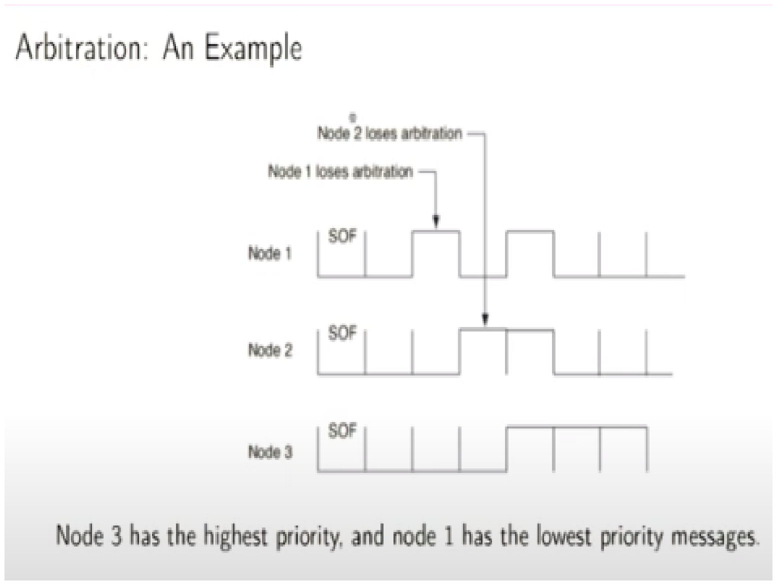
Foundations of Cyber Physical System

Prof. Soumyajit Dey
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture No # 09
Module No # 02
Real Time Sensing and Communication for CPS (Contd.)

Welcome back to the lecture series on foundations of Cyber Physical Systems.

(Refer Slide Time: 00:36)

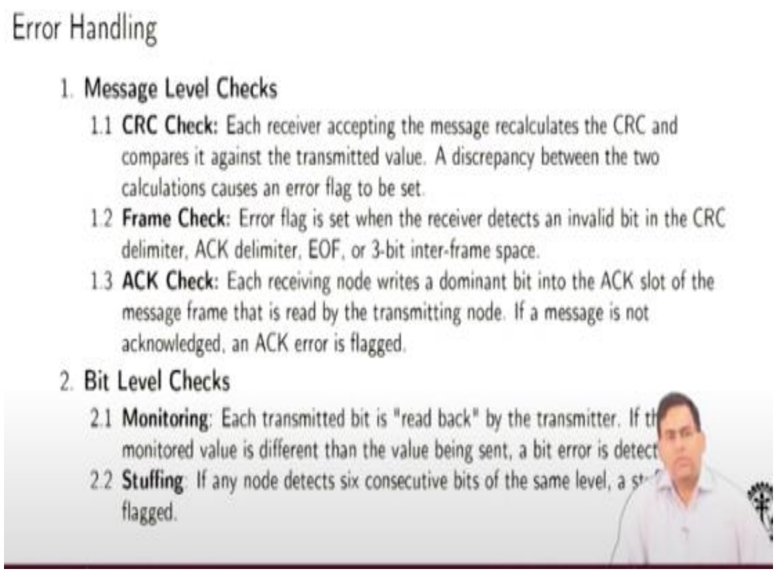


Arbitration: An Example

Node 3 has the highest priority, and node 1 has the lowest priority messages.

So, if you remember in the previous lecture, we discussed about the arbitration scheme that we have in CAN bus. So, with that we will now move to some other properties here.

(Refer Slide Time: 00:43)


Error Handling

1. **Message Level Checks**
   1.1 **CRC Check:** Each receiver accepting the message recalculates the CRC and compares it against the transmitted value. A discrepancy between the two calculations causes an error flag to be set.
   1.2 **Frame Check:** Error flag is set when the receiver detects an invalid bit in the CRC delimiter, ACK delimiter, EOF, or 3-bit inter-frame space.
   1.3 **ACK Check:** Each receiving node writes a dominant bit into the ACK slot of the message frame that is read by the transmitting node. If a message is not acknowledged, an ACK error is flagged.
2. **Bit Level Checks**
   2.1 **Monitoring**: Each transmitted bit is "read back" by the transmitter. If th monitored value is different than the value being sent, a bit error is detect
   2.2 **Stuffing**: If any node detects six consecutive bits of the same level, a st flagged.

For example, what are the different error handling schemes that you have in CAN bus. So, there are several checks for example, these are things we have already described while we define the CAN packets. And now we will just point out the different checks that were there. For example, you have this CRC check. So, if receiver will accept the message and by looking at the message payload it will recalculate the CRC using the known polynomial between both the sender and the receiver.

And, it will compare the value and if you see that the CRC that is computed is different, then, it knows that the original value change from X to some X prime and it will accordingly now send an error flag. Now whenever the receiver will detect an invalid bit in the CRC delimiter then also it will send an error flag. And similarly, if it files find such a thing happening in any such well-defined space like the ACK delimiter or the end of file.

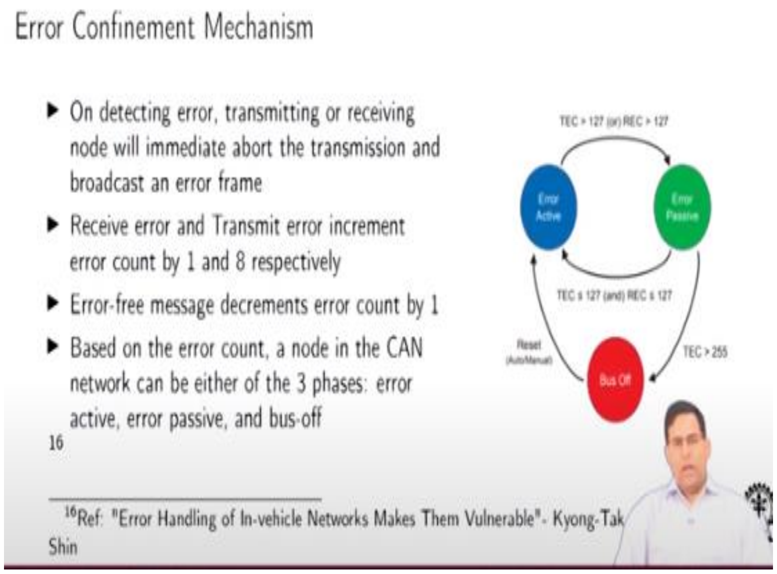
So, if you remember the CRC the ACK delimiter and the end of file, they have well-defined things that they are going to be of 1 bit or 5 bit and they will be recessive. So, if there is any violation of those things it will be sent as an error flag, right. Similarly, there is a 3-bit inter-frame space. So, if that is not maintained that is also an error. And finally, you also have the ACK check. That means each receiving node will write a dominant bit into the ACK slot.

