Packet Arrival Rate

Related terms:

Active Queue Management, Congestion Control, Average Packet Size, Congestion Level

View all Topics

Stream Sessions: Stochastic Analysis

Anurag Kumar, ... Joy Kuri, in Communication Networking, 2004

5.5.2 Invariance of Mean System Time

In spite of its simplicity, Little's theorem can be a powerful analytical tool. We now demonstrate its use for establishing a basic result for a class of multiplexing problems. It will be useful to review the material in Section 4.1.1.

Packets enter the multiplexer at arrival instants ak, $k \geq 1$; the packet arriving at ak has the length Lk. The packet arrival rate is (packets per second). We make the following assumptions about the system:

- The packet lengths (Lk, $k \geq 1$) are independently and identically distributed, with some distribution $L(l)$; that is, Pr (packet length $\leq l$) = $L(l)$.
- After the completion of the transmission of a packet, the scheduler chooses the next packet for transmission without any regard to the service times of the waiting packets. Thus, for example, a scheduling policy that transmits shorter packets first is not under consideration in this discussion.
- When the transmission of a packet is initiated, the link is dedicated to the packet and the packet is transmitted completely.

The second and third assumptions ensure that when the link completes the transmission of a packet and the scheduler looks for a new packet to transmit, the service times of the waiting packets are as if they were "freshly" sampled (in an independent and identically distributed (i.i.d.) fashion) from the distribution $L(l)$.

Let *NFCFS*(*t*), *t* ≥ 0, denote the number of packets in the system when the scheduling policy is FCFS. *NFCFS*(*t*) is a random process; observe that it is completely determined by the random sequences *ak* and *Lk*. Let *P* denote another policy that satisfies the preceding assumptions, and let *NP*(*t*) denote the corresponding packet queue length process. Let us now modify the system under policy *P* in the following way. The packet arrival instants are unchanged, but the kth packet *to be transmitted* is assigned the length *Lk when it is scheduled for transmission*. Note that for a non-FIFO policy the *k*th packet to be transmitted need not be the *k*th packet to arrive. Denote the multiplexer queue length (for this alternative way of sampling packet lengths) by Ñ*P*(*t*). Because of the assumptions about the policies, a little thought shows that *NP*(*t*) and Ñs*P*(*t*) are statistically indistinguishable; a probability question about either of these processes yields the same answer. By the preceding construction, it can also be easily seen that N˜*P*(*t*) = *NFCFS*(*t*) for all *t*. Further assume that with the scheduling policies under consideration, the system is "stable" (formally discussed later in this chapter; see Section 5.6.2), and hence the following time averages exist with probability 1: *NFCFS* = Ñ*P* = *NP*, *WFCFS* and *WP*. By Little's theorem we have

Let *NFCFS*(*t*), *t* ≥ 0, denote the number of packets in the system when the scheduling policy is FCFS. *NFCFS*(*t*) is a random process; observe that it is completely determined by the random sequences *ak* and *Lk*. Let *P* denote another policy that satisfies the preceding assumptions, and let *NP*(*t*) denote the corresponding packet queue length process. Let us now modify the system under policy *P* in the following way. The packet arrival instants are unchanged, but the kth packet *to be transmitted* is assigned the length *Lk when it is scheduled for transmission*. Note that for a non-FIFO policy the *k*th packet to be transmitted need not be the *k*th packet to arrive. Denote the multiplexer queue length (for this alternative way of sampling packet lengths) by Ñ*P*(*t*). Because of the assumptions about the policies, a little thought shows that *NP*(*t*) and Ñs*P*(*t*) are statistically indistinguishable; a probability question about either of these processes yields the same answer. By the preceding construction, it can also be easily seen that N˜*P*(*t*) = *NFCFS*(*t*) for all *t*. Further assume that with the scheduling policies under consideration, the system is "stable" (formally discussed later in this chapter; see Section 5.6.2), and hence the following time averages exist with probability 1: *NFCFS* = Ñ*P* = *NP*, *WFCFS* and *WP*. By Little's theorem we have

$NFCFS = \lambda WFCFS NP = \lambda WP$

We conclude that *WFCFS* = *WP*; that is, the mean time that a packet spends in the system is invariant with the choice of policy in this class. It can be shown that the higher moments of the system time do depend on the policy and that the FCFS policy is optimal in a certain sense.

> Read full chapter

Performance and Architectural Issues

Anurag Kumar, ... Joy Kuri, in Communication Networking, 2004

9.1.1 Packet Switches

First consider the data plane functions in a packet switch. These functions are performed on every packet and are thus high-volume, fast-timescale functions.

We immediately see that the *packet-processing capacity* of the switch is an important performance measure. The packet-processing capacity should minimize packet-processing delays even when the packet arrival rate is high. The packet arrival rate to the switch depends on the data rates on the links that the switch interconnects. Furthermore, if the links allow variable-length packets, the packet length distribution also influences the packet arrival rate. To appreciate the dependence between the packet length distribution and the packet arrival rate, it is instructive to see the distribution of packet lengths in a real packet-switched network. Figure 9.1 shows the packet length distributions from the packet traces collected at NASA Ames Internet Exchange (AIX) over one week in May 1999 and is representative of the packet length distribution seen on the Internet. We see from the figure that whereas nearly 50% of the bytes are from 1500-byte packets (read the dashed curve), nearly 50% of the packets are small, 40-byte packets (read the solid curve). This means that although small packets do not contribute much to the utilization, they consume significant packet processing power. The packet-processing rate of a switch must be dimensioned with this in view.

distribution of packet lengths in a real packet-switched network. Figure 9.1 shows the packet length distributions from the packet traces collected at NASA Ames Internet Exchange (AIX) over one week in May 1999 and is representative of the packet length distribution seen on the Internet. We see from the figure that whereas nearly 50% of the bytes are from 1500-byte packets (read the dashed curve), nearly 50% of the packets are small, 40-byte packets (read the solid curve). This means that although small packets do not contribute much to the utilization, they consume significant packet processing power. The packet-processing rate of a switch must be dimensioned with this in view.

Figure 9.1. Example cumulative packet length distribution collected at the NASA Ames Internet Exchange. Adapted from the graphs in http://www.caida.org/analysis/AIX/plen_hist/as seen on 03 Aug 2003. The dashed plot is the fraction of packets with length less than x bytes, and the solid plot is the fraction of bytes in packets with size less than x bytes.

Exercise 9.1

Derive the solid plot from the dashed plot in Figure 9.1; that is, given the distribution function of the packet lengths, obtain the fraction of bytes contributed by packets of a given size. Observe the *length biasing* (see Section D.1.4) that occurs: If a random packet is observed on the link, it is more likely that a longer packet is observed than a shorter packet.

Figure 9.2 summarizes the discussion from Chapter 2 on the functions of a packet switch. To keep the discussion general, assume a QoS-enabled switch that can provide delay and loss rate guarantees to the packets. We will discuss the performance and design issues associated with each of the blocks shown in Figure 9.2. The receiving and transmission of bits and the extraction of packets are what can be called “physical layer functions,” and we do not discuss these functions in this book.

and design issues associated with each of the blocks shown in Figure 9.2. The receiving and transmission of bits and the extraction of packets are what can be called "physical layer functions," and we do not discuss these functions in this book.

Figure 9.2. Block diagram of a view of the passage of a packet through a switch.

Packet header processing typically has two important functions: *route lookup*, which determines the output link for the packet, and *packet classification*, which determines the specific service grade that is to be provided to the packet. Packet classification may be based on source and destination addresses of the packet and the application that generated it. In virtual-circuit networks, such as ATM networks, route lookup and packet classification are typically a table lookup and, with very high probability, can be completed with a small, constant delay.

