I/O Bus Usage Control in PC-Based Software Routers[†]

Oscar-Iván Lepe-Aldama and Jorge García-Vidal

Department of Computer Architecture, Universitat Politècnica de Catalunya
c/ Jordi Girona 1-3, D6-116, 08034 Barcelona, Spain
{oscar,jorge}@ac.upc.es

Abstract. This paper presents a performance analysis of a fair sharing mechanism for PC-based software routers, required when the I/O bus and not the CPU is the bottleneck. The mechanism involves changes to the OS kernel and assumes the existence of certain NIC functions, but does not require any changes to the PC hardware architecture.

1 Introduction

We can define a software router as a computer that executes a program capable of forwarding IP datagrams among network interface cards (NIC) attached to its I/O bus. It is well known that software routers have performance limitations. However due to the ease with which they can be programmed for supporting new functionality software routers are still important at the edge of the Internet. After this, the question of how to optimize software routers performance arises. In addition, if we want to provide QoS guarantees for traffic going through the router, we must find a suitable way of sharing resources. In other pieces of work the problem of fairly sharing router resources is tackled in terms of protecting [1,4] or sharing [6] the use of the CPU amongst different packets or data flows. However, the increase in CPU speed in relation to that of the I/O bus makes it easy for this bus to constitute a bottleneck, which is why we address this problem.

This paper presents our proposal for a resource sharing mechanism that allows QoS levels to be guaranteed in software routers by jointly controlling I/O bus activity and CPU operation. It is a software mechanism that does not require changes to the PC hardware architecture and which introduces low overhead and avoids intrusion. It requires that NICs dispose of several direct memory access (DMA) channels—one for each traffic flow—working independently and having a set of descriptors that store usage information—NIC's buffer occupancy or the total number of arrivals to the channel. Moreover, this paper presents a study of the properties of the mechanism, when considered in isolation, and a system performance evaluation, when the mechanism is incorporated into a software router. We will concentrate on software routers built on desktop PCs running general purpose, open source operating systems—FreeBSD, which implement networking functions within the kernel.

[†] This piece of work was supported in part by the Mexican Government through CICESE Research Center's and CONACyT's grants. Also supported by the Spanish Minister of Science and Technology through project TIC2001-0956-C04-01

E. Gregori et al. (Eds.): NETWORKING 2002, LNCS 2345, pp. 1135-1140, 2002.

2 A Mechanism for Implementing I/O Bus Sharing

The mechanism we propose for implementing I/O bus sharing, and that we call Bus Utilization Guard (BUG), manipulates the vacancy space of the message buffer reception input queue of each DMA channel, so the overall activity at the I/O bus follows a schedule similar to one produced by a WFQ server. (For now on we referred to the I/O bus simply as the bus, and to a MBUF queue simply as a queue.) For minimizing intrusion, the mechanism is activated each T cycles and it is executed either by the CPU or by a suitable coprocessor placed at the AGP connector. For reducing overhead, the mechanism uses a two state behavior, monitoring and enforcing.

Assume that the mechanism is in monitoring state at cycle $k \bullet T$. Then, the mechanism gathers $D_{i,k}$—number of bytes transferred through the bus during period $((k-1) \bullet T, k \bullet T)$ by channel i. If $sum(D_{i,k}) < T/\beta_{BUS}$, where β_{BUS} is the cost per bit of bus transfer, the mechanism remains at monitoring state and no further actions are taken. On the contrary, the mechanism detects the start of a busy period and enters enforcing state. When at this state, the mechanism polls each NIC to gather $N_{i,k}$—number of bytes stored at the NIC associated with channel i—and computes the amount of bus utilization granted to each channel, or γ_{ik}, after the outputs of an emulated general processor sharing (GPS) server [5] with batched arrivals, or $G_{i,k}$. The input for the emulated GPS are the $N_{i,k}$ at the start of the busy period. Afterwards, the inputs are the amount of arrived traffic during the last period or $A_{,ik}=N_{i,k}-N_{i,k-1}+D_{i,k}$. BUG is work-conservative and thus

$$\gamma_{i,k} = G_{i,k} + (T/\beta_{BUS} - (G_{1,k}+\ldots+G_{N,k})) \quad (1)$$