So that means whenever the transmitter says the acknowledgment is a plot, the receiver will set it to be dominant, right. So that is a switch in signal value that the transmitter can see and if we cannot see that it will go to a retransmission, right. So that then also an acknowledgment error will be flagged. There are also some bit level checks, that means, so these are the previous things are at the packet payload level.

Now, at the bit level if you remember the property of CAN that whenever a bit is being transmitted it is simultaneously also read by the receiver. So, if we can see that well whatever I am transmitting the bus is not having that value then that is a bit level error that will be immediately detected. And also, you have this bit-stuffing which will mean that whenever there are 6 consecutive bits of the same level. That is not allowed, right, due to the bit-stuffing rule, which will which is supposed to change the polarity there.

So, if that is not maintained then a stuffing error at the bit level will also be flagged. So, all these things are there and they should be able to prevent the CAN data getting corrupt due to some random noise or due to some random attack happening on the system through this kind of error handling scenarios.

(Refer Slide Time: 03:44)



Error Confinement Mechanism

- On detecting error, transmitting or receiving node will immediate abort the transmission and broadcast an error frame
- Receive error and Transmit error increment error count by 1 and 8 respectively
- Error-free message decrements error count by 1
- Based on the error count, a node in the CAN network can be either of the 3 phases: error active, error passive, and bus-off

16

[16]Ref: "Error Handling of In-vehicle Networks Makes Them Vulnerable"- Kyong-Tak Shin

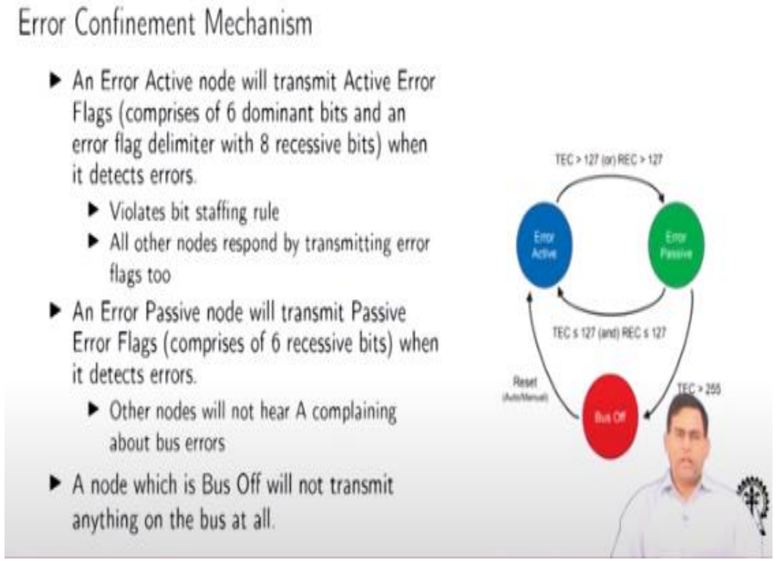
Now what happens if there are errors? So, these are designed to be very robust systems, right. So, that means if there are 1 or 2 errors, the system is supposed to tolerate them, right. And in each CAN node there is a transmission error count and there is a reception error count, the TEC and REC count that is maintained. So, on detecting the error this node whether, the transmitting node or the receiving node, they will immediately abort that particular transmission and they will broadcast an error frame.

And the respective nodes, the transmit node and the receiving node will increment respectively the transmit error flag and the receiver flag, the error counts by different values. Now the transmitter count increases by step of 8 and the receiving error count will increase by a step of 1, as per the protocol.

Now, after this, if there is an error free message transmission as well as reception, then this TEC and REC values both will decrement by 1. Now, so these are the rules based on which the transmission and the reception error values increase or decrease. And based on those values the system can be in 3 possible states in, in the error node. It can either be in an error active state it can be an error passive step. So, error active statements it is normal then the transmission and reception errors are below 127.

If any of them is greater than 127, then they go to error passive mode, where they have they are highlighting that they will send an error frame. But at the same time, they also have the ability to transmit other messages although they are sending an error frame which is kind of a signal that well I have got I am often getting lot of errors, okay. However, if this transmission error count of some particular node goes beyond 255, then it is temporarily taken off the CAN network and it goes to what we call as the bus-off mode.

(Refer Slide Time: 05:50)



Error Confinement Mechanism

- An Error Active node will transmit Active Error Flags (comprises of 6 dominant bits and an error flag delimiter with 8 recessive bits) when it detects errors.
  - Violates bit staffing rule
  - All other nodes respond by transmitting error flags too
- An Error Passive node will transmit Passive Error Flags (comprises of 6 recessive bits) when it detects errors.
  - Other nodes will not hear A complaining about bus errors
- A node which is Bus Off will not transmit anything on the bus at all.

So just repeating the previous states in an elaborate way, an error active mode will transmit and the active error flags. So basically, there is like we said that, in this node I am doing my normal transmissions, but if there are some errors happening. Then I will transmit this error flags which is basically 6 dominant bits and the error flag delimiter which is 8 recessive bits so that is a set pattern, and this pattern represents the error flag.

Whenever it details the following kinds of error like violating of bit-stuffing rule. And then all other nodes are suspended by and it will just. 1 minute. So, it will just send, see that whether this bit patterns are not following that the standard bit-stuffing rule of CAN. And in that case all other nodes will respond by transmitting the error flags too. So, that is the error active mode.

Let me. And in the error passive mode, they will transmit the passive error flag. So, as you can see that the, whether I am in the active mode or in the passive mode, it is kind of represented by what kind of error flag I am sending. In the error active mode, I am sending 6 dominant and 6 and 8 recessive bits, and in case of the error passive mode I am sending 6 recessive bits instead of the 8 recessive bits, right.

