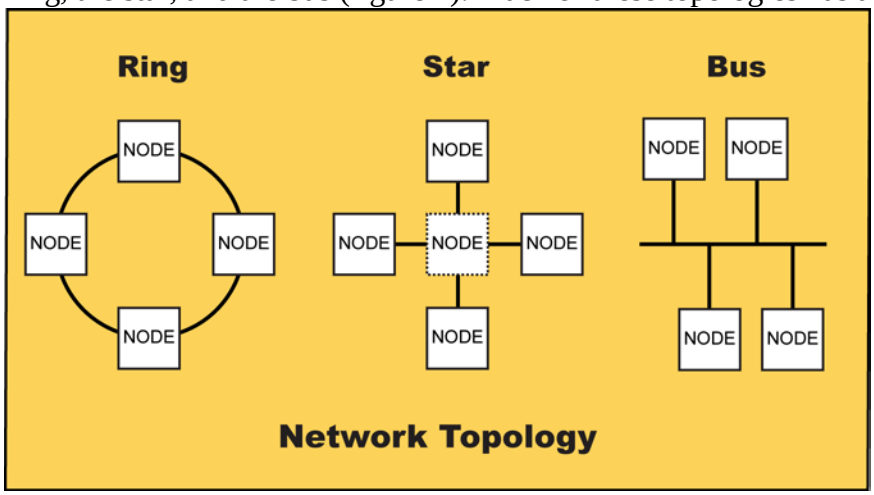
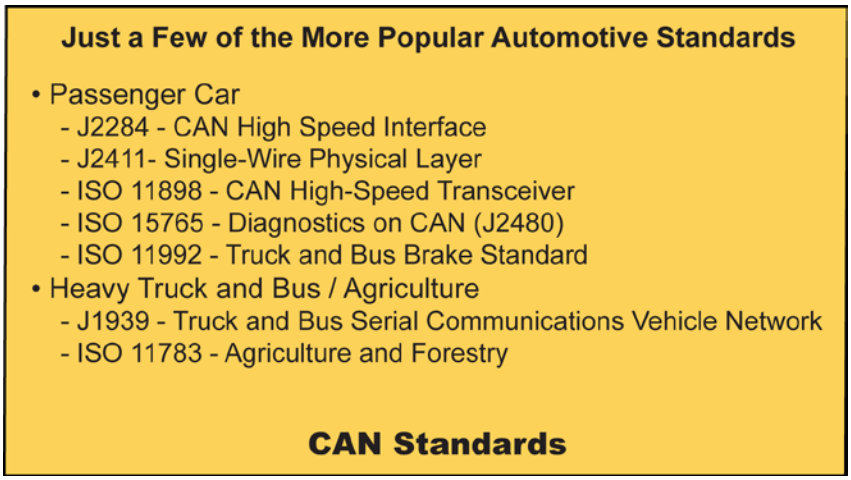
UNDERSTANDING THE CONTROLLER AREA NETWORK (CAN)

The unsuspecting troops had come under heavy enemy fire just before dawn and the garrison was caught totally by surprise. The fort commander had been awakened by the gun fire and issued a message to be dispatched immediately. The message was taken to the telegraph office where an elderly telegraph operator furiously typed out the message in the format of on and off electrical pulses called Morse code. This coded message went out on the telegraph wire where other telegraph operators listened and then went to work decoding the on and off pulses. This example may seem outdated, but the truth is that something very close to this scenario occurs in the modern automobile. If we were to alter a few words such as changing the fort commander to microprocessor, the telegraph operator to transceiver chip, the telegraph wire to bus network and Morse code to controller area network (CAN); it will become clear what each component of the CAN contributes to the operation of the system. The CAN protocol was developed by Bosch in the early 1980's for automotive in-vehicle networks. The first vehicle in which the CAN language was used was the 1991 Mercedes S class. The CAN protocol has emerged over the last seventeen years as the dominant automotive programming language as seen in the Federal law that will be implemented in 2008 model year vehicles in which all vehicles sold in the US must use the CAN protocol to communicate with emission related microprocessors. This CAN standard is ISO15765. It is important to be aware that a vehicle platform can have several different protocols and several different network topologies being used on the same vehicle. It will be imperative to know which system topology and which protocol speeds are being used on the vehicle before you start to diagnose it. The network topology can be laid out in three distinct configurations; the ring, the star, and the bus (figure 1). Each of these topologies has advantages and disadvantages in their use. For instance in the ring topology, if the communications wire becomes shorted or broken on dual ring networks, all the modules or nodes can still communicate. This is a closed architecture network making it difficult to add nodes in the field. In the star topology, if the communication wire becomes shorted or broken only one node will be lost. The problem with the star topology is the center node is the master and the other nodes are referred to as slaves. If the master node goes down, the entire network loses communication. The CAN network uses a bus topology and, if the communication wire becomes shorted or broken in the network, it will maintain communications as long as a dual wire bus is used. The CAN bus topology uses a multi-master system where each node is self sufficient.

