



An Overview of Controller Area Network (CAN) Technology

November 12, 2003

Daniel Mannisto
Senior Developer
Machine Bus Corporation
Evanston, Illinois
www.machineBus.com

Mark Dawson
Technical Writer
msjddawson@sbcglobal.net

Contents

What is Controller Area Network?	1
A History of Controller Area Network	2
What is the Open Systems Interconnection (OSI) Model?	3
How the OSI Model Applies to Controller Area Network Systems	5
The CAN Physical Layer: Bit Representation.....	6
The CAN Physical Layer: Bit Timing and Synchronization.....	9
The CAN Data Link Layer: Bus Arbitration.....	12
The CAN Data Link Layer: Frame Formats.....	14
The CAN Data Link Layer: Error Detection and Handling	18

What is Controller Area Network?

Controller Area Network (CAN) is a network protocol that allows multiple processors in a system to communicate efficiently with each other. In the early 1980s microprocessors became small enough and powerful enough to start appearing everywhere, not just inside personal computers. The typical automobile built since 1985 has dozens of embedded processors. Electronic fuel injection systems have replaced carburetors. Anti-lock braking systems use fast microprocessors to make split second decisions, as do safety systems like air bags and seat belts. Other systems have built-in processors to work as sensors, keeping track of and reporting on temperature and pressure changes for the coolant system, transmission, and oil. Still others monitor emissions, steering performance, and brake fluid levels. As more and more electronic systems were added to automobiles and other machines to make them safer, more efficient, more reliable, and easier to maintain, it became increasingly important for these systems to communicate with each other. To this end the Controller Area Network protocol was introduced in 1983. Today Controller Area Network is the standard for high-speed, mission critical, real-time control networks in many machines. CAN chips are reliable, simple, and cheap. They last a long time and are able to endure temperature and pressure extremes.

CAN makes it possible for all of the separate microprocessors in a system to send and receive messages without relying on some form of central control. This makes for much more efficient control of message traffic, and it reduces the need for internal computer wiring by as much as 90 percent in some cases. Also, CAN systems are flexible and easy to maintain. Mechanics or technicians can repair or replace computer hardware in the system without affecting the rest of the network. Design engineers can rework CAN systems and add or remove network nodes easily. Finally, this leads to a system that lends itself well to project management. Because of the industry-wide CAN standard, each functional unit in a CAN system can be developed and tested separately. When every individual part is shown to work independently, there is a high probability that when all of the components are assembled they will work as a system.

A CAN system sends messages using a serial bus network, with the individual nodes (processors) in the network linked together in a daisy chain. Every node in the system is equal to every other node. Any processor can send a message to any other processor, and if any processor fails, the other systems in the machine will continue to work properly and communicate with each other. Any node on the network that wants to transmit a message waits until the bus is free. Every message has an identifier, and every message is available to every other node in the network. The nodes select those messages that are relevant and ignore the rest.

The CAN protocol was designed for short messages, no more than eight bytes long. It is generally used for sending signals to trigger events, such as to lock seat belts during heavy braking, and measurement values, such as temperature and pressure readings. The protocol never interrupts an ongoing transmission, but it assigns priorities to messages to prevent conflicts and to make sure that urgent messages are delivered first, and includes error checking to make the traffic highly reliable. Even so CAN systems are fast. A CAN system is able to transmit up to 7600 8-byte messages or 18,000 trigger signals per second.

A History of Controller Area Network

Robert Bosch introduced the CAN serial bus system at the Society of Automotive Engineers (SAE) congress in Detroit in 1986. It was called the "Automotive Serial Controller Area Network" because CAN systems were developed first for the automotive industry. It quickly became obvious, however, that the protocol would work extremely well in a wide variety of embedded systems applications. In 1990 CAN technology was introduced for weaving machines, and today the textile industry makes heavy use of CAN systems. CAN chips are found in elevator systems in large buildings, ships of all kinds, trains, and aircraft, in X-Ray machines and other medical equipment, logging equipment, tractors and combines, coffee makers, and major appliances. Today almost every new car built in Europe has at least one CAN system, and CAN has become the industry standard for communications systems in vehicles. Industry leaders expect steady growth in CAN systems for all types of machines and equipment.

Intel delivered the first CAN chip in 1987, and Phillips Semiconductors responded with a CAN chip of their own shortly after that. Motorola and NEC followed; today some 15 different semiconductor vendors build CAN chips. The International Standards Organization (ISO), based in Geneva, Switzerland, is a network of standards institutes from 147 countries that defines standards for international cooperation in technology. ISO published standard 11898 in November of 1993 to define CAN for general industry use. The CAN in Automation (CiA) user group was founded in March of 1992. Now in its 20th year, the CAN protocol is still being enhanced. Early in 2000 an ISO task force defined a protocol for scheduled transmission of CAN messages called Time Triggered CAN (TTCAN). The CAN user's group estimates that the TTCAN extension will allow Controller Area Network to continue its rapid growth for another ten to fifteen years into a variety of other embedded systems applications. In years to come, CAN systems may be working in just about every type of machine or process.

What is the Open Systems Interconnection (OSI) Model?

Open Systems Interconnection is a standard reference model for how messages should be sent between any two points in a telecommunications network. Engineers follow OSI standards to make sure that products they develop will communicate properly with other products. Like Controller Area Network itself OSI was created in 1983 by a committee of professionals from several major computer and telecommunications companies. ISO published standard 7498 to define the ISO/OSI model.

