The Packet Starvation Effect in CSMA/CD LANs and a Solution

Brian Whetten, Stephen Steinberg, Domenico Ferrari
University of California at Berkeley

Abstract

In this paper we explore the *packet starvation effect* (PSE) that occurs in Ethernet controllers due to the unfairness of the CSMA/CD algorithm. The PSE causes some packets to experience latencies up to 100 times the average or to completely starve out due to 16 collisions. The PSE causes some packets to experience high delays at realistic offered loads as low as 40% and causes complete starvation of some packets at offered loads as low as 60%. The PSE makes CSMA/CD LANs unsuitable for real-time traffic except at offered loads much less than 100%. It is the limiting factor in the usable bandwidth of the bus.

As an alternative to CSMA/CD, we present the Fair Dual Distributed Queue (FDDQ) algorithm. Under high load, FDDQ uses a single reservation mini-slot per packet and a tree-based collision resolution algorithm (CRA) to maintain two distributed queues of waiting senders. This provides two priority FCFS access to the network. FDDQ provides utilizations and average latencies very similar to those of CSMA/CD but is fair even at extremely high offered loads. The protocol is stable for a constant number of senders, is simple enough to be practical, should be implementable in firmware, and completely eliminates the PSE.

1. Introduction

Much literature has been devoted to the study of the CSMA/CD algorithm for collision resolution, but most of it has concentrated on the stability of the algorithm, the average fairness of it and the average delay and utilization it can provide. These papers have shown that while CSMA/CD is unstable at offered loads much higher than 100% and with small packets[5], in practice it can achieve utilizations in the range of 90-97%[2]. This contradicts the conventional wisdom of network administrators, which holds that Ethernet LANs[15] (as defined by the IEEE 802.3 standards) behave very badly at offered loads much higher than 50%. Part of this problem is due to errors in Ethernet implementations, but the majority of the problem is due to the inherent unfairness of CSMA/CD.

We used an accurate Ethernet simulator to quantify the unfairness of CSMA/CD for a range of realistic offered loads. We tested loads similar to the continuous queue model used for the measurements published by Boggs, Mogul and Kent[2], a data traffic model based on the measurements by Gusella[8] and a combination of this data traffic with loads designed to model MPEG video streams.

For the simulated loads, the standard deviation of packet latency is often two to five times the average latency, and at offered loads above 60%-70% a significant portion of packets starve out and never gain access to the network. We term this the *packet starvation effect (PSE),* and it occurs because packets have a probability of getting access to the network inversely proportional to their current backoff value. When a new packet competes with an old one, the newer packet has a higher probability of gaining access and the probability of the older packet winning a competition for access exponentially decreases with the number of failed attempts the old packet participates in. At high load, when there are usually two or more packets in contention for the network, this effect causes a significant portion of packets to experience excessive delays or to be dropped after 16 failed retransmission attempts.

While the PSE has not been much of a problem in the past (as shown by the overwhelming success of the Ethernet) it is becoming a problem for modern traffic mixes which require higher bandwidths and real-time assurances on packet latency. In addition, the PSE completely disallows the use of admission control schemes, such as that described by Ferrari[6], which provide real-time traffic guarantees.

While the simulations in this paper have been based on the 10Base5 and 10Base2 versions of the IEEE 802.3 standard, the results of this paper should also be applicable to the 10BaseT standard, although the different topology of 10BaseT will result in different collision patterns and may change the results slightly. In addition, because the emerging 100 Mbps 802.3 standards (100BaseT, etc.) are designed to be virtually identical to

the 10BaseT standard when viewed from the MAC layer, these results should apply to these new standards as well.

Since real-time traffic is the primary driving force behind faster LANs, we suggest that the MAC layer should be considered carefully. The IEEE 802.12 committee is working on the 100BaseVG standard which solves these problems. As an alternative to reworking the entire standard, we propose a new MAC scheme called the Fair Dual Distributed Queue (FDDQ) algorithm. FDDQ is simple enough to be easily implemented in firmware as an alternative to CSMA/CD, provides two priority FCFS access to the LAN, is stable for constant numbers of senders, and provides utilization and average delays very close to CSMA/CD. By removing the packet starvation effect, it will allow LANs to be run at up to 95% utilization as opposed to the 60% to 70% the PSE limits current Ethernets to.