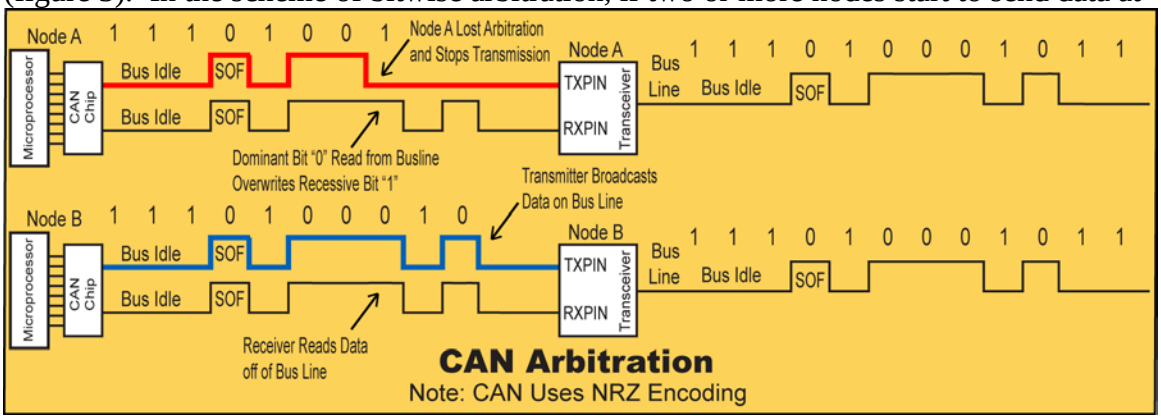


This means that each node has a crystal oscillator and can control the voltage levels on the bus. If one node fails, this bus network configuration will still function. CAN is a language and like Morse code, it uses on and off electrical pulses against time to convey data messages. These data messages are comprised of digital high and low voltage changes that are read as a binary code. This binary code is made up of 1's and 0's. The way in which the 1's and 0's are organized against time is what makes each network protocol different. Just as in different spoken languages, such as English and German, the same alphabet is used. What makes each of these languages unique is the way in which the letters and spaces are organized. In both of these spoken languages the organization of the letters and spaces will have rules applied to them that will have to be followed. Network protocol languages will also have certain rules that will govern them. These regulations will be set by the Society of Automotive Engineers (SAE) and the International Standards Organization (ISO). The regulations mandated by these organizations will insure that a system standard will be followed so that all systems under the standard will operate and work together. When Bosch wrote the CAN language, no physical layer or transfer media were given. This open architecture was done on purpose to allow the CAN language to be more versatile, thus making it more powerful. By not confining the use of the CAN language to a set physical layer, the engineering teams can be more creative. This is why CAN is able to run at various speeds from 1-1000 kilobytes per second (kbps) and to be transmitted on different physical media such as 1 wire, 2 wire, twisted pair, shielded twisted pair, coax, or fiber optic. The CAN network can also utilize different voltage levels to control the bus communications. To make sure that each of these CAN embodiments will work correctly, the SAE and ISO wrote standards for each of the various speeds and physical layers (figure 2). CAN protocol has been divided into three groups based on the transmission/receive rates. These rates are based on how many bytes can be transmitted in one second. CAN A is a low speed CAN network that operates at 10kbps. CAN A is used in a low cost network scheme that will control body functions such as seat nodes, door nodes, mirror nodes, etc. These systems are not safety critical so the transmission speed is not crucial. CAN B is a medium speed CAN network that can operate from 33kbps to 250kbps. This would be used in body control functions or diagnostic functions which need to transmit larger amounts of data. CAN C is a high speed CAN network that operates from 250kbps to 1 megabytes per second (mbps). This would be used in safety critical systems such as air bag, ABS, traction control, power train, etc. The network data transmission speed costs money. The faster the data transmission speed the greater the cost.

