Techniques For Bandwidth Reduction In Controller Area Network

Name of Students: **Geetika Gupta (071040)**
Sudip Shukla (071049)
Hemant Kumar (071066)

Name of Supervisor: Dr. Rajiv Kumar

विद्या तत्व ज्योतिसम

May 2011

Submitted in partial fulfillment of the Degree of
Bachelor of Technology
B. Tech

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY,WAKNAGHAT

TABLE OF CONTENTS

CERTIFICATE

This is to certify that the work titled " **Techniques for Banwidth Reduction in Controller Area Network** "submitted by "**Geetika Gupta, Sudip Shukla, Hemant Kumar**" in partial fulfillment for the award of degree of. Bachelor Of Technology of Jaypee University of Information Technology in 2011 has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

....

Signature

Dr. Rajiv Kumar

Assistant Professor

Department of Electronics and Communication

Jaypee University Of Information Technology,

Waknaghat.

Date :

ACKNOWLEDGEMENT

We are extremely grateful to our project in charge **Dr. Rajiv Kumar** for motivating and mentoring us in our project. He has been our guiding light and source of inspiration throughout the entire project. Besides providing us with technical assistance he has also been a moral support and a motivating figure during difficult times.

We once again express our sincere gratitude towards him and look forward to his assistance and expertise for any future endeavour.
We would also like to thank all the staff members of **Jaypee University of Information Technology,** Waknaghat, for providing us all the facilities required for the completion of this project report.

Geetika	Sudip	Hemant
Geetika Gupta	**Sudip Shukla**	**Hemant Kumar**
Enrollment no. 071040	**Enrollment no. 071049**	**Enrollment no. 071066**
ECE	**ECE**	**ECE**
Date: 23/05/2011	**Date:** 23/5/11	**Date:** 23/05/11

SUMMARY

As automobile industry prospered it incorporated more and more advanced features into vehicles, there developed a growing need for enhanced processing power. Moreover functional integration of individual sensors was considered necessary for applications such as collision avoidance These additional functionalities were achieved by increasing the number of nodes or Electronic Control Modules (ECM's), sensors and actuators that could exchange data among various other nodes in the network.

As the complexity grew in the functions implemented in these systems there was a growing need for an exchange of data between them .With traditional systems, data was transferred by means of point to point networks , but it became increasingly difficult and expensive as control functions became more complex. In the case of complex control systems, the number of connections could not be increased much further as it demanded a large number of multipoint networks,which had lots of complexities .Hence ,a number of systems were developed which needed to implement functions covering more than one control device. For instance, vehicles requires the synchronization of engine timing and carburetor control in order to reduce torque when drive wheel slippage occurs. Another example of functions spanning more than one control unit is electronic gearbox control, where ease of gear changing can be improved by a brief adjustment to ignition timing. To overcome the limitations of conventional control device linkage, there developed a need of networking the system components using a serial data bus system.

It was for this reason that Robert Bosch developed the "Controller Area Network" (CAN), which has International Standardization Organization (ISO) and the Society of Automotive Engineers (SAE) standardization and has been used extensively by several semiconductor manufacturers. Using CAN, peer stations (controllers, sensors and actuators) are connected via a serial bus.

However, the data traffic over the high-speed communication bus Controller Area Network (CAN) increased significantly with the increase in the number of nodes. Another important requirement was to communicate data between the various ECM's in real time for safety-critical applications. Failure to communicate data within a given period of time lead to degradation in system performance.

Solution to satisfy the bandwidth requirements of future vehicle networks was to use a higher bandwidth bus or to use multiple buses. But, the use of a higher bandwidth bus would increase the cost of the network. Similarly, the use of multiple busses would increase the cost and the complexity of the wiring. Another option was the development of a higher layer protocol to reduce the amount of data to be transferred. For this reason several data reduction techniques were developed. These techniques help to transfer large amounts of data while consuming very less bandwidth thereby increasing the efficiency of the network.

A data reduction technique that consumes least network bandwidth and zero message latency would be considered the best.

Our project deals with the use of adaptive data reduction algorithm for reduction of bandwidth in CAN. It also elaborates on the use of Huffman Coding for reducing the bandwidth consumption.

...Geetika.......
Geetika Gupta
...Sudip............
Sudip Shukla
...Hemant..............
Hemant kumar

....
(Signature)
Dr. Rajiv Kumar

Date:

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Development Of CAN

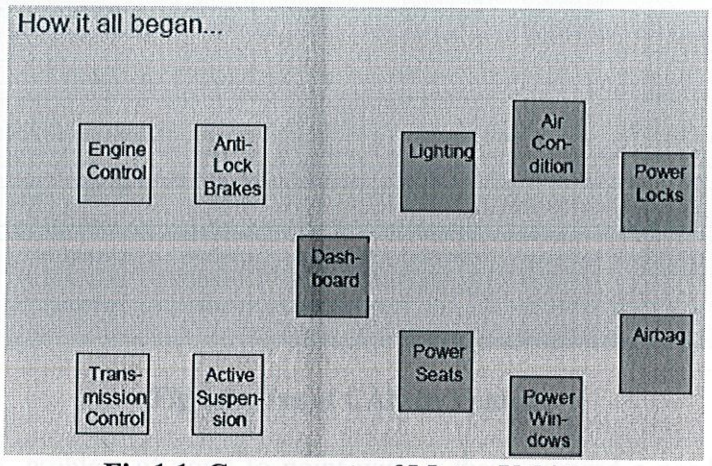


Fig 1.1: Components of Motor Vehicles

The above figure [Fig.1.1] shows how the various electronic devices that are implemented in vehicles.These devices include power seats,power windows,brakes,accelerator etc. As the number of devices increased , there developed a need for CAN.

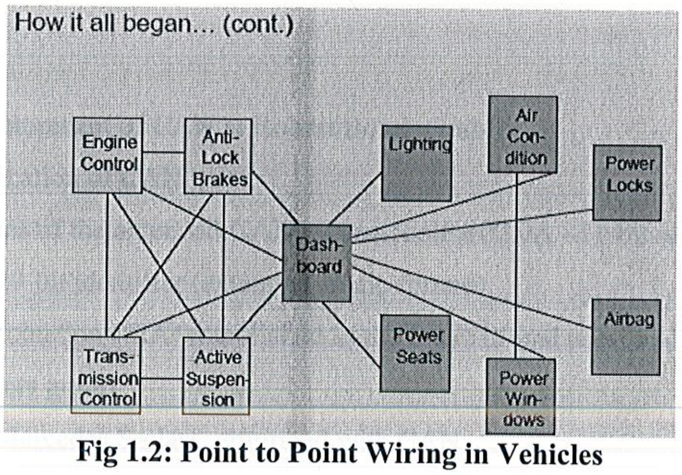


Fig 1.2: Point to Point Wiring in Vehicles

Earlier the information exchange between the various devices in any vehicle was point to point [Fig.1.2] But as the need for information exchange increased , it required connectors of very large length which led to an increase in the overall cost and complexity of the system.

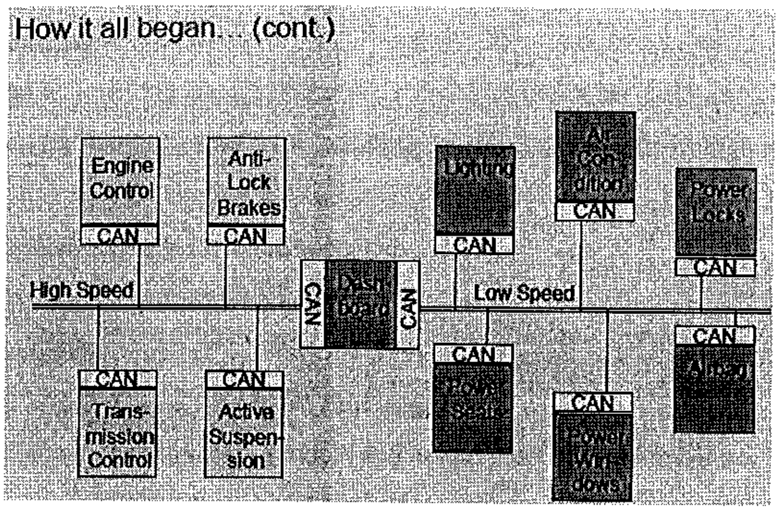


Fig 1.3: Use of CAN in Vehicles

To overcome the problem of point to point wiring, there occured a need for a serial bus communication system[Fig 1.3] which was fufilled by the development of CAN(Controller Area Network).Using CAN, various devices can be connected through one single communication bus.

1.2 History

1985	Start of development of CAN at Robert Bosch GmbH
1986	V1.0 specification of CAN
1991	Specifications of the extended CAN2.0 protocolPart 2.0A –11-bit identifier Part 2.0B –29-bit identifier (extended frame format)
1992	CAN in Automation (CiA) established as the international users and manufacturers group
1993	First car, a Mercedes S-class, equipped with CAN
1994	First standardization at ISO is completed
1998	Development phase of time-triggered CAN (TTCAN) networks
1999	Explosion of CAN-linked equipment in all motor vehicle and industrial applications.