In the OSI model, data communications are organized into a hierarchy of seven different functions. Each function is a separate layer in the model. The top layer relates directly to the software application or hardware device that seeks to send messages across a network, while the bottom layer refers to the actual bit traffic. This is the protocol stack:

Layer 7	Application	Defines communications partners, type of service, and security
Layer 6	Presentation	Converts data from one format to another, such as from a text file to a popup window displaying that text
Layer 5	Session	Opens, coordinates, and ends conversations and exchanges of data between two applications
Layer 4	Transport	Manages error checking and verifies that packets are delivered
Layer 3	Network	Routes and forwards data to the proper destination
Layer 2	Data Link	Builds data packets and synchronizes network traffic
Layer 1	Physical	Conveys the actual bit stream across the network. Manages the hardware and the mechanical process for sending and receiving data.

Figure 1: ISO/OSI Model

When an application seeks to communicate across the network, it processes the message through the seven layers of the OSI model one by one, starting at layer 7 and ending at layer 1. Then, the bit traffic is sent across the network to the application or device meant to receive the message. The application receiving the message processes the message in reverse, from layer 1 to layer 7.

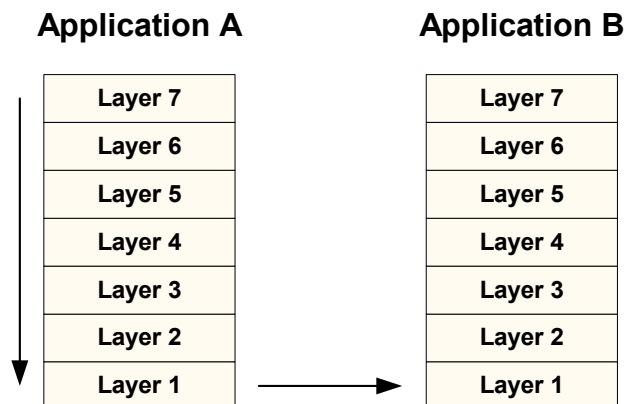


Figure 2: Inter-Node Communication Using the ISO/OSI Model

This means that each of the layers provides services to the layer immediately above it and uses the services of the layer immediately below it, but each layer otherwise works independently of all the others. The function of each layer is hidden from and transparent to the function of every other layer, except for the layers immediately above and below. For example, layer 5 has no direct contact with layer 2. The essence of the OSI model, then, is that a network can pass data between two applications or processes using means that are completely transparent to those applications.

Each layer in the system is in effect communicating with the equivalent layer in a separate application or process. So layer 4 for a software application that seeks to send data across a network will have a “corresponding element” in layer 4 in the database utility that is to receive that data. Each layer has its own set of rules to govern interactions with corresponding layers, and these rules are called layer protocols.

How the OSI Model Applies to Controller Area Network Systems

The ISO/OSI model is designed to be comprehensive enough for the most complex networks. Many communications systems, however, do not use all seven layers in the reference model, and that includes Controller Area Network systems. Controller Area Network systems only involve the transmission of brief, simple messages to trigger events or to provide monitoring values from sensors, such as temperature or pressure. Also, a CAN system is usually closed, and does not need to account for system security, presenting data in a user interface, or monitoring network logins or sessions. Hence only layers 1 and 2, Physical and Data Link, are included in the ISO/OSI 11898 specification.

Physical Layer

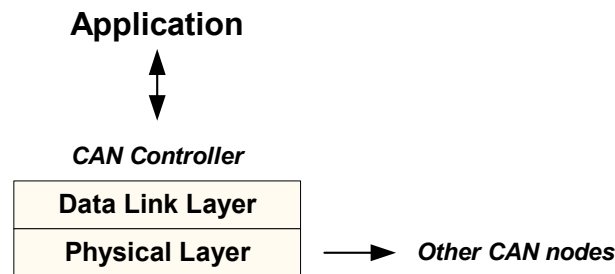


Figure 3: CAN Physical Layer

The *physical layer* governs the connection between the nodes in a network, and the actual transmission of electrical impulses across a copper wire, coax or fiber optic cable, or wireless signal. The transmitter's physical layer translates data drawn from the data link layer into an electronic signal. On the receiving end, the physical layer translates those electronic signals back into a data format that is then passed up to the data link layer. Thus the physical layer provides standards for bit representation, bit timing and synchronization, and often the type of pin connectors and cables to use. (See [“The CAN Physical Layer: Bit Representation”](#) and [“The CAN Physical Layer: Bit Timing and Synchronization”](#))

Data Link Layer

The *data link layer* builds data frames—or packets—to hold data and control information. This control information is used to identify frames, determine access to the bus, and to detect errors. Part of the control information is used to prevent conflicts when two messages seek to access the same part of the network at one time, a function known as Medium Access Control (MAC). In the CAN protocol, the MAC function will prevent a conflict from happening by determining which data frame has the highest priority to the bus.

Other portions of the control information contain codes that allow a node to tell if a message has been corrupted. Random errors are common when data is transmitted across any kind of network, so one of the most important roles of the data link layer is to limit the number of errors. (For more information about MAC see [“The CAN Data Link Layer: Bus Arbitration.”](#) Also see [“The CAN Data Link Layer: Frame Formats”](#) and [“The CAN Data Link Layer: Error Detection and Handling.”](#))

The CAN Physical Layer: Bit Representation

Typically the physical media used by Controller Area Network systems is a differentially driven pair of wires. This provides very reliable signal transmission despite low signal levels and common mode errors. The two wires are named CAN_H and CAN_L and are terminated using 120-ohm resistors. It is also typical to use twisted pair wires to reduce electromagnetic interference.

