CSE 538 Notes

Roch Guérin
guerin@wustl.edu

January 14, 2014

Chapter 1

Connecting Inputs to Outputs, a.k.a. Switching

Consider the basic problem of connecting a set of *inputs* to a set of *outputs*. Inputs and outputs operate independently, and connectivity between them is provided through a *fabric* often referred to as a *switch*. The system overall structure is shown in Fig. 1.1 and consists of M inputs and N outputs connected through the fabric.

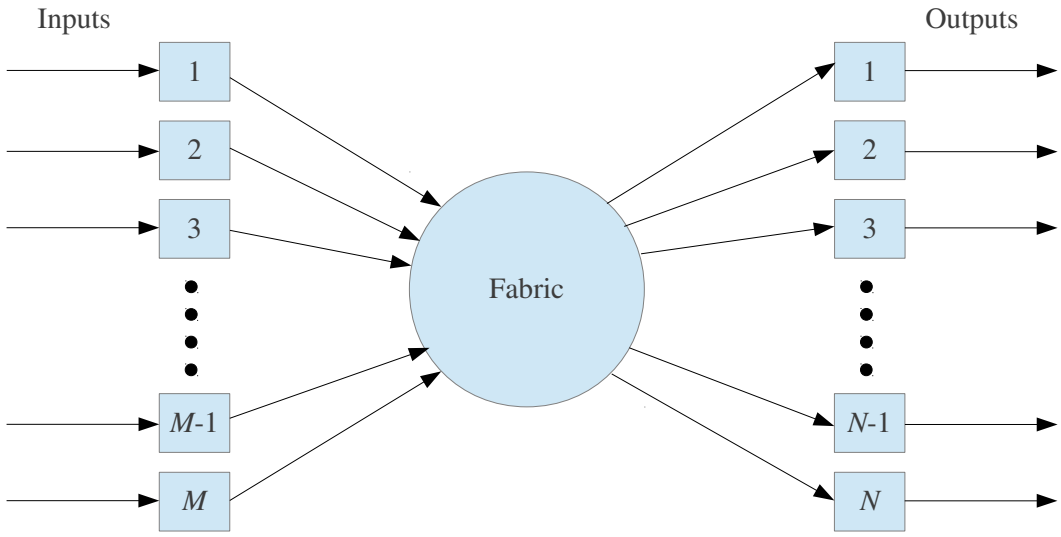

Figure 1.1: Basic switch configuration.

Performance metrics of interest for such a system include its *throughput, i.e.,* how many jobs/requests can be successfully transferred between inputs and outputs, and *latency, i.e.,* how it takes for such a transfer to take place. These depend in turn on factors such as

1. The *arrival process* of new work/requests on the inputs,
2. The characteristics of new work in terms of fabric processing/transfer time, a.k.a. *service time process*,
3. The fabric access control process or switch arbitration mechanism,
4. The switch processing/transfer capabilities, *i.e.,* its ability to transfer units of work from inputs to outputs.

Our goal is to formulate models to characterize (estimate) the performance of different types of switch fabric under different operating conditions.

1.1 The Basic Crossbar

In this base configuration, the fabric is a simple crossbar, *i.e.,* it allows each input (output) to communicate with a single output (input) at the time, so that there can be at most $\min\{M, N\}$ simultaneous communications through the fabric at any one time. The crossbar is further assumed to operate in a time-slotted manner, *i.e.,* during every clock cycle the crossbar can transfer one unit of work from each input to each output. In other words, at most one unit of work can be removed from an input during one clock cycle, and conversely each output can receive at most one unit of work from any given input during one clock cycle. The model is a reasonable representation of memory accesses in a multi-processor system consisting of M processors and N memory units connected through a crossbar, where the clock cycle corresponds to a single read/write (R/W) operation from a processor to a memory unit. In such a setting, each new arrival (a R/W request from a processor) brings one unit of work.