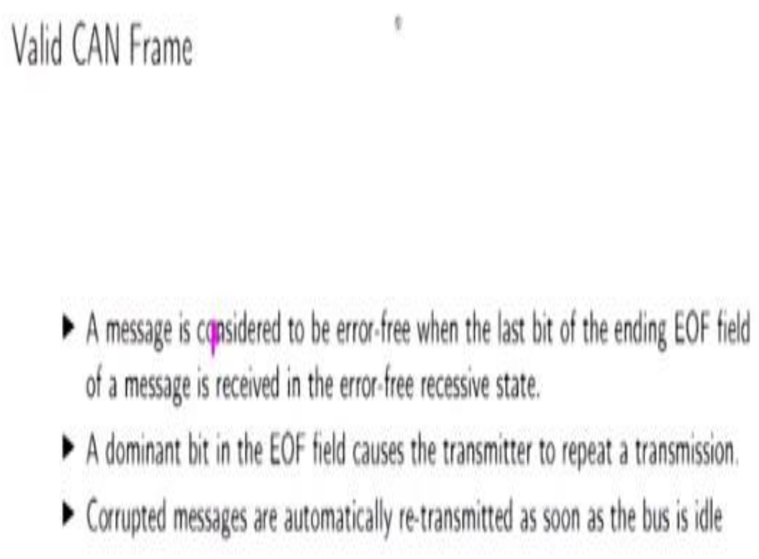
So, in this case the other nodes will not hear this node complaining about the bus error. So, if I am in the active mode then, the other nodes respond back by transmitting error flags if I am in the passive mode, then the other nodes will not hear me complaining about the bus errors. That the difference, but still, in both these nodes in this specific more specific in the active mode. Fine I mean, we are able to do our normal transmissions also, but whenever we are facing this particular kind of errors we are transmitting the error messages, error flags.

And depending on whether this error flag is acknowledged by the other nodes. That is kind of the distinguishing factor, that whether I am in the error active mode or I am in the error passive mode, right. Now as I am repeatedly saying that even when I am in these modes, they just tell me that what are I mean they just kind of make a distinction in terms of what is my history of error counts. Is it too high or its moderate, right.

But in both these modes I am still able to do my other works that means I can transmit other messages also. But once it goes to this thing, that is, TEC is beyond 255 then I am in the bus-off mode then I am not allowed to transmit anything on the bus at all. So that is the bus-off mode when I am temporarily taken out of the network. I mean that am kind of ignored by all the other CAN nodes in the system.

So, in general when so, we will get back to this discussion in future when we talk about security of automotives in specifically CAN bus security. We will see that how this node has got associated vulnerabilities and they can be exploited by some attacker ECU to kind of intentionally make a victim ECU go to the bus-off mode, and then the attacker ECU can actually assume the identity of the victim ECU. So that is a possible attack that happens and we will see how that happens.

(Refer Slide Time: 09:38)



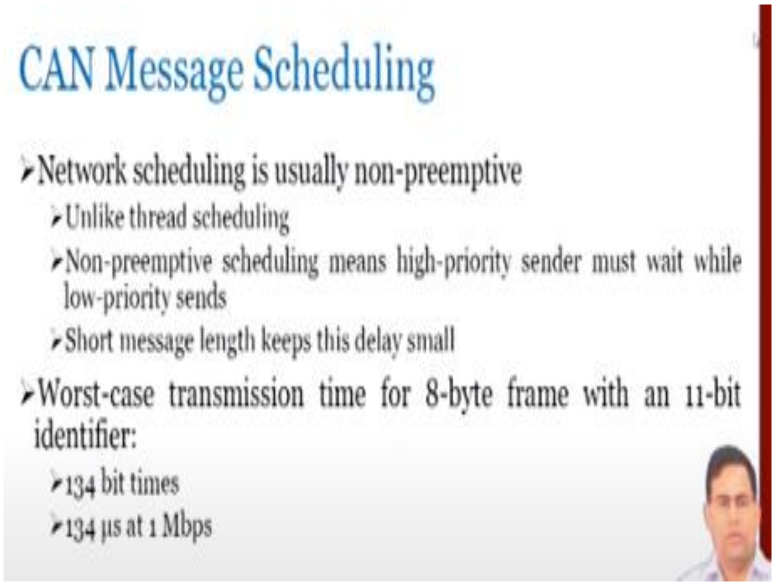
Valid CAN Frame

- A message is considered to be error-free when the last bit of the ending EOF field of a message is received in the error-free recessive state.
- A dominant bit in the EOF field causes the transmitter to repeat a transmission.
- Corrupted messages are automatically re-transmitted as soon as the bus is idle

So now coming here we can talk about what is a valid CAN frame. So, it is a frame where the message is considered to be error free when the last bit of the ending End of Field of is received in the error-free recessive state. So, the EOF is again going to have a well-defined bit pattern which is the recessive state. If I have not got any kind of error in the previous bits and if the EOF is also received in the correct pattern, then I will consider the, it to be a valid CAN frame, right.

Now if I see that well EOF is dominant then I will think that there is an error and I will ask the transmission to be repeated, okay. And in general, such corrupt messages are automatically retransmitted as soon as the bus is idle. That means, in the next transmission cycle.

(Refer Slide Time: 10:31)



CAN Message Scheduling

- Network scheduling is usually non-preemptive
  - Unlike thread scheduling
  - Non-preemptive scheduling means high-priority sender must wait while low-priority sends
  - Short message length keeps this delay small
- Worst-case transmission time for 8-byte frame with an 11-bit identifier:
  - 134 bit times
  - 134 μs at 1 Mbps

So just coming to a small discussion on how CAN messages are scheduled. From the arbitration thing we have understood something that in general for such CAN messages when you have to send multiple messages you have to schedule them inside a bus, there is a difference between scheduling of packets in a network and scheduling of processes or tasks in general in an embedded system or a computer.

