way actual parallel applications work. We proposed to model workloads 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). See Fig. 2.

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 (see BR, PS and TR permutations) 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. The improvement for a larger value of b (10000) is not that spectacular. 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.

In short, congestion control is beneficial for most of the traffic patterns under consideration, provided that the burst size b is large enough. The only exception is the BC pattern, for which neither benefit nor harm are observed; this is because of a peculiarity of this permutation: it forces one half the traffic passing through link 0-63, and the other half through link 31-32. These two links act as bottlenecks (utilization is 100%) and the order in which packets traverse them is irrelevant.

B. Experiments using application traces

We claimed in the previous section that burst-synchronized 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 well-known NAS Parallel Benchmarks (NPB, see [12]).

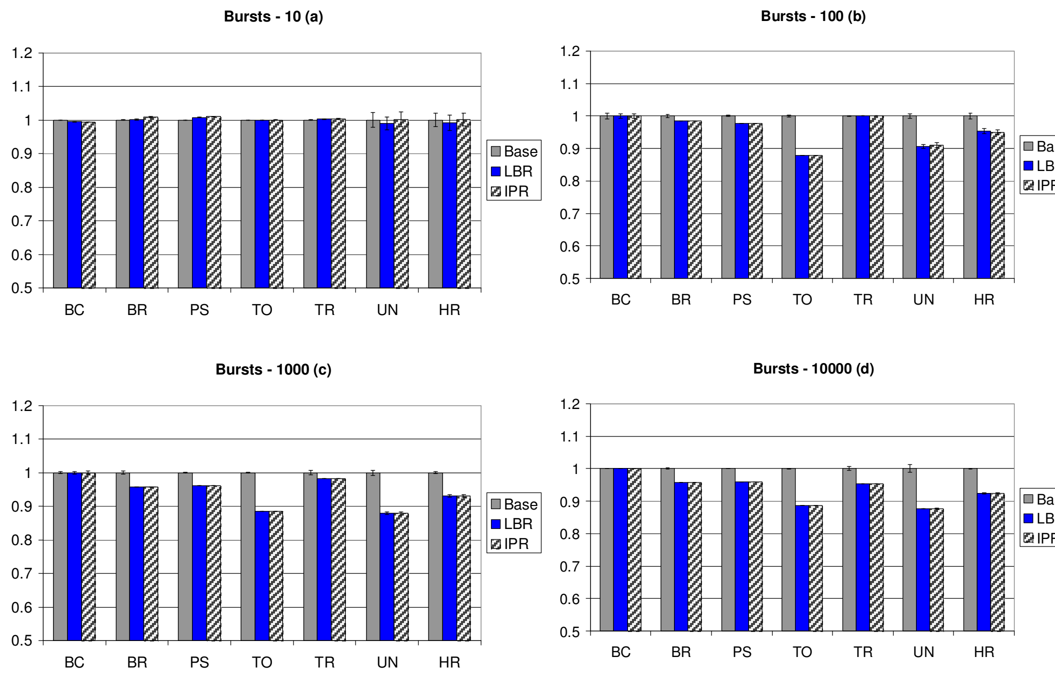

Fig. 2. Performance for burst-synchronized traffic, for burst sizes 10, 100, 1000 and 10000. Normalized times to consume a burst, averaged from 10 runs.

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. 3.

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%.

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 (injected in the network) 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. 4 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.

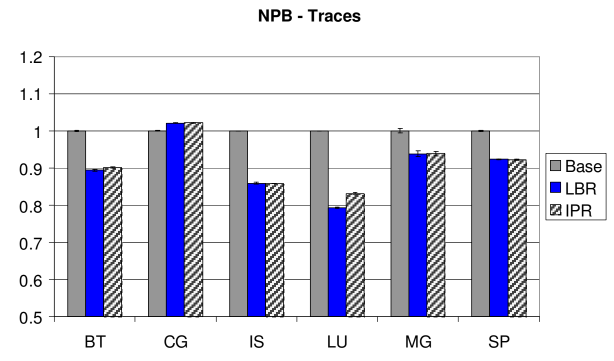

Fig. 3. Normlized results with traces of the NPB (A.64). Averages of 10 runs, and 99% confidence intervals.

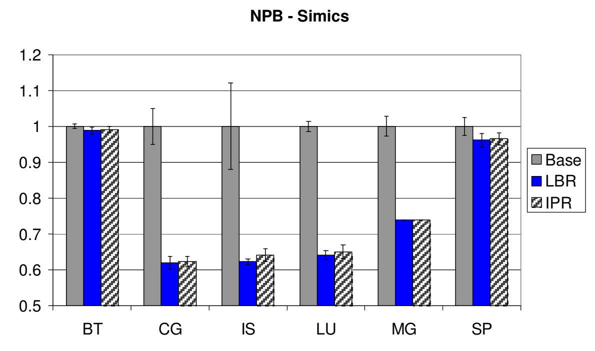

Fig. 4. Normalized results of the execution of the NPB (A.64) in a Simics+FSIN environment. Averages of 10 runs, and 99% confidence intervals.