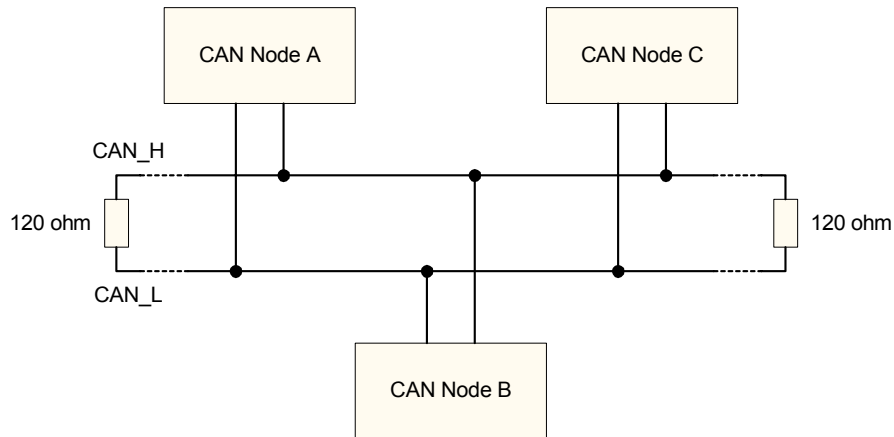


Figure 4: CAN Wiring Diagram

CAN uses a *bus topology* that is inexpensive, allows for easy node connection, and is less prone to network failures. For example, in a star network configuration, the entire network depends on a central hub. If that hub fails, the entire network stops working. A token ring network does not have a central hub, but if any of the individual nodes in the network fails, the ring is broken, and the surviving nodes on the network can no longer communicate with each other.

Non-Return to Zero vs. Manchester Bit Encoding

There are various ways to encode bits in digital systems. We describe two here, Non-Return to Zero and Manchester. Both methods have advantages and disadvantages.

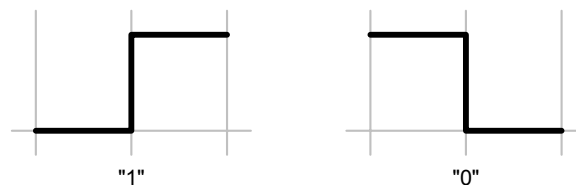


Figure 5: Manchester Bit Encoding

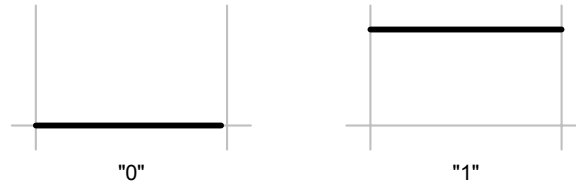


Figure 6: Non-Return to Zero Bit Encoding

The Manchester encoding scheme represents bits as transitions from '0' to '1' or '1' to '0'. Even if a frame has a string of '1's or '0's the transition is always necessary. The Manchester encoding scheme is very useful in asynchronous communication systems since there is always a signal transition for bit synchronization. The receiver can always tell when it has reached the edge, or the end, or a bit signal. The primary disadvantage of Manchester encoding is that it requires more bandwidth because each bit must have two time slots to be encoded.

Non-return to zero (NRZ) encoding does not require signal transitions to represent each bit. The signal remains '0' or '1' for the entire time slot. If a frame has a string of '1's or '0's, the signal will remain constant for as many bit times as necessary. The disadvantage of NRZ is that there is no easy way to tell where each bit starts or ends when there are more than two '1's or '0's in a row. The only way to know where a bit starts or ends is for the receiver to have a clocking source that is identical to the transmitter so that it can decipher the bit stream. This is called synchronous communication. The oscillators used with CAN controllers are accurate enough to make the large number of checkpoints provided in Manchester encoding unnecessary. By eliminating the unnecessary transitions required by Manchester encoding a CAN system can communicate at almost twice the speed for a given clock rate. However, it is very difficult to keep two oscillators exactly synchronized for very long at the bit rates used by CAN. To overcome this CAN uses a technique called *bit stuffing*.

Bit Stuffing

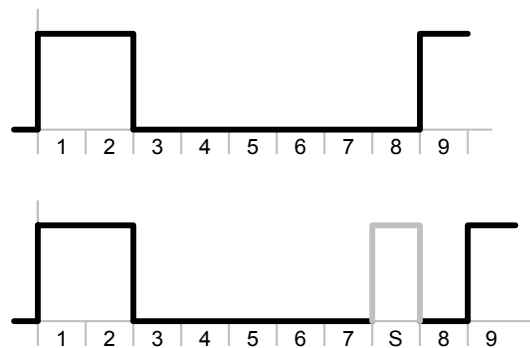


Figure 7: Bit Stuffing

Because NRZ signals can remain constant for a long period of time (that is, for a long series of '1's or '0's) it is possible that the network's individual oscillators (one per node) could become out of synch. The CAN protocol overcomes this by inserting a signal transition bit every time five identical bits appear in a row. This signal transition is called a stuff bit. The receivers use the stuff bit to synchronize their clocks. Every node in the network knows to look for stuff bits. Whenever any receiver detects five '0's or five '1's in a row, it automatically disregards the next bit that follows.

