Doubling Memory Bandwidth for Network Buffers

Youngmi Joo Nick McKeown

Department of Electrical Engineering,
Stanford University, Stanford, CA 94305-9030
{jym,nickm}@leland.stanford.edu

Abstract

Memory bandwidth is frequently a limiting factor in the design of high-speed switches and routers. In this paper, we introduce a buffering scheme called ping-pong buffering, that increases memory bandwidth by a factor of two. Ping-pong buffering halves the number of memory operations per unit time, allowing faster buffers to be built from a given type of memory. Alternatively, for a buffer of given bandwidth, ping-pong buffering allows the use of slower, lower-cost memory devices. But ping-pong buffers have an inherent penalty: they waste a fraction of the memory. Unless additional memory is used, the overflow rate is increased; in the worst case, half of the memory is wasted. Although this can be compensated by doubling the size of the memory, this is undesirable in practice. Using simulations, we argue that the problem is eliminated by the addition of just 5% more memory. We show that this result holds over a wide range of traffic and switch types, for low or high offered load, and continues to hold when the buffer size is increased.

1 Introduction

The speed of high-performance switches and routers is often limited by the bandwidth of commercially available memories. Compared to the continued rapid growth in network bandwidth, memory bandwidth is growing relatively slowly. Memory access time for the fastest available commercial DRAM is about 50ns, and has increased little in recent years.[1] Simply put, network switches and routers are running out of memory bandwidth.

Although the problem occurs equally in switches and IP routers, it is most easily understood in the context of ATM switches, which use fixed-sized cells. Consider a memory buffer with arrival and departure processes of cells, as shown in Figure 1. In each cell time, which we call a time-slot, zero ($A_n = 0$) or one ($A_n = 1$) new cell may arrive, and zero ($D_n = 0$) or one ($D_n = 1$) cell may depart from the buffer. This means that two independent memory operations are required per cell time: one write, and one read. If dual-ported memory is used, it would be possible for both operations to take place simultaneously. However, commercial considerations generally dictate that conventional single-ported memory be used. As a result, the total memory bandwidth (the sum of the arrival rates and departure rates) must be at least twice the line rate. For example, a memory buffer for an OC-48 (2.4Gb/s) line requires a memory bandwidth of 4.8 Gb/s. This corresponds to an access time of 6.7ns for a 32-bit wide memory.

In this paper, we consider a simple, and perhaps obvious, technique that eliminates the need for the two memory operations during each time slot. We call the technique "ping-pong buffering." A ping-pong buffer uses two conventional single-ported memories, where each memory need perform only one memory operation per time slot. The ping-pong buffer is similar, but not identical, to memory interleaving widely-used in computer systems [1].

The main benefit of a ping-pong buffer is that using conventional memory devices, it allows the design of buffers operating twice as fast. But ping-pong buffer's benefit comes with a penalty. As we will describe shortly, if the amount of memory is not increased, the rate of overflow from a ping-pong buffer is larger than for a conventional buffer.

It is the goal of this paper to: (1) explain the bandwidth advantages of ping-pong buffering, and (2) evaluate the penalty measured by the increased rate of overflow. As we will see, the increased rate of overflow can be removed by the addition of a small amount of memory. We limit the scope of our study to the buffers in input-queued ATM switches. However, ping-pong buffering can be used in any system that has at most one read and one write within a single unit of time. For example: the reassembly

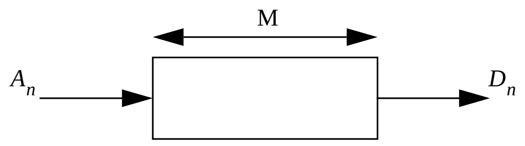

Figure 1: A buffer of capacity M, with arrival process A_n and the departure process D_n.

1.Newer memory architectures such as SDRAM and RAMBUS are now available. These devices use special techniques to increase bandwidth to and from the chip. However, the memory access time is unchanged.

buffers in switches and routers, or disk block data transfer within computer systems.

2 Ping-pong Buffer

Figure 2(a) shows a ping-pong buffer of total capacity M (cells), with the arrival and the departure processes denoted as A_n and D_n, respectively. The buffer consists of two physically separate memory devices, each of size $M/2$. The two memories are arranged so that from the outside, they appear to be a single buffer.

In a ping-pong buffer, read and write operations can take place simultaneously, but only in physically separate memory devices. When a cell arrives and finds that one memory is being read, as shown in Figure 2(b), the arriving cell is directed into the other memory device. We call this type of write operation a ‘constrained write’; we have no choice into which memory to write the arriving cell. On the other hand, if a cell arrives when there is no read operation, as in Figure 2(c), then we are free to choose which memory to write the cell into. We call this an ‘unconstrained write’. One possibility, chosen in our study, is to write the data into the less occupied memory.