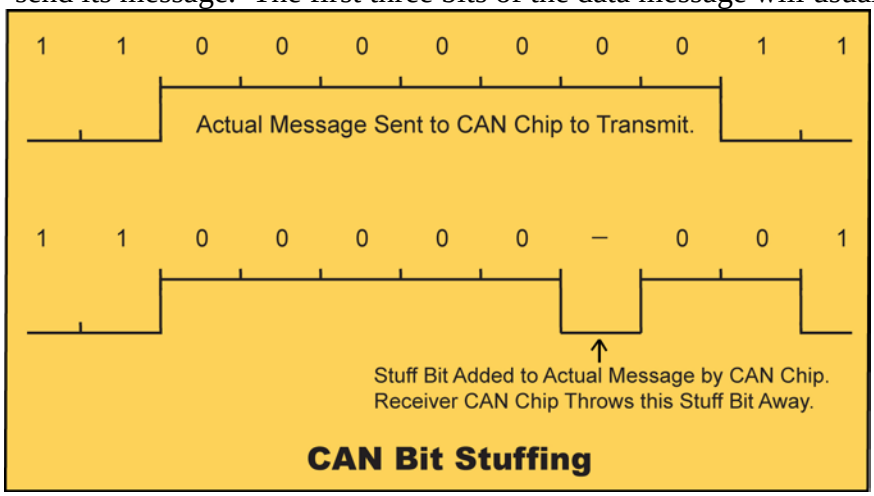


Since the cost of these networks will raise the cost to manufacture the vehicle, the appropriate CAN bus will be utilized. What this means is there may be three different CAN networks operating at the same time in the same vehicle but at different speeds. It is also possible that other network protocols and media types as well as the CAN protocol may be in operation on the same vehicle. CAN A, CAN B, and CAN C run at different speeds, however each of these can be implemented on the same vehicle. If all the CAN speeds were to run on the same wire with this much variation in speed, there would be data collisions. Think of this as a single lane highway with VW Beetles driving at 10mph and Formula 1 cars overtaking the VW Beetles at a speed of 250mph. This would insure that collisions would occur. To avoid this problem each network is isolated to the protocol and speed at which the network is transmitting and receiving. Now that each network protocol and speed is contained on its own isolated network, the collisions from running at various speeds will be avoided, but the networks have been isolated and there will be no way for these networks to share information. One example of this would be the ABS node. The ABS node receives the vehicle speed from the wheel speed sensors and transmits these data on the high speed bus. The power train control node could utilize this information because it operates on the same high speed bus; however, the instrument panel will also need this data message in order to display the vehicle speed. The instrument panel is on the low speed bus and can not receive this information directly. In order for data messages to be shared between different networks on the same vehicle, a special node is used. This special node contains all of the transceivers for the different networks that are utilized on the vehicle. In this way, the node can receive a high speed message on the high speed transceiver and convert this data message out to the low speed transceiver. This node is called a gateway and it enables the transfer of data messages between the different networks on the vehicle. A single vehicle can have several gateways to help transfer data messages between different networks. There are several different configurations with which a bus network can use to communicate. The first style is a token slot protocol. In this protocol, a token is passed between all the nodes on the bus from one node to the next. When a token is received by the node it can send a data message out on the bus. Once the message has been sent the token is passed to the next node so it can send its message out on the bus. In this token scheme, an urgent message will have to wait until a token is received by the node needing to send the urgent message. This creates a long latency period that is undesirable. Another bus scheme is random access. Random access is what the Ethernet uses. This protocol also has long delay that is also undesirable. Yet another bus scheme is CSMA/CD. This is the bus network scheme that CAN uses. CS stands for carrier sense (listen until the network is idle and it must wait if another node is transmitting). MA stands for multiple access (multiple nodes can access an idle bus at any time). CD stands for collision detection (method of resolving data collisions). With up to 40 nodes operating on one CAN network, it will be imperative that an urgent data message is given priority over a normal data message by taking control of the bus. An example of this would be a single telephone with a line of 40 people waiting to talk on it. The person currently on the phone is calling to see if his laundry is ready for pickup. This person now puts the phone down and the next person in line is going to call home to let them know they are on their way. The last person in line needs to call 911 to report an accident with injures, thus this message is deemed urgent and must be sent next.