Recessive and Dominant States

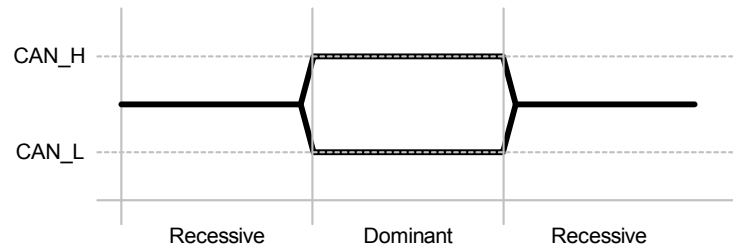


Figure 8: Dominant & Recessive States

When describing CAN signals it is common to use the terms *recessive* and *dominant* to describe the state of the bus. For a two wire bus the recessive bus state occurs when the CAN_L and CAN_H lines are at the same potential (CAN_L = CAN_H = 2.5V), and the dominant bus state occurs when there is a difference in potential (CAN_L = 1.5V and CAN_H = 3.5V). The CAN bus remains in the recessive state when it is idle. (See [“Figure 4: CAN Wiring Diagram”](#))

It is important to make the distinction between ‘1’s and ‘0’s and recessive and dominant bus states. ‘1’s and ‘0’s are useful for representing data using the binary number system, but they don’t tell you anything about the state of the bus. In fact, CAN defines ‘0’ as a dominant bus state and ‘1’ as a recessive bus state. This is somewhat counterintuitive. But the concept of dominant and recessive bus states is an especially important concept when discussing bus arbitration and CAN control fields. (See [“The CAN Data Link Layer: Bus Arbitration”](#) and [“The CAN Data Link Layer: Frame Formats”](#))

The CAN Physical Layer: Bit Timing and Synchronization

The Controller Area Network protocol uses the synchronous data transmission method. “Synchronous” means two events that are coordinated in time. For CAN Systems, this means that every node sends and receives using the same clock rate, and all of the clock rates in the network are based on a single reference point. This makes data transmissions more efficient, but it is difficult to keep any two clocks synchronized over time without some sort of reference signal. Clocks commonly lose their synchronization due to oscillator drift, propagation delays, and phase errors.

CAN nodes use two different methods to synchronize their clocks, Hard Synchronization and Resynchronization. *Hard Synchronization* occurs only once during a message transmission, at the beginning of a new message frame. Before a frame begins, the CAN bus is in a recessive (idle) state. The first bit of a frame is a Start of Frame bit that is always transmitted dominantly. Every node synchronizes its clock using the transition created by this Start of Frame bit. The clocks are not able to remain synchronized throughout the entire frame so they must continually resynchronize.

Resynchronization occurs every time the bus transitions from recessive to dominant. If there is a string of ‘0’s or ‘1’s then the CAN nodes depend on the transition created by the stuff bit to resynchronize their clocks (see [“The CAN Physical Layer: Bit Representation”](#)).

Bit Timing

Every CAN system is configured with a *bit rate*. This is the number of bits that pass a given point in a network per second. The bit rate (or data rate) is measured in thousands of bits (kilobits or Kbps) or millions of bits (megabits or Mbps) per second. The *nominal bit rate* is the number of bits per second transmitted by an ideal transmitter with no resynchronization. Every node on the CAN bus must have the same nominal bit rate. Oscillator drift and other problems can make the nominal bit rate differ from the actual bit rate, making a means to synchronize message traffic necessary.

The nominal *bit time* is the amount of time needed to transmit a single bit across the network. CAN systems use the bit time to make sure that the nodes sample the bus at the appropriate time to determine whether the bus is in a recessive or dominant state. To accomplish this, the nominal bit time is broken into segments. Each segment is divided into units called *time quanta*.

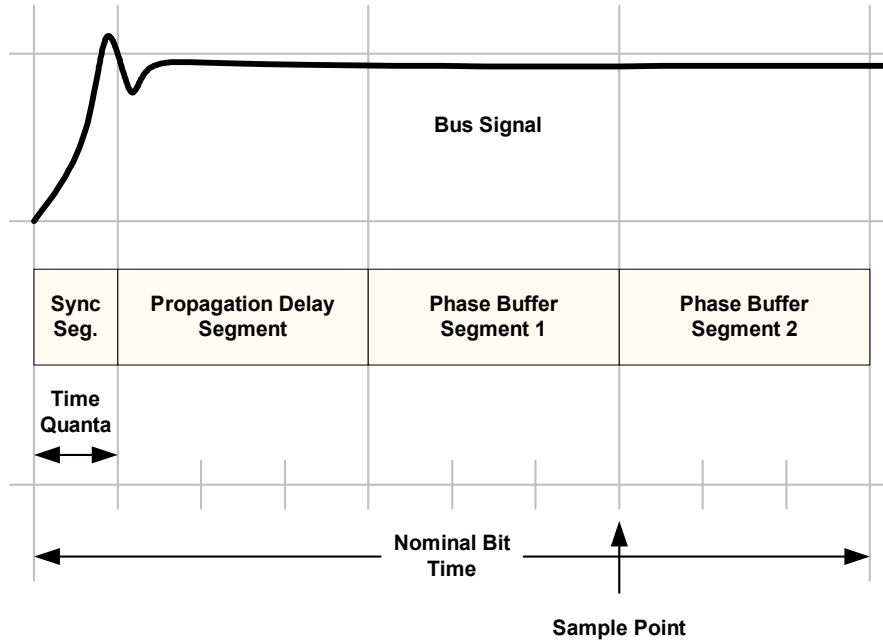


Figure 9: Bit Time Segments

These are the segments of the nominal bit time:

- **Synchronization Segment:** The Synchronization Segment is the first segment of the nominal bit time. This is where the signal transition is expected to occur. The Synchronization Segment is always fixed at one time quanta.
- **Propagation Delay Segment:** A period of time is necessary for signals to travel from one end of the bus to the other and back again. A period of time is also necessary for the electronic components to react. The Propagation Delay Segment exists to compensate for these delays.
- **Phase Buffer Segments 1 & 2:** The Phase Buffers exist to compensate for oscillator drift and other problems with timing. This is where the resynchronization takes place. CAN System designers can make the Phase Segment 1 time period longer or the Phase Segment 2 shorter to compensate for timing errors. The two segments are adjusted in units called the Synchronization Jump Width.
- **Sample Point:** The Sample Point is where the state of the bus is measured. It always occurs between Phase Segments 1 and 2. The Sample Point can be set to be either single or multiple. The CAN controller samples the bus one time in single sample mode and three times in multiple sample mode. In multiple sample mode the value is determined by majority decision. That is, if all three samples are dominant, or if two of the three are dominant, the sample is recorded as dominant.

Bit Rates and Bus Lengths

The *bus length* refers to the actual length of the wiring in a network. For a Controller Area Network, the maximum possible bus length depends on the *bit rate* or data rate. Every CAN system must trade bus length for bit speed. This is not a limitation set by the CAN protocol, but by the laws of physics. A period of time is necessary for a signal to go from one end of the bus to the other back again before the next signal is transmitted. The electronic circuitry also needs time to transmit and receive these signals. This period of time is called the Propagation Delay.

Bit Rate	Bus Length
1Mbit/s	25 m
800 kBit/s	50 m
500 kBit/s	100 m
250 kBit/s	250 m
125 kBit/s	500 m
50 kBit/s	1000 m
20 kBit/s	2500 m
10 kBit/s	5000 m

Figure 10: Maximum Bit Rate vs. Bus Length

The bit rate can always be slower than the maximum possible speed for a given bus length. Conversely, the bus length can be shorter than the maximum possible bus length for a given transmission speed. The CAN system designer needs to keep in mind that transmissions tend to become more reliable with a slower bit speed and shorter bus lengths.

The CAN Data Link Layer: Bus Arbitration

The Data Link layer performs a function called Medium Access Control (MAC) to prevent conflicts on the network. If two or more transmitters seek to send messages across the network at the same time, MAC will make sure that they are each given an opportunity to transmit one at a time so that the messages do not interfere with each other. The method of medium access control a network uses has a significant impact on performance.

Access control methods have two varieties, Determined and Random. With Determined access control the right to access the bus is defined before a node tries to access the bus, guaranteeing that no conflicts will occur. Determined access control requires either a central entity to manage network access or a decentralized agreement between nodes (such as token passing). Centrally controlled access methods are more vulnerable to system failures. One of the reasons is that if the central entity fails then the entire network fails. Decentralized determined access methods are more complex than centralized ones. It also becomes difficult to dynamically assign priority to nodes using decentralized methods.

With Random access control any node can access the bus as soon as it is idle. Most random access control methods are based on "Carrier-Sense Multiple Access" (CSMA). This means that all nodes monitor the network and wait for it to become idle. Once the network is idle all of the nodes that have a message to transmit will attempt to access the network at the same time. Of course only one node is able to transmit at a time, so a method must be found to sort out which node has priority. If the CSMA network is set up to limit or prevent collisions between messages, it is called a Collision Avoidance method. If the network is set up to allow for message conflicts, but then intervenes to detect and clean up these conflicts, it is called a Collision Detect method. With the Collision Detection method each node will check to see if the bus is clear before transmitting. If two or more nodes transmit simultaneously, the nodes that are transmitting will detect the conflict, stop transmitting, and then try to re-transmit at a randomly determined time in the future. The primary problem with the Collision Detection method occurs when there is a lot of contention on the network. Frames are constantly aborted and retransmitted which wastes bandwidth and creates long delays.

Non-Destructive Bit-Wise Arbitration

Controller Area Network systems use "*Carrier-Sense Multiple Access with Collision Avoidance*" (CSMA/CA). The CAN protocol calls this "*Non-Destructive Bit Wise Arbitration.*" It is not centralized, and it grants nodes access to the bus based on priority.