FDDQ is one of a class of Collision Resolution Algorithms (CRAs) that relies on all controllers keeping track of the current state of the network. FDDQ provides two priority FCFS access to the network by maintaining two globally distributed queues of waiting senders across the controllers. FDDQ works like CSMA/CD until a collision occurs, at which time all sites note that the network is congested, enter *congested mode,* and create two *buckets* of senders, one on each queue. The senders who were involved in the collision put their packets into a bucket, and only attempt to send that packet when their bucket comes to the top of the highest priority non-empty queue. Each packet sent while in congested mode is succeeded by an *ordering slot*. Hosts reserve spots in the distributed queues by sending 35 byte ordering packets in these ordering slots.

When multiple packets occur in the same bucket, a CRA with the same effect as the tree-based Capetanakis algorithm[4] is used to arbitrate between them. This algorithm repeatedly divides the top bucket into two buckets, with each host that was in that bucket randomly choosing one of the new buckets until there is only a single sender in each bucket.

Section 2 describes previous work. Section 3 describes the simulator methodology used in the study and the metrics used. Section 4 describes and quantifies the packet starvation effect, and Section 5 presents the Fair Dual Distributed Queue algorithm. Finally, Section 6 compares FDDQ to CSMA/CD, and Section 7 draws some conclusions.

2. Previous work

A large number of papers have been written on the analysis, simulation, and measurement of the CSMA/CD algorithm. Boggs et al. provides a representative overview of these papers[2]. While some of these papers point out that at high loads the standard deviation of packet latency is two to five times that of the average, and Bux[3] shows that Ethernets are not as suitable for real time traffic as token ring LANs, we are not aware of any papers that explain and analyze the packet starvation effect. Nichols[17] analyzes the suitability of Ethernets for real-time video traffic, and concludes that it is suitable only up to an offered load of 60%, which agrees with our conclusions.

At least as many papers have been published on collision resolution algorithms (CRAs) for broadcast channels. Like FDDQ, almost all of these algorithms have the controllers monitor the network constantly to provide better collision resolution behavior. The first of this class of algorithms was published in 1977. It is the tree-based collision scheme of Capetanakis[4]. It was followed up with improved variations by Massey[13] and Gallager, Tsybakov, and Mikhailov [7, 21]. All of these algorithms assume a fully slotted network with fixed packet lengths, where the cost of a collision is equal to that of a packet send. While some of these algorithms claim FCFS scheduling, they only divide the senders into the group of senders currently being resolved and the waiting senders. Since the current group of senders can be large, this greatly limits the scheduling resolution of their FCFS access. Some protocols have been proposed which use "control minislots" (CMS) to provide better utilization bounds, including the announced arrival random access protocols[18] and a distributed queuing algorithm[22]. These protocols involve having senders announce their arrival in one of a series of minislots before they can send, similar to the single ordering slot used by FDDQ. They both assume that the length of the minislots is very small compared to the length of a data packet, and so typically use 3 or more minislots per packet send. This is too expensive for LANs such as the Ethernet. The distributed queuing algorithm is similar to FDDQ in many regards, but it uses three minislots to achieve stability and stops using the minislots for ordering while the top group of senders in the queue is being resolved. Because of this, it is less efficient for LANs than is FDDQ, and it has a lower scheduling resolution for its FCFS delivery.

3. Methodology

In order to analyze the performance of CSMA/CD Ethernets, we used a detailed network simulator[1]. We chose to use a network simulator rather than analytical methods because of the known problems with analytical models of CSMA/CD: overly-pessimistic estimation of delays at high network loads [17], and simple

approximations of network layout and packet length distributions. Furthermore, using a network simulator to analyze the performance of Ethernet allowed us greater flexibility in setting loads than if we had taken measurements of a real network. Network simulators have been shown to be highly accurate when used correctly[20]. We have taken great care to use realistic parameters and we have validated our Ethernet simulation performance against published measurements in [2].

3.1 Simulator methodology

For the simulations in this study, a 10 Mbps Ethernet was modeled. The maximum bus length of 2579m was used for all simulations, with the hosts uniformly distributed along a single cable. The simulator we used takes into account the position of hosts on the bus and uses this to accurately simulate collisions. All simulation runs included at least 30,000 packets sent over the network.

We modeled continuously queued sources, data sources, and video sources in this study. Continuously queued sources each have an offered load close to 100%. Every time that a continuously queued source successfully transmits a packet it waits for the 9.6 μs inter-packet gap and then offers another packet of the same length to the network.

