Modifying the CSMA/CD (IEEE 802.3) Protocol to Involve a Network Queue (CSMA/CDNQ)

Abbey Wineland
George Mason University
awinelan@gmu.edu

Abstract

The carrier sense multiple access protocol with collision detections (CSMA/CD) often suffers from performance issues when it comes to transmission collisions within a network. This paper proposes a solution to the packet starvation effect [3], which is similar to Ethernet capture [2] of CSMA/CD by combining findings from previously performed research to improve CSMA/CD. The proposed solution, carrier sense multiple access with collision detection and network queue (CSMA/CDNQ) modifies the current CSMA/CD protocol by using a network queue to avoid multiple collisions while allowing behavior to resort back to IEEE standard 802.3 (CSMA/CD) in the event of queue failure, thus classifying it as a sometimes centralized, sometimes decentralized multiple access protocol. The Binary Exponential Backoff algorithm is slightly modified in CSMA/CDNQ to allow the queue dominance over the network and each node on the channel will utilize a lock, which is also referred to as the send variable, that controls when the node may transmit.

1. Introduction

The Carrier Sense Multiple Access with Collision Detection (CSMA/CD), also referred to by IEEE standards as 802.3, is the currently used Ethernet multiple access protocol. CSMA/CD has an algorithm that controls every transmitted frame of data by every node on the network. The frame is prepared and the adapter listens to the transmission channel for any current transmissions. Once the node detects a free channel, the frame is sent. If another frame is detected on the channel during transmission the collision detection element takes over. The first node that detects a collision aborts frame transmission and begins transmitting a 48 bit jam signal instead. The jam signal notifies other current transmitters that have not yet sensed the interference that a collision has occurred within the channel and thus all frames are corrupted. The nodes involved in the collision each wait a randomly determined amount of time based on its number of consecutive collisions. This exponential backoff phase implements the Binary Exponential Backoff (BEB) algorithm [6]. During BEB, the node must wait a random amount of time based on the number of consecutive collisions it has been part of. Each node has a collision counter that is incremented with each consecutive collision, up to sixteen collisions. BEB is exponential because each transmitters wait time is randomly chosen between 0 and *two raised to the value of counter* minus 1 [6], multiplied by 512 bit times [5]. From now on in this paper a time slot will be considered 512 bit times, where one bit time is .1 microsecond real time in a 10 Mbps network [5]. The exponent in the algorithm cannot exceed ten, even if there the collision counter exceeds that, thus limiting the wait time to 1023 time slots.

As an example, suppose a packet is sent and becomes part of a collision. So its wait period is randomly chosen between 0 and (2^1 – 1) time slots, thus having a fifty percent success chance. If upon retransmission it collides *again*, it will wait a random amount between 0 and (2^2 -1) time slots, with a success probability of twenty five percent. It's during this time that the winner of the BEB, which is the node with the smaller waiting period, gets to retransmit before the other(s) and thus obtains and maintains an unfair channel monopoly referred to often as Ethernet capture. Ethernet capture is especially common where a node is transmitting high volume media, such as video packets [2]. It causes other nodes to time out by manipulating the network, thus preventing other nodes from transmitting. This results in packet loss by the "losing nodes", which is referred to as the packet starvation effect by [3]. Only once a node has waited its assigned time may it start back into the algorithm again to try and retransmit [5]. The CSMA/CD protocol operates with minimal user disruption and with few aborted transmissions under most conditions. However, the collision detection algorithm is unfair when resulting in Ethernet capture and packet starvation. Available channel time is also often wasted when nodes are waiting to transmit. This is because when multiple nodes have frames to send, but are each waiting in the BEB, the channel is left unoccupied.

2. Research Problem

CSMA/CD is least efficient when the network is busy, due to collisions wasting bandwidth and Ethernet capture allowing one host to dominate the channel. Many proposed modifications to CSMA/CD would work well in a busy network, but decrease CSMA/CDs efficiency in low to medium traffic networks [6]. Thus the difficulty in resolving the issues with CSMA/CD is finding the right balance to optimally utilize the available resources in high network occupancy while not impairing performance during low occupancy. My solution includes introducing a network-wide queue to and altering the behavior of the existing CSMA/CD protocol.