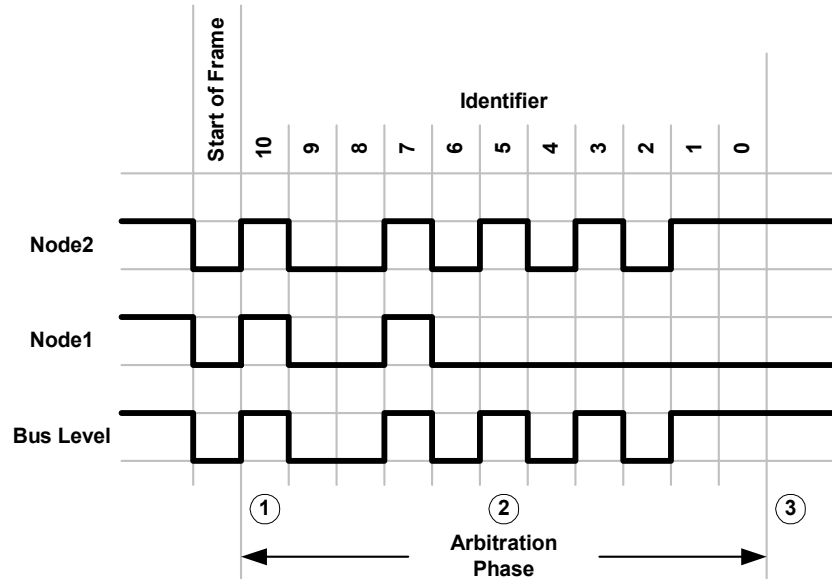


Figure 11: Example of Bit Wise Arbitration. Nodes 1 & 2 start arbitration at point 1. Node 1 yields to Node 2 at point 2, and stops transmitting. Only Node 2 can continue to transmit over the bus.

The CAN protocol controls bus traffic by allowing high-priority messages access to the bus over lower-priority messages. Every message begins with the Arbitration field, which identifies the message and determines its priority. Each node attaches this field to a message it seeks to transmit. Whenever a node transmits the arbitration field, it listens to the bus at the same time. If the node that is transmitting a recessive bit detects a dominant bit on the bus, it automatically stops transmitting and becomes a receiver of the dominant message. Once the more dominant message has been transmitted and the bus becomes idle, the less dominant nodes are able to try again. The result is that no bandwidth is wasted, there are methods a designer can use to predict the longest possible delay before any message is delivered. Note that every node in the network will look at every message transmitted on the bus. Most of the time any given node will ignore most of the messages it sees.

The CAN Data Link Layer: Frame Formats

A frame is a packet of information that contains a complete message from a transmitter.

Controller Area Network systems have four kinds of frames:

- **Data Frame.** A standard message used to transmit data over the network.
- **Remote Frame.** A message sent by a receiver to request data from another node on the network.
- **Error Frame.** A message sent out by a receiver to destroy a frame that contains errors. The Error Frame tells the transmitter to send the message again.
- **Overload Frame.** An Overload Frame is similar to an error frame. A receiver would typically send out an Overload Frame to ask a transmitter to delay the next message sent.

Data Frames

CAN systems use Data Frames to transmit data over the network. A Data Frame contains an identifier and various pieces of control information, and can hold up to eight bytes of data. CAN systems provide two versions of the Data Frame, the *Base Format* and the *Extended Format*. The Robert Bosch Corporation introduced the Extended Format Data Frame in the early 1990's as part of CAN Specification 2.0B. When CAN systems were developed for large systems with heavy message traffic, such as for buses and trucks, the existing Base Format was inadequate. The number of messages created by transmitters on the network was greater than the number of possible ID codes that the CAN system could assign to them to make sure that each message was unique. By adding a longer identifier field with 29 bits, the CAN system can create as many as 512 million different unique messages and priorities.

Base Format Data Frame

Start of Frame	1 bit
Arbitration Field	11 bits
Remote Transmission Request	1 bit
Identifier Extension	1 bit
r0	1 bit
Data Length Code	4 bits
Data Field	0-8 bytes
CRC Sequence	15 bits
Delimiter	1 bit
Acknowledgement Slot	1 bit
Delimiter	1 bit
End-of-Frame Field	7 bits
Intermission Field	3 bits

Figure 12: Base Format Data Frame

These are the fields in a Base Format Data Frame:

- **Start of Frame.** A single dominant bit marks the beginning of the Data Frame. The CAN bus is in an idle (recessive) state prior to the transmission of this bit. All receivers on the bus use this bit to synchronize their clocks (see [“Hard Synchronization”](#))
- **Arbitration Field.** The arbitration field contains the identifier field and the remote transmission request (RTR) bit. This field serves a dual purpose. It is used to determine which node has access to the bus (see [“Non-Destructive Bit-Wise Arbitration”](#)) and to identify the type of data the message contains.
 - **Message Identifier.** The Message Identifier is 11 bits long in the Base Format Data Frame. This means that 2048 (2^{11}) unique messages are possible. The lower the value of this field, the higher the priority.
 - **Remote Transmission Request (RTR) bit.** The RTR bit indicates whether the frame is a Data Frame or a Remote Frame. This bit must be dominant for a Data Frame.
- **Control Field** The Control Field contains the Identifier Extension bit, a reserved bit (r0), and the Data Length Code.
 - **Identifier Extension (IDE) bit.** If the IDE bit is dominant, then the frame is a Base Format Frame. Otherwise the frame is an Extended format frame (see [“Extended Format Data Frame”](#)).
 - **Reserved bit 0 (r0).** The reserved bit is not used and should always be dominant.
 - **Data Length Code.** The number of bytes is stored in the Data Length Code field. This field typically holds values from zero to eight. Since the field is four bits long values greater than eight can be indicated. If the value of the Data Length Code is greater than eight then it is assumed that the frame contains eight bytes.
- **Data Field.** The Data Field is the payload of the Data Frame. It contains from 0 to 8 bytes and is transmitted most significant bit first. This is the only field in the Data Frame that does not have a fixed length.
- **Cyclic Redundancy Check (CRC) Field.** The CRC field is 15 bits long. The receiver uses the value in this field to see if the data bit sequence in the frame was corrupted during delivery. (See [“The CAN Data Link Layer: Error Detection and Handling”](#))
- **Acknowledgement Field.** The Acknowledgement field is two bits long. It contains the Acknowledgement Slot bit and the Acknowledgement Delimiter Bit. The node that transmits the Data Frame sends these two bits recessively. The transmitter expects at least one receiver to acknowledge the receipt of the transmitted message. Any node that receives the message without error will overwrite the Slot with a dominant bit. The Acknowledge Delimiter bit always remains recessive to distinguish a positive acknowledgement from the start of an Error Frame (see [“Error Frame”](#)). If no receiver acknowledges a message, the transmitter will continue to try to send the message until the node turns itself off (see [“Bus Off”](#) state).
- **End-of-Frame and Intermission Fields.** Every Data Frame ends with a seven-bit End-of-Frame Field and a three-bit Intermission Field.
 - **End-of-Frame Field.** The End of Frame Field must be transmitted recessively to mark a completely error-free transmitted frame. If the Acknowledgement Delimiter bit or any of the End-of-Frame bits are transmitted dominantly then this marks the beginning of an Error or Overload Frame (see [“Error Frame”](#) or [“Overload Frame”](#))
 - **Intermission Field.** The Intermission Field represents the minimum amount of spacing between adjacent Data or Remote Frames. All three bits must be transmitted recessively. The bus may continue to remain idle after this, or a new frame will be indicated with the dominant Start-of-Frame bit. If one of the first two bits of this field are dominant then it is assumed an Overload Frame has begun (see [“Overload Frame”](#))