For such a system its throughput[1], is the expected number of R/W operations from processors to memory units that can be successfully performed during one clock cycle. This depends on the arrival patterns of R/W requests coming from each processor, as well as the set of memory units they target. For example and assuming for simplicity that $M = N$, if processor i only issues R/W requests for memory unit i, then the fabric is trivially seen to be able to sustain a throughput of 100%, *i.e.,* it can accommodate a successful R/W request from each processor in every clock cycle, so that all inputs and outputs are busy at all times. Conversely, if all processors systematically generate R/W requests to the same memory unit, say, j, then the fabric throughput is at most $1/N$. Intermediate values are to be expected for other distributions of R/W requests from processors to memory units.

For the sake of anchoring the discussion, consider the specific scenario where all M processors are equally likely to generate a R/W request to any of the N memory units. In other words, during a clock cycle, processor $i, 1 \leq i \leq M$, generates a R/W request to memory unit $j, 1 \leq j \leq N$, with probability p/N, where $p, 0 \leq p \leq 1$ is the probability that a processor has a (new) R/W request during a given clock cycle, *i.e.,* the average number of requests generated per clock cycle by each processor. To simplify the analysis we (initially) assume that R/W requests that cannot be accommodated during one cycle (when more than one processor wants to access a given memory unit during that cycle), are discarded *and* do not affect the request generation statistics in subsequent cycles. This *memorylessness* assumption is clearly inaccurate, but facilitates the analysis and will enable us to obtain an initial approximation for the system's throughput.

Assume that in a given cycle, $m, 0 \leq m \leq M$, R/W requests originate from the processors. The system's throughput $\Theta^m_{(M,N)}$ given m R/W requests depends on the number of distinct memory units they target. Let δ^m_j be the indicator function that takes value 1 if memory unit $j, 1 \leq j \leq N$ is requested by at least one of the m memory requests that were generated, and 0 otherwise. This gives

$$\Theta^m_{(M,N)} = \sum_{j=1}^{N} \delta^m_j \tag{1.1}$$

There are N^m ways to map m requests to N units (each request can choose from among N units). Conversely, for any unit $j, 1 \leq j \leq N$, there are $(N-1)^m$ ways of generating m memory requests that *do not* select unit j. Hence, under the assumption that all memory units are equally likely to be selected, the probability that any unit $j, 1 \leq j \leq N$ is selected is simply $[N^m - (N-1)^m]/N^m$. Combining this with Eq. (1.1), the expected system throughput $E[\Theta^m_{(M,N)}]$ given that m requests were generated in a cycle is given by

$$E[\Theta^m_{(M,N)}] = N\frac{N^m - (N-1)^m}{N^m} = \left[1 - \left(\frac{N-1}{N}\right)^m\right] N\,,$$

where we have used the fact that the mean of a sum of random variables is the sum of the means, irrespective of whether the variables are dependent or independent (and the variables δ^m_j are clearly dependent).

[1] As an historical note, this system was first analyzed in [**?**].

Additionally, the probability $q_m(p)$ that the M processors generate m requests is given by

$$q_m(p) = \binom{M}{m} p^m (1-p)^{M-m} ,$$

So that the expected throughput $\Theta_{(M,N)}(p)$ of a system with M inputs (processors) and N outputs (memory units), when processors generate a new (random) R/W request in each clock cycle with probability p is equal to

$$\begin{aligned} \Theta_{(M,N)}(p) &= \sum_{m=0}^{M} q_m \overline{\theta}_m = \sum_{m=0}^{M} \binom{M}{m} p^m (1-p)^{M-m} N \left[1 - \left(\frac{N-1}{N} \right)^m \right] \\ &= N \left[1 - \sum_{m=0}^{M} \binom{M}{m} \left[p \left(1 - \frac{1}{N} \right) \right]^m (1-p)^{M-m} \right] \\ &= N \left[1 - \left(1 - \frac{p}{N} \right)^M \right] \end{aligned} \tag{1.2}$$