The queue is originally empty, and will receive input only after a packet collision has occurred. The queue acts similarly to nodes on the network, but with precedence and authority over the other transmitters. In current CSMA/CD, when a node is ready to transmit a frame, it listens to the network and if it detects the channel is empty it begins transmitting. I propose setting a flag, or 'send variable,' in each host with the default value equal to true. When the value is true, the host can send as soon as it senses an empty channel. The only incident that changes the value to false is when the queue sends out a signal similar to a jam signal. The signal notifies all the hosts in the network that the queue is preparing to transmit. The only time the queue will transmit a packet is subsequent to a collision. Once the queue finishes all its transmissions, it sends another signal that resets the send variable back to the default value and hosts may once again transmit as in regular CSMA/CD.

The difference between CSMA/CD and CSMA/CDNQ is the behavior following the occurrence of a collision. Upon detection, the host information from each datagram involved is stored in the queue. The queue locks the network by changing all send variables network wide to false. It then sends an additional signal to the first host in the queue changing its send variable value back to true, at which point the host has exclusive access to the channel and can retransmit. Once the queue senses the channel is empty again it changes the send variable value back to false, and the next host in the queue's send variable to true. Once the queue is empty and hears that the channel is empty, it resets all hosts' send variables back to true. From there, regular CSMA/CD continues.

The existing BEB will be used as a backup if the queue goes down, with one slight change. To simplify CSMA/CDNQ while keeping BEB, the BEB has been modified to give the queue time to take control of the adapters on the network. Otherwise immediate retransmissions would prevent it from communicating with the hosts. When the queue gives full channel access to a node it puts a limit on how long it may send. This is the way CSMA/CDNQ eliminates Ethernet capture and thus the packet starvation effect.

3. Related Research

While researching the shortcomings of the CSMA/CD protocol, I came across different collision handling and avoidance mechanisms. This section briefly describes previous research performed related to my research problem.

3.1. Frequency Division Multiplexing (FDM)

FDM is a channel partitioning protocol that divides the available bandwidth into smaller channels and assigns a frequency to each node on the network [5]. This technique avoids collisions altogether, since each node has a chunk of the channel for its use only. FDM is inefficient in low traffic channels. Even if only one node in the network has something to send, it has access to only a portion of the channel, specifically the bandwidth of the channel divided by the number of nodes in the network. FDM is a fair technique, however, since no node can dominate the network. This prevents the packet starvation effect from occurring.

3.2. ControlNet

The deterministic network ControlNet passes a token around the network, allowing only the token holder the capability to transmit if so desired [6]. ControlNet's sending time restriction inspired part of my solution to provide full channel access to one transmitter at a time when necessary to eliminate channel domination and thus the packet starvation effect. However, I chose not to use a token passing mechanism in my solution as it wastes bandwidth in a network with low-channel traffic and causes unnecessary waiting by nodes with frames ready to send [6]. In other words, the network is accessible to only one host at a time and if that host has nothing to transmit, the channel is idle and unusable by other hosts that may be waiting to send a frame.

3.3. Distributed Queue Dual Bus protocol (DQDB)

The distributed queue dual bus media access protocol uses slots and two buses to regulate transmissions. The buses each have a slot generator and each flow one way in opposite directions. The slot generators send empty slots down the network. When a node has data to transmit, it fills in an empty slot with a reservation request. It must then wait until all other nodes that have already requested to send complete their transmissions. This is accomplished by every node having a queue, a request counter, and a countdown counter [1].

For every slot full with a request that passes a node, the request counter of that particular node is incremented. The request counter is decremented for every empty slot passing by on the opposite bus. When a node receives data to transmit, it copies the request counter value to the countdown counter, and decrements the countdown counter for every empty slot that is on the bus opposite of the bus that causes the request counter to increment. Once the countdown counter is zero, the node transmits using the next available empty slot. As you can see, a node must wait a considerable amount of time between when it has data to send, when it puts in a request to send, and when it can actually begin transmitting [1]. Although collisions are avoided, DQDB is impractical for non-linear network layouts and tends to distribute the bandwidth unfairly [4]. A very similar protocol, the fair distributed queue (FDQ), is described in detail in [4].

3.4. Fair Dual Distributed Queue (FDDQ)

The Fair Dual Distributed Queue (FDDQ) algorithm aims to eliminate the packet starvation effect (PSE) while also attempting to accommodate real time traffic. The authors in [3] suggest adding two global queues to each controller on the network that are in use only during a phase referred to as *congested mode*. Each queue is priority based, determined by whether a packet is real-time or not [3]. Like my proposed CSMA/CDNQ, FDDQ uses current CSMA/CD until the network enters congested mode, triggered by a collision within the channel. During congested mode packets are prioritized in each controller's queue using buckets. The top bucket of the high priority queue is sent. When both queues are out of buckets, the controllers exit congested mode and terminate queue use until the next collision [3].