1.3 User Benefits Of CAN

CAN is low cost

• Low cost protocol device available driven by high volume production in the automotive and industrial markets

CAN is reliable

• Sophisticated error detection and error handling mechanisms results in high reliability transmission

• Erroneous messages are detected and repeated

• Every bus node is informed about an error

CAN means real-time

• Short message length (0 to 8 data bytes / message)

• Low latency between transmission request and actual start of transmission.

• Multi Master using CSMA/CD + AMP method

CAN is flexible

• CAN Nodes can be easily connected / disconnected (i.e. plug & play)

• Number of nodes not limited by the protocol

CAN allows Multi-Master Operation

• Each CAN node is able to access the bus

• Bus communication is not disturbed by faulty nodes

• Faulty nodes self swith-off from bus communication

CAN means Broadcast Capability

• Messages can be sent to single/multiple nodes

• All nodes simultaneously receive common data

CAN is standardized

• ISO-DIS 11898 (high speed applications)

• ISO-DIS 11519-2 (low speed applications)

1.4 Applications Of CAN

CAN is used in a variety of fields:

1) Vehicles (cars, trucks, buses, trains)

Enables communication between ECUs like engine management system, anti-skid braking, gear control, active suspension, used to control units like dashboard, lighting, air conditioning, windows, central locking, airbag, seat belts etc. (body control), used in construction vehicles, forklifts, tractors ,power train and hydraulic control.

2) Industrial Automation

For connecting all kinds of automation equipments like control units, sensors and actuators, for initialization, programming of various components, Machine control ,connection of the different intelligent subsystems.

3) Medical Equipments

Computer tomographs, X-ray machines, dentist & wheel chairs

4) Building Automation

For heating, air conditioning, lighting, surveillance etc.

5)Household Appliances

Dishwashers, washing machines, even coffee machines.

CHAPTER 2

PROBLEM FORMULATION AND LITERATURE SURVEY

Problem formulation

Robert Bosch developed the ”Controller Area Network” (CAN), which has ISO 11898 standardization and was used extensively by several semiconductor manufacturers. Using CAN, peer stations (controllers, sensors and actuators) are connected via a serial bus.
However, the data traffic over the high-speed communication bus Controller Area Network (CAN) increased significantly with the increase in the number of nodes. Failure to communicate data within a given period of time lead to degradation in system performance.
One straightforward solution to satisfy the bandwidth requirements of future vehicle networks was to use a higher bandwidth bus or to use multiple buses. However, the use of a higher bandwidth bus would have increased the cost of the network. Similarly, the use of multiple busses would also have increased the cost and the complexity of the wiring. Another option was the development of a higher layer protocol to reduce the amount of data to be transferred. For this reason several data reduction techniques were developed. These techniques help to transfer large amounts of data while consuming very less bandwidth thereby increasing the efficiency of the network.
A data reduction technique that consumes least network bandwidth and zero message latency would be considered the best.Hence adaptive data reduction was considered a good solution and served as a data reduction algorithm which solved some of the bandwidth concerns.

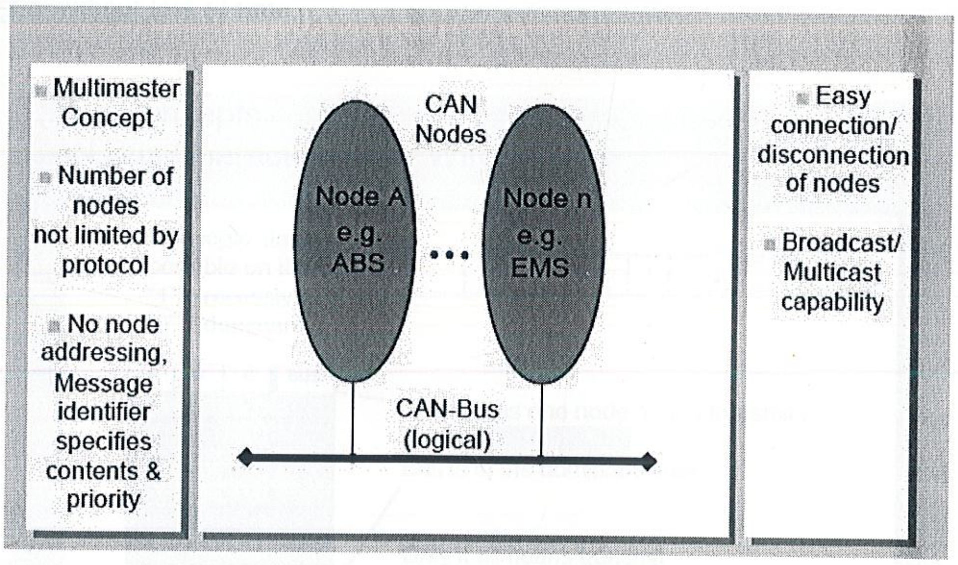
Our project deals with the use of adaptive data reduction algorithm for reduction of bandwidth in CAN. Huffman Coding is also discussed as an alternate solution approach for data reduction

Literature Survey

2.1 Important Features Of CAN

CAN was developed for the use in motor vehicles by Robert Bosch in 1980s. It is an advanced serial bus system used for support of distributed control systems. However there are a number of higher level protocols available for CAN, it uses the Data Link Layer and the Physical Layer in the ISO - OSI model. There are about 20 million CAN nodes in use worldwide.

2.2 Basic Concepts



Multimaster Concept
Number of nodes not limited by protocol
No node addressing, Message identifier specifies contents & priority
CAN Nodes
Node A e.g. ABS
...
Node n e.g. EMS
CAN-Bus (logical)
Easy connection/ disconnection of nodes
Broadcast/ Multicast capability

Fig 2.1: Basic Concepts

• There is no limitation in number of nodes in CAN .It is a multi-master bus with a linear structure [Fig.2.1] , and multiple nodes can be connected to it. .

•In the CAN protocol the address is identified by the identifiers of the transmitted messages, indicating the message content and the priority of the message.

•One major advantage of CAN protocol is that the number of nodes can be changed without affecting the communication of the bus.

•CAN allows multicasting and broadcasting, it has several error detection and error handling mechanisms like the CRC check . With their use errors can be removed, and faulty messages can be retransmitted.

•The CAN protocol uses NRZ bit coding. For synchronization purposes, bit stuffing is used.

•There is a high data transfer rate of 1000 kilobits per second at a maximum bus length of 40 meters or 130 feet when using a twisted wire pair which is the most common bus medium used for CAN. Message length is short with a maximum of 8 data bytes per message and there is a low latency between transmission request and start of transmission.

•The CAN protocol follows Carrier Sense Multiple Access/Collision Detection with Non-Destructive Arbitration. This means that collision of messages is avoided by bitwise arbitration without loss of time.

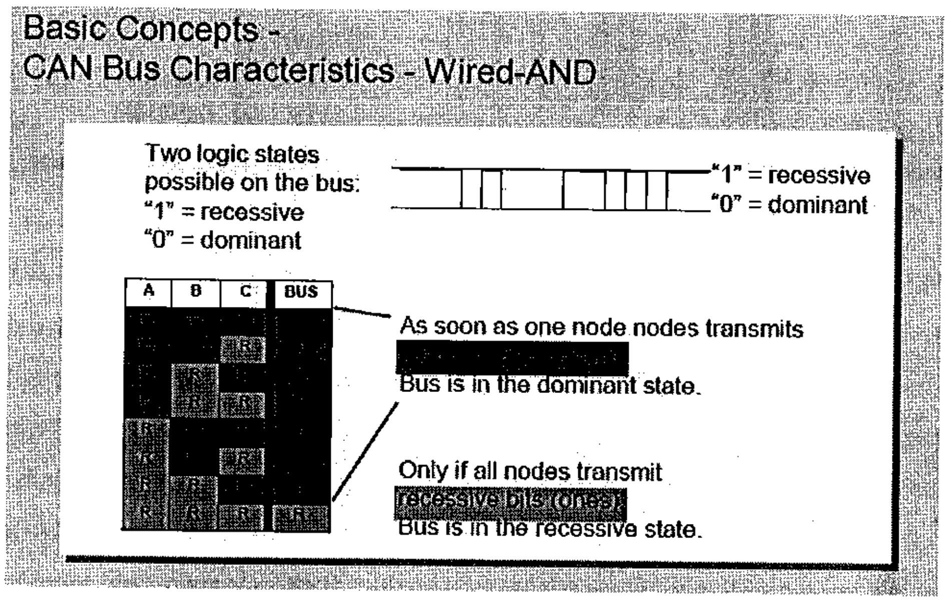

Basic Concepts -
CAN Bus Characteristics - Wired-AND
Two logic states possible on the bus:
"1" = recessive
"0" = dominant
"1" = recessive
"0" = dominant
A
B
C
BUS
R
As soon as one node nodes transmits
Bus is in the dominant state.
Only if all nodes transmit
recessive bits (ones)
Bus is in the recessive state.

Fig 2.2 : Bus Characteristics

• In CAN there are two bus states, it uses a "Wired-AND" mechanism, that is, "dominant bits" (equivalent to "Zero" logic level) and the "recessive" bits (equivalent to the logic level "One").