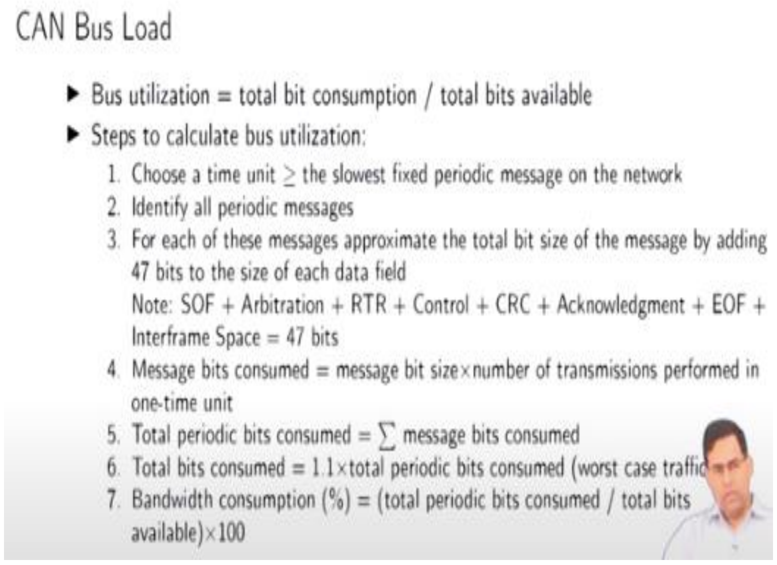
Because we have schedulers in normal embedded processors or standard desktop processors which are preemptive. That means, the high priority message or high priority task can preemptive a low priority tasks, things like that happen right. So, that is that is the standard preemptive scheduling. But unlike that when we have network messages in the CAN bus, it is non-preemptive.

That means, while a low priority sender is sending, his message, even if there is a high priority message, he cannot start sending it sending the priority message and make the low priority one preempt the bus. So no such thing happens in the CAN bus. So, there is an arbitration. Whoever wins the arbitration he will transmit and while he is transmitting is another low priority message comes that guy has to wait. So, it is non preemptive in that sense.

But the thing is, as we have seen that the data length are too small it is up to 8 bytes so the overall packet size is also small. So, this will keep I mean one may think that well if there is no preemption, then sometimes some message has to some specific messages to wait for lot of time, there can be lot of delay. But in this this case no and these delays can be easily bounded and the short message length will keep these delays to be quite small. So that is the advantage of having such scheduling.

Now coming to the other issue that, what about the worst-case transmission time for CAN bus so if I have an 8 byte data frame with 11 bit identifier. So typically, we can see that well we can reach up to a speed of 1 Mbps in the worst-case.

(Refer Slide Time: 12:38)



CAN Bus Load

- Bus utilization = total bit consumption / total bits available
- Steps to calculate bus utilization:
  1. Choose a time unit $\geq$ the slowest fixed periodic message on the network
  2. Identify all periodic messages
  3. For each of these messages approximate the total bit size of the message by adding 47 bits to the size of each data field
     Note: SOF + Arbitration + RTR + Control + CRC + Acknowledgment + EOF + Interframe Space = 47 bits
  4. Message bits consumed = message bit size$\times$number of transmissions performed in one-time unit
  5. Total periodic bits consumed = $\sum$ message bits consumed
  6. Total bits consumed = 1.1$\times$total periodic bits consumed (worst case traffic
  7. Bandwidth consumption (%) = (total periodic bits consumed / total bits available)$\times$100

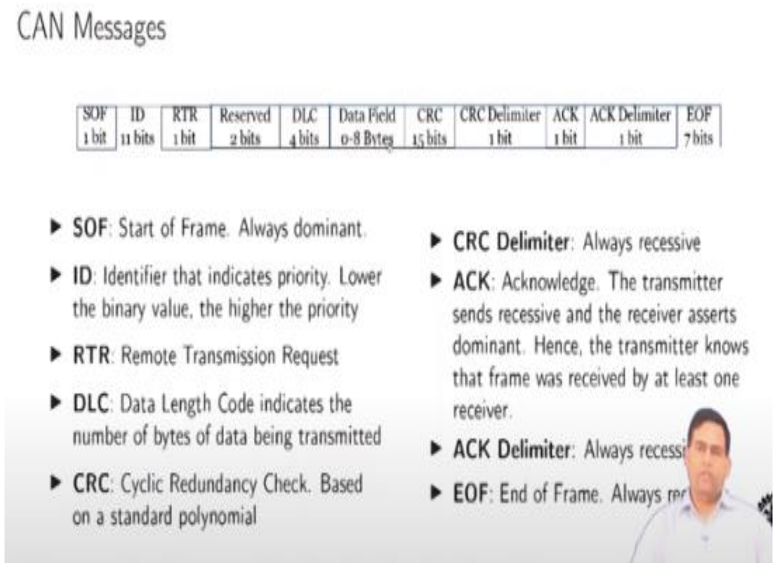
So now let us get into an idea like how CAN bus loads are computed in general. That will let us say I have a set of CAN packets to send we give their periodicities. Then how do I know that what is the total CAN bus load? So, this is a very simple computation so the bus utilization is nothing but the total number of bits that have been consumed divided by the total number of bits that are available.

So, you have a bandwidth of the bus which will allow these many bits to send and you actually send some number of bits so just that fraction, right. So how do you calculate it? To calculate it first of all you have to choose the time window. Now that amount of time you will choose is typically something greater than the slowest periodic message on the network. So, you will have multiple messages with different periodicities.

So let us say the slowest message of the period of 100 milliseconds. So, if you take a window of something more than 100 milliseconds, that mean, in this window you will have at least one instance of all the messages. So, you have a time window inside which you can see all the different kinds of bus loads and you can you can choose something greater than this. The reasonable choice would be to choose a hyper period that means the LCM of all the periods.

So, that would be choice of period inside which you will have all the messages repeating integer number of times and this is the pattern that will keep on happening on the system so you can actually do that. So, you will like to identify what all are the periodic messages and for each of these messages you can approximate the total bit size of the message by adding 47 bits to the data field. So let us understand why this 47 choice is coming.

(Refer Slide Time: 14:23)



CAN Messages

SOF	ID	RTR	Reserved	DLC	Data Field	CRC	CRC Delimiter	ACK	ACK Delimiter	EOF
1 bit	11 bits	1 bit	2 bits	4 bits	0-8 Bytes	15 bits	1 bit	1 bit	1 bit	7 bits