Extended Format Data Frame

Start of Frame	1 bit
Base Message Identifier	11 bits
Substitute Remote Request	1 bit
Identifier Extension	1 bit
Extended Message Identifier	18 bits
Remote Transmission Request	1 bit
r1	1 bit
r0	1 bit
Data Length Code	4 bits
Data Field	0-8 bytes
CRC Sequence	15 bits
Delimiter	1 bit
Acknowledgement Slot	1 bit
Delimiter	1 bit
End-of-Frame Field	7 bits
Intermission Field	3 bits

Figure 13: Extended Format Data Frame

The Extended Format Data Frame is nearly identical to the Base Format Data Frame. The only differences between these two formats are found in the Identifier Extension (IDE) bit in the Control Field, and the size and arrangement of the Arbitration Field. Both the Base Format and the Extended Format can coexist in the same CAN system. The rule is that Base Format frames always have priority over Extended Format frames. (See [“Base Format Data Frame”](#))

The following is a description of the Arbitration and Control Fields:

- **Arbitration Field.** The Arbitration Field in the Extended Format Data Frame is longer to accommodate the 29-bit identifier. Over 536 million (2^{29}) unique messages are possible using the Extended Format.
- **Base Message Identifier.** The Base Message Identifier is identical to the Message Identifier Field in the Base Format Frame. It is 11 bits long and contains the most significant bits of the 29-bit identifier.
- **Substitute Remote Request (SRR) Field.** The SRR bit replaces the RTR bit in the Base Format. Its sole purpose is to be a placeholder so that the Identifier Extension bit remains in the same location as it is in the Base Format. This bit is always transmitted recessively. The RTR bit for the Extended Format has been moved to the end of the Arbitration Field.
- **Identifier Extension (IDE) bit.** If the IDE bit is recessive, then the frame is an Extended Format Frame otherwise the frame is a Base Format Frame.
- **Extended Message Identifier Field.** The Extended Identifier Field adds another 18 bits on to the Base Message Identifier. This brings the total number of identifier bits to 29.
- **Remote Transmission Request (RTR) bit.** The RTR is identical to the RTR bit in the Base Format Frame. It indicates whether the frame is a Data or Remote. This bit must be transmitted dominantly for a Data Frame.
- **Control Field.** The Control Field in the Extended Format Frame contains two reserved bits and the Data Length Code (DLC). The reserved bits are unused and must be transmitted dominantly. The DLC is identical to that used in the Base Format Frame (see [“Base Format Data Frame”](#)).

Remote Frame

Remote Frames are used for receivers to request information from another node. They are often sent out on a regular schedule, to draw updates from sensors. The format for a Remote Frame is identical to that of a Data Frame. Both frame types also feature base and extended formats, and both have a single Remote Transmission Request bit at the end of the Arbitration Field. With a Remote Frame, this bit is transmitted recessively to identify it as a Remote Frame. For Data Frames the RTR bit is always transmitted dominantly. The RTR bit is considered part of the bitwise arbitration so a Data Frame will always dominate over a Remote Frame with the same identifier. This is reasonable since a request for data should not take precedence over the data being requested.

Error Frame

Receivers send out Error Frames whenever they detect that a frame contains an error. The Error Frame can be sent during any point in a transmission and is always sent before a Data Frame or Remote Frame has completed successfully. The transmitter constantly monitors the bus while it is transmitting. When the transmitter detects the Error Frame, it aborts the current frame and prepares to resend once the bus becomes idle again.

The Error Frame contains the following fields:

- **Error Flag Field.** The Error Frame deliberately violates the bit stuffing rule by sending either six recessive bits or six dominant bits in the Error Flag Field. The determination of whether the Error Flag is recessive or dominant depends on the Error State of the node (see [“Fault Confinement”](#)).
- **Error Delimiter Field.** The Error Delimiter is a sequence of eight recessive bits and indicates the end of the frame. The bus will then enter an idle state or a new Data Frame or Remote Frame will start

Note that not every node in a network is permitted to send out Error Frames. The CAN protocol has a method of keeping faulty nodes from participating on the bus (see [“Fault Confinement”](#)).