Observe that $sum(\gamma_{i,k}) = T/\beta_{BUS}$, a situation that can lead to an unfair share. Consequently, BUG is prepared with an unfairness-counterbalancing algorithm. This algorithm computes an unfairness level per channel and if it detects at least one deprived flow, then it reduces $\gamma_{i,k}$ of every depriver flow by the corresponding unfairness value. One problem with this approach is that if unfairness is detected then

$$(\gamma_{1k} + \ldots + \gamma_{Nk}) / \beta_{BUS} = T \quad (2)$$

That is, the unfairness-counterbalancing algorithm may artificially produce some bus idle time. This problem also arises when packetzing bus utilization grants, as shortly explained. Happily, a single mechanism, one that allows BUG to vary the length of its activation period, solves both problems. The length T of BUG's activation period, in general, keeps no relationship with any packet bus-transmission time—besides having to be at least larger than the largest. Consequently, when packetizing utilization grants it may happened that $mod(\gamma_{i,k}, L_i)? 0$, where L_i is the mean packet length for channel i. Hence, some rounding off is required. We have tested rounding off both down and up and both produce particular problems. However, the former gave us a more stable mechanism. If nothing else is done, some bus idle time is artificially produced and the overall share assigned to that flow would be much less of what it should be. This problem can be solved if we let BUG reduced its next activation period length by some dt time value, where dt is the time due to

rounding off. Evidently, this increases BUG's overhead. But as long as *dt* is a small fraction of *T*, the increase will remain at acceptable levels.

BUG will switch from enforcing to monitoring state, resetting the emulated GPS, any time that $sum(D_{i,k}) < T/\beta_{BUS}$.

3 BUG's Dynamics

We devised a series of simulation experiments to assess the performance of a PCI bus controlled by a BUG. For all experiments we compared the responses of three simulated buses: a plain PCI, a WFQ bus and a BUG regulated PCI. We are approximating the PCI operation by a Round Robin scheduler. Operational parameters where computed after a 33 MHz, 32-bit bus. Besides, we set queue spaces to infinity and set BUG's nominal activation period to 0.1 ms. Traffic load for all experiments was composed of three packet-flows soliciting each 1/3 of router resources. Flows differentiate themselves by the size of their packets: small (172 bytes), medium (558 bytes) and large (1432 bytes). Different experiments used different inter-arrival processes to show particular behavior.

In Fig.4.a we show responses to unbalanced constant bit rate traffic. Each line at every chart denotes the running sum of output bytes over time. The traffic pattern is as follows. At time zero, flow 1 and flow 2 start loading the system with a load level equivalent to 50% of a PCI bus capacity each; that is, 528 Mbps. Two ms later (first arrow; 20T = 2 ms) flow 3 starts loading the system also at 528 Mbps. Then, 2 ms later (second arrow) flow 3 multiplies its bit rate by four. From the first chart we can see that the ideal bus behavior allows a 50% bus share between flow 1 and 2 during the first 2 ms. Then, after flow 3 gets active, it allows a 33% bus share irrespectively of the load level of flow 3. From the second chart we can see that a plain PCI bus only adequately follows the ideal behavior during the first 2 ms—first arrow. Then, the round robin scheduling deprives flow 1 in favor of flow 3. Moreover, although flow 2 is lightly affected it also receives more than its solicited share. After time 4 ms—second arrow—all gets worst. From the third we can see that the BUG equipped bus behaves very much like the ideal bus does. Observe that when flow 3 gets on, the reactive nature of BUG is reflected. For the first two activation periods, or so, flow 3 gets bus use-grants above its solicited share, depriving the other flows. But then, BUG adjusts and before 1 ms has passed all flows start receiving their solicited share. Before time 10 ms, flow 1 starts lagging a little behind flow 2. This is due to rounding off mismatches. By algorithm definition, when this mismatch accumulates to a whole packet BUG will allow flow 1 to catch up. We have practiced more experiments like the above varying the order of the flows and the length and size of the load changes and we have always found congruent results.

In Fig.4.b and Fig.4.c we analyze the dynamic behavior of BUG under highly variant random load. For this pair of experiments each packet flow was run by an on-off source. On-state period lengths were set to a constant value. Packet inter-arrival processes were Poisson with mean bit rate equal to 3520 Mbps, or 300% of the PCI bus capacity. Off-state period lengths were drawn after an exponential random process with mean value set to 9 times the on-state period-length. Consequently, all flows overall mean bit rates were equal to 30% of the PCI bus capacity or 352 Mbps. Besides observing the system response to this kind of traffic, with these experiments

we wanted to see if we could find any BUG pathology related to operating-mode cycles, where the continuous but random path into and out of enforcing mode may produce some wrong behavior. Consequently, we ran several experiments with different on-off cycle lengths. Here we present results for an on-state period-length 8 times the BUG activation period T (Fig.4.b) and for one of 0.5T (Fig.4.c). In both these figures, each chart left to right separately compares for each flow (flows 1, 2 and 3) the resulting output processes for each of the considered buses. Each line denotes the running sum of output bytes over time, and thus horizontal segments correspond to off-state periods. For reference, each chart also draws, as a running sum over time, the corresponding flow's input process. From both figures we can see that despite the traffic's fluctuations BUG quite well follows the ideal WFQ policy, while the PCI