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 [1], situation that does not allow for the full utilization of this resource.

Congestion control inside the network prioritizes in-transit 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.

VI. Conclusions 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 synchronized traffic, traces and real workloads generated by applications as they run. The three methods provide similar insights: congestion control provides important performance benefits for loosely-coupled applications that have communication intensive phases. However, it may penalize tightly-coupled applications which already have significant levels of self-throttling. In the worst scenarios, introduced performance fall is not very high (less than 2%). LBR performs slightly better than IPR, as show in the trace-based and execution-driven simulations.

In the experiments we have used a small, 64-node network. 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 behaviour of actual traffic; in particular, they must reflect how communicating processes synchronize.

The simple, burst-synchronized traffic used in this work appears to be a basic but good model to represent the self-throttling nature of the traffic produced by parallel applications. Further work is required to identify the range of values of b that better represent different workloads. This set of experiments has also shown some limitations in our trace-based simulation environment. We need to improve it in order to take into consideration computation times between message interchanges.

Also, we need to improve our execution-driven simulation setup to avoid the utilization of TCP, using instead a fast-messaging mechanism on top of our simulated network device; this would allow us to avoid interferences introduced by this protocol.

VII. References

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

[2] E. Baydal, P. Lopez and J. Duato, "A Familiy of Mechanisms for Congestion Control in Wormhole Networks" IEEE Trans. on Parallel and Distributed Systems, Vol. 16, No. 9, September 2005, pp 772-784.

[3] M. Blumrich, D. Chen, P. Coteus, A. Gara, M. Giampapa, P. Heidelberger, S. Singh, B. Steinmacher-Burrow, T. Takken, P. Vranas. "Design and Analysis of the BlueGene/L Torus Interconnection Network" IBM Research Report RC23025 (W0312-022) December 2003.

[4] 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

[5] W.J. Dally, B. Towles. "Principles and Practices of Interconnection Networks". Morgan-Kaufmann, 2004.

[6] J. Duato. "A Necessary and Sufficient Condition for Deadlock-Free Routing in Cut-Through and Store-and-Forward Networks". IEEE Trans. on Parallel and Distributed Systems, vol. 7, no. 8, pp. 841-854, 1996.

[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] C. Izu, J. Miguel-Alonso, J.A. Gregorio. "Effects of Injection Pressure on Network Throughput", in Proc. PDP 2006 14th Euromicro Conference on Parallel, Distributed and Network based Processing. Montbéliard-Sochaux - France- February 15-17 2006.

[9] R. Jain. "Congestion control in computer networks: issues and trends". IEEE Network, vol.4 no.3, pp 24-30, May 1990.

[10] J. Miguel-Alonso, C. Izu, J.A. Gregorio. "Improving the Performance of Large Interconnection Networks using Congestion-Control Mechanisms". Technical report EHU-KAT-IK-06-05. Department of Computer Architecture and Technology, The University of the Basque Country. Submitted.

[11] MCS Division – Argonne National Laboratory. "MPICH Home Page". Available (May 2006) at http://www-unix.mcs.anl.gov/mpi/mpich/

[12] NASA Advanced Supercomputing (NAS) division. "NAS Parallel Benchmarks" Available (May 2006) at http://www.nas.nasa.gov/Resources/Software/npb.html

[13] V.S. Pai, P. Ranganathan, S.V. Adve. "RSIM: An Execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors". In Proceedings of the Third Workshop on Computer Architecture Education, February 1997.

[14] V. Puente, C. Izu, J.A. Gregorio, R. Beivide, and F. Vallejo, "The Adaptive Bubble router", Journal on Parallel and Distributed Computing, vol 61, no. 9, pp.1180-1208 September 2001.

[15] F.J. Ridruejo, J. Miguel-Alonso. "INSEE: an Interconnection Network Simulation and Evaluation Environment". Lecture Notes in Computer Science, Volume 3648 / 2005 (Proc. Euro-Par 2005), pp. 1014 - 1023.

[16] J Shin, TM Pinkston. "The Performance of Routing Algorithms under Bursty Traffic Loads". Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications. Las Vegas (USA), Jun. 2003.

[17] A. Singh, W. J. Dally, A. K. Gupta, B. Towles. "Adaptive Channel queue routing on k-ary n-cubes". Proceedings of the Sixteenth Annual ACM Symposium on Parallel Algorithms, June 27-30, 2004, Barcelona, Spain SPAA'04 pp 11-19

[18] Y. H. Song, T. M. Pinkston. "Distributed Resolution of Network Congestion and Potential Deadlock Using Reservation-Based Scheduling". IEEE Trans. On Parallel and Distributed Systems, vol. 16, No. 8, August 2005, pp 686-701

[19] M. Thottethodi, A.R. Lebeck, S.S. Mukherjee. "Exploiting Global Knowledge to Achieve Self-Tuned Congestion Control for K-Ary N-Cube Networks". IEEE Trans. on Parallel and Distributed Systems, Vol. 15, No. 3, March 2004, pp 257-272.

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