College of Computer Science
Northeastern University
COM 3510

Lecture 4
18 October 2001
Computer Networks: Theory, Modeling, and Analysis

Analysis of ARQ, framing, and introduction to queueing theory

Handouts: Lecture notes for Lecture 3. Sample solution to Problem Set 1.

Lecture Outline:

- Analysis of the efficiency of ARQ protocols.
- Framing.
- Introduction to Queueing Theory.
  - Delay models.
  - Little's Theorem.

1 Analysis of the efficiency of ARQ protocols

In this section, we will derive formulae for the efficiency of the stop and wait and go back n ARQ protocols. Our calculations will take into account a number of parameters that affect the efficiency. Consider the transmission of a long sequence of packets from station A to B; assume that B sends ack (or request) frames only. Let n_f, n_h, and n_a denote the length in bits of the data frame, the header in a frame, and the ack frame, respectively. Let t_{prop} denote the propagation time on the channel and let t_{proc} denote the time taken to process any single frame. Let R denote the transmission rate for the channel.

Stop and wait with no errors. We begin by considering the stop and wait protocol under the assumption that no packets are lost and where there are no transmission errors. Figure 1 presents the breakup of the time T_0 that it takes between the time that the sender begins transmitting a packet i and the time the sender begins transmitting packet $i+1$. We obtain:

$$T_0 = 2(t_{\text{proc}} + t_{\text{prop}}) + \left(\frac{n_f}{R} + \frac{n_a}{R}\right) \tag{1}$$

Since the number of bits of information transmitted during time T_0 equals $n_f - n_h$, we obtain that the effective throughput achieved is $(n_f - n_h)/T_0$. Dividing the effective throughput by R yields the transmission efficiency μ_0:

$$\mu_0 = \frac{1 - \frac{n_h}{n_f}}{1 + \frac{n_a}{n_f} + \frac{2(t_{\text{proc}} + t_{\text{prop}})R}{n_f}} \tag{2}$$

Go back n with no errors. Equation 2 identifies three sources of inefficiency for the stop and wait protocol. First is the term $1 - n_h/n_f$, which cannot be avoided as long as we need headers. Second is the term $(2(t_{\text{proc}} + t_{\text{prop}})R)/n_f$ which corresponds to the time the sender waits while the processing and propagation of the packets is going on. The third is the term n_a/n_f, which

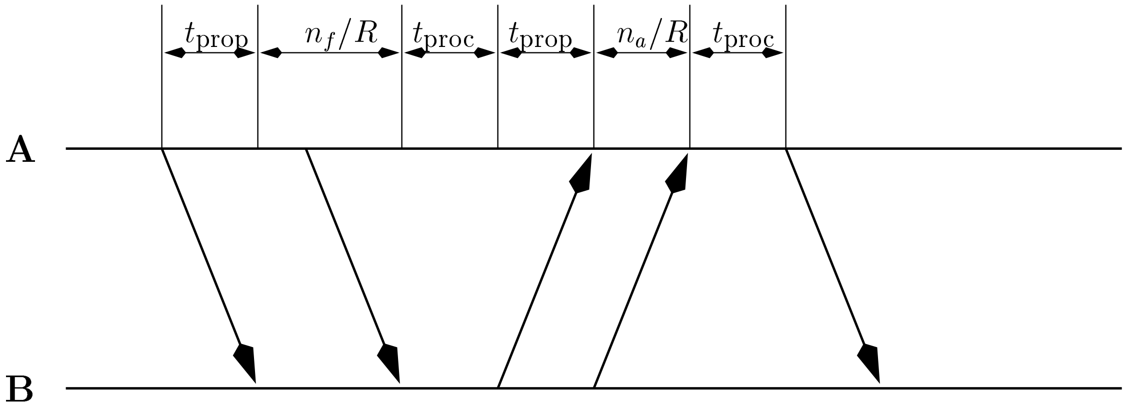

Figure 1: Transmission of a frame using stop and wait protocol without errors

corresponds to the time the sender waits during the transmission of the ack frame. Both the second and the third sources of inefficiency can be alleviated if the sender continues to send frames during those periods. This is what the go back n protocol is designed for.