•Recessive state is achieved only if all nodes transmit recessive bits (ones), even if one node transmits a dominant bit (zero), the bus comes in the dominant state.

2.3 Data Link Layer

2.3.1 Bus Access And Arbitration

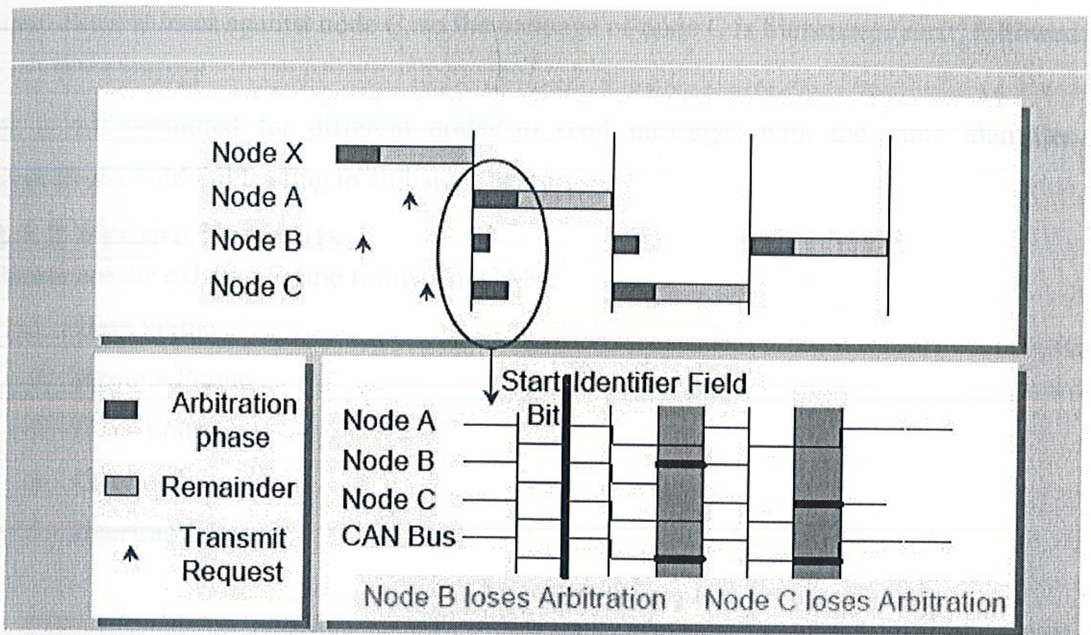


Fig 2.3 : Bus Access and Arbitration [18]

•The CAN protocol handles bus accesses according to the concept called "CSMA with Arbitration on Message Priority". This arbitration concept avoids collisions of messages whose transmission was started by more than one node simultaneously and makes sure the most important message is sent first without time loss. We see [Fig.2.3] the trace of the transmit pins of three bus nodes called A, B and C, and the resulting bus state according to the wired-AND principle.

•If two or more bus nodes start their transmission at the same time after having found the bus to be idle, collision of the messages is avoided by bitwise arbitration. Each node sends the bits of its message identifier and monitors the bus level.

•At a certain time nodes A and C send a dominant identifier bit. Node B sends a recessive identifier bit but reads back a dominant one. Node B loses bus arbitration and switches to receive mode. Some bits later node C loses arbitration against node A. This means that the message identifier of node A has a lower binary value and therefore a higher priority than the messages of nodes B and C. In this way, the bus node with the highest priority message wins arbitration without losing time by having to repeat the message.

•Nodes B and C automatically try to repeat their transmission once the bus returns to the idle state. Node B loses against node C, so the message of node C is transmitted next, followed by node B's message.

•It is not permitted for different nodes to send messages with the same identifier as arbitration could fail leading to collisions and errors.

2.3.2 Frame Formats

•These are the existing Frame formats in CAN :

1. Data Frame
2. Remote Frame
3. Error Frame
4. Overload Frame
5. Interframe Space

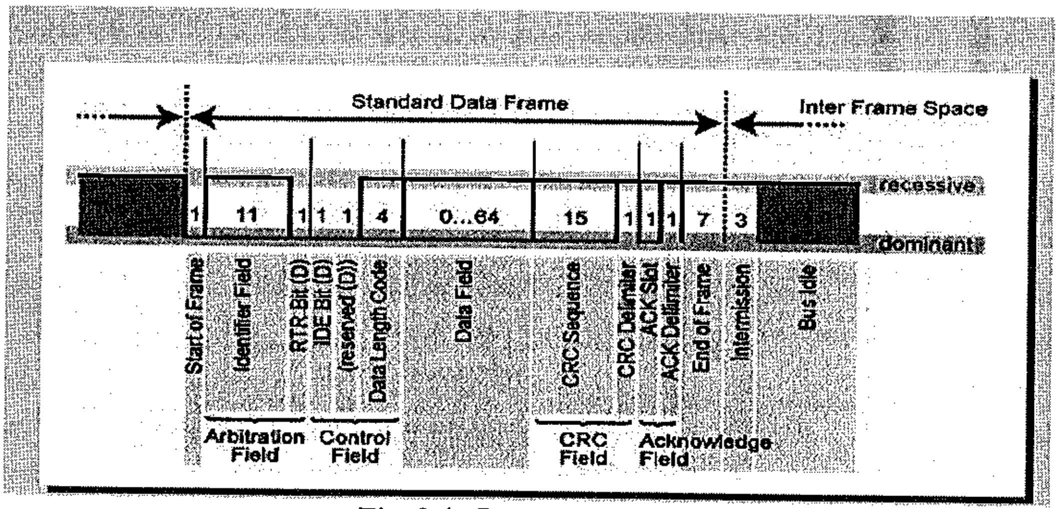


Fig. 2.4 : Data Frame [18]

•A "Data Frame" is generated by a CAN node when the node wishes to transmit data. The Standard CAN Data Frame is shown above. The frame begins with a dominant Start Of Frame bit for hard synchronization of all nodes.

•The Start of Frame bit is followed by the Arbitration Field consisting of 12 bits. The 11-bit Identifier,which reflects the contents and priority of the message, and the Remote

Transmission Request bit. The Remote transmission request bit is used to distinguish a Data Frame (RTR = dominant) from a Remote Frame (RTR = recessive).

•The next field is the Control Field, consisting of 6 bits. The first bit of this field is called the IDE bit (Identifier Extension) and is at dominant state to specify that the frame is a Standard Frame. The following bit is reserved and defined as a dominant bit. The remaining 4 bits of the Control Field are the Data Length Code (DLC) and specify the number of bytes of data contained in the message (0 - 8 bytes).

•The data being sent follows in the Data Field which is of the length defined by the DLC above (0, 8, 16, ..., 56 or 64 bits).

•The Cyclic Redundancy Field (CRC field) follows and is used to detect possible transmission errors. The CRC Field consists of a 15 bit CRC sequence, completed by the recessive CRC Delimiter bit.

•The next field is the Acknowledge Field. During the ACK Slot bit the transmitting node sends out a recessive bit. Any node that has received an error free frame acknowledges the correct reception of the frame by sending back a dominant bit (regardless of whether the node is configured to accept that specific message or not). From this [Fig. 2.4] it can be seen that CAN belongs to the "in-bit-response" group of protocols. The recessive Acknowledge Delimiter completes the Acknowledge Slot and may not be overwritten by a dominant bit.

•Seven recessive bits (End of Frame) end the Data Frame.

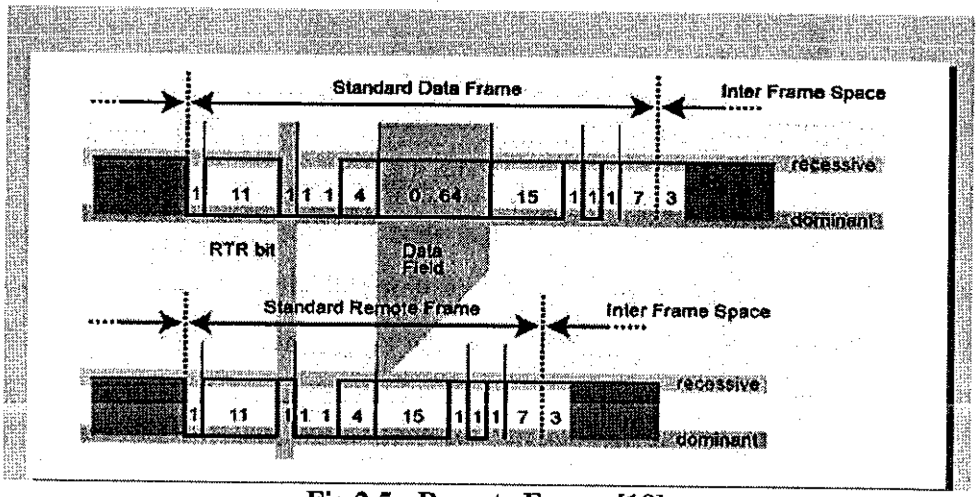


Fig 2.5 : Remote Frame [18]

•Generally data transmission is performed on an autonomous basis with the data source node (e.g. a sensor) sending out a Data Frame. It is also possible, however, for a destination node to request the data from the source by sending a Remote Frame.