Overload Frame

The Overload Frame can be considered a special form of the Error Frame. It is used to ask a transmitter to delay further frames or to signal problems with the Intermission Field (see [“Base Format Data Frame”](#)). The Overload Frame has the same format as the Error Frame but unlike the Error Frame it does not cause the retransmission of the previous frame. If the Overload Frame is being used to delay further transmissions, then no more than two Overload Frames can be generated successively.

An Overload Frame includes a Flag Field that contains a sequence of six dominant bits followed by a Delimiter Field, a series of eight recessive bits. When one node sends out an Overload flag all of the nodes on the network detect it and send out their own overload flags, effectively stopping all message traffic on the CAN system. Then, all of the nodes listen for the sequence of eight recessive bits. The maximum amount of time needed to recover in a CAN system after an Overload Frame is 31 bit times.

The CAN Data Link Layer: Error Detection and Handling

The Data Link Layer in Controller Area Network systems is very effective in detecting errors. Every frame is simultaneously accepted by every node in the network or rejected by every node. If a node detects an error it transmits an error flag to every other node in the network and destroys the transmitted frame. We describe error detection and fault confinement with CAN systems below.

Error Detection Mechanisms

Controller Area Networks use the following methods to detect errors:

- **Bit Check.** Every node monitors its own messages as it transmits them across the bus, checking the bits in each frame for errors. The node will detect a bit error if it sends out a dominant bit when it was supposed to transmit a recessive bit, or a recessive bit when the message called for a dominant bit.
- **Frame Check.** Each type of frame contains specific bit fields that always have fixed values. For example, the first bit in every frame must always be a dominant bit. Every node monitors these fixed fields in every frame they receive. A node will detect a frame error if a fixed bit field contains an illegal bit value. Errors of this type are called Form Errors.
- **Cyclic Redundancy Check.** Cyclic redundancy checking is a very effective method of looking for errors in a network by applying a polynomial equation to a block of transmitted data.

A polynomial is a complex mathematical expression where a series of values are added together, and each value includes a variable or variables raised to a power and multiplied by a coefficient. This simple polynomial has one variable:

$$a_n x^n + a_{n-1} x^{n-1} + \dots + a_2 x^2 + a_1 x^1 + a_0 x^0$$

Here a is the coefficient and x represents the variable.

The node calculates the cyclic redundancy code (CRC) for any message it sends, and then attaches the resulting value to a field at the end of the frame. The node that receives the message applies the same polynomial to the frame and compares its result to the CRC code provided by the sender. If the two values do not match, the receiver sends an error message to the transmitter. Cyclic redundancy is a very reliable method for looking for errors in transmissions.

- **Acknowledgement Check.** Every node that transmits a message listens for an acknowledgement in the ACK slot from at least one other node in the network (see [“Base Format Data Frame”](#)). If the transmitter does not receive a response then this is considered an acknowledgement error. The node will continue to retransmit the frame until it receives an acknowledgement.
- **Stuff Rule Check.** Every node checks for a violation of the bit stuffing rule. If a node detects more than five identical bits in a row then a Stuff Error has occurred and an Error Frame is sent. The node records a bit stuffing error, and sends out an Error Frame to the transmitter.

Fault Confinement

A major risk with any serial bus system is that a single defective node can shut down the entire network. To deal with this the CAN protocol is designed to automatically detect a faulty node and disconnect it from the network. CAN requires two error counters for every node, one to keep track of transmit errors and the other to keep track of receive errors. When a transmission or reception error occurs, the respective counter is increased by a weighted value depending upon the type error and which node caused the error. For every successful transmission or reception the respective counter is decreased by one. Generally, if a node recognizes that it is the source of the error

then the counter is incremented by nine for receive errors and eight for transmission errors. Otherwise, each counter is increased by one.

Based on these counters, a node can be in one of three states:

- **Error Active.** The node is working normally. It takes part in network communications and sends an error flag when it detects an error, destroying the transmitted frame.
- **Error Passive.** One of the node's error counters has exceeded 127. The node can still send and receive messages on the bus, but it is no longer permitted to send out Error Frames with active error flags. An Error Passive node can only respond to an error by sending out an Error Frame with a passive error flag (see "[Error Frame](#)"). Also, if an Error Passive node sends out a Data Frame with an error, it must wait for eight bit times before it tries to retransmit. Both of these measures restrict nodes that have a higher than average error rate so that they can't interfere with communication across the bus. A node can become error active again when both error counters fall below 128.
- **Bus Off.** The CAN node that is bus off is not allowed to influence the bus in any way. It is only allowed to receive messages. A node can only become bus off due to transmit errors. No amount of receive errors can put a node into the bus off state. A node enters the bus off state when the transmit counter exceeds 255. This means that a faulty node will be turned off if it transmits 32 consecutive messages with errors. A node that has entered the bus off state can only leave this state by resetting the counters and CAN controller, and then only after it has detected 128 sequences of eleven consecutive bits. This allows 128 frames to be transmitted undisturbed if the node continues to fail after being reset.

The rest of the network will continue to work even if a node is set to Bus Off. Note that errors sometimes appear in a network because of outside factors, such as if the network is exposed to temporary electromagnetic interference. CAN systems are therefore designed so that troublesome nodes can reduce their error counts if they successfully send and receive a series of Data Frames over time.