In Internet-like datagram networks, route lookup and packet classification are more complex, and the time taken to perform these tasks could be variable, possibly even exceeding packet interarrival times. We discuss the algorithms and performance of route lookups in Chapter 12. The point to note now is that the variability in processing time can cause a packet queue to build up at the input, and a packet will experience variable delay before its service category is determined. Also, because the buffer space available for the packet queue is finite, some packets may be lost. In providing delay and loss guarantees to the packets, it is imperative that the header processing be completed with a constant latency after the arrival of the packet. Otherwise the switch might excessively delay packets with stricter guarantees and not even know it! Thus the variability of packet header processing time at the input becomes an important performance issue.

After the output link and the service category for the packet have been determined, the packet must be switched to a queue in the output link. There are three issues associated with the switching of packets to the output link: placement of a packet queue, scheduling of packets to be switched to the output queue, and the design of the fabric itself. Recall from Chapters 1 and 2 that output contention occurs when two or more packets arriving on different input links want to leave from the same output link at the same time. If the links are not time-slotted, a packet may arrive for an output while a packet transmission is in progress on that output. With time-slotted links, two packets may arrive in the same slot from different inputs and may want to be output on the same link. All but one of the contending packets must be queued, and a packet queue must be maintained. Thus, output contention necessitates queueing of the packets. If the packets queue at the inputs, then the switching capacity should be such that the queueing delay at the input is minimized. The queueing delay and the loss probabilities in the input queue or the output queue are

Figure 9.2. Block diagram of a view of the passage of a packet through a switch.

Packet header processing typically has two important functions: *route lookup*, which determines the output link for the packet, and *packet classification*, which determines the specific service grade that is to be provided to the packet. Packet classification may be based on source and destination addresses of the packet and the application that generated it. In virtual-circuit networks, such as ATM networks, route lookup and packet classification are typically a table lookup and, with very high probability, can be completed with a small, constant delay.

In Internet-like datagram networks, route lookup and packet classification are more complex, and the time taken to perform these tasks could be variable, possibly even exceeding packet interarrival times. We discuss the algorithms and performance of route lookups in Chapter 12. The point to note now is that the variability in processing time can cause a packet queue to build up at the input, and a packet will experience variable delay before its service category is determined. Also, because the buffer space available for the packet queue is finite, some packets may be lost. In providing delay and loss guarantees to the packets, it is imperative that the header processing be completed with a constant latency after the arrival of the packet. Otherwise the switch might excessively delay packets with stricter guarantees and not even know it! Thus the variability of packet header processing time at the input becomes an important performance issue.

After the output link and the service category for the packet have been determined, the packet must be switched to a queue in the output link. There are three issues associated with the switching of packets to the output link: placement of a packet queue, scheduling of packets to be switched to the output queue, and the design of the fabric itself. Recall from Chapters 1 and 2 that output contention occurs when two or more packets arriving on different input links want to leave from the same output link at the same time. If the links are not time-slotted, a packet may arrive for an output while a packet transmission is in progress on that output. With time-slotted links, two packets may arrive in the same slot from different inputs and may want to be output on the same link. All but one of the contending packets must be queued, and a packet queue must be maintained. Thus, output contention necessitates queueing of the packets. If the packets queue at the inputs, then the switching capacity should be such that the queueing delay at the input is minimized. The queueing delay and the loss probabilities in the input queue or the output queue are

important performance measures for the switch and are a function of the switching capacity, the packet buffer sizes, and the packet arrival process. If packets are queued at the input to the switching fabric, then a scheduler must be used to decide when to offer a packet to the switch fabric.

important performance measures for the switch and are a function of the switching capacity, the packet buffer sizes, and the packet arrival process. If packets are queued at the input to the switching fabric, then a scheduler must be used to decide when to offer a packet to the switch fabric.

Now consider the design of the switch fabric. In a slot, the switch fabric may or may not be able to switch all permutations of destination requests. Consider an $N \times N$ switch, with d = [d_1, ..., d_N] representing the destination vector of the packets at the N inputs. In a *nonblocking* switch, if the **d** is *any* permutation (i.e., **d** is *any* permutation of [1, . . ., N]), then the packet at input i can be switched to output d_i. If the switch is blocking, the blocking probability must be evaluated.

Remark: We have seen in Chapters 4 and 5 that QoS for stream sessions can be guaranteed by appropriate packet scheduling on the output link. Thus, if a switch were to provide QoS to the packets in terms of delay and loss guarantees, then the packets should reach the output queue with constant latency (or with a tightly upper-bounded delay) after their arrival at the switch. Alternatively, we could say that the packet should be available for transmission or scheduling on the output link with an approximately constant latency. This is the primary aim of packet switch design.

The control plane functions in a packet switch are slow-timescale functions and include functions such as executing the signaling protocols and evaluating the network state (e.g., network topology) from the signaling messages exchanged. The details of the control plane functions depend on the type of the network. An example of a control plane function is the execution of a routing protocol and the determination of the routing table. In executing a routing protocol, the switch exchanges network topology information with its neighbors, and the routing table is computed after all the information is collected from all the neighbors. How often the routing table can be computed depends on the size of the network and the processing capability of the control processor and is thus a measure of the ability of the switch to adapt to changing network conditions. Typically, the control plane functions are performed in software on general-purpose processors. The MIPS (million instructions per second) rating of the processor is a good measure of control plane performance of the switch.

> Read full chapter

Mesh Networks: Optimal Routing and Scheduling

Anurag Kumar, ... Joy Kuri, in Wireless Networking, 2008

Overview

Overview

In Section 8.1 we first describe the communication graph of a wireless network deployed in a given geographical area. Constraints on the simultaneous transmissions based on SINR, protocol-model, and the network graph are then described. In Section 8.2, for a given set of allowable link activation vectors, we obtain the network stability region, the set of end-to-end packet arrival rates for which the queues in all the network nodes will be stable. In Section 8.3, we consider the joint optimal routing of a set of end-to-end packet flows and the corresponding link scheduling. A static link schedule using graph coloring techniques is derived. In Section 8.4 we develop the important, queue-length based, backpressure algorithm for joint routing and transmission scheduling on the links. This algorithm is optimal in the sense that it can stabilize any stabilizable end-to-end arrival rate vector. The algorithm is a maximum weight scheduling algorithm and the stability proof makes use of stochastic Lyapunov functions. In Section 8.5 we consider end-to-end elastic traffic and, for a given set of users, we obtain the jointly optimal routing of the packet flows and the transmission schedule; in this case, a utility function on the allocated rate is defined for each user and the sum of the total utilities of all the users is maximized. Using convex programming and Lagrangian duality we obtain optimal joint packet flow rate allocation, routing, and link scheduling policies. In this section, we also consider optimal scheduling of one-hop packet flows in a slotted Aloha network. In the optimal algorithm the nodes update their transmission probabilities using local information to maximize the sum of link utility functions.