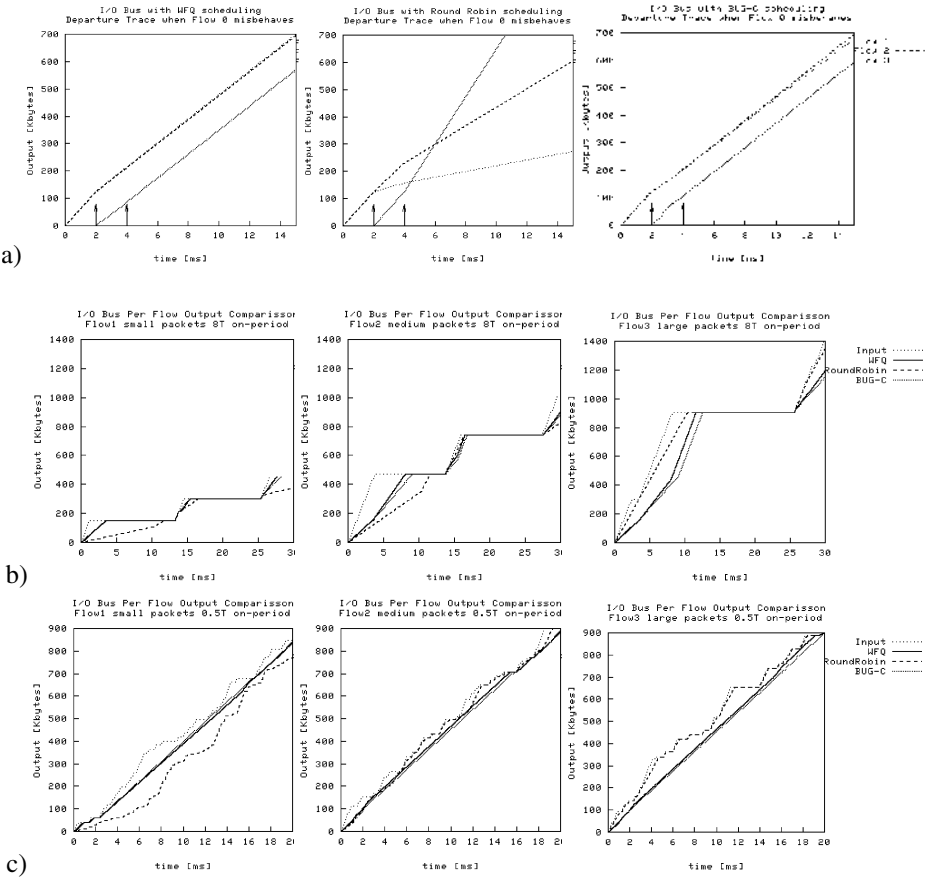

Fig. 1. Simulation results from BUG dynamics contrasting study under (a) unbalanced CBR traffic and (b,c) random and highly variable traffic. BUG's behavior is contrasted to the behavior of the ideal WFQ policy and the behavior of a PCI bus (approximated by a round-robin policy). Note that each chart at (a) contrasts the output processes of the three traffic flows described in the main text for a particular scheduling policy. While at (b,c) each chart contrasts the output processes produced by the three scheduling polices for one traffic flow.

like Round Robin policy again favors the largest-packet flow and affects the most to the smallest-packet flow. Of particular interest is what Fig.4.c show to us about BUG behavior. It seams that BUG is not macroscopically sensitive to a traffic pattern that repeatedly takes it in and out of enforcing mode.

4 System Performance Study

Here we study the performance of a PC based software router whose PCI bus in regulated by BUG. Operational parameters for the queuing network model were determined using software profiling, as described in [2]. The target system had a 600 MHz Pentium III CPU, a 100 MHz system bus, 10 ns EDO RAM chips and a 33 MHz, 32-bit PCI I/O bus. Software wise, the system was power by FreeBSD 4.1.1. Measurements were not taken for the bus service times. Instead, we used the description of the system operation [2]. We assume that data phases are of 1 cycle and that frame transfer is never pre-empted. We have considered Poisson traffic as input traffic, and which has a three-flow configuration as for the previous section.