If the window size n in go back n is large enough that the channel is kept busy while acknowledgments are returned, then the efficiency μ_0' of the protocol equals

$$1 - \frac{n_h}{n_f}.$$

Efficiency in the presence of errors. We now analyze the efficiency of stop and wait and go back n in the presence of errors. (We assume, however, that no packets are lost.) Furthermore, we disregard timeouts in our analysis. Let P_f denote the probability that an error occurs in a data frame. If p is the bit error rate, then we have $P_f = 1 - (1-p)^{n_f}$. For $pn_f \ll 1$, we have $P_f \approx pn_f$. In our analysis, we are going to assume that there are no errors in the acknowledgments. This is a reasonable assumption if the acks are much smaller than the data frame, since then the ack error probability would be small. In any case, the analysis that follows can be modified to accomodate ack errors.

The stop and wait protocol continues to transmit a particular data frame until it is acknowledged by the receiver. If a frame incurs an error i times, it is transmitted $i+1$ times. The probability of the frame incurring an error i times and then being successfully delivered is $P_f^i(1-P_f)$. It takes T time units (Equation 1) between the time that the sender sends out a frame and the time that it has processed the following ack packet from the receiver. Thus, the expected time T between the transmission of two frames is

$$\sum_{i \ge 0} i P_f^i (1-P_f) T_0.$$

It can be shown using standard calculus that $\sum_{i \ge 0} i P_f^i$ equals $1/(1-P_f)^2$. Therefore, T equals $(1-P_f)T_0$. This yields the efficiency μ to equal $\mu_0(1-P_f)$.

In the case of go back n, when a frame is retransmitted, the entire window of n frames starting from the smallest-numbered frame not yet acknowledged is retransmitted. Thus, if T_0' is the average time between frames in go back n when there is no error, the expected time T' between frames in the presence of frame error rate P_f is

$$\sum_{i \ge 0} (in+1) P_f^i (1-P_f) T_0'.$$

Algebraic simplification yields T' to be

$$\frac{(1+(n-1)P_f)T_0'}{1-P_f}.$$

Thus the efficiency μ' of go back n equals $(n_f - n_h)R/T'$. On substituting the value of T' and noting that $T_0' = n_f/R$, we get

$$\mu' = (1-P_f)\frac{1-\frac{n_h}{n_f}}{1+(n-1)P_f}.$$

We note that larger the window size smaller the efficiency. Therefore, the window size must be set to the smallest value that ensures that the channel is kept busy while acknowledgments are returned.

2 Framing

Framing is the task of encapsulating packets such that they are separated from one another and from the idle period of a data link transmission. We consider three types of framing techniques: *byte-oriented*, *bit-oriented*, and *length-based.*

2.1 Byte-oriented framing

In byte-oriented framing, a frame is presented as a collection of bytes or characters (typically, ASCII characters). Communication control characters are added at the start and end of the packet and possibly within the body of the packet, as we will see shortly. A canonical frame structure used in byte-oriented framing protocols is illustrated in Figure 2. This structure is followed (at a high-level) in the BISYNC and ARPANET protocols.

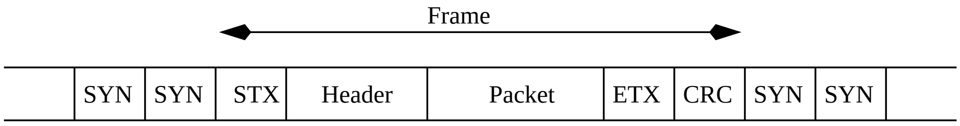

Figure 2: Framing structure in byte-oriented framing

A string of SYN characters forms the idle fill. The STX and ETX characters indicate the beginning and the end of a frame. Since the body of the packet, the header, and the CRC bits are arbitrary strings of bits, they might, through chance, contain a communication control character. Such occurrences in the header and the CRC pose no serious problem for the receiver since the header and the CRC bits are of fixed size and appear immediately after an STX and ETX character, respectively. Occurrences of control characters within the body of the packet, however, pose problems. Byte-oriented protocols insert a special control character called DLE (data link escape) before the communication control characters STX and DLE. They are not inserted before the possible appearances of these characters in the packet body. A problem caused by the DLE character is: how do we distinguish between the DLE control character and the appearance of the character within the body? The answer is to insert an additional DLE before each appearance of DLE in the packet body. The receiving DLC then strips off the first DLE. The next character indicates whether the following characters are part of the body or are control characters. Thus, for example, DLE ETX is interpreted as the end of the body, while DLE DLE ETX is interpreted as the appearance of the characters DLE ETX within the body.