Note that from Eq. (1.2) we have the following relationships:

$$\begin{aligned} \lim_{M \to \infty} \Theta_{(M,N)}(p) &= N \\ \lim_{N \to \infty} \Theta_{(M,N)}(p) &= pM \end{aligned}$$

The first expression simply states that as the number of processors grows large (compared to the number of memory units), then unless $p = 0$, the odds that at least one processor wants to access each memory unit approaches 1. Hence, the throughput approaches N. Conversely, the second expressions states that as the number of memory units grows large (compared to the number of processors), then the odds of request conflicts go to 0, and the system throughput is simply equal to the expected number pM of (new) R/W requests originating from the M processors in each clock cycle.

N	2	3	4	5	6	7	8	9	10
$\frac{\Theta_{(N,N)}(1)}{N}$	0.75	0.704	0.684	0.672	0.665	0.660	0.656	0.654	0.651

Table 1.1: Maximum throughput of a symmetric system as a function of system size (N).

Eq. (1.2) also indicates that the throughput is maximized for $p = 1$, *i.e.,* the inputs are saturated with processors generating a new R/W request every clock cycle. Table 1.1 gives the normalized (per output) throughput value for a saturated $(p = 1)$ and symmetric $(M = N)$ system for increasing values of N in the range $N = 1$ to $N = 10$. We note that the system throughput decreases as N increases. In general, the asymptotic value of the normalized (*i.e.,* the per input/output) throughput when both M and N grow large while maintaining a fixed proportion, *i.e.,* $N = \alpha M$, is equal to

$$\lim_{M,N \to \infty} \frac{\Theta_{(M,N)}(p)}{N} = \left(1 - e^{-\alpha p} \right) . \tag{1.3}$$

This implies that when $\alpha = 1$ (*i.e.,* $M = N$) and $p = 1$ (a saturated scenario, where processors always seek to write to a memory unit) the normalized throughput is asymptotically $1 - 1/e \approx 0.63$. In other words, contentions between processors seeking to access memory units limits the feasible throughput of each memory unit to 63% of their R/W bandwidth. Note that under the assumption that requests that cannot be satisfied are dropped and forgotten, this result is actually independent of the switch itself, *i.e.,* it holds even if the switch was infinitely fast (totally transparent) and simply arises because of the random contention for memory units among processors.

The impact of the switch only manifests itself, when requests that cannot be immediately accommodated can be preserved/queued and reattempt. Queues can be located at either the input (holding requests from a processor, which could not be transferred through the switch) or the output (holding requests from multiple processors attempting to access the memory unit). The switch operation and performance affect where queueing takes place, and play a role in determining the system's throughput. The next sections offer some insight into those issues.

1.2 Output Queues and the Impact of Speedup

For simplicity, consider a system where $M = N$ and $p = 1$, *i.e.,* the previous scenario of a saturated system with as many processors as there are memory units. Under these assumptions, if the switch is "infinitely fast" (transparent), then competing requests (from multiple processors) for a memory unit will be delivered to the corresponding output and queued. Assuming queues are large enough to store as many R/W requests as necessary, then we can achieve a memory unit load of 100% under a uniform distribution of R/W requests among memory units (the queues absorb fluctuations in how requests are distributed across memory units). It is of interest to evaluate how close we can get to this result when the switch is not infinitely fast.

Consider for that purpose a scenario where the switch can transfer $k \geq 1$ R/W requests in a single processor/memory clock cycle, *i.e.,* the switch clock is k times faster than the processor and memory clocks. The factor k denotes the switch *speedup* compared to its inputs (processors) and outputs (memory units). When $k = 1$, the analysis is trivially equivalent to that of the previous section. The more interesting case is for $k > 1$, so that more than one R/W request can be delivered to a memory unit in one of its clock cycle. Excess requests are queued for subsequent access to the memory unit. When more than k processors simultaneously request the same memory unit, the additional requests are, as before, dropped and "forgotten."