For the ping-pong buffer depicted in Figure 2, let the occupancies of memory devices 1 and 2 be X_1 and X_2, respectively. The state of this ping-pong buffer, represented by the pair (X_1, X_2), is shown on a plane in Figure 3(a). At each time slot, there are five possible movements for this point, marked as (i) to (v) in the figure. If a cell arrives when there is no read operation occurring, then the point moves up by one (i), since $X_1 > X_2$ and we chose to write into the less populated memory. When an incoming cell finds memory 1 being read, then the point moves towards upper left (ii), and if memory 2 was being read, the next point would move to the lower right (iii). A read operation without any arrival results either in the movement to the left (iv) or one below (v), when memory 1 or 2 was being read, respectively. Figure 3(b) shows one example ‘trajectory’ of the state of a ping-pong buffer.

As an example of how a ping-pong buffer might be used,

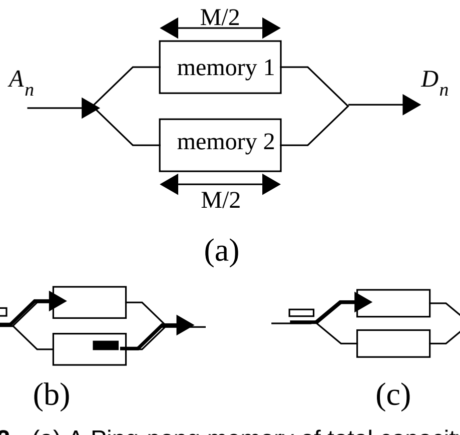

Figure 2: (a) A Ping-pong memory of total capacity M, (b) Constrained write, (c) Unconstrained write.

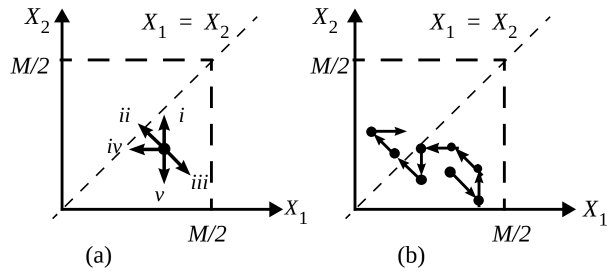

Figure 3: 2-dimensional representation of the state of a ping-pong memory of size M, (a) Possible movements at one point, (b) Example ‘trajectory’.

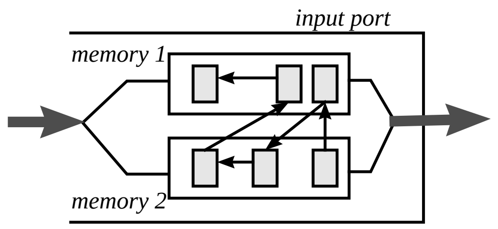

Figure 4: Schematic diagram of a ping-pong buffer in an input-queued ATM switch.

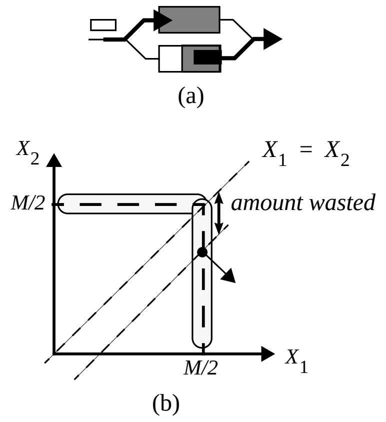

Figure 5: (a) An overflow in a ping-pong buffer, (b) Overflow seen on the ‘occupancy plane’.

Figure 4 shows the schematic diagram of a ping-pong buffer used in an input-queued ATM switch. A cell in the singly linked list is shown as a gray rectangle; each cell points to the cell that arrived behind it. Note that two cells arriving consecutively may reside in different memories, or in the same memory; the location of an arriving cell depends on what was occurring at the time of its arrival. Consequently, the logical ‘queue’ of cells is shared across both memory devices.

3 Increased Overflow Rate

Although a ping-pong buffer doubles memory bandwidth, it can lead to overflows even when the buffer has space available. Figure 5(a) shows a cell arriving to a ping-pong buffer when one memory is full, while the other is not. If the less occupied memory is being read, then the incoming cell can only be directed to the memory that is already full, causing an overflow. The overflows caused by ping-pong buffering correspond to the case where the point (X_1,X_2) attempts to move across the boundary '$X_1 = M/2$' or '$X_2 = M/2$', shown as shaded regions in Figure 5(b). Note that the total number of cells in the ping-pong buffer is less than its total capacity. A conventional buffer of the same capacity, subject to the same arrival and departure processes, wouldn't have caused this overflow. Hence, we can see that a ping-pong buffer would have a higher overflow rate than the conventional buffer of the same size. Or, equivalently, ping-pong buffer wastes a certain amount of space at each overflow, marked as "amount wasted" in Figure 5(b).

