Techniques for Fast Packet Buffers

Sundar Iyer, Ramana Rao Kompella, Nick McKeown,
Department of Computer Science,
Stanford University,
Stanford, CA 94305.
{sundaes,ramana,nickm}@.stanford.edu

Abstract

The goal of our work is to design practical schemes for packet buffers operating at OC768 line rates and beyond. All packet switches (e.g. IP routers and ATM switches) must have packet buffers to hold packets during times of congestion. Frequently, the maximum speed at which a packet switch can operate is determined by the speed of its packet buffers. This is because a packet buffer must have enough bandwidth to both write (store) and read (retrieve) every packet. For example, an OC768c interface must be able to write arriving packets into its packet buffer, and retrieve them again, as fast as they arrive. For 40-byte minimum length IP packets, this corresponds to at least one write and read operation every 8ns – within the capabilities of commercially available SRAM, but beyond the speeds of commercially available DRAM. Unfortunately, because the buffers must be large, there is a need to use DRAM instead of SRAM. Amongst the memory technologies available today, only DRAMs support large buffer sizes but do not give fast random access into memory. Thus the design of packet buffers for fast line interfaces poses a real challenge which will increase as line rates increase.

Since neither DRAM or SRAM is suitable alone, a number of different (usually proprietary) techniques are used. These include: using a bank of DRAMs; cell striping and packet interleaving amongst multiple DRAMs; a combination of SRAMs and DRAMs etc. These techniques are generally implemented in an ad hoc manner, and often provide only statistical guarantees (hoping that the worst case traffic patterns do not happen). We aim to provide a variety of deterministic results and techniques that are useful to the design community at large. Armed with these results, we hope that designers need not re-invent the wheel.

We will assume a very general model for a packet buffer. The buffer consists of Q FIFO queues. Q could be one (if the buffer contains only a single FIFO queue), Q could be a small number (if the packet switch performs class-based queueing), Q could be quite large (if the packet buffer contains the virtual output queues of an input-buffered switch), or Q could be very large (if the packet buffer contains the per-flow queues of a traffic manager). When packets arrive to the packet buffer, they are added to the tail of the appropriate FIFO. Packets leave according to the decisions of an external scheduler. This could be, for example, a switch arbiter, or an output link scheduler. All that matters here is that the scheduler tells the packet buffer to give it the packet from the head of one of its FIFO queues. The packet buffer cannot predict which queue the scheduler will request next and must operate correctly no matter what sequence of requests it receives from the scheduler. It will be useful to consider two cases: One in which the scheduler is impatient and cannot wait for packets. In other words, when the scheduler requests the packet from a FIFO queue, the packet buffer must respond immediately and with - as near as possible - zero delay. The second case is one in which the scheduler is more patient and can tolerate a small, but bounded, delay for each packet. In this case, it is important to bound the delay so that the retrieval process can be pipelined and can keep up with the rate of new requests from the scheduler.

We will consider the following packet buffer architecture to meet these needs. It consists of b large off-chip DRAMs operating in parallel, and a small on-chip SRAM buffer. The b DRAMs are used to match the line rate. The SRAM is used as temporary storage so that packets can be retrieved quickly from the buffer. We can think of the SRAM as a "cache" holding just the heads and tails of each of the FIFO queues. The body of each FIFO resides in the DRAM banks. With sufficiently large b and a large enough SRAM cache, an arbitrarily fast packet buffer can be built. An obvious, and important, question to ask is: How large does the on-chip SRAM buffer need to be to meet our design requirements? And is there an algorithm for deciding which packets should be in the SRAM cache so as to minimize the SRAM buffer size?

Our results are as follows:

1. *An impatient scheduler.* If all the DRAMs share the same address bus, then all requests from the scheduler can be served immediately if the SRAM buffer is of size $Qb(2+lnQ)$. We will present an algorithm, which we call MDF (Maximum Deficit First) that achieves this bound.
2. *A patient scheduler.* If all the DRAMs share the same address bus, then all requests from the scheduler can be served within a bounded latency of $Q(b-1)+1$ time slots if the SRAM buffer is of size $Q(b-1)$. The

latency arises because the buffer management algorithm uses a lookahead buffer to gain a "sneak preview" of the requests it will receive next. We show that the buffer of size $Q(b-1)$ is necessary, and can be achieved using an algorithm we call ECQF (earliest critical queue first).

3. *A patient scheduler, and b address busses.* If each DRAM is addressed using a separate address bus, then the SRAM buffer can be eliminated completely, so long as the scheduler can wait for $2Q$ DRAM random access times.

As an example consider the design of the output buffers for an OC768c linecard with 32 Diffserv queues. We can meet the bandwidth requirements using 51.2ns DRAMs and cells segments of 64bytes, with an SRAM buffer of 115kbits, and the ECQF algorithm. When the output link scheduler decides which FIFO queue to serve next, the packet will be available within 2.9us.