> Read full chapter

Operating Systems Overview

Peter Barry, Patrick Crowley, in Modern Embedded Computing, 2012

Polling Interface

In many cases, the network interface driver provides a mechanism to poll the network interface for received packets. The network stack may be set up to poll for network packets for two reasons. The first is to improve the overall performance of the device driver under varying packet arrival rates. When a slow rate of packets arrives at the network interface, it is best for the device driver to interrupt the processor when each packet arrives; in this case there is little delay in processing the packet that has arrived. If the network load is high and a burst of packets arrives with the minimum interpacket arrival time, it is often better to be interrupted on the arrival of the first packet and then to poll the interface a number of times before the packet related interrupts are re-enabled from the device. This has the effect of reducing the total number of interrupts processed for the bursty traffic. In some cases where the network traffic is at a sustained high rate, a scenario known as *live lock* can occur. Live lock in this case is when the CPU is spending all of the time servicing network receive interrupts but never has an opportunity to process the packets. The Linux stack implements a concept known as New API (NAPI) (http://www.linuxfoundation.org/collaborate/workgroups/networking/napi) to switch between interrupt driven operation and polled operation based on the network load. There are also interrupt moderation schemes developed for some network interfaces to help moderate the interrupt rate using inter packet timers and network traffic load.

of the first packet and then to poll the interface a number of times before the packet related interrupts are re-enabled from the device. This has the effect of reducing the total number of interrupts processed for the bursty traffic. In some cases where the network traffic is at a sustained high rate, a scenario known as *live lock* can occur. Live lock in this case is when the CPU is spending all of the time servicing network receive interrupts but never has an opportunity to process the packets. The Linux stack implements a concept known as New API (NAPI) (http://www.linuxfoundation.org/collaborate/workgroups/networking/napi) to switch between interrupt driven operation and polled operation based on the network load. There are also interrupt moderation schemes developed for some network interfaces to help moderate the interrupt rate using inter packet timers and network traffic load.

The other reason to provide a network polling API implementation is to allow the network interface to be used as a debug interface when debugging the operating system. For example, you can connect to the Linux target system and use the GDB debugger to debug the kernel from a host machine across an Ethernet interface that supports the polling Network interface. Network interfaces that only operate with interrupts cannot be used for this purpose, as interrupt processing is prevented when a breakpoint is hit in the kernel.

> Read full chapter

Multiple Access: Wireless Networks

Anurag Kumar, ... Joy Kuri, in Communication Networking, 2004

Instability of Aloha

In the throughput discussion of Aloha and S-Aloha, we overlook an important feature of random access MAC protocols: A packet that has suffered a collision stays in the network and makes retransmission attempts until it is successful. Such packets are called backlogs. Let *Bk* denote the backlog at the beginning of slot *k*. We assume that all fresh arrivals during a slot will attempt transmission at the beginning of the next slot. We can write the evolution equation for *Bk* as follows:

where *Ak*, is the number of arrivals, and *Dk* ∈ {0, 1} is the number of departures in slot *k*. If new packet arrivals form a Poisson process of rate and if the backlogs attempt retransmission independently in each slot with probability *r*, then {*Bk*} is a discrete time Markov chain. Consider the *drift* from state *n* denoted by *d*(*n*) and defined as

$d(n)$ is the average change in the backlog in one slot when the backlog is n. The backlog decreases by 1 if no new arrivals occur and if only one of the backlogs attempts transmission in the slot. The backlog increases by 1 if exactly one arrival occurs and if at least one of the backlogs attempts a retransmission. If the number of new arrivals is more than 1, the backlog increases by that amount. For all other combinations the backlog does not change. Thus we can write

$d(n)$ is the average change in the backlog in one slot when the backlog is n. The backlog decreases by 1 if no new arrivals occur and if only one of the backlogs attempts transmission in the slot. The backlog increases by 1 if exactly one arrival occurs and if at least one of the backlogs attempts a retransmission. If the number of new arrivals is more than 1, the backlog increases by that amount. For all other combinations the backlog does not change. Thus we can write

Using this and simplifying, we get

Clearly, as n becomes large, irrespective of the value of λ and r, the second term becomes very small and $d(n)$ is positive. Thus the mean drift for all large values of backlog (except finitely many values) is nonnegative. Because at most one packet can depart the system in each slot, we can apply Theorem D.10 to conclude that the backlog Markov chain is not positive recurrent. This means that if the backlog becomes large, the network has a tendency to increase the backlog rather than decrease it. This in turn implies that the Aloha protocol, if it is running for a long time, can develop a large backlog that may never be cleared.

The assumption of an infinite number of nodes is for analytical convenience. It is also a worst case analysis because with finite nodes, packets from the same node do not compete with each other, and in that case the performance can only improve. However, it can be shown that even when the number of nodes in the network is finite, the behavior is qualitatively similar to that we have derived; that is, if the backlog becomes large, the network has a tendency to operate with a large backlog for very long times. An obvious issue now is to design mechanisms to make the network stable. We do this by making the retransmission probabilities adaptive. To see how this can be done in the S-Aloha network, assume that all the nodes know the size of the backlog at the beginning of every slot and also know the stationary packet arrival rate. A packet is successfully transmitted in a slot either if exactly one new packet arrives and none of the backlogs attempt retransmission, or if no new packet arrives and exactly one of the backlogs attempts retransmission. Thus, Ps is given by

$$P_s = \lambda e^{-\lambda}(1-r)^n + e^{-\lambda} n r (1-r)^{n-1}$$

Given $Bk = n$, the r that will maximize the probability of success—say, $r(n)$—is given by

If an adaptive retransmission probability $r(n)$ is used instead of fixed r, the drift will become

In this case $d(n) \to \lambda - e^{-\lambda}$. Hence, using Theorem D.9 with $f(i) = i$, we conclude that for the Markov chain $\{Bk\}$ is positive recurrent; that is, the S-Aloha network is stable.

It is not practical for the nodes to know *Bk*, and a node should learn the network state from the events that it can observe. Let *Zk* be the event in slot k, with the possible events being idle (denoted by 0) if no transmission was attempted in it; success (denoted by 1) if exactly one transmission was attempted; and an error (denoted by *e*) if more than one transmission was attempted. Typically two kinds of event observations are used. For example, we could use

It is not practical for the nodes to know *Bk*, and a node should learn the network state from the events that it can observe. Let *Zk* be the event in slot k, with the possible events being idle (denoted by 0) if no transmission was attempted in it; success (denoted by 1) if exactly one transmission was attempted; and an error (denoted by *e*) if more than one transmission was attempted. Typically two kinds of event observations are used. For example, we could use

$$S_{k+1} = \max\{1, S_k + aI_{\{Z_k=0\}} + bI_{\{Z_k=1\}} + cI_{\{Z_k=e\}}\}$$

and transmit with probability in slot *k* if the node has either a fresh packet or a backlog. Note that a, b, and c are fixed constants. A possible choice for the parameters is $a = -1$, $b = 0$, and $c = 1$. Alternatively,

$$S_{k+1} = \max\{1, a(Z_k) \times S_k\}$$

where *a*(*Zk*) are predetermined constants for the network. It has been shown that *a*(*Zk*) exist to achieve the maximum possible throughput of .

Using the feedback from the network in adapting the retransmission times requires that all nodes be active all the time. Also, they all should see the same channel state at all times. This is clearly a very strong requirement, but it is avoidable. To make the protocol more robust, in many random access protocols nodes use their transmission attempt history to adapt their retransmission times. After a collision is detected by a transmitting node, the node stops the transmission and does not attempt another transmission for a backoff period of *x* units of time. Here *x* is a uniformly distributed random integer in the interval [0, *B* − 1]. *B* is updated by the node at every event (collision or success). A typical update equation has the form

(8.18)

where *a*, *b*, $B_{\min}$, and $B_{\max}$ are predefined.