4. Solutions/Analysis

My solution aims to resolve the current imperfections within the Ethernet's multiple access protocol: CSMA/CD. I've named my multiple access protocol CSMA/CDNQ because it uses the existing carrier sense multiple access protocol with the addition of a network queue. Since CSMA/CDNQ is partially centralized there is a need for a plan in case of failure. For this reason the current collision detection procedure is kept as a backup in the event of queue failure. My goal is not to replace the current CSMA/CD, but to enhance it.

4.1. Protocol characteristics

CSMA/CDNQ can be considered a hybrid multiple access protocol, since it has features of two of the three multiple access protocol categories, including *random access* and *taking turns* protocols [5]. Most of CSMA/CDNQ's behavior would lean towards its classification as a random access protocol, considering that CSMA already belongs to this group and that the nodes are free to transmit close to whenever they choose. Also, when an adapter transmits a frame it uses the full bandwidth available since it is not divided amongst the nodes into different frequencies. For example, if the nodes are connected through a 100 Mbps channel, each transmission of every node will broadcast at a rate of 100 Mbps. CSMA/CDNQ has characteristics belonging to the taking-turns protocol group during the time when the network queue has control. This is because only one adapter is able to transmit at a time, even if other nodes have frames ready to transmit.

4.2. Normal Behavior

CSMA/CDNQ's behavior when there are no collisions is just like regular CSMA, with the addition of a *send variable* that acts like a lock. Figure 1 shows the behavior of an adapter in CSMA/CDNQ without the details of how the collision handling is implemented. Section 4.3 and

4.4 go into more detail of the behavior in different situations.

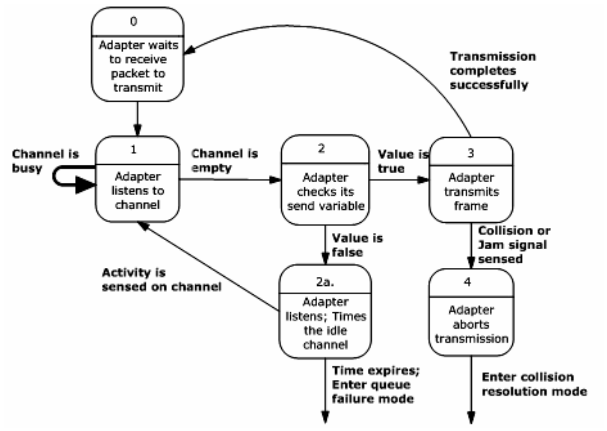
0
Adapter waits to receive packet to transmit
Transmission completes successfully
Channel is busy
1
Adapter listens to channel
Channel is empty
2
Adapter checks its send variable
Value is true
3
Adapter transmits frame
Value is false
Collision or Jam signal sensed
Activity is sensed on channel
2a.
Adapter listens; Times the idle channel
4
Adapter aborts transmission
Time expires; Enter queue failure mode
Enter collision resolution mode

Figure 1: Normal adapter behavior

When an adapter has a frame to transmit, it first listens to the channel. If busy, it waits until it's empty. Otherwise, it checks its send variable. If the send variable is equal to false, the adapter begins timing the empty channel while simultaneously listening for action in the channel (see section 4.4). However, if the send variable is true, the adapter sends the frame. The whole time while transmitting the adapter is also listening for any interference, or change in frequency, that would indicate a collision. As soon as a collision is sensed, the transmitter aborts and sends out a 48 bit jam signal to notify other adapters there has been interference and that their frames may also be corrupted.

4.3. Network Queue

The network queues behavior can be described as passive most of the time. It is more of an observer until an event triggers it to act otherwise. The network queue, which is actually another node in the network that has a transmitter and also a queue, constantly monitors the network channel for activity. It also regulates the send variable of every node in the network. The instant the network queue is connected to the network it sends out a signal that sets each node's send variable value to true. This allows them to send whenever the channel is empty. Similarly, each time it senses a new node in the network it ensures its send variable is initialized to true.