Data traffic has long been the primary use for LAN's. Numerous measurement studies[2, 8] have shown that packet lengths have a primarily bi-modal or multi-modal distribution. We based our length distribution on that measured in Gusella's study[8]. This distribution is shown in table 3.1. The lengths include the packet headers, and the average packet length is 649.1 bytes.

Describing the inter-packet arrival times of data traffic is more difficult and an active area of research; see for example the work in [12] on the self-similar nature of the distribution. For our simulations we have used an exponential distribution for inter-packet arrival times. Although this is a simplification, it has been shown to closely approximate real measurements over all but the shortest time intervals[20].

Length	Probability
64	0.304
144	0.083
220	0.08
576	0.1
1072	0.25
1500	0.183

Figure 3.1: Data length distribution

For our simulations of video traffic, we have used a rough approximation of MPEG, believing that the more complex proposed approaches[14, 23] offer few advantages since the small increase in accuracy is achieved at the expense of generality. In our simulation a video stream is modeled with a packet-train for each frame[11]. A packet-train is sent off 25 times a second and is of variable length with a mean of 11 KB. Every packet, or train car, is full except for the last and there is only a minimal delay of 70μs between cars, representing the time required to get the data to the Ethernet controller. The resulting video stream is 2.2 Mbps, or 22% of the available bandwidth. When multiple video streams are modeled together, the start of their packet trains is synchronized. This produces the burstiest possible combination of the video streams.

3.2 Performance metrics

Performance metrics are defined as functions of offered load. The individual offered load for each station is calculated by running the station by itself on the network and measuring its throughput. The combined offered load is the summation of the offered loads of each station.

The primary metrics analyzed in this study are:

- **Utilization:** the actual throughput of the bus, measured by dividing the sum of the lengths of the packets successfully sent (including their headers) by the amount of time of the simulation. Unlike some studies[2], we do not include the inter-packet gap or the propagation delay in the utilization calculation.
- **Average Delay:** the average time it takes to send a packet, measured from the time it arrives at the controller to the time it is successfully sent or is canceled due to 16 collisions. We also call this statistic the average packet latency. Because this study deals with the MAC layer, no concept of higher level buffering was taken into account.
- **Standard Deviation of Delay:** the standard deviation of the packet latency.
- **Percentage of Starved Packets:** the percentage of packets that suffer unacceptable delays or completely starve out due to 16 collisions. Packets that are involved in 16 collisions are dropped by the controller and lost. In this study, three separate statistics of this type are measured: the percentage of packets that suffer delays of at least 50 ms, the percentage of packets that suffer delays of at least 100 ms, and the percentage of packets that starve out completely.
- **Stability:** if the utilization of the bus decreases as the offered load increases, the protocol is unstable.

4. The packet starvation effect (PSE)

The packet starvation effect (PSE) causes some packets to experience extremely long delays and some to starve out due to 16 collisions. In this section, we describe the reasons for the packet starvation effect, and then quantify the extent of this effect as a factor of offered load. We

show that for the tested loads, the effect usually becomes significant starting at an offered load of between 60% and 70%, and gets worse as the offered load increases and as the number of stations increase. In some cases, the effect is significant at offered loads as low as 40%.

4.1 Explanation of the PSE

The reason for the packet starvation effect is that when two packets compete for access under CSMA/CD, the probability of one packet getting access over the other is approximately proportional to the ratio of their maximum backoff values. When two packets become ready (due to a new arrival or to the end of a backoff) at approximately the same time, the two controllers will both wait until they see that the network is free and then attempt to send, colliding with each other. When this occurs, they both backoff a random amount based on the number of collisions (N) that the packet has been a part of. If N is less than or equal to 10, the backoff is between 0 and 2^N-1. It is between 0 and 1023 otherwise. The probability that an older packet selects a smaller backoff value than a newer packet with fewer collisions is less than the ratio of the newer packet's maximum backoff (2^N-1 or 1023) divided by the older packet's maximum backoff. Because this value increases exponentially, unless a packet comes ready when no other host is ready to send, it will usually either get access to the bus very quickly or it will experience 16 collisions and starve out. Under high load, there is usually another packet waiting to send, and so long delays and packet starvation occurs to a significant percentage of packets.

We offer an example of this problem here. Consider two packets, an old packet which has collided three times and a new packet which has not been in a collision. If these two collide, the old packet (N=4 now) will select a backoff between 0 and 15 and the new packet (N=1) will select a backoff between 0 and 1. The only case where the old packet will get access before the new packet is if the new packet selects 1 and the old packet selects 0, which has a probability of 1/32. If they both select the same value, they will collide and backoff again (probability 2/32). Otherwise the new packet gets to request the bus again before the old packet (probability 29/32). Because of this effect, the old packet will usually keep on backing off until it is the only packet trying to access the network or until it starves out after 16 collisions.