We have performed the simulation with systems configured with two different CPUs. CPU1 works at 1 GHz and CPU2 works at 3 GHz. The system's I/O bus works at 33 MHz and has a 32-bit data path. Note that for the considered traffic, the CPU is the bottleneck for the system with CPU1 while the I/O bus is bottleneck for system with CPU2.

In Fig.5.a we show results for the basic software router. The left chart shows aggregated throughputs for offered loads in the range of [0, 1400 Mbps]. The other two charts show the share obtained for each traffic flow, firstly for CPU1 and then for CPU2. It can be seen that the system with CPU1 has a linear increase of the aggregated throughput for offered loads below 225 Mbps. At this point the CPU utilization is 100% while the bus utilization is around 50% and the systems enters into a saturation state. If we further increase the offered load the throughput decreases until a live lock condition appears, at an offered load of 810 Mbps. During the saturation state most losses occur in the IP input buffer. The system with CPU2 gets its bus saturated before its CPU at an offered load of 500 Mbps. The system behavior for increasing offered loads depends on which priorities are used by the bus arbiter. Summarily, the basic system cannot provide a fair share of the resources when it is in saturation. Fig.5.b shows results for the system with a WFQ scheduling for the CPU and the BUG mechanism for controlling bus usage. We see that the obtained results correspond to almost an ideal behavior, as under saturation throughput does not decrease with increasing offered loads and the system achieves a fair share of both router resources: CPU and bus.

5 Conclusions

Under quite normal operation conditions for today's PC hardware and tele-communication links, the plain PCI bus arbitration mechanism impedes a software router to fulfill QoS guarantees. The mechanism that we proposed and called BUG, for bus usage control, is effective in controlling the bus share between different flows.

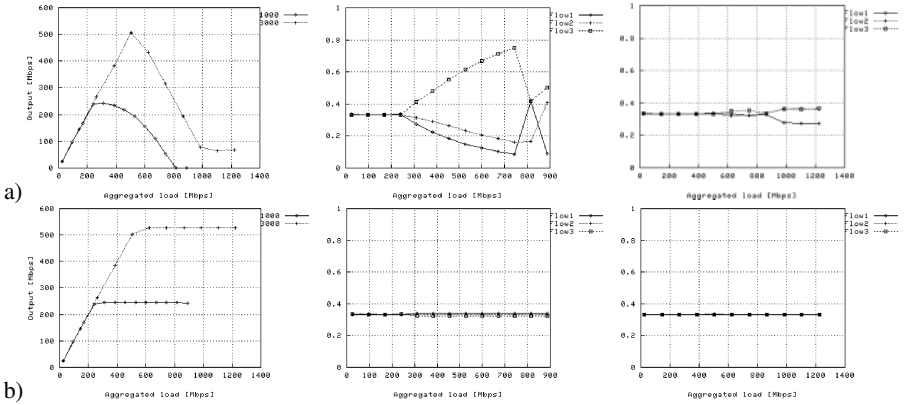

Fig. 2. Performance results for (a) base BSD router (b) a router with WFQ for the CPU and BUG for the I/O bus. The charts at the left contrast the router throughput when it uses a CPU of 1GHz and a CPU of 2GHz. The charts at the middle and at the right show the throughput share obtained by each of the three flows described in the main text. The charts at the middle are for a router using a 1GHz CPU, while the charts at the right are for a router using the 2GHz CPU.

When we use this mechanism in combination with the known techniques for CPU usage control, we obtain a nearly ideal behavior of the share of the software router resources for a broad range of workloads.

References

1. Indiresan, A. Mehra and K. G. Shin, "Receive Livelock Elimination via Intelligent Interface Backoff", December 1997, http://citeseer.nj.nec.com/366416.html
2. O. I. Lepe and J. García, "A Performance Model of a PC-based IP Software Router", to appear at Proc. IEEE ICC2002, April 2002.
3. M. L. Loeb, A. J. Rindos, W. G. Holland and S. P. Woolet, "Gigabit Ethernet PCI Adapter Performance", IEEE Network, 15(2): 42-47, March/April 2001.
4. Mogul and K. K. Ramakrishnan, "Eliminating receive livelock in an interrupt-driven kernel", ACM Trans. Computer Systems, 15(3): 217-252, August 1997.
5. K. Parekh and R. G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks- The Multiple Node Case", Proc. IEEE INFOCOM 1993, pp. 521-530 vol.2
6. X. Qie, A. Bavier, L. Peterson and S. Karlin, "Schedulling Computations on a Software-Based Router", Proc. SIGMETRICS 2001, June 2001.