For purpose of illustration[2], consider the case $k = 2$, *i.e.,* the switch can transfer up to 2 R/W requests from different processors to a given memory unit in a single (input/output) clock cycle. Reusing our earlier notation, let $\delta_j, 1 \leq j \leq N$ be an indicator function that takes values: 0 if there are no R/W request for memory unit j, 1 if there is one and only one R/W request for memory unit j, and 2 if there are two or more R/W requests for memory unit j. This captures the fact that the switch can now transfer up to two R/W requests to memory units per clock cycle (R/W requests above that limit are still discarded). Following a similar reasoning as in the previous section, we have

$$P(\delta_j = i) = \begin{cases} \left(\frac{N-1}{N}\right)^N & \text{for } i = 0 \\ \frac{N(N-1)^{N-1}}{N^N} = \left(\frac{N-1}{N}\right)^{N-1} & \text{for } i = 1 \\ 1 - \left(\frac{N-1}{N}\right)^{N-1} - \left(\frac{N-1}{N}\right)^N & \text{for } i = 2 \end{cases}$$

The normalized throughput or equivalently, the corresponding R/W load (arrival rate of R/W requests) on a given memory unit is then given by

$$\begin{aligned} \theta_N^{(2)} &= \sum_{i=0}^{2} iP(\delta_j = i) = \left(\frac{N-1}{N}\right)^{N-1} + 2 - 2\left(\frac{N-1}{N}\right)^{N-1} - 2\left(\frac{N-1}{N}\right)^N \\ &= 2 - 2\left(1 - \frac{1}{N}\right)^N - \left(1 - \frac{1}{N}\right)^{N-1} \end{aligned}$$

So that when N grows large, the asymptotic memory load is

$$\lim_{N\to\infty} \theta_N^{(2)} = 2 - \frac{3}{e} \approx 0.896 \tag{1.4}$$

[2]And simplicity, as the approach becomes quickly intractable for larger speedup factors.

In other words, although some requests are still lost due to input contentions, *i.e.,* when three or more processors have R/W requests for the same memory unit, a speedup of 2 now allows the system to reach approximately 90% of the maximum possible load of 100% feasible under a uniform distribution of memory accesses. This is to be compared with a throughput of only 63% in the absence of output queueing and/or speedup. Note that if in addition to queueing on the outputs queueing is also allowed on the inputs (R/W requests that cannot be transferred through the switch upon their arrival are allowed to wait and reattempt in a subsequent cycle), then it is possible to show that the input queues are stable as long the arrival rates on the inputs is less than the switch saturation throughput [**?**]. In the next section, we explored how the presence of input queues that hold requests that cannot be transferred through the switch affects its saturation throughput.

1.3 A Finer Grain Analysis

The results of the previous sections assumed a *memoryless* arrival process, where R/W requests that cannot be transferred through the switch are discarded without affecting future request arrivals. As mentioned earlier, this is clearly an approximation. In practice, R/W requests that are "blocked" are often queued and proceed to reattempt in the next clock cycle.

A natural question is whether and to what extent this affects the switch throughput, *i.e.,* because it changes arrivals statistics and introduces dependencies over time and between inputs. The analysis of such a system is complex, but it is possible to formulate a relatively simple model that captures its behavior at *saturation*, *i.e.,* when processors generate new R/W requests in every cycle. For simplicity, we assume again $M = N$ and a speedup factor of $k = 1$.

As before, each processor generates R/W requests aimed at randomly selected memory units. R/W requests from a processor join an (input) queue, with the first or head-of-line (HOL) request competing with requests from other processors for transfer through the switch. When multiple HOL requests from different processors compete for access to the same memory unit, one of them is randomly selected and delivered to the output. Other requests remain in their input queue and reattempt in subsequent cycles until they eventually succeed. As soon as a request is successfully transferred through the switch, its HOL position in the input queue is immediately filled by another request (either from the queue or freshly generated by the processor). The new HOL request is again destined for a randomly selected memory unit.