•There are 2 differences between a Data Frame and a Remote Frame .Firstly the RTR-bit is transmitted as a dominant bit in the Data Frame and secondly in the Remote Frame there is no Data Field. In the very unlikely event of a Data Frame and a Remote Frame with the same identifier being transmitted at the same time, the Data Frame wins arbitration due to the dominant RTR bit following the identifier. In this way [Fig.2.5] , the node that transmitted the Remote Frame receives the desired data immediately.

•If a node wishes to request the data from the source, it sends a Remote Frame with an identifier that matches the identifier of the required Data Frame. The appropriate data source node will then send a Data Frame as a response to this remote request.

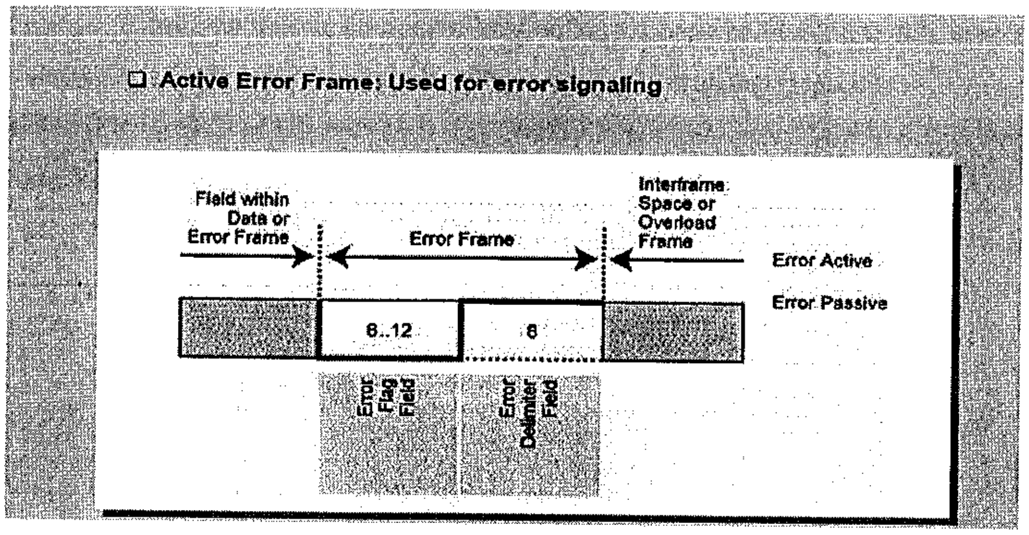

Active Error Frame: Used for error signaling
Field within Data or Error Frame
Error Frame
Interframe Space or Overload Frame
Error Active
Error Passive
6..12
8
Error Flag Field
Error Delimiter Field

Fig 2.6 : **Error Frame [18]**

•An *Error Frame* is generated by any node that detects a bus error. The Error Frame consists of 2 fields, an Error Flag field followed by an Error Delimiter field. The Error Delimiter consists of 8 recessive bits and allows the bus nodes to restart bus communications cleanly after an error. There are, however, two forms of Error Flag fields. The form of the Error Flag field depends on the "error status" of the node that detects the error.

•If an "error-active" node detects a bus error then the node interrupts transmission other current message by generating an "active error flag"[Fig 2.6]. The "active error flag" is composed of six consecutive dominant bits. This bit sequence actively violates the bit stuffing rule. All other stations recognize the resulting bit stuffing error and in turn generate Error Frames themselves. The Error Flag field therefore consists of between six and twelve consecutive dominant bits (generated by one or more nodes). The Error Delimiter field completes the Error Frame. After completion of the Error Frame bus activity returns to normal and the interrupted node attempts to resend the aborted message.

•If an "error passive" node detects a bus error then the node transmits an "passive Error Flag" followed, again, by the Error Delimiter field. The "passive Error Flag" consists of six consecutive recessive bits, and therefore the Error Frame (for an "error passive" node) consists of 14 recessive bits (i.e. no dominant bits). From this it follows that, unless the bus error is detected by the node that is actually transmitting (i.e. is the bus master), the transmission of an Error Frame by an "error passive" node will not affect any other node on the network. If the bus master node generates an "error passive flag" then this may cause other nodes to generate error frames due to the resulting bit stuffing violation.

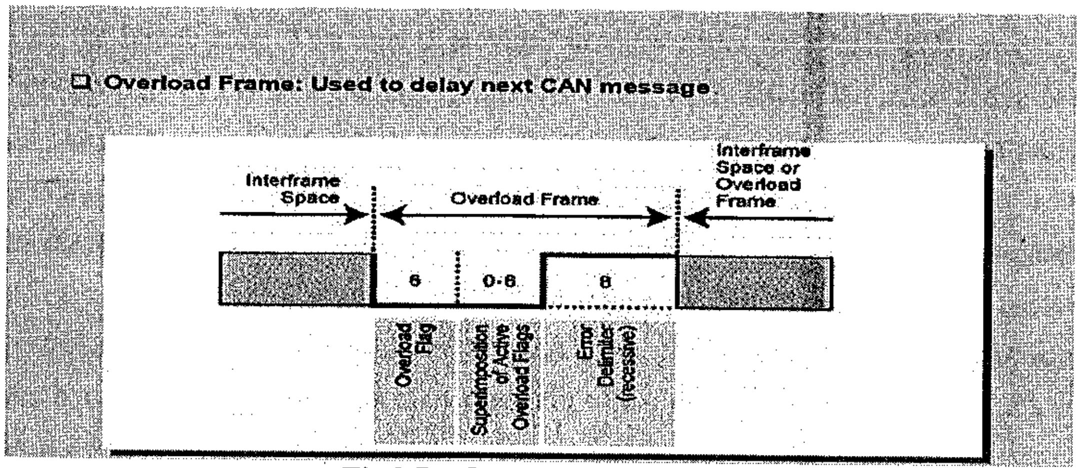

Overload Frame: Used to delay next CAN message.
Interframe Space
Overload Frame
Interframe Space or Overload Frame
6
0-6
8
Overload Flag
Superimposition of Active Overload Flags
Error Delimiter (recessive)

Fig 2.7 : Overload Frame [18]

•An Overload Frame has the same format as an "active" Error Frame. An Overload Frame, however can only be generated during Inter-frame Space. This is the way then an Overload Frame can be differentiated from an Error Frame (an Error Frame is sent during the transmission of a message). The Overload Frame consists of 2 fields, an Overload Flag followed by an Overload Delimiter[Fig.2.7]. The Overload Flag consists of six dominant bits followed by Overload Flags generated by other nodes (as for "active error flag", again giving a maximum of twelve dominant bits).The Overload Delimiter consists of eight recessive bits. An Overload Frame can be generated by a node if due to internal conditions the node is not yet able to start reception of the next message. A node may generate a maximum of 2 sequential Overload Frames to delay the start of the next message.

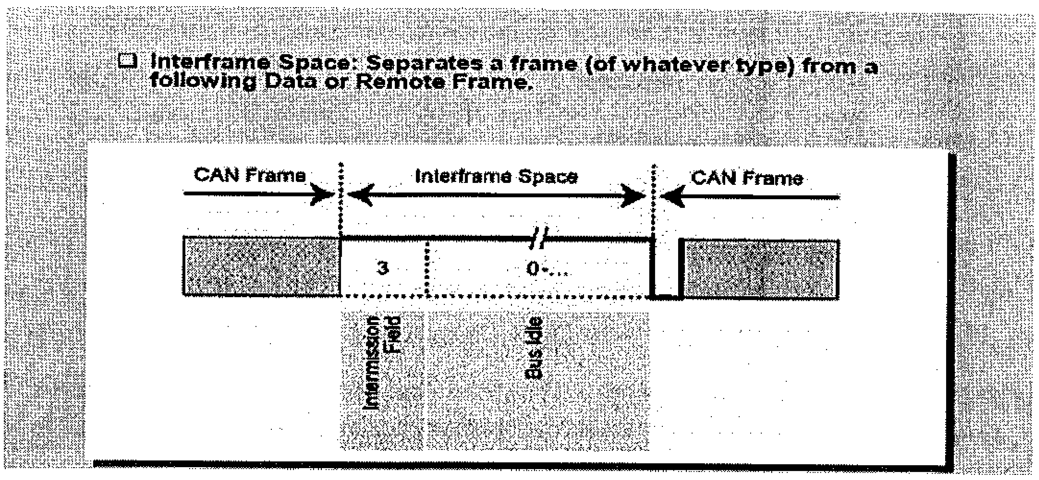

❑ Interframe Space: Separates a frame (of whatever type) from a following Data or Remote Frame.
CAN Frame
Interframe Space
CAN Frame
3
0-...
Intermission Field
Bus Idle

Fig 2.8: Interframe Space [18]

•Interframe Space separates a preceeding frame (of whatever type) from a following Data or Remote Frame. Interframe space is composed of at least 3 recessive bits, these bits are termed the Intermission. This time is provided to allow nodes time for internal processing before the start of the next message frame. After the Intermission, for error active CAN nodes the bus line remains in the recessive state (Bus Idle) until the next transmission starts.