The increased frequency of overflows may limit the benefit of ping-pong buffering. Before using such a buffering scheme, we should understand the increased overflow rate. First, we take an intuitive look at the increased rate of overflow, then in the next section we will take a more quantitative approach.

If we choose to write into the less occupied memory at each unconstrained write, we can expect that the two memories will maintain almost the same level of occupancies. In other words, the point (X_1,X_2) would spend most of its time inside the shaded oval in Figure 6(a). When the overflows due to ping-ponging occur, the memory that is not full would have only a small amount of space left. This can be interpreted as the ping-pong buffer performing like a conventional buffer only slightly smaller in size.

However, one can create a much worse sequence of read and write operations leading to more frequent overflows. If a burst of cells is written into one memory, which later causes another longer burst to be driven into the other memory; the resulting trajectory of (X_1,X_2) can look like Figure 6(b). We demonstrate this using the following example in Figure 7. Assume, for this example, that cells are read out in first-in-first-out order, and that an unconstrained write chooses to put the cell in memory 1 when $X_1 = X_2$. Each cell, drawn as a square, is marked by the order in which it has arrived. Departing cells at each time slot are shaded.

In the example shown in Figure 7, beginning at time-slot 14 the discrepancy in memory occupancies increases from 0 to 6 to 12. Had the size of each memory been 8, the ping-pong buffer in Figure 7 would have had 3 overflows by time slot 28. Even for a larger memory size, the widely oscillating occupancies could repeatedly cause overflows.

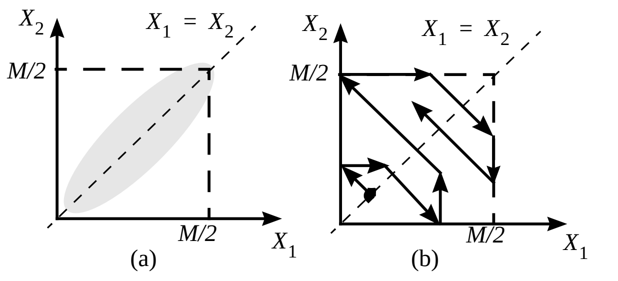

Figure 6: (a) Ping-pong buffer maintaining even level of occupancies, (b) Oscillating occupancies.

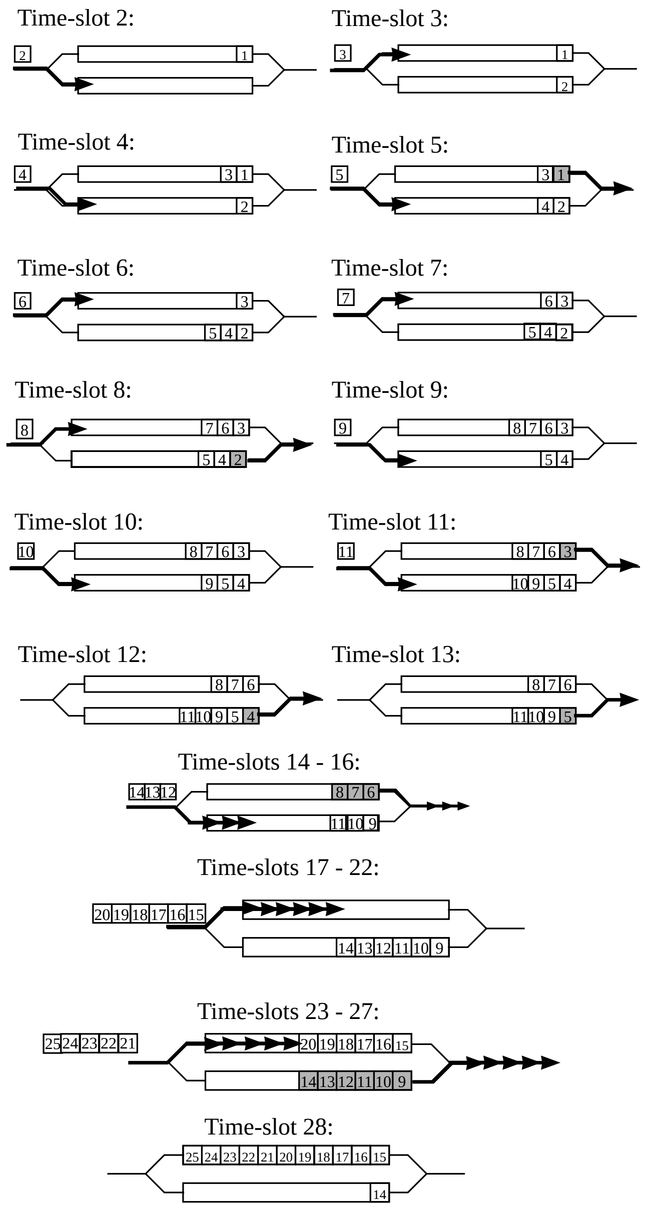

Figure 7: Example of increasing imbalance in memory occupancies

In summary, we see that pathological cases are possible, and may lead to only half the memory being used. A ping-pong buffer may perform, at worst, like a memory that is twice as fast, but only half the size. Fortunately, in practice, such pathological cases are rare. A 'typical' simulation result[2] shows something in-between the two extremes. Figure 8 shows the overflow rates of conventional buffer and a ping-pong buffer when used in an input-queued ATM switch.[3] The results confirm our intuition; a ping-pong buffer clearly has higher overflow rate than a conventional buffer of the