Because the HOL position of all input queues is assumed to remain occupied, the behavior of this saturated system can be represented using a *closed network* of N "virtual" HOL queues. Each virtual queue is associated with HOL requests destined for a given output. In every (switch) cycle, one request is removed from each non-empty HOL virtual queue. Those departing requests then re-enter one of the N virtual queues at random, *i.e.,* each queue is selected with probability $1/N$. An illustration of this representation is shown on the left-hand-side of Fig. 1.2 for a simple 2×2 switch. The right-hand-side of the figure shows the different states that the two virtual queues can be in, with state (i_1, i_2) identifying the number of requests in virtual queues 1 and 2, respectively, *i.e.,* the number of HOL request destined for outputs 1 and 2. The arrows associated with each state identify the state transition that can take place and their respective probabilities.

The switch (saturation) throughput can then be readily obtained once state probabilities have been computed, *i.e.,* the switch successfully transfers two R/W requests when in state $(1, 1)$ and one R/W request when in state $(0, 2)$ or $(2, 0)$. Given the small size of the state space, there are many ways to compute the state probabilities. By symmetry, we known that $P_{(2,0)} = P_{(0,2)}$. It is also immediate to see that $P_{(1,1)} = 2P_{(2,0)}$, which when combined with the fact that $P_{(1,1)} + P_{(2,0)} + P_{(0,2)} = 1$ gives $P_{(1,1)} = 0.5$ and $P_{(2,0)} = P_{(0,2)} = 0.25$. This then translates into an aggregate switch saturation throughput of 1.5, or output saturation throughput of 0.75. Note that this is identical to the saturation throughput value of Table 1.1 for $N = 2$ and without input queues. This is because in the case of a 2 switch, the presence of input queues does not change the statistics of HOL patterns. As we shall see, this does not hold for larger switches.

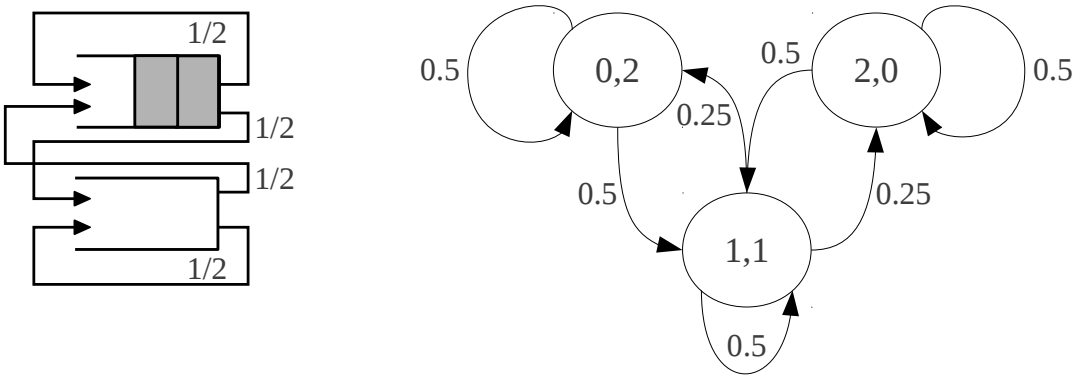

Figure 1.2: Saturated 2×2 model and associated (Markov) chain.

For a 2×2 switch, the analysis can also be extended to a switch with a speedup of $k = 2$. The only difference is that in this case, all three states are associated with a throughput of two R/W requests successfully transferred during a single (input/output) clock cycle (two switch clock cycles). This ensures that for all patterns of R/W requests to outputs, it is never possible for a backlog of requests to remain at the end of an input clock cycle. Hence, a throughput of 100% can be realized through a combination of speedup and input queues.