> Read full chapter

Random Access and Wireless LANs

Anurag Kumar, ... Joy Kuri, in Wireless Networking, 2008

Stabilizing Aloha

An obvious issue now is to design mechanisms to make the network stable for some □ > 0 so that the network can support packet arrivals at that rate. This is done by making the retransmission probabilities adaptive. To see how this can be done, assume that all the nodes know the size of the backlog at the beginning of every slot and also the stationary packet arrival rate. A packet is successfully transmitted in a slot if either of the two conditions is satisfied: (1) Exactly one new packet arrives and none of the backlogs attempts a retransmission or (2) no new packet arrives and exactly one of the backlogs attempts a retransmission. Thus, the probability of a successful transmission when the backlog is n, $P_s(n)$, is given by

able. To make the protocol more robust, in many random access protocol standards, a node uses its own transmission attempt history to adapt the retransmission times. Usually, the history is reset after every successful transmission. A node will make the m-th transmission attempt after a backoff period of x_m units of time. Here x_m is a uniformly distributed random integer in the interval $[0, B_m - 1]$. B_m is updated by the node at every event (collision or a success). A typical update equation has the form able. To make the protocol more robust, in many random access protocol standards, a node uses its own transmission attempt history to adapt the retransmission times. Usually, the history is reset after every successful transmission. A node will make the m-th transmission attempt after a backoff period of x_m units of time. Here x_m is a uniformly distributed random integer in the interval $[0, B_m - 1]$. B_m is updated by the node at every event (collision or a success). A typical update equation has the form

(7.3)

(7.3)

where $a, b, B_{\min}$, and $B_{\max}$ are predefined.

> Read full chapter

Multiple Access Techniques

Vijay K. Garg, in Wireless Communications & Networking, 2007

6.11 Random Access Methods

So far we have discussed the reservation-based schemes, now we focus on random-access schemes [8]. When users have a steady flow of information to transmit (for example, a data file transfer or a facsimile transmission), reservation-based access methods are useful as they make an efficient use of communication resources. However, when the information to be transmitted is bursty in nature, the reservation-based access methods result in wasting communication resources. Furthermore, in a cellular system, where subscribers are charged based on a channel connection time, the reservation-based access methods may be too expensive to transmit short messages. Random-access protocols provide a flexible and efficient methods for managing channel access to transmit short messages. The random-access methods give freedom for each user to gain access to the network whenever the user has information to send. Because of this freedom, these schemes result in contention among users accessing the network. Contention may cause collisions and may require retransmission of the information. The commonly used random-access protocols are pure ALOHA, slotted-ALOHA, and CSMA/CD. In the following section we briefly describe details of each of these protocols and provide the necessary throughput expressions.

6.11.1 Pure ALOHA

In the pure ALOHA [18, 23] scheme, each user transmits information whenever the user has information to send. A user sends information in packets. After sending a packet, the user waits a length of time equal to the round-trip delay for an acknowledgment (ACK) of the packet from the receiver. If no ACK is received, the packet is assumed to be lost in a collision and it is retransmitted with a randomly selected delay to avoid repeated collisions.* The normalized throughput *S* (average new packet arrival rate divided by the maximum packet throughput) of the pure ALOHA protocol is given as:

acknowledgment (ACK) of the packet from the receiver. If no ACK is received, the packet is assumed to be lost in a collision and it is retransmitted with a randomly selected delay to avoid repeated collisions.* The normalized throughput *S* (average new packet arrival rate divided by the maximum packet throughput) of the pure ALOHA protocol is given as:

(6.20)

where G = normalized offered traffic load

From Equation 6.20 it should be noted that the maximum throughput occurs at traffic load G = 50% and is $S = 1/2e$. This is about 0.184. Thus, the best channel utilization with the pure ALOHA protocol is only 18.4%.

6.11.2 Slotted ALOHA

In the slotted-ALOHA [23] system, the transmission time is divided into time slots. Each time slot is made exactly equal to packet transmission time. Users are synchronized to the time slots, so that whenever a user has a packet to send, the packet is held and transmitted in the next time slot. With the synchronized time slots scheme, the interval of a possible collision for any packet is reduced to one packet time from two packet times, as in the pure ALOHA scheme. The normalized throughput *S* for the slotted-ALOHA protocol is given as:

(6.21)

where G = normalized offered traffic load

The maximum throughput for the slotted ALOHA occurs at G = 1.0 (Equation 6.21) and it is equal to $1/e$ or about 0.368. This implies that at the maximum throughput, 36.8% of the time slots carry successfully transmitted packets. The best channel utilization with the slotted ALOHA protocol is 36.8% — twice the pure ALOHA protocol.

6.11.3 Carrier Sense Multiple Access (CSMA)

The carrier sense multiple access (CSMA) [8, 18] protocols have been widely used in both wired and wireless LANs. These protocols provide enhancements over the pure and slotted ALOHA protocols. The enhancements are achieved through the use of the additional capability at each user station to sense the transmissions of other user stations. The carrier sense information is used to minimize the length of collision intervals. For carrier sensing to be effective, propagation delays must be less than packet transmission times. Two general classes of CSMA protocols are nonpersistent and p-persistent.

•

Nonpersistent CSMA: A user station does not sense the channel continuously while it is busy. Instead, after sensing the busy condition, it waits for a randomly selected interval of time before sensing again. The algorithm works as follows: if the channel is found to be idle, the packet is transmitted; or if the channel is sensed busy, the user station backs off to reschedule the packet to a later time. After backing off, the channel is sensed again, and the algorithm is repeated again.

- **p-persistent CSMA**: The slot length is typically selected to be the maximum propagation delay. When a station has information to transmit, it senses the channel. If the channel is found to be idle, it transmits with probability p. With probability q = 1 – p, the user station postpones its action to the next slot, where it senses the channel again. If that slot is idle, the station transmits with probability p or postpones again with probability q. The procedure is repeated until either the frame has been transmitted or the channel is found to be busy. If the station initially senses the channel to be busy, it simply waits one slot and applies the above procedure.
- **1-persistent CSMA**: 1-persistent CSMA is the simplest form of the p-persistent CSMA. It signifies the transmission strategy, which is to transmit with probability 1 as soon as the channel becomes idle. After sending the packet, the user station waits for an ACK, and if it is not received within a specified amount of time, the user station waits for a random amount of time, and then resumes listening to the channel. When the channel is again found to be idle, the packet is retransmitted immediately.

For more details, the reader should refer to [18].

The throughput expressions for the CSMA protocols are:

- **Unslotted nonpersistent CSMA**(6.22)
- **Slotted nonpersistent CSMA**(6.23)
- **Unslotted 1-persistent CSMA**(6.24)
- **Slotted 1-persistent CSMA**(6.25)

where:

S = normalized throughput
G = normalized offered traffic load
$a = \square/T_p$
□ = maximum propagation delay
T_p = packet transmission time

> Read full chapter

Overview

Overview

Michał Pióro, Deepankar Medhi, in Routing, Flow, and Capacity Design in Communication and Computer Networks, 2004

1.3.1 Traffic in the Internet

When a user employs an application such as email, the message generated is broken down into smaller data packets for transporting over the Internet. A large portion of applications on the Internet use the TCP/IP (Transmission Control Protocol/Internet Protocol) stack. The end computers are responsible for breaking an application's messages (e.g., web pages, email) into smaller packets on one end and then re-assembling them in the right order at the other end before handing it over to the application; in the process, the two end computers need to ensure that if a packet is by chance lost somewhere, they need to work together to ensure notification of the lost packet and retransmission so that the message is correctly delivered to the application. For more details on how the TCP/IP protocol works, refer to books such as [Com00], [KR02], [PD03], and [Tan03]. The network's job is to route these packets from one end to the other end, in fact, without considering reliability in delivery as per TCP/IP protocol. The packets are also known as IP datagrams.

There are many reasons why an IP datagram may not be delivered at the other end; for example, a physical transmission may have corrupted the datagram along the way making it meaningless to forward it any further, or due to congestion, a router in transit has run out of buffer space at the instant this particular datagram arrived. Unlike congested road networks where you are queued and delayed, in the Internet this delay can be addressed only to the extent that buffer capacity is still available and is subject to the capacity of the buffer at the arriving router as well. Going by the basic principle of the TCP/IP protocol stack, based on routing decision, a router's job is to forward packets towards the destination without necessarily taking into consideration whether there is any buffer space left at the arriving router; this is perfectly acceptable since the protocol rule allows for the end computers to re-generate any lost packets. While the protocol allows the rate to be adjusted due to congestion at the end system, any packet already in transit can still possibly be dropped.