2. We will see more simulation results later.

3. The switch maintains a single FIFO queue at each input.

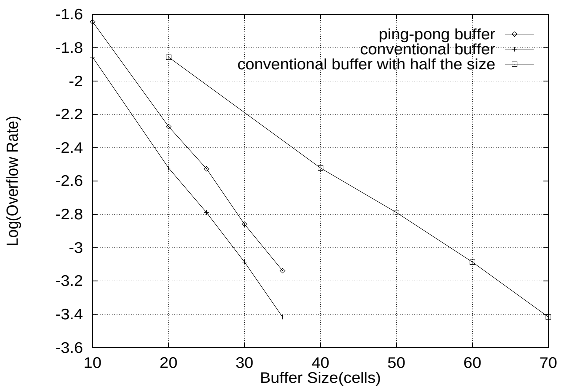

Figure 8: Log(overflow rate) vs. buffer size(Cells). Uniform i.i.d. Bernoulli traffic; utilization 57%.

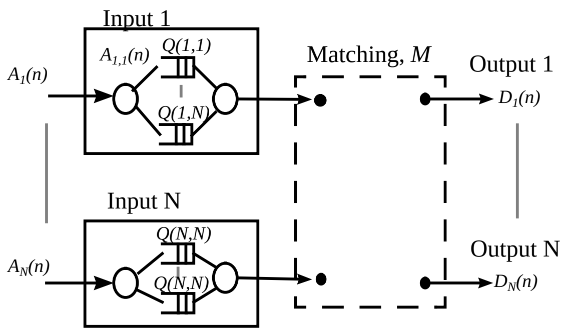

Figure 9: An Input-Queued Cell-Switch with virtual output queues.

same total size. However, a ping-pong buffer has an overflow rate much lower than a conventional buffer of half the size.

4 Estimating the Overflow Rate

4.1 Our Model

We focus our study of ping-pong buffering on its application to input-queued switches. Figure 9 shows our switch model with N inputs and N outputs. At the beginning of each time slot, either zero ($A_i(n) = 0$ for input i at the n-th time slot) or one ($A_i(n) = 1$ for input i at the n-th time slot) cell arrive at each input. We call $A_i(n)$ the arrival process to the input port i, where n represents the time index. To prevent head-of-line blocking, a cell destined for output j arriving at input i is placed in the FIFO queue $Q(i,j)$. We call $Q(i,j)$ a "virtual output queue". At the beginning of each time slot, a scheduling algorithm (sometimes called an 'arbiter') decides which cells to deliver to which output ports. The algorithm selects a *matching*, M, between the inputs and outputs, such that each input is connected to at most one output, and each output is connected to at most one input. At the end of the n-th time slot, if input i is connected to output j, one cell is removed from $Q(i,j)$ and sent to output j, corresponding to $D_j(n) = 1$.

In this paper, we assume that all the arrival processes, $A_i(n)$, $1 \le i \le N$, are described by a single random process, denoted $A(n)$, and that arrival processes at different inputs are independent of each other.

For the arrival processes, we first consider uniform independent identically distributed (i.i.d.) Bernoulli arrivals. Although this traffic is not representative of real network traffic, it can help us understand the way that ping-pong buffering behaves. Next, we consider whether burstiness in arriving traffic causes more frequent overflows. Intuitively, one might expect that the burstiness in arrivals would lead to burstiness in read operations(departures). This, in turn, could produce an increasing imbalance in memory occupancies, hence increasing the rate of overflow. To see the effect of burstiness on the performance of ping-pong buffer, we consider 2-state Markov-modulated Bernoulli arrivals. Often used as a simple model of burstiness, it models a simple on-off process that alternately produces a burst of cells, all with the same destination, followed by an idle period. The length of bursty and idle periods are geometrically distributed. Although studies have shown that this type of model fails to capture long-range dependence [9], we can expect that the results from this simple Markov-modulated traffic will provide a qualitative understanding of the effect of burstiness on ping-pong buffering.

We studied three switches:

1. a conventional input-queued switch with a single FIFO queue at each input. This switch is limited to approximately 58% throughput due to Head-of-Line blocking.
2. an input-queued switch with virtual output queues and a RANDOM scheduler. With RANDOM scheduling, the scheduler arbitrarily selects one out of all the possible matching patterns of inputs and outputs, with uniform probability, independently each time slot.
3. an input-queued switch with virtual output queues and an *i*SLIP scheduler [2]. *i*SLIP is an iterative round-robin scheduler that attempts to provide equal access to each output, and is simple to implement in hardware. This algorithm is representative of a variety of similar algorithms that attempt to achieve a maximal matching [3][4][5][6][7][8].