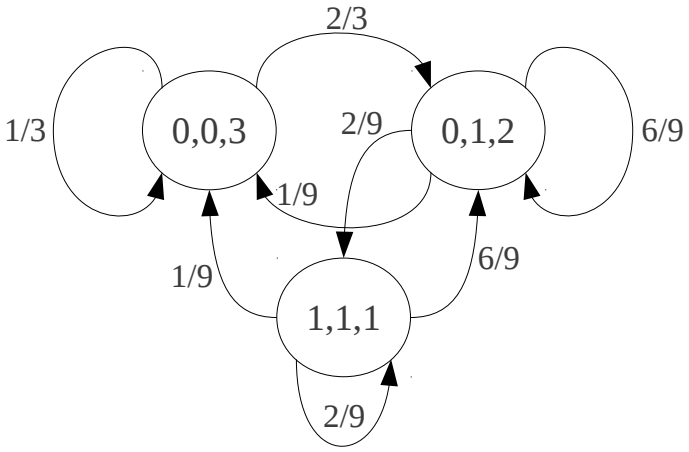

Figure 1.3: (Markov) chain of saturated 3×3 switch.

The direct approach of Fig. 1.2 can be extended to larger switches, but complexity increases rapidly as the number of distinct states for an $N \times N$ switch is equal[3] to $\binom{2N-1}{N}$. For $N = 3$ this comes to a total of 40 states, though this can be reduced down to just 3 through symmetry arguments, *i.e.,* ignore individual input/output identities and focus on distinct patterns of input to output assignments, namely, three requests to the same output, two requests to one output and one to another, and all three requests to distinct outputs. State transitions between such aggregate states are as shown in Fig. 1.3, whose probabilities can be computed and again used to obtain the switch throughput, *i.e.,* $P_{(0,0,3)} = 1/7, P_{(0,1,2)} = 2/3, P_{(1,1,1)} = 4/21$, so that the saturation output throughput is approximately 0.6825. Note that this is less than for the 2×2 switch, and also different from the corresponding entry in Table 1.1 for a switch without input queues. In general, the introduction of input queues results in a lower throughput, and this throughput value also decreases (to a limit) as N increases. Characterizing this asymptotic

[3]This is easily seen by casting the question as a standard "stars and bars" problem, *i.e.,* the number of N-tuple of non-negative integers that add up to N.

value, however, requires the use of a different approach.

1.3.1 Asymptotic Switch Throughput

In this section we consider the case of a symmetric switch, *i.e.,* $M = N$, that again operates at the same clock rate as its inputs (the processors), and where requests generated by the processors can queue while waiting to be transferred through the switch. Our goal is to derive the maximum possible input/output port throughput under a symmetric, uniform load assumption (each input targets each output with probability $1/N$) as $N \to \infty$. As in the previous section, processors generate enough requests to ensure that their input queues are never empty, *i.e.,* each time a request is transferred through the switch, a new one is available to take its place.

Under the assumption of a symmetric, uniform load, all outputs (and inputs) are equivalent and we can focus on HOL queues that will be the primary vehicle in our analysis. Recall from our earlier discussion that the HOL queues form a closed network through which a total of N requests circulate. In each clock cycle, every non-empty HOL queue transfers one request through the switch. Let T_n be the number of requests successfully transferred during the n^{th} clock cycle, $n \geq 0$. Using our closed network model of the HOL queues, those T_n requests correspond to the arrival of new requests to the HOL queues in clock cycle $(n+1)$. In other words, we have

$$T_n = N - \sum_{i=1}^{N} Q_n^{(i)} = \sum_{i=1}^{N} A_{n+1}^{(i)} \,,$$

where Q_n is the number of requests left behind at all HOL queues at the *end* of the n^{th} clock cycle, and $A_{n+1}^{(i)}$ corresponds to the number of requests for HOL queue $i, 1 \leq i \leq N$ at the *beginning* of clock cycle $(n+1)$. Note that the above expression implies

$$\theta_N = E[T_n/N] = 1 - E[Q_n^{(i)}] \,, \tag{1.5}$$

where we have used the fact that all outputs are statistically identical (symmetric, uniform load), and θ_N is the saturation throughput of an input (or output) or equivalently the probability of a new request arriving to an HOL queue. Our goal is to estimate $\lim_{N\to\infty} \theta_N = \theta_\infty$.