Let us recap a couple of items from this discussion thus far: Traffic congestion can occur in a network (or in parts of a network), delay is possible, and packets may be dropped. Thus, the job of a network designer, a skilled technical professional, is to design a network that keeps delay to an acceptable minimum, and to minimize the loss of packets (due to congestion). Note that congestion is a fact of life; it is impossible to avoid it completely since traffic can be unpredictable at times. However, we can design a network in such a way that the congestion does not happen *all* the time, or, rather, happens only infrequently. Essentially, our situation would be equivalent to stating the following: in a particular highway in the road network, the delay is really bad; we need more lanes constructed. Precisely the same way, in the Internet, we need to have enough *lanes* (bandwidth or capacity) so that we can give an acceptable level of service; in addition, we need router buffers in place with sufficient memory to deal with traffic burst and real-time traffic so that packet dropping is minimized.

distribution, and the *M*/*M*/1 queueing system, for example, see [Med02]); then, there happens to be a nice analytical formula for computing the average packet delay due to queueing phenomenon. Specifically, if the average packet size is denoted by Kp bits, and the link capacity (speed) is given by C bits per second (e.g., T1-rate: 1.54 Mbps), then the average service rate of the link is $\mu = C/Kp$ pps. If the average arrival rate is denoted by λ_p pps, then the average delay (in seconds), $D(\lambda p, \mu p)$, is given by:[11]

distribution, and the *M*/*M*/1 queueing system, for example, see [Med02]); then, there happens to be a nice analytical formula for computing the average packet delay due to queueing phenomenon. Specifically, if the average packet size is denoted by *Kp* bits, and the link capacity (speed) is given by *C* bits per second (e.g., T1-rate: 1.54 Mbps), then the average service rate of the link is μ = *C*/*Kp* pps. If the average arrival rate is denoted by □ₚpps, then the average delay (in seconds), *D*(□*p*, μ*p*), is given by:11

(1.3.1)

(1.3.1)

This simple relation can provide an insightful information. First, we can certainly assess the average delay; for example, for an arrival rate of 100 pps and service rate of 190 pps, the average delay would be 1/(190 − 100) second, i.e., 11.11 millisecond (ms). If the average arrival rate increases to 150 pps, then the average delay increases to 25 ms. It is usually helpful to also look at the average utilization of the system, □, which is given by □*p*/μ*p*, i.e., the average arrival rate divided by the average service rate (□ = 100/190 = 0.555... in the considered case). To illustrate the typical delay behavior, we have plotted the average delay function with utilization, □, in the *x*-axis and the average delay in the *y*-axis, keeping the service rate fixed at 190 pps (Figure 1.9).

FIGURE 1.9. Average Delay Using *M*/*M*/1 Delay Formula

Now, when you see a number such as 11.11 ms, you might wonder why such a minor delay would be of any interest or concern. There are a couple of ways to answer this question: 1) this delay is only the queueing related delay (other delays such as propagation and nodal processing delay to this number), and 2) this value is only for a single-link case (for end-to-end delay, a packet could be required to traverse many links through the Internet).

We now illustrate the second answer further by returning to the web page access example from Warsaw to Kansas City. Beyond networks in different domains visited, which we have discussed earlier, it actually goes through several routers *within* each network. Specifically, in this case, we found that the path goes through 18 hops or routers.12 To simplify the illustration, assume that each link between two adjacent routers has the average delay of 11.11 ms. Then the end-to-end delay will be at least $11.11 \times 19 \approx 200$ ms! It is evident that when we consider the end-to-end delay, the delay components do add up due to the instantaneous store-and-forward nature of the Internet routing. Thus, it is important to keep the average delay on each link/network as low as possible.

routers has the average delay of 11.11 ms. Then the end-to-end delay will be at least 11.11 × 19 ≈ 200 ms! It is evident that when we consider the end-to-end delay, the delay components do add up due to the instantaneous store-and-forward nature of the Internet routing. Thus, it is important to keep the average delay on each link/network as low as possible.

Returning to the plot (Figure 1.9), notice that the average delay drastically increases as the average arrival rate is closer to the average service rate of the link. Thus, the average delay is a highly non-linear phenomenon which becomes worse when the utilization, □, is close to 1 (i.e., 100% link utilization). This graph can be used in another way by asking what is the acceptable delay that we would like users to tolerate for a good quality service? If we know this number, this graph (or the formula) can be used to determine what is the acceptable level of link utilization. Suppose, the acceptable average delay is 15 ms; then, we can determine that the acceptable average utilization is to be no more than 64.5% on the link. The good news is that at least for the purpose of network design considerations, it may be acceptable to use a maximum link utilization as an alternative criterion to the delay criterion. However, we need to take into account another factor that we have assumed thus far, i.e., that the packet arrival rate behavior follows the Poisson process. Unfortunately, measurements from the Internet indicate that the arrival process does not follow the Poisson process and the delay is *worse* than the one calculated using the Poisson assumption. This simply means that the delay curve is above the *M/M/1* delay curve; to illustrate this, we have plotted such a delay curve in addition to the *M/M/1* curve in Figure 1.10. There are important network design implications that arise from this discrepancy: The idea that the 64.5% average utilization would be acceptable at 15 ms of average delay is perhaps an overestimate; in reality, we might need to hold the average utilization even lower, perhaps at about 50% on average to achieve the 15 ms delay requirement.

Returning to the plot (Figure 1.9), notice that the average delay drastically increases as the average arrival rate is closer to the average service rate of the link. Thus, the average delay is a highly non-linear phenomenon which becomes worse when the utilization, □, is close to 1 (i.e., 100% link utilization). This graph can be used in another way by asking what is the acceptable delay that we would like users to tolerate for a good quality service? If we know this number, this graph (or the formula) can be used to determine what is the acceptable level of link utilization. Suppose, the acceptable average delay is 15 ms; then, we can determine that the acceptable average utilization is to be no more than 64.5% on the link. The good news is that at least for the purpose of network design considerations, it may be acceptable to use a maximum link utilization as an alternative criterion to the delay criterion. However, we need to take into account another factor that we have assumed thus far, i.e., that the packet arrival rate behavior follows the Poisson process. Unfortunately, measurements from the Internet indicate that the arrival process does not follow the Poisson process and the delay is *worse* than the one calculated using the Poisson assumption. This simply means that the delay curve is above the *M/M/1* delay curve; to illustrate this, we have plotted such a delay curve in addition to the *M/M/1* curve in Figure 1.10. There are important network design implications that arise from this discrepancy: The idea that the 64.5% average utilization would be acceptable at 15 ms of average delay is perhaps an overestimate; in reality, we might need to hold the average utilization even lower, perhaps at about 50% on average to achieve the 15 ms delay requirement.

FIGURE 1.10. Average Delay Using *M/M/1* Delay Formula and a Fictitious Delay Formula

FIGURE 1.10. Average Delay Using *M/M/1* Delay Formula and a Fictitious Delay Formula

We now consider another important design issue referred to as the *scaling* (or packing) factor. Our *M*/*M*/1 illustration so far has been shown for a service rate of 190 pps (corresponding to T1-link rate). Now consider a link with 10 times the capacity of T1-link,[13] and an arrival rate 10 times more, i.e., with the average rate of 1,900 pps. Reflecting back to the average delay function (Section 1.3.1), with a ten-fold increase in both the average service rate and the average arrival rate (i.e., while the utilization remains the same), the average delay *reduces* to one-tenth of the previous value since $1/(10\mu p - 10\lambda p) = 0.1/(\mu p - \lambda p)$. Thus, it is better to have one higher-speed link instead of having 10 parallel lower-speed links to carry the same amount of total traffic. This statement is valid without taking into consideration the cost of the link; in general, while taking the typical cost structure of links into consideration, the gain resulting from using high capacity links is even more profound.[14] This is often referred to as the *statistical multiplexing* gain. Similar phenomena also occur with air travel network where big aircraft (fleets) are used in many segments to reduce cost by better packing.