To make sure that urgent messages are sent in the correct order, they must be prioritized. In this example, the 911 call would come before any other call. If two 911 accident calls needed to be made at the same time, the next digit would indicate if there were injuries. An accident with injuries would be assigned a '0' and accident without injuries would be assigned a '1'. So, 9110 would take a higher priority than 9111. The accident with injuries in this example would take control of the phone line. This is similar to the way the CAN protocol prioritizes data messages so that each node on the bus can access the bus at any time; however, the node with a higher priority will take control of the bus to send its data message first. This style of communication is called bitwise arbitration (figure 3). In the scheme of bitwise arbitration, if two or more nodes start to send data at

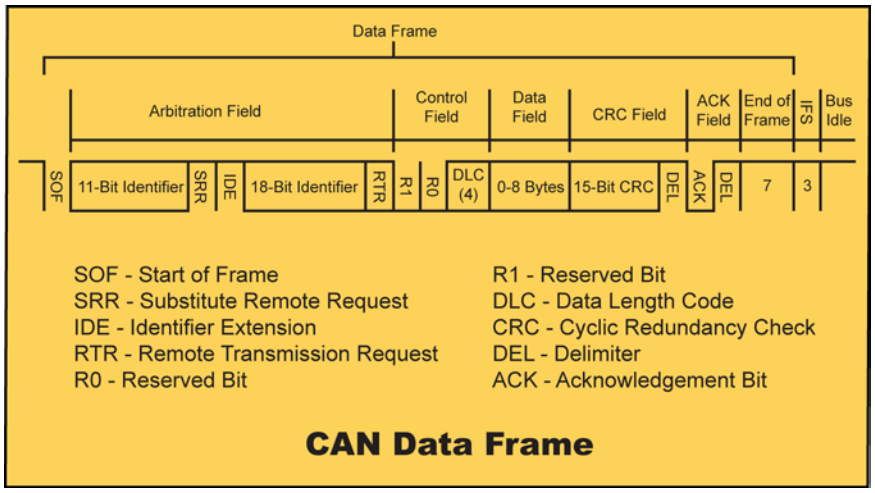


exactly the same time, the node sending the higher priority message will take control of the bus. This is accomplished by the priority of the message containing more '0' than '1'. '0' has a higher priority and is the dominant bit. '1' has a lower priority and is the recessive bit. If node A's message had the first two bits at '0' and the next bit at '1' and node B's message had the first three bits at '0', node B would have a higher priority assigned and it would win the bitwise arbitration and take control of the bus. Thus, node B would send its data message first. Node A would have to wait and then try to send its message after node B was done. If the bus were idle then, node A could send its message. If another node needed to send a message and had a higher priority, it would win the bitwise arbitration and then node A would have to wait again before trying to send its message. The first three bits of the data message will usually determine the outcome of the bitwise arbitration and therefore, are assigned the main priority of the message. A wakeup call is assigned the highest priority and it will start with three '0's'. A scan tool is assigned the lowest priority and will start with three '1's'.



During the transmission of the data message, the voltage level is changing between a high voltage state, indicating a ‘1’, and a low voltage state, indicating a ‘0’. The fewer voltage transitions that take place within the data message the better in order to limit radio-frequency interference (RFI). To minimize the RFI, the CAN protocol uses a method referred to as non-return to zero (NRZ). In this format, the voltage level does not have to change for each bit, but it can stay at a low voltage level, ‘0’, or a high voltage level, ‘1’ for up to five bits of length. If the data message is held at a given voltage level for more than five bits, the CAN chip will force a change in the voltage level opposite to the 5 bit level (figure 4). This is called bit stuffing and is done so all the nodes on the bus can resynchronize to the rising or falling voltage edge. This bit stuffing also assures that the bus voltage level has not become stuck. This will help with the CAN diagnostics.