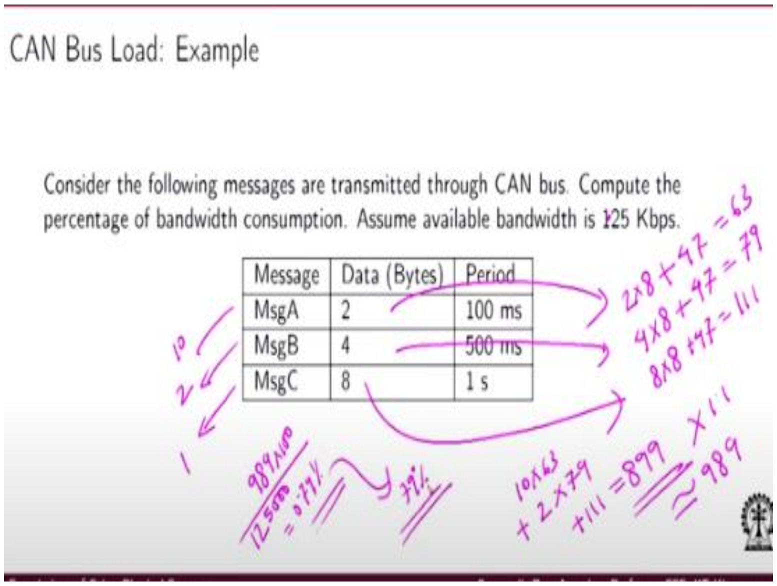
- **SOF**: Start of Frame. Always dominant.
- **ID**: Identifier that indicates priority. Lower the binary value, the higher the priority
- **RTR**: Remote Transmission Request
- **DLC**: Data Length Code indicates the number of bytes of data being transmitted
- **CRC**: Cyclic Redundancy Check. Based on a standard polynomial
- **CRC Delimiter**: Always recessive
- **ACK**: Acknowledge. The transmitter sends recessive and the receiver asserts dominant. Hence, the transmitter knows that frame was received by at least one receiver.
- **ACK Delimiter**: Always recess
- **EOF**: End of Frame. Always re

So, if you see here apart from the data field which is a variable 0 to 8, all other filter, of fixed length, right. And they would sum up to 47. So, if you just sum up the number of bits of all those other fields you will get an interference space, I mean, up to interference space you will get this number of bits as 47. So, get your data size plus 47, that is your packet size right. So, that is the total number of message bits.

Now you multiply that by the number of transmissions that are happening inside a time unit and so that would be the number of messages that are consumed, and you just sum it up for the total I mean over the all the messages, right. So that would give you that well inside the time window that you took what is the total number of bits that have been transmitted and for what you budget for worst case.

That means in the worst case there can be some more transmission. So, you can have a suitable multiplication factor let us say 10% more, okay. Now that would give you a value which of the worst-case estimate of total bits consumed divided by total bits available and that is your percentage bandwidth.

(Refer Slide Time: 15:34)



CAN Bus Load: Example

Consider the following messages are transmitted through CAN bus. Compute the percentage of bandwidth consumption. Assume available bandwidth is 125 Kbps.

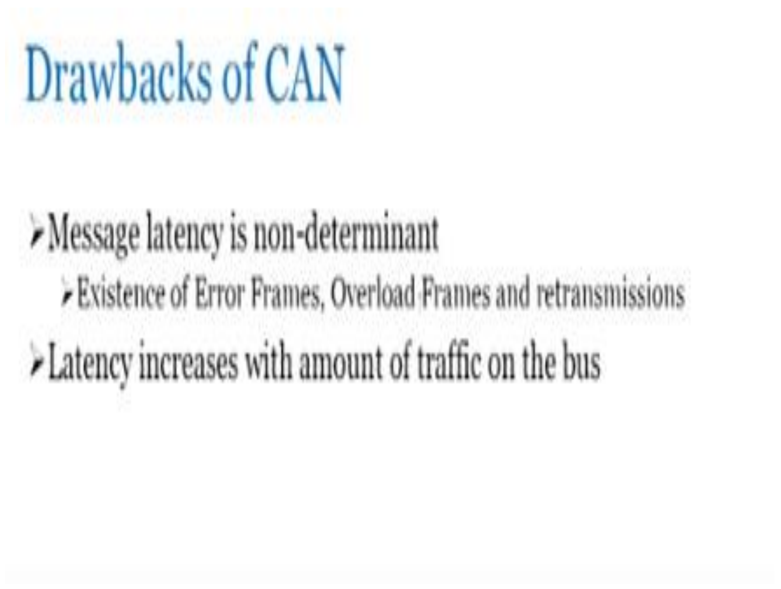
Message	Data (Bytes)	Period
MsgA	2	100 ms
MsgB	4	500 ms
MsgC	8	1 s

So let us take a sample example. Suppose in a CAN system you have nodes which are sending messages like this. So, you have messages with periodicity 100, 500 and, 100 and 500 millisecond and 1 second and the data bits are like data size is like, the payload is 2 bytes, 4 bytes, and 8 bytes. So, well it is an easy calculation then. So, if I just take this example. So, if we just work it out. So, that it is given in number of bytes, right.

So, you just have to take this plus 47. Similarly, you take this plus 47. So, the periods are all given, right. Now let us say we have this transmission window that is 1 second, right, and 1 second is also interval inside which I have integral number of all these message instances occurring here, right. So, if we choose that, then inside that how many this message instances I can see? So, I will see 10 instances of this and I will see 2 instances of this and I will see 1 instance of this, right.

So, overall, you will have 10 times 63. So, over this 1 second window the number of bits that you are consuming is this 899 and you can consider that worst case factor of 10%, right. So, this would roughly go to something like 989, right. Now let us say you are given this baud rate of 125 kilo bits per second, right. So essentially what do you have, is something like. So it is pretty low. This is a very small percentage. You can have so many more messages there with high priorities and other things. Even if you have something less you can have a 1.25 kbps then this will become something like 79%, right.

(Refer Slide Time: 18:56)



Drawbacks of CAN

- Message latency is non-determinant
  - Existence of Error Frames, Overload Frames and retransmissions
- Latency increases with amount of traffic on the bus

So that is a very simple example here. So, that is overall summary of CAN and just to think of what can be the drawbacks. So, the drawback is for example, you have this kind of error frames, overload frames and retransmissions. So, if it is a non-ideal scenario where these things keep on happening then the message latency is becomes non-deterministic. As long as these things do not happen is everything is well-defined, right.

And also, with amount of traffic increasing, I mean, CAN is designed to be a low data rate thing, right. So, then the message latencies will increase because you will be arbitrated out multiple times and all these things will keep on happening, right. So, one question that may come is how do the ECUs, multiple ECUs can be connected on the CAN bus and how the periodicities can then be defined.

Well, we have understood one thing that if, I mean, there is another this arbitration happens that means you should if you have a low priority message you should not try to send it at the times when the high priority messages are being sent, right. Now it depends on how these messages are generated. If these messages are getting generated by some periodic sensor let us say or some periodic controller if it is a control message, right.