In the above approach, we insert a DLE before the control characters. An alternative approach is to insert a DLE before every occurrence of STX, ETX, or DLE in the text portion of the frame. In this case, the receiver strips off every DLE and assumes that the following character is the part of the message. If an STX or ETX occur in isolation, then they are assumed to be control characters. One problem with this approach, in comparison with the first, is that if the number of occurences of the control characters is large, then a large number of DLE characters need to be introduced, thus causing a substantial increase in the size of the frame.

2.2 Bit-oriented framing

Bit-oriented framing protocols use a special flag to indicate the start and end of a frame, as well as idle fill. Unlike byte-oriented framing, the packet body and the flag can be of any length and, thus, need not be separated at byte boundaries. A commonly used protocol that employs bit-oriented framing is the High-level Data Link Control protocol (HDLC).

The basic control character in bit-oriented framing is a flag or a *sentinel*, which is the string $01^6 0 = 01111110$ in practice. The frames are separated from one another by sequences of sentinels. As in byte-oriented framing, we need to address the problem of the occurrence of the pattern $01^6 0$ within the body of the packet. This problem is solved using *bit stuffing*. After every string of 5 consecutive 1s, a 0 is inserted (stuffed). When the receiver reads in 5 consecutive 1s, it checks the next bit. If the next bit is a 0, then this indicates the body of the packet; if it is a 1, the receiver checks the following bit, which if 0 indicates a sentinel, and otherwise indicates an error. Bit stuffing is illustrated in Figure 3.

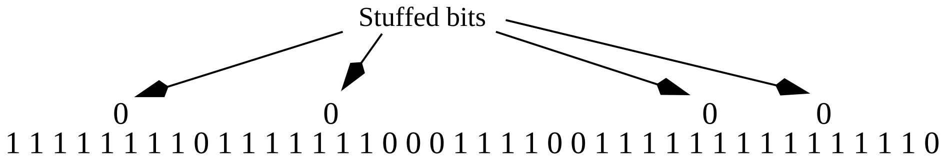

Figure 3: Bit stuffing

Longer the length of the sentinel, larger is the overhead at the start of the frame, but smaller is the expected number of bits stuffed in the body of the packet. Section 2.5.2 of Bertsekas and Gallager analyzes this tradeoff.

2.3 Length-based framing

An alternative to adding the start of text and end of text characters or special flags at the start and end of a frame is to add the length of the frame at the start. The DECNET protocols uses this technique.

2.4 Errors and framing

When we studied error-detection techniques, we noted that errors in the packet body or the CRC code bits are detected by the receiver with high probability, when the code bits are checked against the body. Framing adds special communication control characters or sentinels or length information. We now consider the problems that arise when errors corrupt the framing information.

If bits in the control characters or flags are corrupted, then typically the end of a frame will not be detected, and the CRC of the following frame will be assumed to be the CRC for the two frames put together. With high probability ($1 - 2^{-L}$ for an L-bit CRC code assuming random errors), a mismatch will be detected and the frame(s) will be assumed lost. If errors cause a flag (such as 01^60) to appear within the body of a frame, then the receiver will perceive the end of a frame earlier than later. Consequently, the bits that are perceived by the receiver as CRC bits will essentially be a random string, thus yielding a probability of 2^{-L} of failing to detect this error.

If a length field in the header provides framing, an error in this length will cause the receiver to look for the CRC in the wrong place. With probability $1 - 2^{-L}$ (for an L-bit CRC string), the error will be detected. Once the error is detected, however, the receiver will not know where to look for the next frame. This is why length-based framing protocols add a synchronization string at the beginning of a frame on retransmission.

3 Delay Models

An important performance measure in a data network is the average delay incurred by a packet. Early on in the course, we saw that the delay of a packet equals the sum of the propagation delay, the transmission delay, and the queueing delay. The propagation delay is a function of the physical distance and the speed of light in the medium. The transmission delay is a function of the length of the packet and the bandwidth of the link(s). Finally, the queueing delay is the waiting time of the packet and depends on a variety of parameters, including the service discipline, the arrival process, the bandwidth of the link(s), buffer sizes, etc. Another component of delay that is usually considered is the *processing delay*, which is the time spent processing a packet before transmission on a link.