For that purpose, note that under the assumption of symmetric, uniform load, $A_{n+1}^{(i)}$ has a binomial distribution with a mean of T_n/N. In other words,

$$P\left(A_{n+1}^{(i)} = k\right) = \binom{T_n}{k} \left(\frac{1}{N}\right)^k \left(1 - \frac{1}{N}\right)^{T_n - k}$$

It is possible to show (see [**?**, Appendix 1] that as $N \to \infty$

$$\lim_{N\to\infty} P\left(A_{n+1}^{(i)} = k\right) = \lim_{N\to\infty} \binom{N\theta_N}{k} \left(\frac{1}{N}\right)^k \left(1 - \frac{1}{N}\right)^{N\theta_N - k}$$

In other words, the impact of T_n's deviations from its average value become negligible. Given this, the arrival process of new requests to any HOL queue can be shown to be given by a Poisson process of rate θ_∞, *i.e.,*

$$\lim_{N\to\infty} P\left(A_{n+1}^{(i)} = k\right) = \frac{e^{-\theta_\infty} \theta_\infty^k}{k!} \tag{1.6}$$

Given this arrival process and wlog, we now focus on the evolution of a given HOL queue, say, queue i, as $N \to \infty$, which is captured in the following expression

$$\begin{aligned} Q_{n+1}^{(i)} &= \max(0, Q_n^{(i)} - 1 + A_{n+1}^{(i)}) \\ &= Q_n^{(i)} + A_{n+1}^{(i)} - I\left(Q_n^{(i)} + A_{n+1}^{(i)}\right) \,, \end{aligned} \tag{1.7}$$

where $I(.)$ is the indicator function that takes value 1 when its argument is positive, and 0 otherwise.

In order to solve the above expression, we resort to moment generating functions ($z-$transform). Specifically, denote as $Q_n(z)$ and $A_{n+1}(z)$ the z-transforms of $Q_n^{(i)}$ and $A_{n+1}^{(i)}$, respectively, where for notational simplicity we are omitting the HOL queue index i. To obtain $Q_n(z)$, we write

$$\begin{aligned} Q_{n+1}(z) &= E\left[z^{Q_{n+1}^{(i)}}\right] = \sum_{k=0}^{\infty} P\left(Q_{n+1}^{(i)} = k\right) z^k \\ &= E\left[z^{Q_n^{(i)} + A_{n+1}^{(i)} - I\left(Q_n^{(i)} + A_{n+1}^{(i)}\right)}\right] \\ &= p_0 + \sum_{k=1}^{\infty} P\left(Q_n^{(i)} + A_{n+1}^{(i)} = k\right) z^{k-1} \\ &= p_0 + \frac{1}{z}\left(Q_n(z) A_{n+1}(z) - p_0\right) , \end{aligned}$$

where $p_0 = P\left(Q_n^{(i)} + A_{n+1}^{(i)} = 0\right)$. Note that the last equality relies on the assumption that the number of arrivals to HOL queue i are independent from its current occupancy. This is obviously not true for finite values of N, but is consistent with Eq. (1.6) that holds as $N \to \infty$ and assumes a constant rate arrival process of intensity θ_∞. Using further the fact that in steady-state (as $n \to \infty$), neither queue occupancy nor the number of arrivals depend on the clock cycle under consideration (n or $n+1$), and in particular that $Q_n(z) = Q_{n+1}(z) = Q(z)$ and $A_{n+1}(z) = A(z)$, we obtain

$$Q(z) = \frac{p_0(1-z)}{A(z) - z} \tag{1.8}$$

Eq. (1.6) also yields

$$A(z) = e^{\theta_\infty (z-1)} ,$$

So that we only need to determine p_0 to fully characterize $Q(z)$. Taking expectations on both sides of Eq. (1.7) and using again the fact that in steady-state $E\left[Q_{n+1}^{(i)}\right] = E\left[Q_n^{(i)}\right]$, we get

$$E\left[I\left(Q_n^{(i)} + A_{n+1}^{(i)}\right)\right] = E\left[A_{n+1}^{(i)}\right] = \theta_\infty$$