•The Interframe Space has a slightly different format for error passive CAN nodes which were the transmitter of the previous message.In this case, these nodes have to wait another eight recessive bits called Suspend Transmission before the bus turns into bus idle for them after Intermission and they are allowed to send again. Due to this mechanism error active nodes have the chance to transmit their messages before the error passive nodes are allowed to start a transmission.

2.3.3 Error Detection

Following types of errors are detected in CAN :

1) CRC Error
2) Form Error
3) Stuff Error
4) ACK Error
5) Bit Error

•The CAN protocol provides sophisticated error detection mechanisms discussed below :

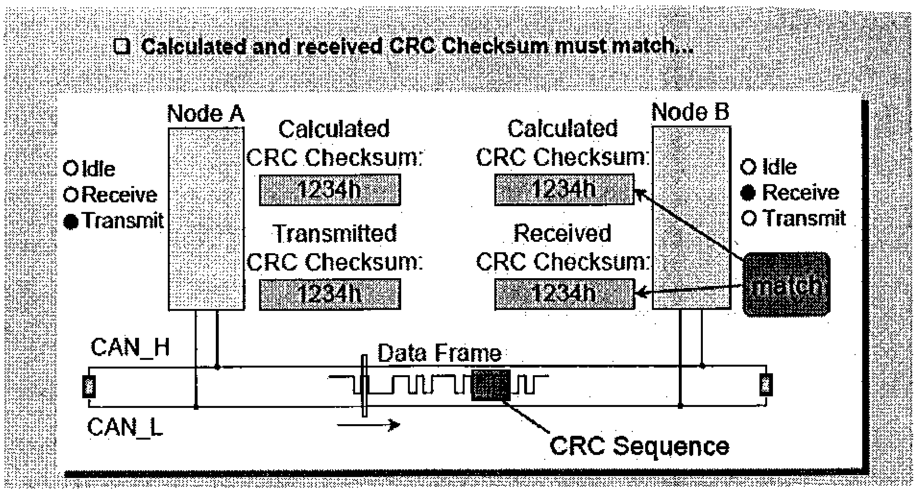


Fig 2.9: Cyclic Redundancy Check [18]

•With the Cyclic Redundancy Check, the transmitter calculates a check sum for the bit sequence from the start of frame bit until the end of the Data Field.

•This CRC sequence is transmitted in the CRC Field of the CAN frame.

•The receiving node also calculates the CRC sequence using the same formula and performs a comparison to the received sequence[Fig.2.9].

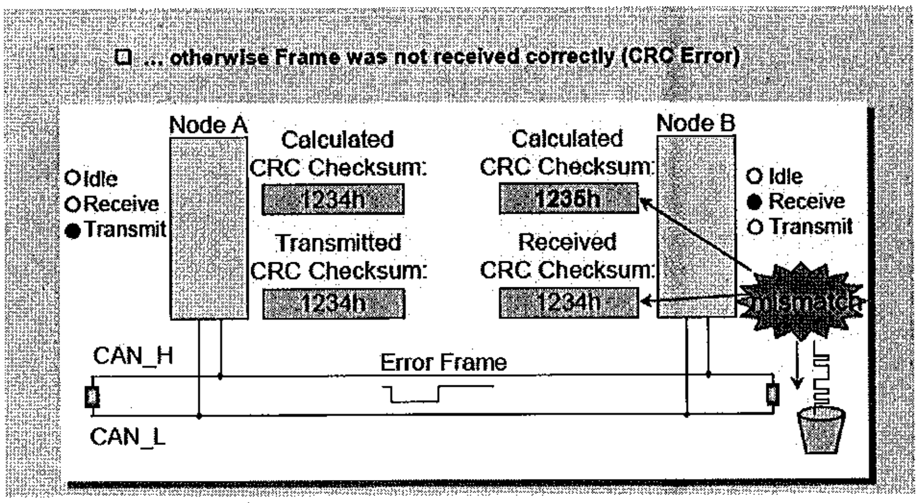


Fig 2.10: Cyclic Redundancy Check(contd.) [18]

•If node B detects a mismatch between the calculated and the received CRC sequence , then a CRC error has occurred.

•Node B discards the message [Fig.2.10] and transmits an Error Frame to request retransmission of the garbled frame.

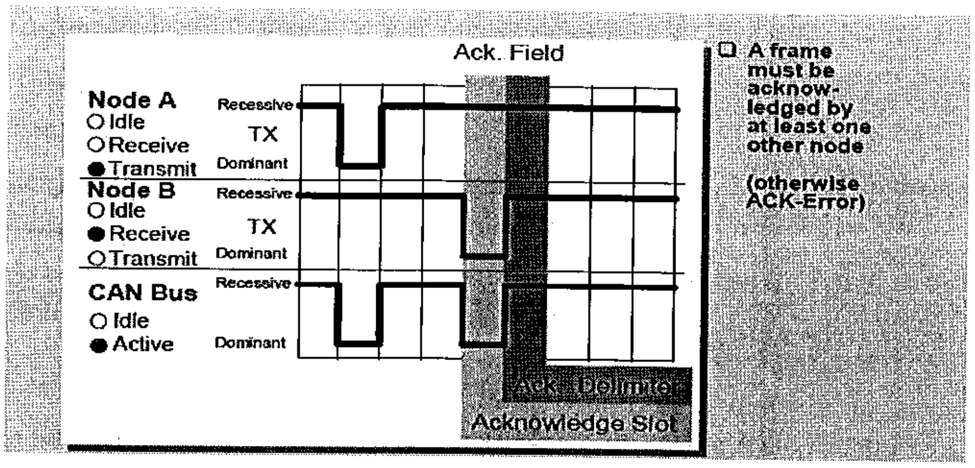


Fig 2.11 : Error Detection- Acknowledge [18]

•With the Acknowledge Check the transmitter checks in the Acknowledge Field of a message to determine if the Acknowledge Slot[Fig.2.11], which is sent out as a recessive bit, contains a dominant bit.

•If this is the case, at least one other node, (here node B) has received the frame correctly.

•If not, an Acknowledge Error has occured and the message has to be repeated. No Error Frame is generated, though.

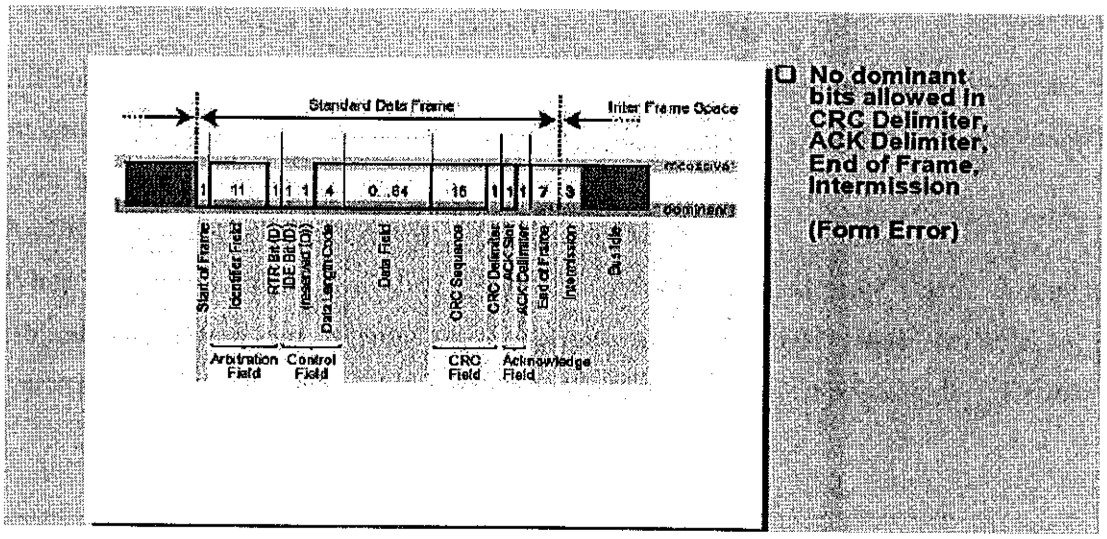


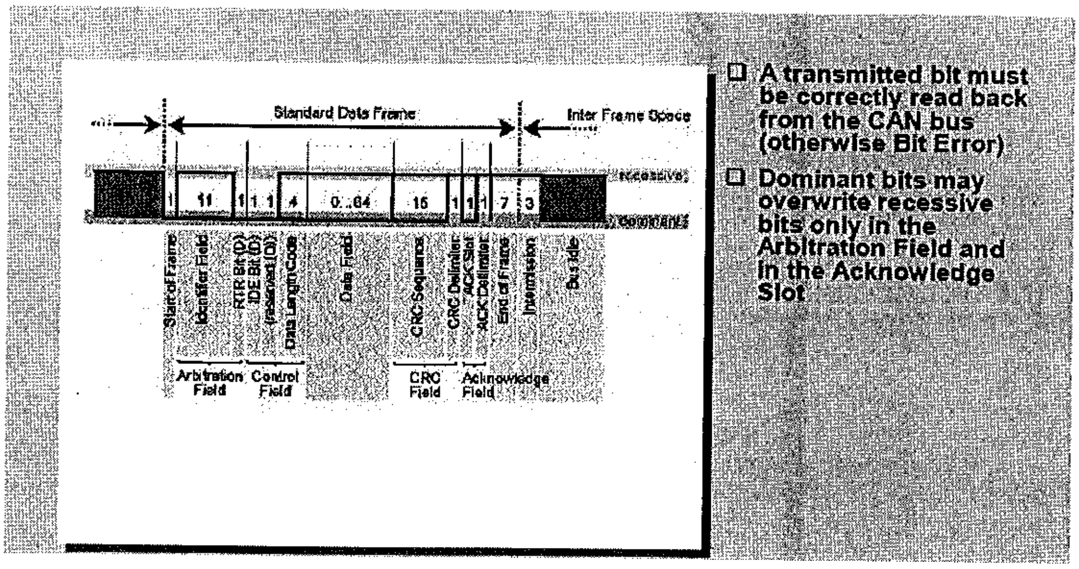
Fig 2.12: Error Detection – Frame Check [18]

•Another error detection mechanism is the Frame Check. If a transmitter detects a dominant bit in one of the four segments:

1. CRC Delimiter,

2. Acknowledge Delimiter,
3. End of Frame or
4. Interframe Space

then a Form Error occurs and an Error Frame is generated [Fig. 2.12]. The message will then be repeated.