Finally, we only present simulation results for the case when offered load is high. This is because we are interested in the overflow rate — overflows only occur in significant numbers (and therefore only become measurable) when the offered load is high. Later in the paper, we will argue that in order to understand the memory wastage due to ping-pong buffering, it suffices to look only at the cases when the offered load is high.

4.2 Our Metric for the Cost of Ping-pong Buffering

The cost of a ping-pong buffer is that it wastes a fraction of the buffer space. For a given overflow rate, R, we measure this cost using a *wastage factor*:

$$\omega(R) \equiv \frac{M(R) - \tilde{M}(R)}{M(R)}, \qquad (1)$$

where $M(R)$ is the size of the ping-pong buffer that yields

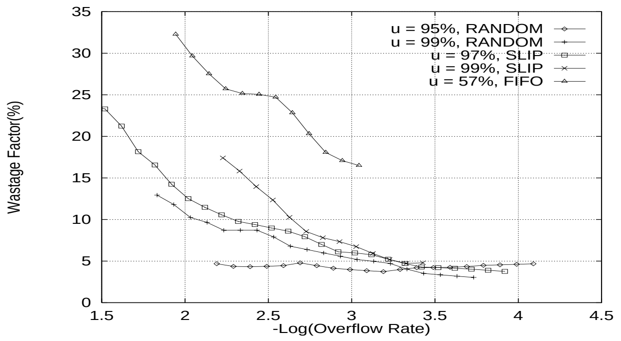

Figure 10: . Wastage factors vs. -Log(overflow rate) -- i.i.d. Bernoulli traffic.

overflow rate R , and $\tilde{M}(R)$ is the size of the conventional buffer that we would need to achieve the same overflow rate. In other words, $\omega(R)$ is a measure of the amount of memory that we waste by using a ping-pong buffer.

We calculate $\omega(R)$ from our simulation results as follows. First, we find many points representing the overflow rate as a function of memory size for ping-pong buffers and conventional buffers. We then interpolate between these points and measure the horizontal distance between values for each kind of buffer. For example, the top plot in Figure 10 is determined from the lower two plots in Figure 8.

4.3 Simulation Results for Benign Traffic

Figure 10 shows the wastage factors of ping-pong buffer subject to uniform i.i.d. Bernoulli traffic with different values of utilization and scheduling algorithms. Recall from Figure 8 that the overflow rate drops rapidly with increasing memory size. We can conclude from Figure 10 that

1. the *wastage factor* decreases as the memory size is increased.
2. the *wastage factor* appears to be independent of buffer size for sufficiently large buffers. In practice, it is likely that the buffer size would be chosen to ensure an overflow rate much lower than 10^{-4} . We may conclude that for i.i.d. Bernoulli traffic the *wastage factor* will be below 5%, and almost independent of buffer size.

Our findings are in agreement with the qualitative argument presented in Section 3, where we considered the discrepancy of the memory occupancies at the times of overflows. The discrepancy is independent of memory size. However, if the buffer is small, then it constrains the discrepancy to small values. A small increase in memory size significantly reduces the probability that a discrepancy will cause an overflow. This is illustrated in Figure 11(a). On the other hand, if the buffer is large, a small increase in memory size barely changes the probability of a discrepancy causing an overflow. This is illustrated in Figure 11(b).

4.4 Simulation Results for Bursty Traffic

We find that, contrary to our expectation, burstiness in the incoming traffic *reduces* the fraction of buffer space wasted by ping-pong buffering. This is shown by Figure 12, which compares wastage factors for Markov-modulated Bernoulli arrivals, with the RANDOM and *i*SLIP scheduling algorithms.

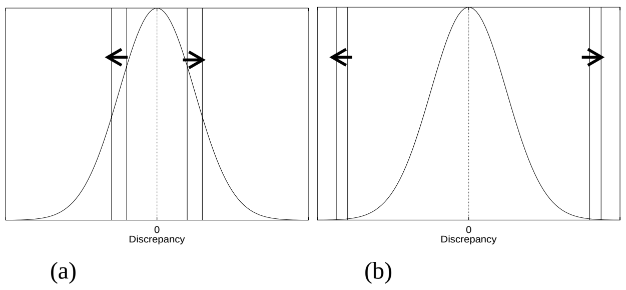

Figure 11: Constraints imposed by memory size on the distribution of discrepancy, (a) small buffer, (b) large buffer size.