Then well you will like to send them with the same periodicity. So, one option is to create a static schedule of such messages where you do not have this issue, or, but that is also not possible right.

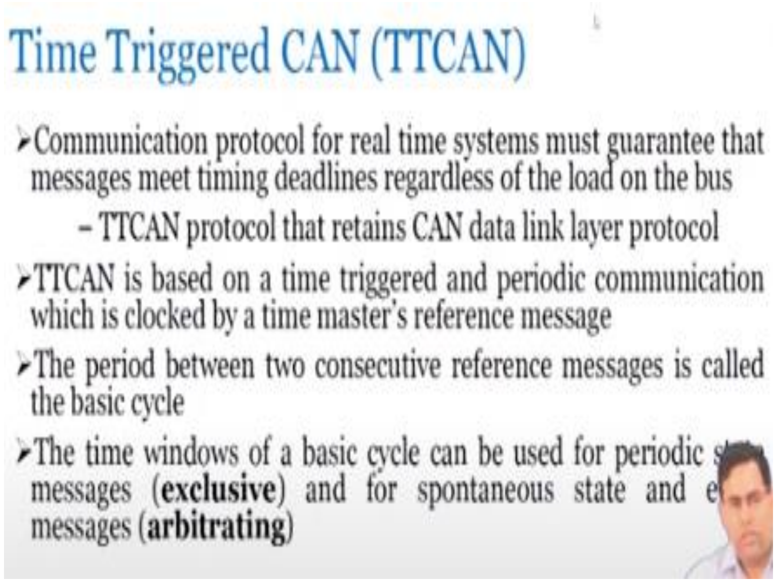
Because then it becomes a fully kind of deterministic system. I mean in CAN the reason you have arbitration is well things may also be event driven that means, some messages will be generated following some events which I do not know when they will happen.

But the fact that that message will be transmitted at that time or not is dependent on the existence of another higher priority message at the same time, right. So, depending on this issue well I can have a statically schedule scheduled set of messages. Then the basically the arbitration is never coming into play at every unique point I will have one message to send from some node. Or like I said that, if messages are generated in an event driven way in other ECUs, then it is not possible to have them statically scheduled they will be coming dynamically.

And whether they can go or not will be depending on the arbitration and existence of other high priority messages. And that will lead to the latency getting increased and it may so happen that some event generated, some event-based message that is generated is delayed quite a bit. I mean it may miss his deadline also. So, all those factors need to be looked into when somebody is designing a CAN system.

That well, suppose right now this CAN system has these these messages and I am attaching another event driven system into that whether it happens that, whenever this event driven system is generating some message in specific cases, those messages are getting overly delayed due to the existence of such other messages or not. So that becomes a question.

(Refer Slide Time: 22:04)



Time Triggered CAN (TTCAN)

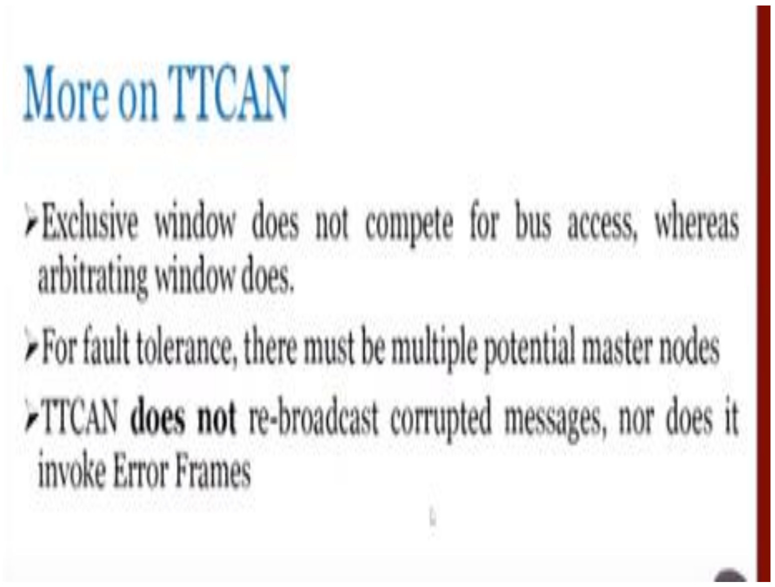
- Communication protocol for real time systems must guarantee that messages meet timing deadlines regardless of the load on the bus
  - TTCAN protocol that retains CAN data link layer protocol
- TTCAN is based on a time triggered and periodic communication which is clocked by a time master's reference message
- The period between two consecutive reference messages is called the basic cycle
- The time windows of a basic cycle can be used for periodic s messages (**exclusive**) and for spontaneous state and e messages (**arbitrating**)

One option is to have Time Triggered CAN, TTCAN. So, this returns the original CAN's data link layer protocol, but it also provides the time triggering and the periodic communication which is clocked by a time master's reference. So, there is a time trigger based on which the communication happens we are not going to the details of this. This is used in certain cases. So, as you can see, we said that CAN is pretty much event driven. Here we have time, event triggered messages in CAN. But here we have time triggered messages also, right.

That means, the period of between any two such references consecutive messages here is a basic cycle and this time window of a basic cycle can be used for periodic messages and for spontaneous state and event messages. That means you have specific time windows which can be used for, exclusively for periodic messages. And so those windows are exclusive for periodic messages and also, and there are windows where we can have them for spontaneous event driven messages.

That means we have 2 different ways, right. We can say time triggered periodic messages or we can send event triggered messages from among which there would be some arbitration that is required. So, I think for us, is just what we are trying to say is that well unlike CAN which is kind of purely event triggered, right. I mean messages coming in arbitration, it may be possible to also have time triggered messages.

(Refer Slide Time: 23:41)


More on TTCAN

- Exclusive window does not compete for bus access, whereas arbitrating window does.
- For fault tolerance, there must be multiple potential master nodes
- TTCAN **does not** re-broadcast corrupted messages, nor does it invoke Error Frames

So, what does this mean? That when we have that exclusive window, right, that means this is the slot when only periodic messages will go, it does not compete for bus access. That means whenever I mean there is it is like static or static scheduling of messages. Whereas for the arbitrating window when I have the event triggered messages they will be competing for bus access, okay.