We now consider another important design issue referred to as the *scaling* (or packing) factor. Our *M*/*M*/1 illustration so far has been shown for a service rate of 190 pps (corresponding to T1-link rate). Now consider a link with 10 times the capacity of T1-link,[13] and an arrival rate 10 times more, i.e., with the average rate of 1,900 pps. Reflecting back to the average delay function (Section 1.3.1), with a ten-fold increase in both the average service rate and the average arrival rate (i.e., while the utilization remains the same), the average delay *reduces* to one-tenth of the previous value since $1/(10\mu p - 10\square p) = 0.1/(\mu p - \square p)$. Thus, it is better to have one higher-speed link instead of having 10 parallel lower-speed links to carry the same amount of total traffic. This statement is valid without taking into consideration the cost of the link; in general, while taking the typical cost structure of links into consideration, the gain resulting from using high capacity links is even more profound.[14] This is often referred to as the *statistical multiplexing* gain. Similar phenomena also occur with air travel network where big aircraft (fleets) are used in many segments to reduce cost by better packing.

Regardless, what we need to know from the measurement is whether the utilization is observed to be higher than the acceptable threshold for a particular link type; if so, it is probably time to add bandwidth/capacity to the network. Certainly, this problem is trivial if we were to have just a single-link network since we can measure the arrival rate, observe the utilization, and determine if bandwidth needs to be increased if the utilization is, for example, more than 50% on average. In large networks (rather than just a set of serial links) where the utilization can be further impacted by routing of traffic *flows*, the network design problem of adding bandwidth is much more complex than this simple single-link network example—this will be covered extensively in this book, and we will illustrate a simple design example later in Section 1.4.

In any case, an important lesson learned from the above discussion is that determining the average arrival rate in pps (based on measurements) is required since this really refers to the traffic *volume* or demand in a particular measuring unit for the Internet; furthermore, we need this information between different nodes in the network, somewhat similar to the air travel example where we need similar information between Kansas City and Warsaw, between Chicago and Detroit, and so on. In summary, we certainly and *minimally* need the traffic demand volume as an input for all our network design problems.

Note that we have characterized traffic only in terms of pps since we have hidden the packet size information while discussing the unit of data traffic. Actually, the average packet size could have been taken into account and the demand for data traffic could be given in *Mbps* instead of pps[15]—thus, both pps and Mbps are valid units for data traffic demand as long as the link capacity is used in the appropriate context. It may be noted that many ISPs in fact measure Mbps (or Gigabits per second, Gbps) as the unit for traffic demand volume for the purpose of routing optimization and network design.

Gbps) as the unit for traffic demand volume for the purpose of routing optimization and network design.

> Read full chapter

IP Traffic Engineering

Deep Medhi, Karthik Ramasamy, in Network Routing (Second Edition), 2018

7.1.4 Average Delay in a Single Link System

First, we assume that packet arrival to a network link follows a Poisson process with the average arrival rate as □ packets per sec. The average service rate of packets by the link is assumed to be μ packets per sec. We consider here the case in which the average arrival rate is lower than the average service rate, i.e., ; otherwise, we would have an overflow situation. If we assume that the service time is *exponentially distributed* (see Appendix B.12), in addition to packet arrival being Poissonian, then the average delay, □, can be given by the following formula, which is based on the 1 queueing model (see Appendix B.15.2):

(7.1.4)

Now consider that the average packet size is □ Megabits, and that the packet size is exponentially distributed. Then, there is a simple relation between the link speed *c* (in Mbps), the average packet size □, and the packet service rate μ, which can be written as:

(7.1.5)

This is then essentially the relation discussed earlier in Eq. (7.1.1). Combining □ with the packet arrival rate □, we can consider the arrival rate, *h*, in Mbps as follows:

(7.1.6)

If we multiply the numerator and the denominator by □, we can then transform Eq. (7.1.4) as follows:

(7.1.7)

This relation can be re-written as:

(7.1.8)

If we now compare Eq. (7.1.4) and Eq. (7.1.8), we see that the average packet delay can be derived directly from the link speed and arrival rate given in a measure such as Mbps; the only difference is the factor □, the average packet size. Second, although it may sound odd, the quantity, , can be thought of as the average "bit-level" delay on a network link where the average traffic is assumed to be *h* Mbps. In other words, if we track the traffic volume in Mbps on a link and know the link data rate, we can get a pretty good idea about the average delay. There are a couple of advantages to this observation: first, we can use traffic volume, *h*, and link speed, *c*, in other units such as Gbps without changing the basic behavior on delay given by ; second, it is not always necessary to track the average packet size; third, if the delay is to be measured in millisec instead of sec, then must be multiplied by the constant, 1000, without changing the basic structure of the formula. Finally, whether we consider measures in packets per sec or Mbps (or Gbps), the link utilization parameter, □, that captures the ratio of traffic volume over the link rate, remains the same regardless of the average packet size since

If we now compare Eq. (7.1.4) and Eq. (7.1.8), we see that the average packet delay can be derived directly from the link speed and arrival rate given in a measure such as Mbps; the only difference is the factor □, the average packet size. Second, although it may sound odd, the quantity, , can be thought of as the average "bit-level" delay on a network link where the average traffic is assumed to be *h* Mbps. In other words, if we track the traffic volume in Mbps on a link and know the link data rate, we can get a pretty good idea about the average delay. There are a couple of advantages to this observation: first, we can use traffic volume, *h*, and link speed, *c*, in other units such as Gbps without changing the basic behavior on delay given by ; second, it is not always necessary to track the average packet size; third, if the delay is to be measured in millisec instead of sec, then must be multiplied by the constant, 1000, without changing the basic structure of the formula. Finally, whether we consider measures in packets per sec or Mbps (or Gbps), the link utilization parameter, □, that captures the ratio of traffic volume over the link rate, remains the same regardless of the average packet size since

(7.1.9)

(7.1.9)

In essence, we can say that, under the 1 queuing assumption, the average delay, , can be given in terms of the link speed *c* and the traffic rate *h* where as

In essence, we can say that, under the 1 queuing assumption, the average delay, , can be given in terms of the link speed *c* and the traffic rate *h* where as

(7.1.10)

(7.1.10)

with utilization given by . Incidentally, Eq. (7.1.10) then gives a functional relation mentioned earlier in Eq. (7.1.2). What happens if we were to consider self-similarity of traffic? Unfortunately, there is no nice formula like the above when traffic is self-similar. It has been reported that the delay behavior with heavy-tail traffic is worse than that of the 1 delay. Thus, we will create a fictitious delay function for self-similar traffic and plot it along with the 1 delay as shown in Figure 7.1; note that in this figure the link speed *c* is kept fixed while the traffic rate *h* is increased—this is why the *x*-axis is marked in terms of link utilization, given in percentage as □ goes from 0 to 100%.

with utilization given by . Incidentally, Eq. (7.1.10) then gives a functional relation mentioned earlier in Eq. (7.1.2). What happens if we were to consider self-similarity of traffic? Unfortunately, there is no nice formula like the above when traffic is self-similar. It has been reported that the delay behavior with heavy-tail traffic is worse than that of the 1 delay. Thus, we will create a fictitious delay function for self-similar traffic and plot it along with the 1 delay as shown in Figure 7.1; note that in this figure the link speed *c* is kept fixed while the traffic rate *h* is increased—this is why the *x*-axis is marked in terms of link utilization, given in percentage as □ goes from 0 to 100%.