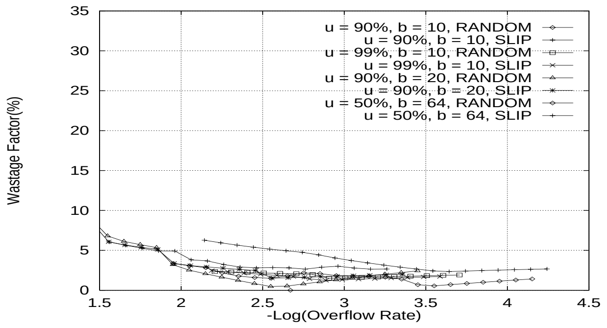

Figure 12: Wastage factors vs. -Log(overflow rate) -- 2-state markov traffic.

We believe that increased burstiness will decrease the wastage factor, but as yet have been unable to prove this result in general. Instead, we offer an intuitive explanation. First, recall the example in figure 7 that shows how discrepancy (and hence increased wastage factor) can arise in a FIFO ping-pong buffer. The discrepancy was caused by a sequence of constrained writes that forced an arriving stream of cells to be written into only one of the memory devices. Later, when this sequence of cells departed (all from one memory), they forced a sequence of arriving cells to be written into the other memory. In other words, when a sequence of arriving cells causes discrepancy, the same sequence of cells will cause further discrepancy when they depart. We say that in this way, discrepancy *propagates* in a FIFO ping-pong buffer.

Now instead, suppose that the cells had not departed in FIFO order; in fact, let's assume that they departed in a random order. This would mean that the first sequence of arriving cells would not force all of the cells in a future sequence to be written into the same memory device. In other words, the random departure order breaks the propagation of discrepancy from one sequence of cells to the next.

Now consider what happens when the buffer is arranged as a set of VOQs rather than a single FIFO queue. The scheduling algorithm will cause the cells to depart in non-FIFO order, and so we can expect the wastage factor to be decreased. Indeed, this is what our results show. Furthermore, we find that if the arrivals are more bursty, then their departure order is even further from FIFO. This is caused by the scheduling algorithm. In an attempt to achieve fairness among the VOQs, the scheduling algorithm is unlikely to service the same VOQ twice in a row. This means that cells arriving in a burst (and hence for the same destination), are unlikely to depart consecutively, or even close to each other in time. If, on the other hand, the arrivals are not bursty, then successively arriving cells are likely to be for different destinations. It is therefore quite possible that the scheduler will cause them to depart consecutively, or close to each other in time.

In summary, we believe that a FIFO departure order leads to the propagation of discrepancy, and hence increased wastage factor. Independent arrivals lead to departures that are closer to FIFO order than bursty arrivals. As the arrivals become more bursty, the discrepancy is less likely to propagate. Hence, bursty arrivals lead to a smaller wastage factor.

4.5 Variation of Wastage Factor with Offered Load

All of the results that we have presented have been for the case when the offered load is high. Although we see that the *wastage factor* is low for a reasonably sized memory, it is worth asking whether the *wastage factor* is still small when we reduce the offered load.

Indeed, we find that as we reduce the offered load for different types of traffic, and for different switches, the *wastage factor decreases*. For brevity, we do not reproduce lots of results here; a typical result is shown in Figure 10, which shows that as the offered load decreases, so does the *wastage factor*.

Intuitively, the relationship between offered load and *wastage factor* is simple. Recall that wastage in a ping-pong buffer is caused by discrepancy, which in turn is caused by constrained writes. When the offered load is high, there are a large number of reads, which constrain the writes of the (plentiful) new arrivals. Therefore, the probability that an incoming cell faces an constrained write increases with the offered load. Discrepancy is increased, leading to larger wastage.

Therefore, although our simulation results focus on switches with high offered load, we can infer that when the offered load is reduced, the *wastage factor* will decrease.

4.6 Variation of Wastage Factor with Buffer Size

We can also ask whether the *wastage factor* will decrease as we increase the buffer size. Because of the time to perform the simulations, our results cannot be extended to large buffers. However, our limited results indicate that the *wastage factor* will decrease for a variety of switches and traffic types. But they cannot prove it conclusively.

We feel that it is an important question to answer. For if the *wastage factor* were to increase, it would decrease the efficacy of ping-pong buffering. We therefore show analytically that the wastage factor does not increase with buffer size. The details of our arguments are contained in Appendix A.

5 Conclusion

Ping-pong buffering is a technique for doubling the bandwidth of a memory buffer. Its advantage is clear: by limiting each memory to just one memory operation per unit time, it enables buffers to be built that operate twice as fast; or for a given speed, it allows buffers that use slower, lower-cost memories.

We have seen that ping-pong buffering comes with a penalty: when compared with a conventional buffer, it will experience more frequent overflows. Caused by discrepancy in the memory occupancies, the overflows can waste up to half of the memory space. A simple solution to this problem is to double the amount of memory used; a rather unsatisfactory solution. Memory buffers are frequently used in systems requiring substantial storage capacity, and often dominate the system cost.