When a complete data message of ‘1’s’ and ‘0’s’ is completed, it is called a data frame (figure 5). The data frame is composed of the start of frame, arbitration field, control field, data field, cyclic redundancy check field, acknowledgement field, and end of frame. The CAN data frame can either be an 11 bit identifier or have a frame extension making it a 29 bit identifier. Most CAN vehicles now on the road use the 11 bit identifier. Newer CAN vehicles are taking advantage of the extended 29 bit identifier because more information than just the priority and identification can be conveyed in the arbitration field. The 29 bit’s can also contain information and source coding. Whenever a data frame is constructed and sent, it will be important for the data being transmitted and received to be correct. In order for this to be checked, the CAN protocol has incorporated an error detection and protection scheme. This is done with a process called cyclic redundancy check or CRC. The CRC is a 15 bit extension that is added on to the data frame. This extension makes the data frame divisible evenly by a mathematic equation called a polynomial. This relationship between the message and the CRC will produce a known value. If the value is anything other than the expected value, an error is marked against the message and the message is discarded. Therefore, any noise or disruption in the data frame will be caught and dealt with by the CAN error detection. This CRC function is performed by the transmitter and receivers located in the CAN chip’s hardware. The transmitting node uses it’s CAN chip to provide the outgoing bit stream with a CRC. At the end of the CRC, the transmitting node checks the CRC value before it stops transmitting. In other words, the transmitting node is checking itself. If the CRC numerical value is incorrect the transmitter sends out an error frame by holding the bit stream down for more than 5 bits.



Any receiving node on the bus can also send an error frame in this manor. This violates the bit stuffing error so all of the nodes on the bus tag this data message as an error and discard it. The transmitting node then retransmits the data message. The CAN protocol determines which node is responsible for an error by penalizing the node that reports the first error. This node increases it's receive error counter by a greater amount than the other nodes. This process is basically "shooting the messenger". The error indicators are watched by each node. In the case of intermittent errors, the CAN protocol will deal with these and will only add a slight latency. In these cases, the CAN error recovery is very good. In the case of persistent problems where the errors counter exceeds a programmed percentage threshold, the CAN protocol will require the transmitter with the most error counts to get off the bus (bus-off state). Once the node has kicked its self off the bus, the only way the node can get back on the bus is if the bus has enough idle time, which is not likely, or if a hardware reset is done. It is important to realize that if a competing node does not stop transmitting once it loses bitwise arbitration, or starts transmitting when another node has control of the bus or if noise interrupts during the message; the data message will be corrupt. The CRC will show a message error and this error will be assigned to the transmitting node. The problem here is the transmitting node is not the problem but it will be assigned the fault. This can set a diagnostic trouble code (DTC) that would not indicate the true problem in the CAN network. Be aware that the scan tool becomes another node on the bus. If the scan tool has a problem it can corrupt data messages and be the cause of false DTCs. Once the data message is sent an acknowledgement or ACK from another node on the bus is sent. If no ACK is received in the data frame, the transmitting node assumes no other node on the bus received the message. This could indicate to the node that the bus is open. Another way CAN uses to confirm the message was received is to have a response generated from the receiving node. On a high speed class C CAN network, if a message is sent to a specific node, a response from that node is expected. If the receiver of the message does not respond, a DTC is set for the receiving node. A node that is experiencing an acknowledgement failure will not get kicked off of the bus. As soon as the node can communicate on the bus again it will recover from the acknowledgement error. In low and medium speed CAN A or CAN B networks, a response is not expected. If a message is sent to a specific node, no receiving response is generated. In this scheme "no news is good news". What can occur is that if a node on these style networks fails quickly, no messages will be sent out, thus, no errors for the failing node will be counted. Since there is no response expected, it is possible for a node to be completely inoperative and to have no codes to indicate that a problem exists. One such node that could fail in this manner would be a driver information module or DIM. It will be important to study the CAN standards and have a good understanding of which network system you are working on. As more problems arise in vehicles with complex networks, the repair of these systems will become common place. So, have your knowledge and service bays ready.