4.2 Quantification of the PSE

This section quantifies the packet starvation effect for three classes of traffic: continuously queued sources, data traffic, and combinations of data and video traffic. The continuously queued sources are very similar to the sources measured in [2], except that the network length of 2759m for our simulations is longer than the 910m used in that study. This study showed that while the average latency scaled pretty much linearly as a function of offered load for their tested cases, the standard deviation of latency was at least 2 to 5 times the mean for the very high offered loads that they tested. Although we omit the graphs here, our simulations both agreed with this result and generalized it to the tested data and video sources. We found that this average latency/standard deviation of latency ratio started small and increased quickly with offered load. It became higher than 2 at between 50% and 65% offered load, and increased to a maximum of between 4 and 5. The reason for this extremely high variance is in large part due to the PSE. Most packets get through in a reasonable amount of time, but some take extremely long times to get through.

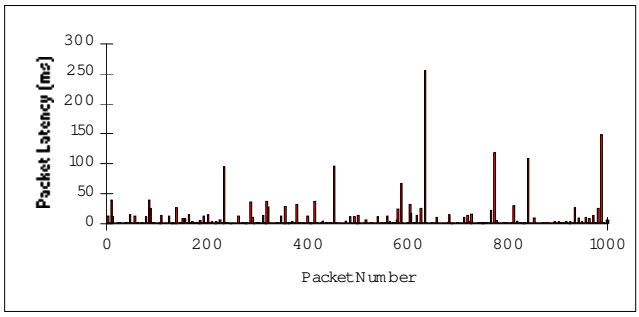

Figure 4.1: Packet Latency for 1000 Video Packets

A graphic example of this is shown in figure 4.1. This graph shows the simulated packet latency experienced by the first 1000 consecutive packets from a video host. This graph is taken from a scenario with 3 video streams (offered load 65.4%) and 40 data stations (offered load 6.5%) which together offer a total load of only 71.9%. The average latency for the packets in this graph is only 3.4 ms, yet there are numerous packets which have delays over 100 ms and the standard deviation of latency is 12.6 ms.

Figures 4.2 through 4.5 show the percentage of packets that experience the PSE at three different levels. The mildest level is packets that experience what we call *partial starvation* for at least 50 ms. These are blocked from access to the bus for at least 50 ms before they finally manage to get through, usually because a backoff ends when no other host is offering a packet. The second level consists of packets that experience partial starvation for at least 100 ms, and the final level consists of packets that completely starve out due to 16 collisions. On average, it took a starved packet approximately 225 ms before it finished its 16th collision and starved out, and this varied all the way from 75 ms to nearly 400 ms.

