Chapter 5

Conclusions and future work

5.1 Conclusions

We have showed that widely used, single-queue performance models of software routers implemented with BSD networking software, or similar, and personal computing (PC) hardware, or similar, incur in significant error when these models are used to study scenarios where the router's central processing unit (CPU) is the system's bottleneck and not the communications links. Furthermore, we have showed that a queuing network model better models these systems under the considered scenarios.

Armed with a mature characterization process, we have showed that it is possible to build a queuing network model of PC-based software routers that is highly accurate, so this model may be use to carry out performance studies at several detail levels. Furthermore, we have showed that model's service times computed after some system may be used for predicting the performance of other systems, if scaled appropriately, and that model's service times scale linearly with CPU's operation speed but can be considered constant with respect to messages' and routing table's sizes. Moreover, model's service times related to the network interfaces layer have linear behavior with respect to CPU's operation speed and their offset varies with respect to the network interface card's and device driver's technology and the cache memory performance.

Using our validated, parameterized, queuing network model, we have quantitatively showed that current CPUs allow a software routers to sustain high throughput when forwarding plain Internet Protocol datagrams if some adjustments are introduced to the networking software, and that current PC's input/output (I/O) buses, however, are the limiting factor for achieving system-wide high performance. Moreover, we have

quantitatively showed how and when current PC's I/O buses hamper a PC-based software router of supporting system-wide communication quality assurance mechanisms, or Quality-of-Service (QoS) mechanisms.

Under the light of the above statement, we proposed a mechanism for improving the resource sharing of the input/output bus of personal computer-based software routers. This mechanism that we called BUG, for bus utilization guard, does not imply any changes in the host computer's hardware, although some special features are required for network interface cards—they should have different direct memory access channels for each differentiated packets flow and they should be able to give information about the number of bytes and packets stored for each of these channels. When we use this mechanism in combination with known techniques for CPU usage control, we quantitatively showed that it is possible to obtain a nearly ideal behavior of the share of the software router resources for a broad range of workloads.

5.2 Future work

A precise analysis of the results obtained when a BUG protected PCI bus was loaded with self-similar traffic is missing in this document; see subsection 4.4.5. Recall that although results confirmed that under the considered load conditions, a BUG protected PCI bus better follows the ideal behavior of a WFQ bus when compare to a plain PCI bus. However, these results also showed a departure from the ideal behavior higher than we expected.

A working implementation for a production system of the BUG is missing in this document. Currently, an undergraduate student of the Facultad de Informática de Barcelona (FIB/UPC) is conducting his final project on this subject. He is implementing the BUG for a software router running FreeBSD release 4.5 and wearing 3COM's 3C996 PCI-X/Gigabit Ethernet network interface cards.

We find naturally pursuing to extend our modeling process to embrace the whole networking software; that is, to model PC-based communication processes involving Internet architecture's transport layer protocols and BSD's sockets layer. This, we think, is no easy task, as most certainly it would involve user-level application programs that are subject to the CPU scheduler. That is, the CPU's extended model would have to include not only the priority preemptive scheduling policy, which models the software interrupt mechanism, but also a round-robin preemptive policy for modeling the UNIX CPU scheduler. Furthermore, the CPU's extended model would have to switch between scheduling policies depending on the kind of tasks pending execution. In turn, this would require extending our characterization process for describing the tasks involve when context switching occurs. Evidently, if such a model could be built it would be valuable for capacity planning and as a uniform test-bed for Internet services.

We also find tempting to simplify the software router model's assumptions and to pursue to solve it analytically. The motivation would be to assess the trade-off between the model simplification and the error incurred.

Current telematic systems research is focusing on embedded systems for personal communications and ubiquitous computing. At the same time, current embedded sys-

tems wear complex microprocessors (some of which provide performance-monitoring counters like Intel's Pentium-class microprocessors do) and execute full-blown PC operating system kernels like OpenBSD or Linux. Under this light we find interesting trying to use our characterization and modeling process for studying the performance of telematic systems when executed over embedded hardware.

Other performance modeling challenges we find interesting are to study emerging software router technologies like SMP PC- and PC clusters-based software routers.