We have studied the amount of memory that is wasted in practice. Our main result is that in the worst case that we studied, a reasonably sized buffer will lead to a wastage of less than 5%. Although our results are based on simple traffic models with high offered load, we argue that our result will hold for a wide range of operating conditions. In particular, when the traffic is more bursty, when the buffers are larger (as they are likely to be in practice), and when the offered load is reduced.

Acknowledgement

We are very grateful for the helpful comments from Professor Peter Glynn at Stanford University.

6 References

[1] Patterson, D. and J. Hennessy, *Computer Architecture: A Quantitative Approach*, second edition.2nd ed. San Francisco: Morgan Kaufmann Publishers, c1996.

[2] McKeown, N., “Scheduling Algorithms for Input-Queued Cell Switches,” PhD Thesis. University of California at Berkeley, 1995.

[3] Ali, M., Nguyen, H., “A neural network implementation of an input access scheme in a high-speed packet switch,” *Proc. of GLOBECOM 1989*, pp. 1192 - 1196.

[4] Anderson, T., Owicki, W., Saxe, J. and Thacker, C., “High-speed switch scheduling for local area networks,” *ACM Trans. on Computer Systems*, pp. 319 - 352, Nov. 1993.

[5] Brown, T., Liu, K., “Neural network design of a Banyan network controller,” *IEEE J. of Selected Areas of Communications*, vol. 8, pp. 1289 - 1298, Oct. 1990.

[6] Chen, M., Georganas, N., “A fast algorithm for multi-channel/port traffic scheduling,” *Proc. IEEE Trans. Information Theory*, vol. 37, no. 1, pp. 114 - 121, 1991.

[7] Obara, H., “Optimum architecture for input queueing ATM switches,” *IEE Electronics Letters*, pp. 555 - 557, March 28, 1991.

[8] Troudet, T., Walters, S., “Hopfield neural network architecture for crossbar switch control,” *IEEE Trans. on Circuits and Systems*, vol., CAS-38, pp. 42 - 57, Jan. 1991.

[9] Leland, W.E., Willinger, W. Taqqu, M., Wilson, D. “On the self-similar nature of Ethernet traffic,” *Proc. of Sigcomm.*, San Francisco, 1993, pp. 183-193

[10] Glynn, P.W. and Whitt, W., “Logarithmic asymptotics for steady-state tail probabilities in a single-server

queue", *Journal of Applied Probability, 31A (1994), pp.131--156.*

[11] Dembo, A., and Zeitouni, O., Large Deviations Techniques and Applications, 2nd edition, Jones and Bartlett, 1993.

Appendix A: Bounding the Wastage Factor

Recall that in Section 4.2, we defined the wastage factor, $\omega(R)$, to be a function of the overflow rate, denoted as R, and plotted $\omega(R)$ against $-\log(R)$. Since $-\log(R)$ is increasing with memory size, it would be equally informative if one looks at the same ratio, but with $\tilde{M}(R)$ defined in a slightly different way.

In particular, we start with the size of a ping-pong buffer, M, and define the following quantities:

$P_{OV}(M) = Pr$ [Overflow in ping-pong buffer of size M],
$\tilde{P}_{OV}(M) = Pr$ [Overflow in conventional buffer of size M].

Let $\tilde{M}$ be the size of a conventional buffer such that

$$\tilde{P}_{OV}(\tilde{M}) = P_{OV}(M). \tag{A.2}$$

For a given M, we will consider a slightly different measure of wastage

$$W(M) = (M - \tilde{M})/M. \tag{A.3}$$

Our goal is to show that when M is sufficiently large, $W(M)$ can be bounded from above.

Since a ping-pong buffer cannot overflow unless at least one buffer is full, we observe the following fact.

Fact 1: $W(M) \le 1/2$ *for all* M.

Before we introduce and prove our main result, we need the following assumption regarding the overflow performance of a conventional buffer [10].

Assumption 1: *For a conventional buffer of size M, its overflow probability is*[4]

$$\tilde{P}_{OV}(M) = \exp(-\gamma \cdot M) + o(\exp(-\gamma \cdot M)), \tag{A.4}$$

where γ *is some positive constant, determined by the arrival and the departure processes.*

Note that with this assumption and (A.2), we can write

4.The exact statement is in the form of logarithmic asymptotics, and the result is originally for the waiting time of a single-server queue. However, the same proof in [10] can carry over to the queue length distribution, which approximates the loss probability.

$$\tilde{M} = -\frac{1}{\gamma} \cdot \log P_{ov}(M) - \frac{1}{o(1/M)}. \tag{A.5}$$

The proof of the lemmas leading to the main result rely on the concept of sub-additivity, defined below.

Definition: *A function* $f: Z_+ \to [0,\infty]$ *is said to be sub-additive if* $f(m+n) \le f(m) + f(n)$ *for all* $m, n \in Z_+$.