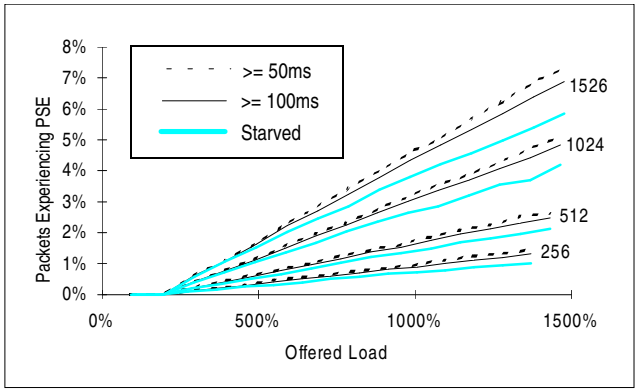

Figure 4.2: PSE for Continuously Queued Sources

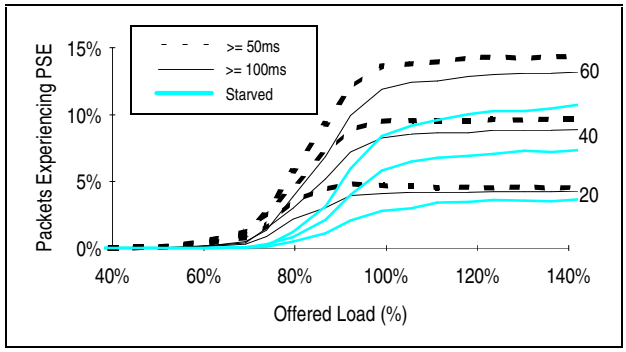

Figure 4.3: PSE for 20, 40, and 60 Data Sources

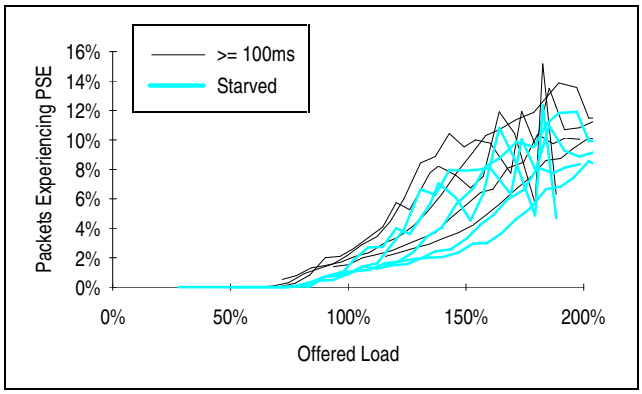

Figure 4.4: PSE for Video Packets,Combined Loads

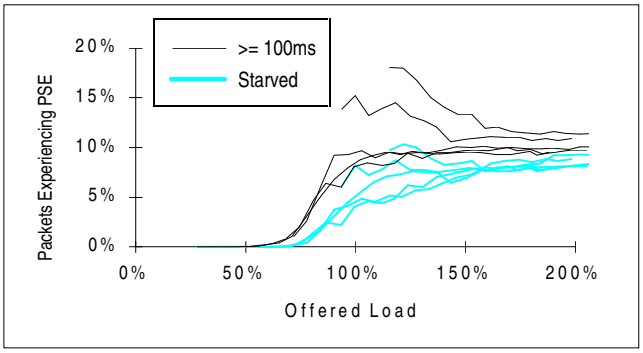

Figure 4.5: PSE for Data Packets, Combined Loads

Figure 4.3 shows the same statistics for 20, 40, and 60 data hosts. The curves for the three host configurations are very similar, with the PSE becoming more extreme as the number of stations increases. Figures 4.4 and 4.5 show the same statistics for combined video and data traffic. For clarity in the graphs, the 50 ms partial starvation level is not shown. The offered load for these graphs consists of between 1 and 5 video streams, combined with data traffic from 40 hosts added in roughly 6% increments.

The tested data and combined loads have between 3% and 15% of their packets completely starve out at offered loads much above 100%. This indicates that Ethernets should not be allowed to reach 100% offered load, even for short periods of time.

A critical question is at what level of offered load a CSMA/CD LAN can run before an unacceptable portion of its packets start experiencing unacceptable levels of the PSE. Figure 4.6 answers this question for the loads tested, for the three levels of packet starvation effect, and for differing percentages of packets that experience that effect. This table lists the minimum offered load that produces the given level of PSE in at least 0.01%, 0.1%, and 1% of the packets sent. The table entries marked as "All" indicate that all offered loads for that number of ideo streams produces the given PSE level. The offered load was increase in increments of approximately 1.5%, which bounds the accuracy of the figures.

From this table, we can draw some generalizations for some sample scenarios. To support "hard" real-time traffic, which can not afford to experience any losses or delays above 50 ms and which conforms to our data model, a CSMA/CD bus must be run at something less than 40% load. As shown by the continuous queuing tests (see Figure 4.2) this number will decrease as the average packet length increases and as the number of sources increases. A second scenario is if the network is to be used primarily for TCP/IP data traffic such as file transfers across a connected WAN. In this case, having more than 1% of the packets experience 100ms delays will greatly degrade the performance of these streams because of the Van Jacobson congestion control algorithm [10]. This algorithm assumes that all dropped packets are due to congestion, and will reduce the window size to 1 packet. For the tested traffic patterns, this occurs at around 75% offered load. The final case is the combination of compressed video streams with data traffic. If differencing based compression is used, then it is not acceptable to have even 0.1% of the packets experience 50 ms delay, which occurs at 69% with one video stream and at 65% with two video streams. This case is most similar to the types of traffic we expect to see in the future.