Figure 7.1. The *M*/*M*/1 average delay curve along with a fictitious delay curve.

Figure 7.1 is, in fact, very helpful in letting us understand the problem from the perspective of traffic engineering. For instance, suppose that to provide acceptable perception to users, we want to maintain the average delay at say 20 millisec. From the graph, we can see that with the 1 average delay formula, the link can handle an arrival traffic rate up to about 75% of the link capacity while maintaining the acceptable average delay. However, if the traffic does *not* follow the Poisson process, then the delay would be much worse at the same utilization value; for instance, in this fictitious graph of delay, we can see that the delay would be about 40 millisec instead. Certainly this is not desirable when the delay is required to be below 20 millisec. Thus, instead of taking a vertical view at a certain utilization, we take the horizontal view at an acceptable average delay. If we do so, we see that to maintain the average delay at or below 20 millisec, the non-Poisson traffic cannot go beyond 50% link utilization.

In regard to traffic engineering, there are two important points to note from the above discussion. First, there is a direct relation between delay and utilization; because of this, requiring a network to maintain a certain delay can be recast as requiring the utilization to be kept below an appropriate level. Second, since there is no simple nice formula to consider delay for self-similar traffic, being conservative on the requirement on utilization can often be sufficient for the purpose of traffic engineering. For instance, in the above example, we observe that keeping utilization at 50% would be more appropriate than letting it grow to 80%. Due to the relation between traffic volume and capacity through utilization (), this means that for a fixed link speed *c*, we need to keep the traffic volume at a lower level than would otherwise be indicated for Poisson traffic in order to address traffic engineering needs.

> Read full chapter

Figure 7.1. The *M*/*M*/1 average delay curve along with a fictitious delay curve.

Figure 7.1 is, in fact, very helpful in letting us understand the problem from the perspective of traffic engineering. For instance, suppose that to provide acceptable perception to users, we want to maintain the average delay at say 20 millisec. From the graph, we can see that with the 1 average delay formula, the link can handle an arrival traffic rate up to about 75% of the link capacity while maintaining the acceptable average delay. However, if the traffic does *not* follow the Poisson process, then the delay would be much worse at the same utilization value; for instance, in this fictitious graph of delay, we can see that the delay would be about 40 millisec instead. Certainly this is not desirable when the delay is required to be below 20 millisec. Thus, instead of taking a vertical view at a certain utilization, we take the horizontal view at an acceptable average delay. If we do so, we see that to maintain the average delay at or below 20 millisec, the non-Poisson traffic cannot go beyond 50% link utilization.

In regard to traffic engineering, there are two important points to note from the above discussion. First, there is a direct relation between delay and utilization; because of this, requiring a network to maintain a certain delay can be recast as requiring the utilization to be kept below an appropriate level. Second, since there is no simple nice formula to consider delay for self-similar traffic, being conservative on the requirement on utilization can often be sufficient for the purpose of traffic engineering. For instance, in the above example, we observe that keeping utilization at 50% would be more appropriate than letting it grow to 80%. Due to the relation between traffic volume and capacity through utilization (), this means that for a fixed link speed *c*, we need to keep the traffic volume at a lower level than would otherwise be indicated for Poisson traffic in order to address traffic engineering needs.

> Read full chapter

Markov Processes

Markov Processes

Scott L. Miller, Donald Childers, in Probability and Random Processes, 2004

9.5 Engineering Application: A Computer Communication Network

Consider a local area computer network where a cluster of nodes is connected by a common communication line. Suppose for simplicity that these nodes occasionally need to transmit a message of some fixed length (or, number of packets). Also, assume that the nodes are synchronized so that time is divided into slots, each of which is sufficiently long to support one packet. In this example, we consider a random access protocol known as *slotted Aloha*. Messages (packets) are assumed to arrive at each node according to a Poisson process. Assuming there are a total of *n* nodes, the packet arrival rate at each node is assumed to be □/*n* so that the total arrival rate of packets is fixed at □ packets/slot. In slotted Aloha, every time a new packet arrives at a node, that node attempts to transmit that packet during the next slot. During each slot, one of three events can occur: (1) no node attempts to transmit a packet, in which case the slot is said to be idle; (2) exactly one node attempts to transmit a packet, in which case the transmission is successful; or (3) more than one node attempts to transmit a packet, in which case a collision is said to have occurred.

All nodes involved in a collision will need to retransmit their packets, but if they all retransmit during the next slot, then they will continue to collide and the packets will never be successfully transmitted. All nodes involved in a collision are said to be backlogged until their packet is successfully transmitted. In the slotted Aloha protocol, each backlogged node chooses to transmit during the next slot with probability *p* (and hence chooses not to transmit during the next slot with probability 1 – *p*). Viewed in an alternative manner, every time a collision occurs, each node involved waits a random amount of time until they attempt retransmission, where that random time follows a geometric distribution.

This computer network can be described by a Markov chain, X_k = number of backlogged nodes at the end of the *k*th slot. To start with, we evaluate the transition probabilities of the Markov chain, $p_{i,j}$. Assuming an infinite number of nodes (or a finite number of nodes, each of which could store an arbitrary number of backlogged packets in a buffer), we note that

(9.65)

(9.66)

Markov Processes

Scott L. Miller, Donald Childers, in Probability and Random Processes, 2004

9.5 Engineering Application: A Computer Communication Network

Consider a local area computer network where a cluster of nodes is connected by a common communication line. Suppose for simplicity that these nodes occasionally need to transmit a message of some fixed length (or, number of packets). Also, assume that the nodes are synchronized so that time is divided into slots, each of which is sufficiently long to support one packet. In this example, we consider a random access protocol known as *slotted Aloha*. Messages (packets) are assumed to arrive at each node according to a Poisson process. Assuming there are a total of *n* nodes, the packet arrival rate at each node is assumed to be □/*n* so that the total arrival rate of packets is fixed at □ packets/slot. In slotted Aloha, every time a new packet arrives at a node, that node attempts to transmit that packet during the next slot. During each slot, one of three events can occur: (1) no node attempts to transmit a packet, in which case the slot is said to be idle; (2) exactly one node attempts to transmit a packet, in which case the transmission is successful; or (3) more than one node attempts to transmit a packet, in which case a collision is said to have occurred.

All nodes involved in a collision will need to retransmit their packets, but if they all retransmit during the next slot, then they will continue to collide and the packets will never be successfully transmitted. All nodes involved in a collision are said to be backlogged until their packet is successfully transmitted. In the slotted Aloha protocol, each backlogged node chooses to transmit during the next slot with probability *p* (and hence chooses not to transmit during the next slot with probability 1 – *p*). Viewed in an alternative manner, every time a collision occurs, each node involved waits a random amount of time until they attempt retransmission, where that random time follows a geometric distribution.