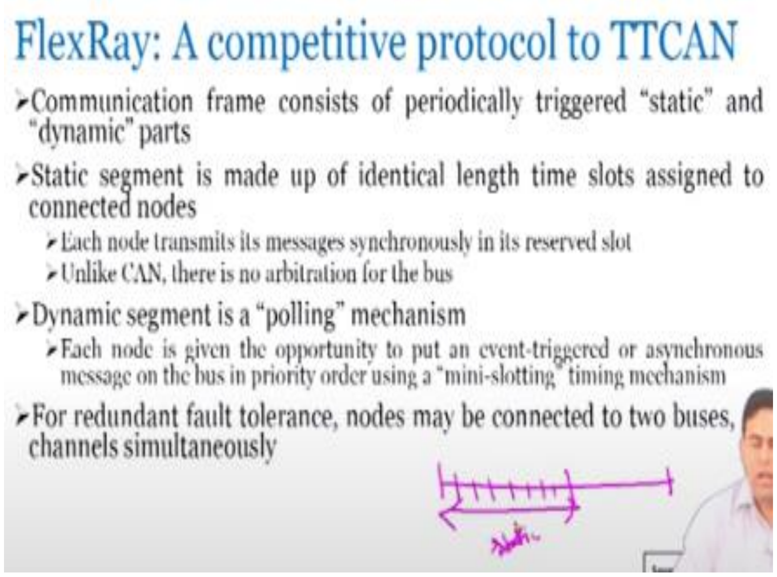
So that is a like bringing in 2 different philosophies of time triggering and evet triggering together. So, just to summarize in in simple CAN, we do not have this kind of a distinction. What we have is well messages can be sent messages can be generated, right, based on some events. And while the messages are sent it has to be arbitrated. But, like I said that, that may also be, an issue because some because due to arbitration latencies become non deterministic.

Because right now I want to send a periodic message whether I can send that message once every H time period. I really do not know because it depends on how the existence of other high priority messages. So, it is always under arbitration. Rather than that if it is the case that there are time windows, some of which are exclusive that means some of the bus cycles are reserved for periodic messages.

And some of the bus transmission cycles are reserved for the standard arbitration-based scheme that we just discussed. So that is a mixture, right. So that is something that TTCAN tries to do and this basic transmission cycle is defined by this idea of basic cycle here. Which is kind of, the difference between 2 consecutive special messages called the reference message.

And this reference message is basically something which is a time instances, which are generated by some periodic clock, okay. And it has got other properties like it does not broadcast, rebroadcast the corrupted messages or invoke error frames something that the CAN do. In TTCAN those things are not there.

(Refer Slide Time: 25:51)



FlexRay: A competitive protocol to TTCAN

- Communication frame consists of periodically triggered "static" and "dynamic" parts
- Static segment is made up of identical length time slots assigned to connected nodes
  - Each node transmits its messages synchronously in its reserved slot
  - Unlike CAN, there is no arbitration for the bus
- Dynamic segment is a "polling" mechanism
  - Each node is given the opportunity to put an event-triggered or asynchronous message on the bus in priority order using a "mini-slotting" timing mechanism
- For redundant fault tolerance, nodes may be connected to two buses, channels simultaneously

Now, this there is another comparative protocol to TTCAN which has those ideas also. Like whether you can have I mean whenever you are having a communication frame it can have 2 parts. One is the static part where you have fixed periodically triggered messages like the ones we want. And you also have the dynamic part where the messages will be sent based on arbitration, okay.

So, these are named as the static and dynamic segments in a FlexRay transmission cycle and the static segment is made of identical length time slots which you can assign to different nodes. And each node transmits this message synchronously in its reserved slot. So, you can understand that in a FlexRay cycle, this is a full transmission cycle. This is your static segment. Now this static segment will be statically divided among these nodes, okay.

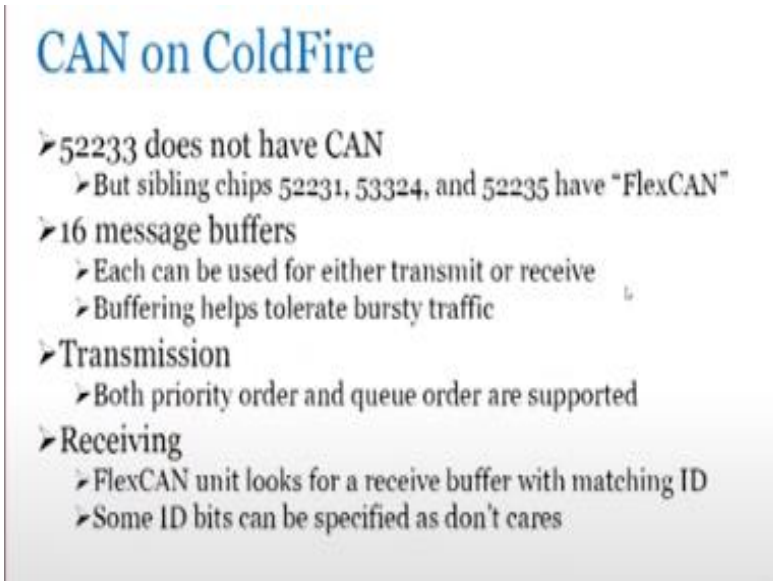
So that means each of these segments are having specific slots which can be reserved by individual nodes, right. And in those nodes, those slots result for the restrictive nodes they, can send their messages. So, there is no arbitration happening right. Now the dynamic segment has something like a polling mechanism. So that means if I have, if I want periodic messages to be sent without latency I must put them in the static segment for that.

But like we said that we will also see soon see that later on when we are doing schedulability analysis, that making things completely static is also a wastage of resources. That is why people like to keep things dynamic also or event driven also, okay. So, in the dynamic segment we need some kind of mechanism to decide who will send and who will not send, right.

So, in case of FlexRay, each of the nodes are given the opportunity to put an event triggered or asynchronous message on the bus in a priority order, using something called a mini slotting timing mechanism. We are not going into the details here because we are just telling you what FlexRay is we are not going to the details of FlexRay here as part of this lecture. Here we were just covering the details of CAN.

The only point we are trying to make here is FlexRay allows you, just like the idea of TTCAN, it allows you to have slots for static messages with fixed slots and some part where you have some arbitration or some polling, where there are messages which are generated at known at non-fixed times based on some events, right.