- ☐ A transmitted bit must be correctly read back from the CAN bus (otherwise Bit Error)
- ☐ Dominant bits may overwrite recessive bits only in the Arbitration Field and in the Acknowledge Slot

Fig 2.13: Error Detection – Bit Monitoring [18]

•All nodes perform Bit Monitoring: A Bit Error occurs if a transmitter sends a dominant bit but detects a recessive bit on the bus line or, sends a recessive bit but detects a dominant bit on the bus line [Fig.2.13].

•An Error Frame is generated and the message is repeated.

•When a dominant bit is detected instead of a recessive bit, no error occurs during the Arbitration Field or the Acknowledge Slot because these fields must be able to be overwritten by a dominant bit in order to achieve arbitration and acknowledge functionality.

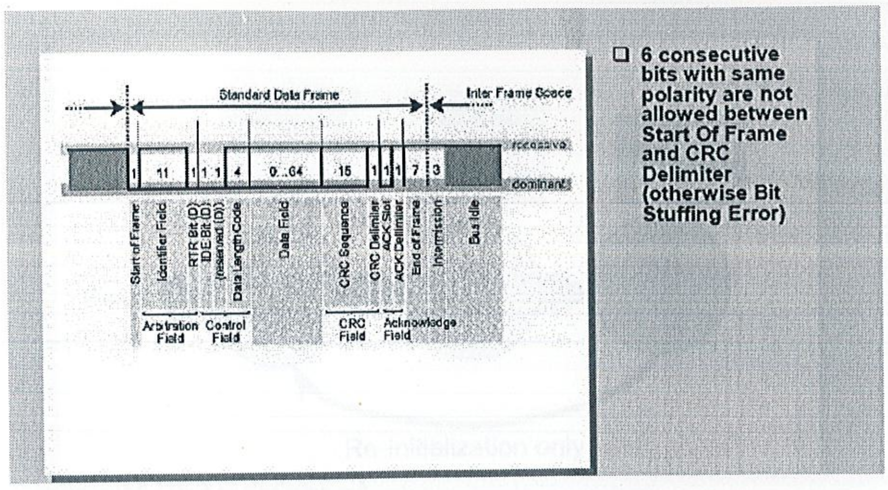


Fig 2.14 : Error Detection- Bit Stuffing Check [18]

•If six consecutive bits with the same polarity are detected between Start of Frame and the CRC
Delimiter, the bit stuffing rule has been violated [Fig. 2.14].

•A stuff error occurs and an Error Frame is generated. The message is then repeated.

2.3.4 Error Handling

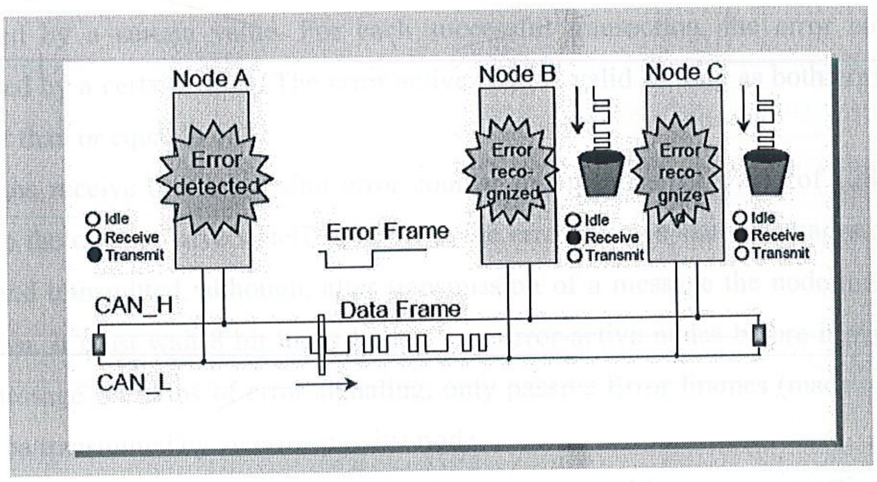


Fig 2.15: Error Handling [18]

•Detected errors are made public to all other nodes via Error Frames.

•The transmission of the erroneous message is aborted [Fig.2.15] and the frame is repeated as soon as possible.

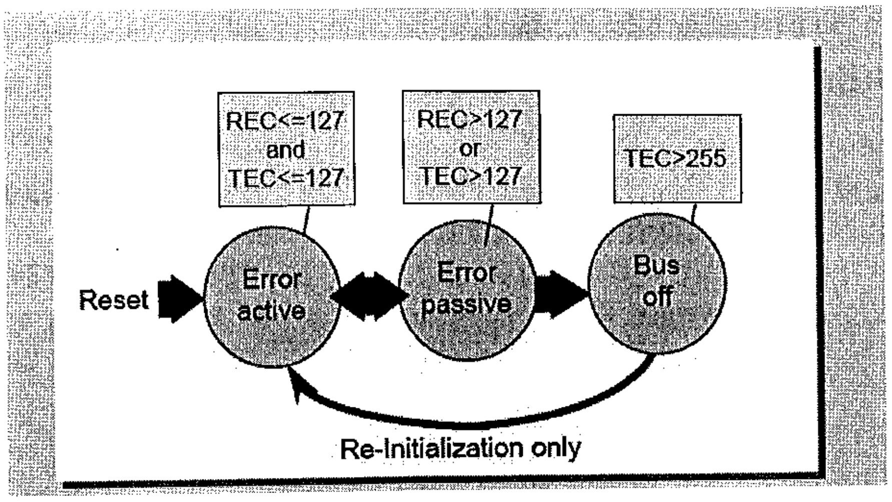


Fig 2.16: Error Handling (contd.) [18]

•Each CAN node is in one of three error states "error active", "error passive" or "bus off" according to the value of their internal error counters.

•The error-active state is the usual state after reset. The bus node can then receive and transmit messages and transmit active Error Frames (made of dominant bits) without any restrictions. During CAN communication, the error counters are updated according to quite complex rules. For each error on reception or transmission, the error counters are incremented by a certain value. For each successful transaction, the error counters are decremented by a certain value. The error active state is valid as long as both error counters are smaller than or equal to 127.

•If either the receive or the transmit error counter has reached the value of 128, the node switches to the error-passive state[Fig.2.16]. In the error-passive state, messages can still be received and transmitted, although, after transmission of a message the node must suspend transmission. It must wait 8 bit times longer than error-active nodes before it may transmit another message. In terms of error signaling, only passive Error Frames (made of recessive bits) may be transmitted by an error-passive node.

•If both error counters go below 128 again due to successful bus communication, the node switches back to the error-active state.

•One feature of the CAN protocol is that faulty nodes withdraw from the bus automatically. The bus-off state is entered if the transmit error counter exceeds the value of 255. All bus activities are stopped which makes it temporarily impossible for the station to participate in the bus communication. During this state, messages can be neither received nor transmitted. To return to the error active state and to reset the error counter values, the CAN node has to be reinitialized.