The queue recognizes collisions very similarly to how regular nodes in the network do, but in addition to sensing its own frequency it senses others. Figure 2 shows how the queue monitors the network and at what point it takes control. The queue constantly listens to the channel, and when it senses activity it listens for interruption or completion. The queue becomes active when it senses a collision in the network or the 48 bit jam signal. Once it has finished controlling which adapters can transmit and when, it goes back to simply monitoring the network and letting the nodes send as per the CSMA/CD protocol.

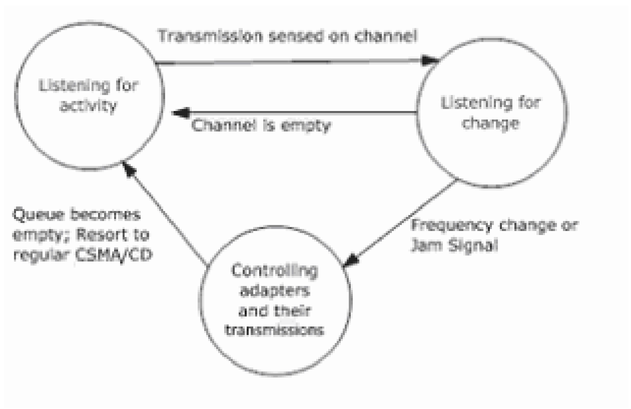
Listening for activity
Transmission sensed on channel
Listening for change
Channel is empty
Queue becomes empty; Resort to regular CSMA/CD
Frequency change or Jam Signal
Controlling adapters and their transmissions

Figure 2: Queue monitoring the network

In normal CSMA/CD, the first time a node is involved in a collision it has a 50 percent chance of immediately resending the frame without waiting at all [5]. In CSMA/CDNQ, the BEB algorithm is modified because there must be wait time to allow the queue's change-value-signal to propagate to all the nodes. Therefore, when the collision counter is one, the random number used to determine wait time cannot be 0. In this case, the BEB is changed so the wait time is automatically one time slot. The following table demonstrates the number of time slots that can be randomly chosen in CSMA/CD vs. CSMA/CDNQ.

Table 1: Possible wait times for up to 2 consecutive collisions

Collision Counter		**CSMA/CD**	**CSMA/CDNQ**
	0	N/A	N/A
	1	0, 1 time slots	1 time slot
	2	0,1,2,3 time slots	0,1,2,3 time slots

The reason a node cannot immediately retransmit as is possible in regular CSMA/CD is because immediately following a collision, the queue sends out a broadcast signal to all nodes in the network that changes all send variables to false. If a node could immediately retransmit with no waiting time, the queue's broadcast would not be able to propagate through.

Due to the queues variable changing signal, no nodes can transmit. The queue then sends out a frame that collects the MAC address of every node involved in the collision. It can determine which nodes were involved in the collision since, as in regular CSMA/CD protocol, each node has a collision counter that is used by the BEB algorithm previously mentioned (see [6]). When it propagates back to itself, it places each MAC address in its queue. This may sound time consuming, but imagine a congested network experiencing frequent collisions involving many hosts. In regular CSMA/CD if there are say, five nodes involved in one collision, many repeat collisions will occur, especially during the early stages of the BEB, since the likelihood that two or more nodes select the same waiting time is higher earlier in the algorithm. By preventing future collisions and making one quick trip around the network, the CSMA/CDNQ actually saves time. It may also be quicker to simply modify the 48 bit jam signal to flip the send variable value, but in the event that the queue goes down, this could be disastrous. Section 4.4 addresses this issue.

As soon as the collector frame returns to the queue, the addresses of the adapters involved in the collision are added to the queue and network queue channel takeover begins. As an example let us refer to the first address in the queue as Collision Address 1 (CA1), the second Collision Address 2 (CA2), and so on. CA1 at this point senses an empty channel and is ready to retransmit, but is waiting for the value of its send variable to change before it can begin. The queue sends a signal to CA1 that 1) resets CA1's collision counter to 0, 2) sets a frame counter in the adapter initialized to 5 that will decrement with every sent frame so that CA1 only controls the network for a period of time, and 3) resets CA1's send variable to true. CA1 now has control of the network to transmit up to 5 frames consecutively. With each completed transmission, the frame counter decrements. Channel dominance leaves this node in one of two ways. The first is in the event that it transmits its maximum number of frames, which is five. The nodes' send variable is then automatically reset to false, thus preventing it from continuing transmitting. The second event is that it runs out of frames to send, thus leaving the channel empty. In both scenarios, as soon as the queue senses an empty channel, it sends out two signals. The first signal goes to CA1, verifying that its send variable is now false. If it is not, meaning it simply ran out of frames to send, the signal changes it to false. CA1 is then removed from the queue. The second signal goes to CA2, which is now the head of the queue. This signal changes CA2's send variable to true, and the same algorithm that CA1 executed is now traversed by CA2.