	PSE >= 0.01 %			PSE >= 0.1%			PSE >= 1%		
Sources	**>=50**	**>=100**	**Starved**	**>=50**	**>=100**	**Starved**	**>=50**	**>=100**	**Starved**
20 Data	43 %	49 %	69 %	53 %	62 %	72 %	71 %	76 %	86 %
40 Data	40 %	52 %	66 %	52 %	60 %	72 %	66 %	72 %	83 %
60 Data	37 %	49 %	62 %	50 %	59 %	74 %	68 %	75 %	80 %
1 Vid, 40 Data (Video)	55 %	69 %	80 %	69 %	75 %	84 %	81 %	88 %	101 %
2 Vid, 40 Data (Video)	48 %	60 %	79 %	65 %	72 %	81 %	77 %	85 %	105 %
3 Vid, 40 Data (Video)	All	All	70 %	All	All	80 %	68 %	80 %	94 %
4 Vid, 40 Data (Video)	All	All	All	All	All	All	All	88 %	94 %
1 Vid, 40 Data(Data)	All	45 %	66 %	51 %	57 %	71 %	66 %	71 %	81 %
2 Vid, 40 Data(Data)	All	All	60 %	48 %	54 %	71 %	65 %	71 %	81 %
3 Vid, 40 Data(Data)	All	All	All	All	All	74 %	All	70 %	84 %
4 Vid, 40 Data(Data)	All	All	All	All	All	All	All	All	All

Figure 4.6: PSE threshold levels in terms of offered load

5. The Fair Dual Distributed Queue algorithm

The Fair Dual Distributed Queue (FDDQ) algorithm remedies the problems that CSMA/CD has by providing two-priority FCFS scheduling of packet delivery. This is done by using *slots* to maintain two globally distributed queues of waiting senders across the bus, one for high priority (real-time) traffic and one for low priority (datagram) traffic. Each position in one of the queues is termed a *bucket*, and may have 0, 1, or more members. Under low load, senders use the same policy as CSMA/CD--they wait until they see that the network is empty and then attempt to send their packet. When a collision occurs, the controllers declare that the network is now congested, and all packets must get a place in one of the distributed queues before they can be sent. By monitoring the network, each controller keeps track of the buckets that currently exist, and which buckets it has packets in. The *top bucket* is defined as the first bucket in the high priority queue if one exists, and as the first bucket in the low priority queue otherwise. Only the packets that are in the top bucket are allowed to attempt to send.

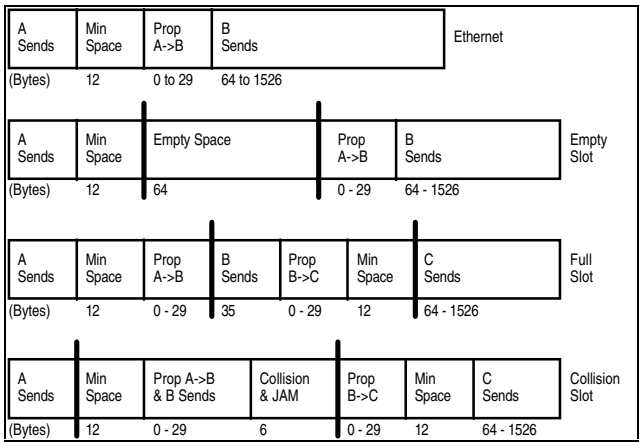

Figure 5.1: FDDQ Slots

5.1 Slots

During congested mode, the FDDQ algorithm makes use of *slots* (also known as mini-slots in some of the literature) to convey information to all of the controllers on the bus. A slot conveys whether 0, 1, or more senders are waiting to send. A slot has the value of either Empty (E), High Priority (H), Low Priority (L), or Collision (C). There are two types of slots--*ordering slots* and *collision resolution slots*.

During the congested mode, one ordering slot follows each packet that is sent. Controllers can reserve space in the distributed queues by sending a 35 byte packet at the beginning of an ordering slot. This packet consists of the 14 byte Ethernet header followed by 21 bytes of either all 0's for a high priority packet or all 1's for a low priority packet. If a collision is detected, a JAM signal is sent out, in exactly the same way as it is done in the IEEE 802.3 standards. Because senders can only start sending at the beginning of an ordering slot, the minimum length packet used in these slots is reduced from the CSMA/CD 64 bytes to 35 bytes.

Zero or more collision resolution slots precede each packet that is sent. These are used to resolve the cases where the top bucket does not contain exactly one sender. If there are no packets in this bucket, an empty event occurs in the slot, and if there are too many packets a collision event occurs.

The overall cost of each slot depends on a number of factors and varies between 18 and 72 bytes. Figure 5.1 illustrates this cost. The first diagram shows two successful sends in a row on a standard Ethernet. The minimum space between packets and the propagation delay between senders always occur between two sends, and so are not considered as part of the cost of a slot. The second diagram shows an empty slot. Each controller