2.4 Physical Layer

2.4.1 Message Coding

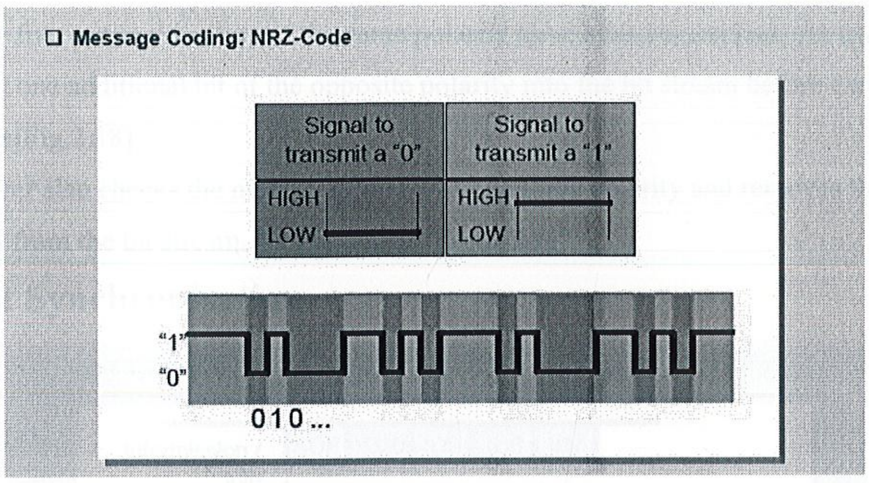
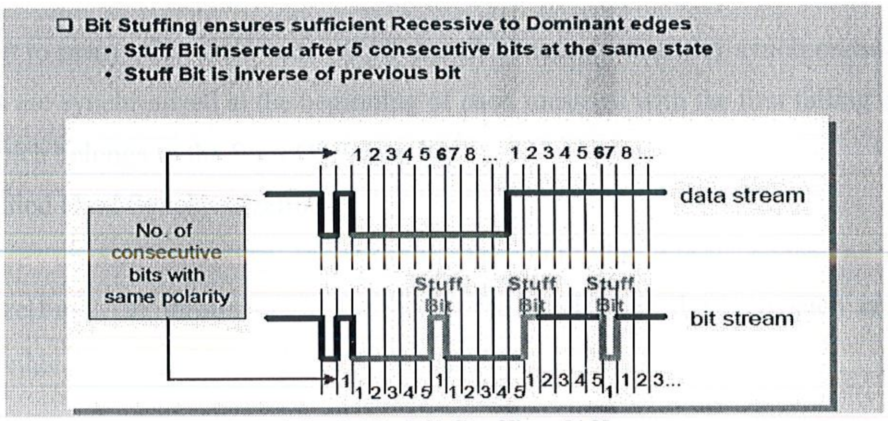


Fig 2.17: Message Coding – NRZ code [18]

•The CAN protocol uses Non-Return-to-Zero or NRZ bit coding. This means that the signal is constant for one whole bit time and only one time segment is needed to represent one bit.

•Usually, but not always, a "zero" corresponds to a dominant bit, placing the bus in the dominant state, and a "one" corresponds to a recessive bit [Fig.2.17], placing the bus in the recessive state.

2.4.2 Bit Stuffing



❑ Bit Stuffing ensures sufficient Recessive to Dominant edges
• Stuff Bit inserted after 5 consecutive bits at the same state
• Stuff Bit is inverse of previous bit
1 2 3 4 5 6 7 8 ... 1 2 3 4 5 6 7 8 ...
data stream
No. of consecutive bits with same polarity
Stuff Bit
Stuff Bit
Stuff Bit
bit stream
1 1 2 3 4 5 1 1 2 3 4 5 1 2 3 4 5 1 1 2 3...

Fig 2.18: Bit Stuffing [18]

•One characteristic of Non-Return-to-Zero code is that the signal provides no edges that can be used for resynchronization when transmitting a large number of consecutive bits with the same polarity.

•Therefore Bit stuffing is used to ensure synchronization of all bus nodes.

•This means that during the transmission of a message, a maximum of five consecutive bits may have the same polarity.

•Whenever five consecutive bits of the same polarity have been transmitted, the transmitter will insert one additional bit of the opposite polarity into the bit stream before transmitting further bits[Fig.2.18].

•The receiver also checks the number of bits with the same polarity and removes the stuff bits again from the bit stream. This is called "destuffing".

2.4.3 Bit Synchronization

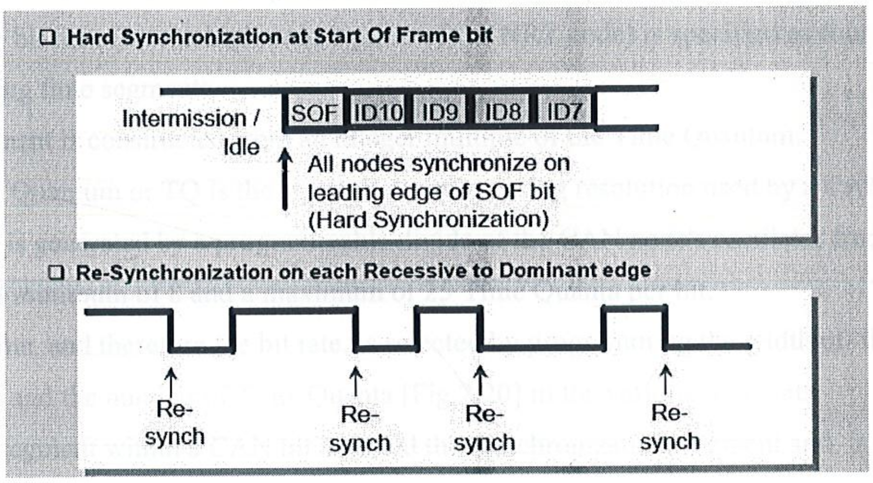

❑ Hard Synchronization at Start Of Frame bit
Intermission / Idle
SOF
ID10
ID9
ID8
ID7
All nodes synchronize on leading edge of SOF bit (Hard Synchronization)
❑ Re-Synchronization on each Recessive to Dominant edge
Re-synch
Re-synch
Re-synch
Re-synch

Fig 2.19: Bit Synchronization [18]

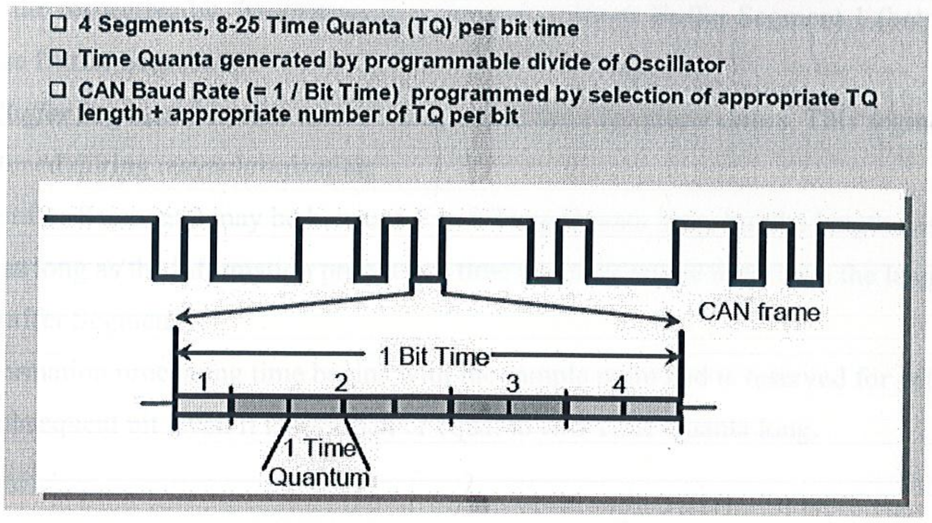
•In contrast to many other field buses, CAN handles message transfers synchronously.

•All nodes are synchronized at the beginning of each message with the first falling edge of a frame which belongs to the Start Of Frame bit.

•This is called Hard Synchronization.

•To ensure correct sampling up to the last bit, the CAN nodes need to re-synchronize[Fig.2.19] throughout the entire frame. This is done on each recessive to dominant edge.

2.4.4 Bit Construction



- 4 Segments, 8-25 Time Quanta (TQ) per bit time
- Time Quanta generated by programmable divide of Oscillator
- CAN Baud Rate (= 1 / Bit Time) programmed by selection of appropriate TQ length + appropriate number of TQ per bit

Fig 2.20: Bit Construction [18]

•One CAN bit time (or one high or low pulse of the NRZ code) is specified as four non - overlapping time segments.

•Each segment is constructed from an integer multiple of the Time Quantum.

•The Time Quantum or TQ is the smallest discrete timing resolution used by a CAN node.

•Its length is generated by a programmable divide of the CAN node's oscillator frequency.

•There is a minimum of 8 and a maximum of 25 Time Quanta per bit.

•The bit time, and therefore the bit rate, is selected by programming the width of the Time Quantum and the number of Time Quanta [Fig.2.20] in the various segments.

•The first segment within a CAN bit is called the Synchronization Segment and is used to synchronize the various bus nodes.

•On transmission, at the start of this segment, the current bit level is output.

•If there is a bit state change between the previous bit and the current bit, then the bus state change is expected to occur within this segment by the receiving nodes.

•The length of this segment is always 1 Time Quantum.

•The Propagation Time Segment is used to compensate for signal delays across the network.

•This is necessary to compensate for signal propagation delays on the bus line and through the electronic interface circuits of the bus nodes.

•*Phase Buffer Segment 1* is used to compensate for edge phase errors. This segment may be between 1 to 8 Time Quanta long and may be lengthened during resynchronization.

•The sample point is the point of time at which the bus level is read and interpreted as the value of the respective bit. Its location is at the end of Phase Buffer Segment 1 (between the two Phase Buffer Segments).

•*Phase Buffer Segment 2* is also used to compensate for edge phase errors. This segment may be shortened during resynchronization.