Since $p_0 = P\left(Q_n^{(i)} + A_{n+1}^{(i)} = 0\right)$ and given the definition of the indicator function $I(.)$, we also have

$$E\left[I\left(Q_n^{(i)} + A_{n+1}^{(i)}\right)\right] = 1 - p_0 ,$$

so that

$$p_0 = 1 - \theta_\infty \tag{1.9}$$

Combining Eqs. (1.8) and (1.9) gives

$$Q(z) = \frac{(1-\theta_\infty)(1-z)}{e^{\theta_\infty (z-1)} - z} \tag{1.10}$$

Differentiating Eq. (1.10) and setting $z = 1$ gives[4]

$$E[Q] = \frac{\theta_\infty^2}{2(1-\theta_\infty)} ,$$

[4]Recall that $Q'(1) = \sum_{k=1}^{\infty} kP(Q^{(i)} = k) = E[Q]$.

which when combined with letting $N \to \infty$ in Eq. (1.5) finally allows us to obtain

$$\theta_\infty = 2 - \sqrt{2} \approx 0.586 \tag{1.11}$$

Note that when comparing the above result to the asymptotic value of Eq. (1.3), we see that the added memory in the arrival process associated with allowing requests to queue results in a decrease in switch throughput from 0.63 to 0.586. This is consistent with the relationship between the results of Table 1.1 and those of Section 1.3, which for finite N values showed lower saturation throughput with queueing than without queueing, *e.g.,* 0.6825 versus 0.704 for $N = 3$.

Having characterized the maximum throughput achievable when using a switch that operates at the same speed as inputs, we turn briefly to another metric of interest, namely, latency.

1.3.2 An Initial Look at Latency

Estimating latency is inherently much harder, if only because by its nature latency calls for keeping track of what happens over a period of time, *i.e.,* monitor a job's progress through the system. Some of the more advanced methods used to estimate throughput can be extended to also yield latency estimates, but unfortunately there are no "simple" methods available to easily quantify a switch latency.

It is, however, possible to estimates for the average latency *if* we can measure some other quantity that captures switch performance. Specifically, assume that at every clock cycle we simply record the number of R/W requests that are present in the input queue of a processor. This is reasonably easy to do, and from those quantities we can determine the *average* number of requests present in an input queue[5]. Lets denote this value as $\overline{Q}$. Conversely, let p denote the probability of a processor generating a (new) R/W request in a given clock cycle. It is then possible to show that the average latency $\overline{L}$ for delivering a R/W request through the switch is given by

$$\overline{L} = \frac{\overline{Q}}{p} \tag{1.12}$$

The is actually the consequence of a very general result called *Little's Law*, which we will formally establish later. Intuitively, Eq. (1.12) says that with $\overline{Q}$ requests on average in the queue, and since the queue's average clearing rate must equal the average queue filling rate p (conservation law), it take on average $\frac{\overline{Q}}{p}$ to clear the queue, which therefore represents the system's average latency.

[5]Note that this implicitly assumes a stable system where the size of the input queue does not grow unbounded.

Bibliography

[1] J. H. Patel, "Performance of processor-memory interconnections for multiprocessors," *IEEE Trans. Commun.*, vol. C-30, no. 10, October 1981.

[2] L. Jacob, "Performance analysis of scheduling strategies for switching and multiplexing of multiclas variable bit rate traffic in an ATM network," Ph.D. Thesis, Indian Institute of Technology, Bangalore, India, December 1992.

[3] M. Karol, M. G. Hluchyj, and S. P. Morgan, "Input versus output queueing on a space-division packet switch," *IEEE Trans. Commun.*, vol. C-35, no. 12, December 1987.