This computer network can be described by a Markov chain, X_k = number of backlogged nodes at the end of the *k*th slot. To start with, we evaluate the transition probabilities of the Markov chain, $p_{i,j}$. Assuming an infinite number of nodes (or a finite number of nodes, each of which could store an arbitrary number of backlogged packets in a buffer), we note that

(9.65)

(9.66)

Using these equations, it is straightforward to determine that the transition probabilities are given by

Using these equations, it is straightforward to determine that the transition probabilities are given by

(9.67) (9.67)

In order to get a feeling for the steady state behavior of this Markov chain, we define the drift of the chain in state *i* as

(9.68) (9.68)

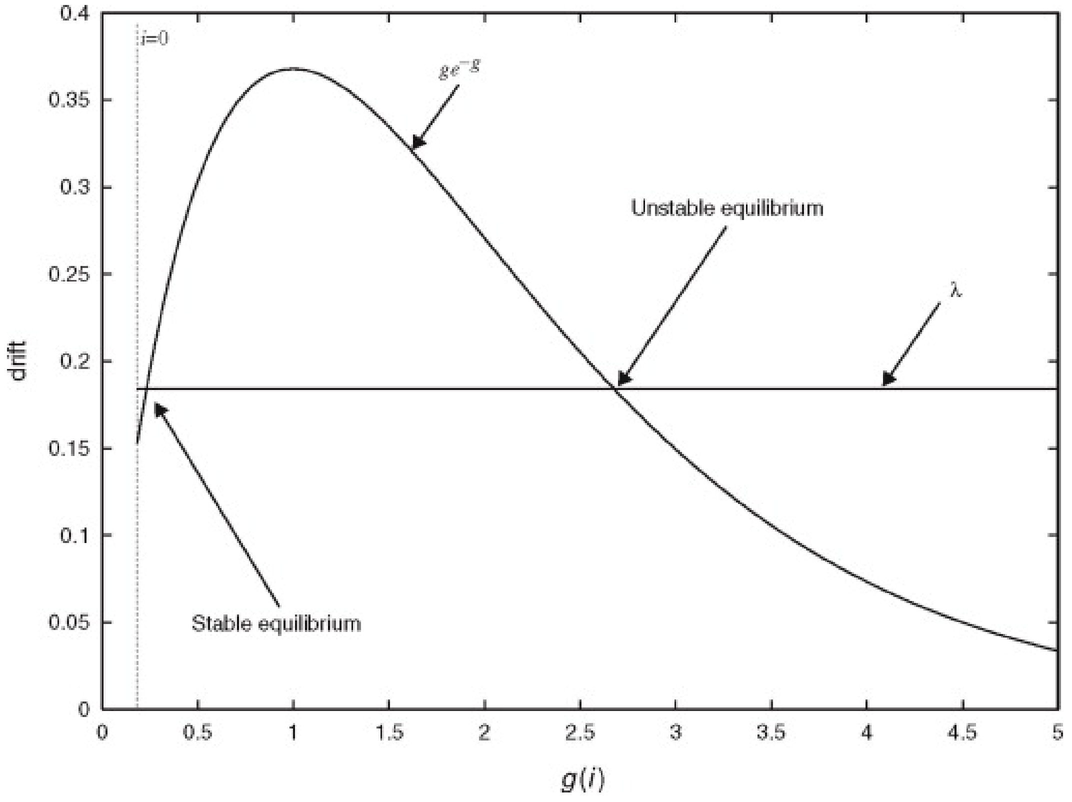
Given that the chain is in state *i*, if the drift is positive, the number of backlogged nodes will tend to increase, whereas, if the drift is negative, the number of backlogged nodes will tend to decrease. Crudely speaking, a drift of zero represents some sort of equilibrium for the Markov chain. Given the preceding transition probabilities, the drift works out to be

(9.69) (9.69)

Assuming that , then we can use the approximations $(1 - p) \approx 1$ and $(1 - p)^i \approx e^{-ip}$ to simplify the expression for the drift:

(9.70) (9.70)

The parameter $g(i)$ has the physical interpretation of the average number of transmissions per slot given that there are *i* backlogged states. To understand the significance of this result, the two terms in the expression for the drift are plotted in Figure 9.6. The first term, λ, has the interpretation of the average number of new arrivals per slot, while the second term, $g \exp(-g)$, is the average number of successful transmissions per slot or the average departure rate. For a very small number of backlogged states, the arrival rate is greater than the departure rate and the number of backlogged states tends to increase. For moderate values of *i*, the departure rate is greater than the arrival rate and the number of backlogged states tends to decrease. Hence, the drift of the Markov chain is such that the system tends to stabilize around the point marked stable equilibrium in Figure 9.6. This is the first point where the two curves cross. Note, however, that for very large *i*, the drift becomes positive again. If the number of backlogged states ever becomes large enough to push the system to the right of the point marked unstable equilibrium in the figure, then the number of backlogged nodes will tend to grow without bound and the system will become unstable.


i=0
ge^{-g}
Unstable equilibrium
λ
Stable equilibrium
drift
$g(i)$
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

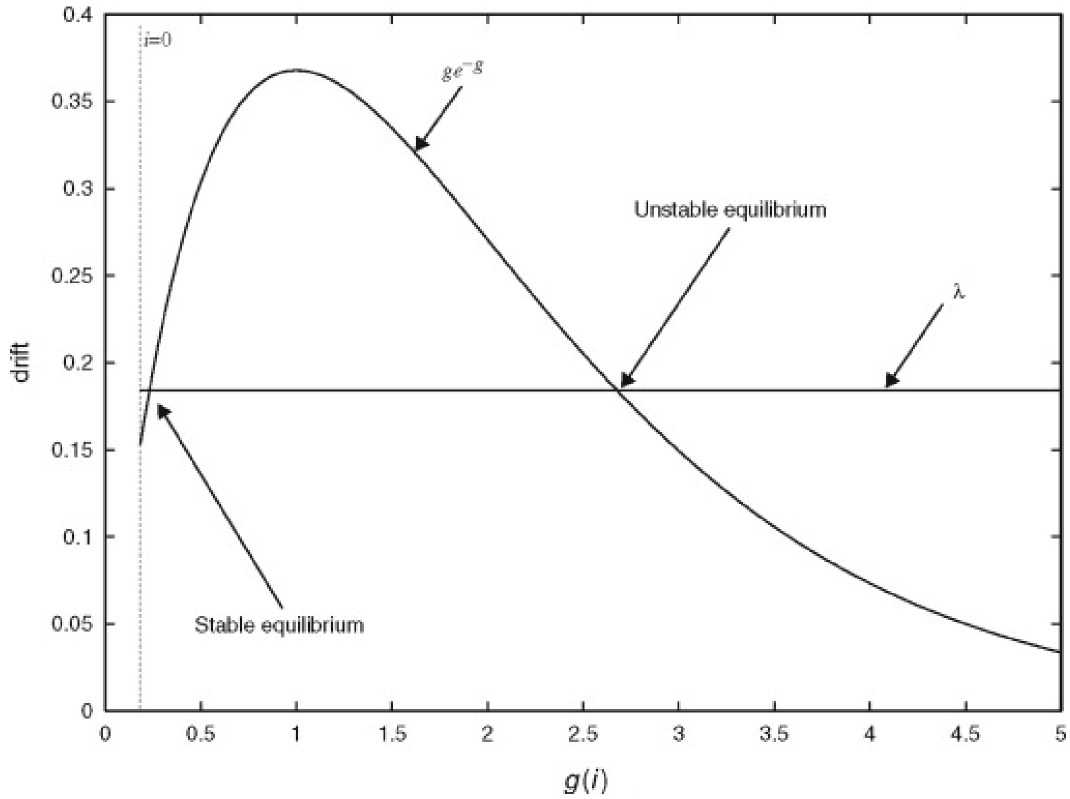


Figure 9.6. Arrival rate and successful transmission rate for a slotted Aloha system.

Note that the value of □ represents the throughput of the system. If we try to use a value of □ that is greater than the peak value of *g* exp(−*g*), then the drift will always be positive and the system will be unstable from the beginning. This maximum throughput occurs when *g*(*i*) = 1 and has a value of □*max* = 1/*e*. By choosing an arrival rate less than □*max*, we can get the system to operate near the stable equilibrium, but sooner or later we will get a string of bad luck and the system will drift into the unstable region. The lower the arrival rate, the longer it will take (on average) for the system to become unstable, but at any arrival rate, the system will eventually reach the unstable region. Hence, slotted Aloha is inherently an unstable protocol. As a result, various modifications have been proposed that exhibit stable behavior.

> Read full chapter