Lemma 1: *There exists a finite number* M_0 *such that* $\log P_{ov}(M)$ *is sub-additive whenever* $M \ge M_0$.

Proof: Proof by contradiction. Suppose that $\log P_{ov}(M)$ fails to be sub-additive for infinitely many M. Then, we can find an infinite sequence of such values, denoted as $\{M_n\}$, so that $\log P_{ov}(M)$ is not sub-additive along the sequence, i.e.,

$$\log P_{ov}(M_m + M_n) > \log P_{ov}(M_m) + \log P_{ov}(M_n) \quad \forall m, n \in Z_+. \tag{A.6}$$

Then, by the sub-additivity lemma [11], the following holds:

$$\lim_{n\to\infty} \frac{\log P_{ov}(M_n)}{M_n} = \sup_n \left[\frac{\log P_{ov}(M_n)}{M_n}\right].$$

Since $P_{ov}(M) \le 1$, $\frac{\log P_{ov}(M_n)}{M_n} \le 0$, and therefore, $\sup_n \left[\frac{\log P_{ov}(M_n)}{M_n}\right] \le 0$. However, $\sup_n \left[\frac{\log P_{ov}(M_n)}{M_n}\right] < 0$ cannot be true, because that implies

$$\log P_{ov}(M_n) = -\alpha \cdot M_n + o(M_n)$$

for some constant $\alpha > 0, c > 0$, which in turn prevents (A.6) from being valid for infinitely many $m, n \in Z_+$.

Therefore, $\sup_n \left[\frac{\log P_{ov}(M_n)}{M_n}\right] = 0$, and we thus have $\lim_{n\to\infty} \frac{\log P_{ov}(M_n)}{M_n} = 0$.

From (A.5),

$$\lim_{n\to\infty} W(M_n) = \lim_{n\to\infty} \frac{M_n - \tilde{M}_n}{M_n} = 1 + \lim_{n\to\infty} \frac{1}{\gamma} \cdot \frac{\log P_{ov}(M_n)}{M_n} = 1 \tag{A.7}$$

However, this contradicts Fact 1. ■

Lemma 2: *Let* $f(M) = M \cdot W(M)$. *Then, there exists a finite number* M_0 *such that* $f(M)$ *is sub-additive whenever* $M \ge M_0$.

Proof: By the definition of $f(M)$ and by (A.5),

$$f(M) = M \cdot W(M) = (M - \tilde{M}) / M$$
$$= M + \frac{\log P_{ov}(M)}{\gamma} - O(M)$$

Note that the first and the third term of the above equation are additive (or, the third term could be sub-additive). And from Lemma 2, $\log P_{ov}(M)$ is sub-additive whenever $M \geq M_0$ for some finite M_0.

Therefore, there exists a finite number M_0 such that $f(M)$ is sub-additive whenever $M \geq M_0$. ■

Theorem 1: *Let $W(M)$ be defined as in (A.3). Then,*

$W(M) \rightarrow \underset{M > M_0}{inf} W(M)$ *as* $M \rightarrow \infty$, *where* M_0 *is defined as in* Lemma 2.

Proof: By the definition of $f(M)$, we have $W(M) = f(M)/M$. By Lemma 3, $f(M)$ is sub-additive for $M > M_0$. Then by the sub-additivity lemma, it holds that:

$$\frac{f(M)}{M} \rightarrow \underset{M > M_0}{inf} \left(\frac{f(M)}{M} \right) \text{ as } M \rightarrow \infty .$$

Therefore, $W(M) \rightarrow \underset{M > M_0}{inf} W(M)$ as $M \rightarrow \infty$. ■

The following corollary of Theorem 1 provides a useful fact.

Corollary 2: *Suppose that a ping-pong buffer of size $\hat{M}$ yields a wastage factor $\hat{W}$. Then, there exists a finite buffer size $M^* \geq \hat{M}$ such that, any ping-pong buffer of size M^* or larger has a wastage factor less than $\hat{W}$.*

Proof: Observe that $\hat{W} = W(\hat{M}) \geq \underset{M > M_0}{inf} W(M)$, and recall that $W(M) \rightarrow \underset{M > M_0}{inf} W(M)$ as $M \rightarrow \infty$. Then, by the property of the limit and by Theorem 1,

$$W(M) - \underset{M > M_0}{inf} W(M) > W(\hat{M}) - \underset{M > M_0}{inf} W(M)$$

holds only for finitely many M.

Therefore, there exists $\hat{M} \leq M^* < \infty$ such that $W(M) \leq W(\hat{M})$ whenever $M \geq M^*$. ■

Note that the above result does not prove that the function $W(M) = (M - \tilde{M}) / M$ is *monotonically* decreasing in M. However, our result shows that when the memory is large enough, the wastage factor will be bounded from above by a constant. This is supported by our simulations results in Figures 10 and 12.