They are not time trigger they are generated due to some events I do not know when they will come and when they are generated, they have to compete through some mechanism, okay. So, that is this other kind of protocol called FlexRay and that also has got some use in automotive systems.
(Refer Slide Time: 28:51)

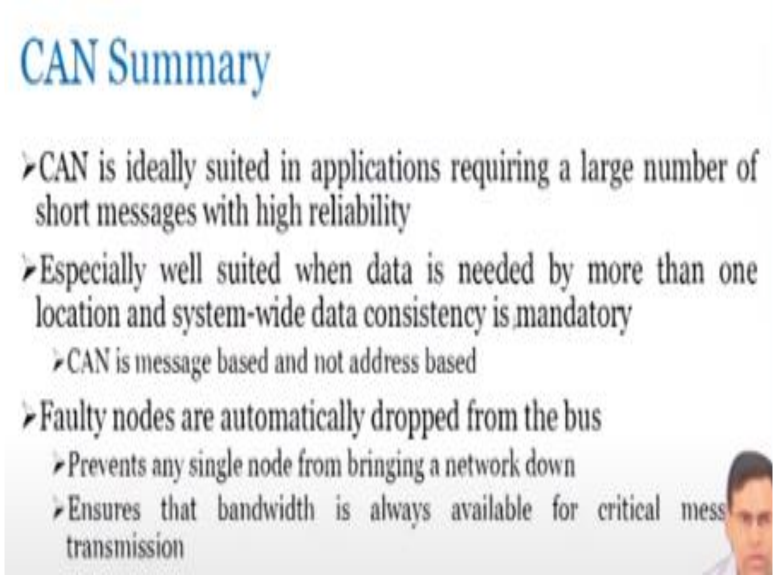


CAN on ColdFire

- 52233 does not have CAN
  - But sibling chips 52231, 53324, and 52235 have "FlexCAN"
- 16 message buffers
  - Each can be used for either transmit or receive
  - Buffering helps tolerate bursty traffic
- Transmission
  - Both priority order and queue order are supported
- Receiving
  - FlexCAN unit looks for a receive buffer with matching ID
  - Some ID bits can be specified as don't cares

Now the CAN is actually used in many standard microcontrollers I mean the CAN interfaces are provided. That means these microcontrollers understand the physics of the CAN protocol the I mean the different protocol rules and all that. Similarly, there are microcontrollers which understand the FlexRay protocol also. And there are I mean those transceivers are often attached to such micro protocol such microcontrollers.

(Refer Slide Time: 29:17)



CAN Summary

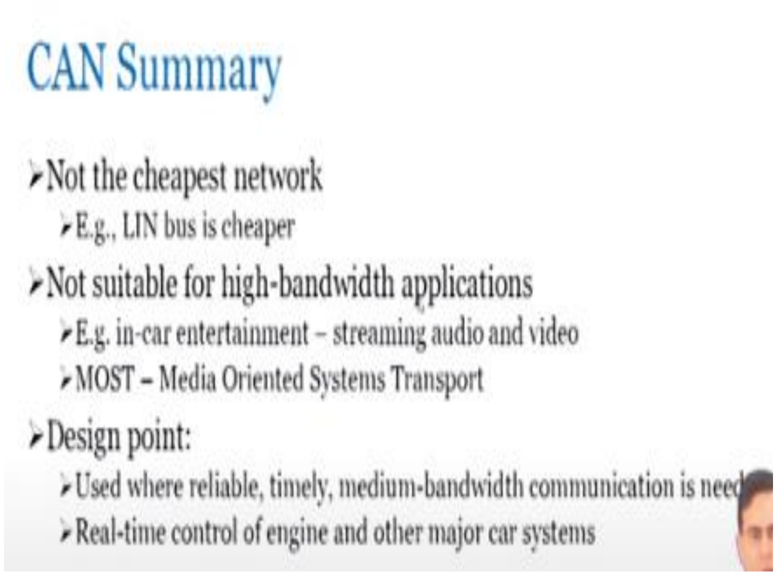
- CAN is ideally suited in applications requiring a large number of short messages with high reliability
- Especially well suited when data is needed by more than one location and system-wide data consistency is mandatory
  - CAN is message based and not address based
- Faulty nodes are automatically dropped from the bus
  - Prevents any single node from bringing a network down
  - Ensures that bandwidth is always available for critical mess transmission

So, in a summary, CAN is ideally suited for applications where you have a large number of short messages. Message lengths are small because of this limited data packet length and you do not want them to suffer due to latency because if and as long as the message lengths are small, the latency is I mean the delays are not much, right.

So, for them CAN is suitable and CAN is highly reliable. Because if there are errors, I can catch them and I mean the well-defined arbitration scheme tells you exactly when your message will go or whether it will not go these things are quite well known nothing is probablistic here. So, it is suited for data I mean cases where the data is needed by more than one location, because you see, CAN is more like a multicast thing. So the message can be read by anyone.

So, as long as the message is transmitted and for situations where multiple nodes are requiring the message CAN is basically quite resource friendly. I mean, it is it is nice to use CAN in those cases, right. And for faulty nodes they get automatically dropped from the bus because of that error-based protocol we saw that the nodes have a count of how much error they have faced. And in case the number of counts are high they will first transmit error flags the, then they will from go from an active to a passive node where the error flag will change and finally they will go to a bus-off mode where they will wait till their transmission counts go further down.

(Refer Slide Time: 30:40)



CAN Summary

- Not the cheapest network
  - E.g., LIN bus is cheaper
- Not suitable for high-bandwidth applications
  - E.g. in-car entertainment – streaming audio and video
  - MOST – Media Oriented Systems Transport
- Design point:
  - Used where reliable, timely, medium-bandwidth communication is need
  - Real-time control of engine and other major car systems

But there is also another cheap network for the LIN bus which is also used for some specific kind of control loops in automotives. But none of these are going to be used for high bandwidth applications and there are, lot of high value, bandwidth applications nowadays in your system, right. For example, I mean self-driving vehicles or vehicles with connected mobility features they use a lot of camera sensors, radar and lidar sensors.

And those sensors stream in lot of high bandwidth data, right. I mean lot of, I mean high resolution data frames with a high bitrate. Now, that is something that CAN cannot actually support and they may be supported by things like automotive Ethernet and also for the infotainment part the media part there are other protocols like most. So, from a design point of view this is used typically for low bandwidth but reliable communication when that is the requirement.

For example, real time control of engine or other car systems like the electronic stability program that is which is the, I mean, vehicle dynamics controller and other packages. So that is the primary usage of CAN. With this I will end this lecture here. Thank you for your attention.