This behavior continues until there are no more addresses in the queue. Consider the last address, CA5, completes its five frames. The queue still sends its signal to CA5 changing the variable to false, but instead of sending a signal to the next address in the queue (since there isn't one) it sends out a broadcast resetting all transmit variables to true. The network is now on its own again, that is, until the next collision occurs, which triggers the whole process over again.

4.4. Queue Failure

In the unlikely yet disastrous event that the network queue goes down, there is a backup plan so that transmissions may continue. While in collision resolution mode, when the queue has control over the network, each adapter is still listening to the channel for other transmissions. This is because when all but one adapter has send variable equal to false, one node in the network should be transmitting. If this is not happening, it indicates that the queue went down while all the send variables are equal to false. This is why when an adapter's send value is false, it counts the idle time in the network. It begins timing after the end of each transmission, which is also as soon as the channel becomes empty. Anytime its send variable is false, it times the idle channel. If a node's send value is false, and the channel is empty for 1023 times slots (1023 * 512 bit times), it automatically resets its send value to true. The number 1023 is selected because in CSMA/CD, this is the maximum amount of time an adapter can possibly randomly select to wait.

Imagine that the queue goes down; all nodes wait the maximum timeout time, and then reset their variables to true. Now imagine that a collision occurs. The same procedure is performed, meaning the nodes enter modified BEB, which is the BEB in which immediate retransmission is not allowed. Without interaction from the network queue, they just continue from the first waiting period into regular CSMA/CD. It is only in this event that packet starvation and Ethernet capture can occur.

Earlier in the paper, I mentioned how it would be quicker to have the jam signal change the value of the send variables network wide to false. After all, it is already being propagated to each node. You should now realize that this is impractical, and that only the queue should be able to lock nodes out from sending. If a regular node had the power to change the send variable by transmitting the jam signal, and the queue goes down, then *every time* a collision occurs each node on the network must wait the maximum timeout period, indicating queue failure, before resetting it's variable and re-entering normal CSMA/CD. This would be a huge waste of time, because in CSMA/CDNQ, even though multiple collisions may occur in the case of the queue being down, the nodes only must wait the maximum timeout time once, instead of every time a collision occurs. Even worse yet, while a node on its second collision timeout waiting period hears a node that it collided with previously, it will assume the transmission is coming from the queue, reset it's waiting time, and be locked out indefinitely!

5. Summary

The CSMA/CDNQ algorithm may appear complicated, but in fact is simply a modified version of the CSMA/CD protocol that is currently in use by the Ethernet. My research indicates weakness in the CSMA/CD protocol that my CSMA/CDNQ resolves. A prototype may be developed in the future to test and examine the CSMA/CDNQ under different network situations to get an accurate analysis of the benefits and performance statistics of the protocol.

6. References

[1] Akyildiz, I.F., and J. Liebeherr, "A Highly Adaptive Media Access Protocol for Dual Bus Metropolitan Area Networks", Distributed Computing Systems, 1992., Proceedings of the 12th International Conference, 9-12 Jun 1992 pp. 186-193.

[2] G. Fairhurst, "Carrier Sense Multiple Access with Collision Detection", 14 Jan. 2004 <http://www.erg.abdn.ac.uk/users/gorry/course/lan-pages/csma-cd.html>.

[3] Ferrari, D., S. Steinberg, and B. Whetten, "The Packet Starvation Effect in CSMA/CD LANs and a Solution", Local Computer Networks, 1994., Proceedings of the 19th Conference, 2-5 Oct. 1994 pp 206 - 217

[4] Kabatepe, M., and K.S. Vastola, "FDQ: The Fair Distributed Queue MAN", INFOCOM '92. Eleventh Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE, 4-8 May 1992 pp 200-209 vol. 1

[5] Kurose, J.F., and K.W. Ross, *Computer Networking,* Pearson Addison Wesley, 2005.

[6] Lian, F.L., J.R. Moyne, and D. Tilbury, "Performance Evaluation of Control Networks: Ethernet, ControlNet, and DeviceNet", IEEE Control Systems Magazine, Feb. 2001.