Our next one and a half lectures will cover models and quantitative analyses for studying delays in data networks, primarily focusing on the transmission and queueing delays. For this purpose, we will adopt a *queueing-theoretic* framework. A *queueing system* consists of a server (or a collection of servers) that serves customers (or users or clients or packets), which arrive on-line. A customer that arrives may have to wait in a queue before getting service. We will assume throughout that the queues may be unbounded. We begin by stating and proving a fundamental result due to J. Little (1961), which is applicable to nearly all queueing systems of interest.

3.1 Little's Theorem

For a given delay model or a queueing system, we will be interested in measuring two quantities:

- T, the average delay incurred by a customer.
- N, the average number of customers in the system at any time.

These quantities, of course, depend on the properties of the underlying system. In particular, they depend on the process describing the arrival of customers and the distribution of the service times required by the customers. Little's Theorem provides a fundamental relationship between N, T, and the rate λ of the arrival process. The beauty of the theorem lies in its generality; it makes virtually no assumptions about the nature of the arrival process or of the service time distribution. Little's Theorem states that

$$N = \lambda T.$$

We now proceed to define the quantities N, T, and λ more formally and derive Little's formula. Let $N(t)$ denote the number of customers in the system at time t, $\alpha(t)$ denote the number of arrivals in the interval $[0,t]$, and T_i denote the time spent by the ith customer. Let N_t denote the time average of $N(\cdot)$ up to time t; that is,

$$N_t = \frac{1}{t}\int_0^t N(\tau)d\tau.$$

Naturally, N_t changes as t increases. For many systems of interest, N_t tends to a steady-state N as t increases. So we define

$$N = \lim_{t\to\infty} N_t.$$

Similarly, we define the time averaged arrival rate λ_t and the steady-state average rate λ as

$$\lambda_t = \frac{\alpha(t)}{t} \text{ and } \lambda = \lim_{t\to\infty} \lambda_t.$$

Finally, the time averaged delay T_t and the steady-state average delay are defined as

$$T_t = \frac{\sum_{i=0}^{\alpha(t)} T_i}{\alpha(t)} \text{ and } T = \lim_{t\to\infty} T_t.$$

We now prove Little's Theorem. The proof is best given by a figure that depicts the arrival and departure of customers in the system. The shaded area in Figure 4 between the graphs $\alpha(\tau)$ (the arrival function) and $\beta(t)$ (the departure function) is the integration of $N(\tau)$. This area can also be bounded in terms of the terms of the delays incurred by the customers. An upper bound can be given in terms of the delays incurred by the customers that have arrived. And a lower bound can be given in terms of the customers that have departed. We obtain the following inequality.

$$\sum_{i=1}^{\beta(t)} T_i \leq \int_0^t N(\tau)d\tau \leq \sum_{i=1}^{\alpha(t)} T_i.$$

We consider the upper bound first. Dividing both sides by t, we get

$$\frac{1}{t}\int_0^t N(\tau)d\tau \leq \frac{\alpha(t)}{t}\frac{\sum_{i=1}^{\alpha(t)} T_i}{\alpha(t)},$$

or equivalently,

$$N_t \leq \lambda_t T_t.$$

Taking the limit as t tends to ∞,

$$N \leq \lambda T.$$

We now consider the lower bound. Again dividing both sides by t, we get

$$\frac{1}{t}\int_0^t N(\tau)d\tau \geq \frac{\beta(t)}{t}\frac{\sum_{i=1}^{\beta(t)} T_i}{\beta(t)},$$

Taking the limit as t tends to ∞ and noting that the steady-state departure rate $\lim_{t\to\infty}\beta(t)/t$ equals the arrival rate λ, we obtain

$$N \geq \lambda T.$$

This completes the proof of Little's Theorem.

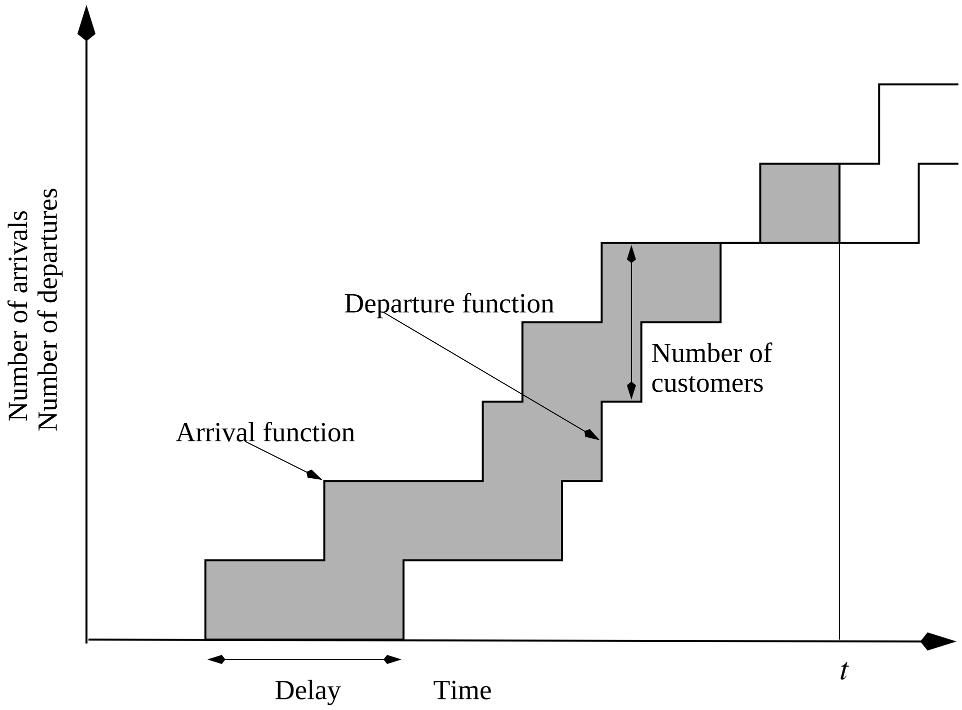

Figure 4: Proof of Little's Theorem

3.2 Applications of Little's Theorem

We consider two example applications of Little's Theorem.

1. **Flow control:** Consider a sliding window transport layer protocol for flow control such as go back n or selective repeat ARQ. Let W denote the size of the window, λ be the arrival rate of packets, and T be the average packet delay incurred. Since the number of packets in flight (or in the system) is always at most W, Little's Theorem says that $W \geq \lambda T$. The implications of the preceding inequality are the following. For a given window size, if T increases, then the arrival rate has to be decreased. Alternatively, if T increases, then for the same arrival rate, the window size has to be increased.

2. **Throughput in a time-sharing system:** Suppose we have N terminals sharing a supercomputer. Assume that whenever a user logs on to a terminal, s/he spends some time reflecting on the nature of job that the user wants to submit to the supercomputer. Let the average reflection time be R, and the average processing time of a job submitted to the processor be P. In this example, we consider the maximum throughput attainable in the system. For this purpose, we will make the assumption that as soon as a user departs, another occupies the freed terminal. Thus, the number of users in the system is always N.

   Let T denote the average delay for a user. Then T equals $R+D$, where D is the average time spent waiting once the job is submitted to the CPU. Since there are at most N jobs in the system at any time, we have $P \leq D \leq NP$. Therefore, $R+P \leq T \leq R+NP$. By Little's Theorem, if λ is the arrival rate of users, then $N = \lambda T$. We thus have the following bounds on λ.

$$\frac{N}{R+NP} \leq \lambda \leq \frac{N}{R+P}.$$

We can place another upper bound on λ by noticing that the supercomputer itself represents a queueing system, in which the number of jobs being processed at any time is at most 1. And the arrival rate into the system can be no larger than the departure rate out of the computer, which in turn equals the arrival rate into the computer in steady state. Therefore, we have, by another invocation of Little's Theorem, that $\lambda P \leq 1$, yielding $\lambda \leq 1/P$. Combining the two upper bounds together, we get:

$$\begin{aligned} \lambda &\leq \min\left\{\frac{1}{P}, \frac{N}{R+P}\right\} \\ &= \frac{1}{P}\min\left\{1, \frac{N}{1+R/P}\right\}. \end{aligned}$$

The preceding upper bound on λ provides the following insight into the time-sharing system. If $N < 1+R/P$, then the throughput bottleneck is the number of terminals. If $N > 1+R/P$, the throughput bottleneck is the processing power of the computer.