•Phase Buffer Segment 2 may be between 1 to 8 Time Quanta long, but the length has to be at least as long as the information processing time and may not be more than the length of Phase Buffer Segment 1.

•The information processing time begins with the sample point and is reserved for calculation of the subsequent bit level. It is less than or equal to two Time Quanta long.

CHAPTER 3

SOLUTION APPROACH: 1

3.1Why Data Reduction Techniques Are Required

The two parameters that affect system performance are bandwidth occupied by a message and latency.

- *latency* is an initial network cost, paid for every message that is transmitted in it, even for the theoretical message of zero byte length.
- *bandwidth* is the maximum network performance, achieved theoretically with messages of infinite size; the network gets close to them in messages of large length.
- If a message consumes less bandwidth, more messages would be able to be passed in the same time , therefore improving system performance.

The following equations show how the time of transmission required by a message is related to the bandwidth occupied.

Let's say the time in seconds that a network needs to send a message with n bytes is given by:

$$T(n) = \alpha + \beta n$$

This is a line equation of the form ***y=mx+c*** where α is the *constant*, and β is the line *slope* .

The parameter α is called ***latência***, or even **"*zero byte latency*"**, which is the time required to transmit a message of zero bytes .

To understand the meaning of β in the formula for T(n), we will first consider the transfer rate B(n) that is achieved for n bytes which is calculated dividing the number of bytes that have been transferred by the time that the transfer spent. In mathematical terms .

$$B(n) = \frac{n}{T(n)} = \frac{n}{\alpha + \beta n}$$

$$\frac{1}{B(n)} = \frac{T(n)}{n} = \frac{\alpha + \beta n}{n} = \frac{\alpha}{n} + \beta, \text{ supposing } n \neq 0$$

$$B(n) = \frac{1}{\frac{\alpha}{n} + \beta}$$

When n is too large, the ratio α/n becomes too little

$$B(n) \cong \frac{1}{\beta}$$

That is the maximum transfer rate that could be achieved in this network. So, it is usual to call the parameter 1/β as network *bandwidth*

If a message consumes less bandwidth, more messages would be able to be passed in the same time , therefore improving system performance.

In any CAN network ECM's i.e the electronic control modules are responsible for exchange of data among various nodes. As they are increased, the data traffic increases. Since CAN is a real time network , it becomes necessary that the data is transmitted within a given period of time. For this it is required that a large number of messages be sent in a short period of time. So every message on the bus should use minimal bandwidth of the network.

"Several data-reduction techniques have been developed using which large amounts of data can be transmitted in a short period of time, consuming very less bandwidth. An optimum DR technique can be characterized as the one that consumes the least network bandwidth and has zero message latency."

We now present a generalized data-reduction algorithm that could be applied to all data classes found in automotive multiplexing environment.

3.2 A Generalized Data Reduction Algorithm[15]

We present an algorithm for data compression(reduction of the storage space required for data by changing its format) and decompression (Decoding the original message) in order to reduce bandwidth and message latency.

- In the given algorithm, the T (DCB)bit is set to "1" and "0" to indicate compression and no compression, respectively.

- Each control module in the multiplexing system consists of a transmit buffer (TRANSBUF) and a receiver buffer (RECBUF)[Fig.25].
- Each Control Module keeps a copy of the most recently transmitted message in the TRANSBUF and a copy of the most recently received message in the RECBUF.
- Suppose an CM transmits a message Mi every t time units.
- The transmitting CM keeps a copy of message Mi in its TRANSBUF.
- The next time the CM transmits a message Mi after t time units, the CM compares the data field of the previous Mi, saved in the TRANSBUF, with the current message being transmitted.
- In the event that two or more of the data bytes of the current transmitted message are of same magnitude as of the message in TRANSBUF[Fig.3.1], the transmitting CM realizes that few of the data bytes have been repeated.
- The transmitter then initiates a series of steps outlined below, to implement data compression

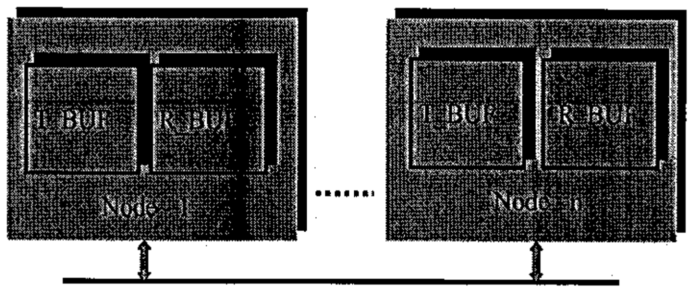


Fig 3.1 : Automotive multiplexing system consisting of 'n' number of ICM's[11]

3.2.1 Data Compression Process

1. The transmitter sets the R bit or data compression bit (DCB) in the PDU to "1".
2. The transmitter prepares a compression code (CC) to indicate repeated bytes in the recently transmitted message.
3. Each bit in the CC indicates a data byte in the message. The bits in the CC are set to "1" or "0" to indicate a repeated byte or non-repeated byte in the message, respectively. For example, if byte 3 in the message is repeated then bit 3 in CC is set to "1" and so on.

4.The non-repeated bytes are concatenated after the CC in the data field of the message being transmitted over the bus.

At the receiver, the ICM keeps a copy of the most recently received message in its R_BUF to perform decompression. The following steps are involved in data decompression process at the receiving end.

3.2.2 Data Decompression Process

1. The receiver checks the DCB of the received message.
2. If DCB is "1", then the receiver treats the first byte in the data field of the received message as the CC.
3. The receiver retrieves the repeated data bytes by indexing through the R_BUF and fetching bytes whose corresponding bits in the CC have a value "1". For example, byte 3 is fetched if bit 3 in CC is "1".
4. The entire message is recreated using the repeated bytes from R_BUF and non-repeated bytes from the received message.

This algorithm works very well when data bytes within the message remain constant with high probabilities. However, in real life situations, one or more parameters in the message may fluctuate at low rates. For example, during braking, car speed may decrease at a constant rate say 8 miles/sec, or during acceleration the speed may increase at 4 miles/sec2. Under such conditions, the above protocol would transmit the entire length of the data bytes used to represent the particular parameter even though the change in value of the parameter is very less. In the worst-case scenario, if all the parameters in the message change by small amounts, the algorithm would transmit all the eight bytes of the message without leading to any DR. This is highly undesirable since it would lead to high bus utilization and consumption of unnecessary bandwidth.

To overcome this drawback of the previous DR algorithm, we use an Adaptive Data-Reduction (ADR) algorithm which consists of adaptive and non-adaptive data reduction.

3.3 Adaptive Data Reduction Algorithm[15]

The proposed ADR algorithm consists of two parts

- Non-Adaptive data-reduction
- Adaptive data-reduction

Adaptive DR is based on the technique of delta modulation that is widely used .It is based on the principle that instead of sending the absolute value of a signal at each time instant, only the changes in the signal values from one time instançe to another (delta) are transmitted.

To achieve data reduction, the first byte in the data field of the CAN message is used to indicate delta compression. This byte is called the delta compression code (DCC).

In the ADR technique[Fig.3.3], if one or more signal fields in the message Ci transmitted at time m + t change their value since their previous transmission at time m, the transmitting node executes the following series of steps.

- The transmitting node computes à delta compression code (DCC). A value of "1" in the DCC indicates the delta change in the value of the signal rather than the absolute value. A value of "0" in DCC indicates no change in the value of the signal.
- Since a copy of the message transmitted at time m is stored in TX_BUF, the transmitting node computes differences (deltas) in the value of the corresponding signals in TX_BUF at time m with that at time m + t.
- The compressed version of the original CAN message having an identifier whose value is one less than the value of the identifier of the original message is encoded with the delta signal values.
- The delta-compressed CAN message carrying the delta signals is sent over the CAN bus to the receiving node.
- . In the event that the value of a delta signal exceeds the length of the assigned field, the absolute values of all the signals (i.e. the original CAN message) are transmitted rather than the delta-compressed version of the message. This assumption is based on the fact that a drastic change in the value of a particular signal could reflect a critical condition in which case it would be necessary to communicate absolute values of all signals within the message.
- The receiving node in the system decompresses the delta-compressed CAN message[Fig3.6] sent by the transmitter. The following series of steps are executed to

perform data-decompression. Assuming that a delta-compressed message DPi is received.

- The receiving node checks the identifier of the CAN message DPi.
  If the message is a delta-compressed message, the first byte in the data
  field of the message is treated as the DCC.
- The receiver then fetches a copy of the most recently received Pi from the REC_BUF.
- The DCC acts as an index to the corresponding signal fields within the delta-compressed message. For example, a value of "1" in the first bit position of the DCC means that the first signal field within the message is a delta signal of the first signal field within the message Pi from REC_BUF. This delta signal is either added to or subtracted from the corresponding signal field within the message Pi from REC_BUF to get the new value of the signal.
- A value of "0" in the DCC means that the corresponding signal has not changed its value since the previous transmission. The absolute value of this signal is fetched from the previous Pi in REC_BUF[Fig.3.2].
- The receiver reconstructs all the signals within the message in a similar fashion.
- The receiver then updates the REC_BUF with this new message, overwriting the previous Pi.

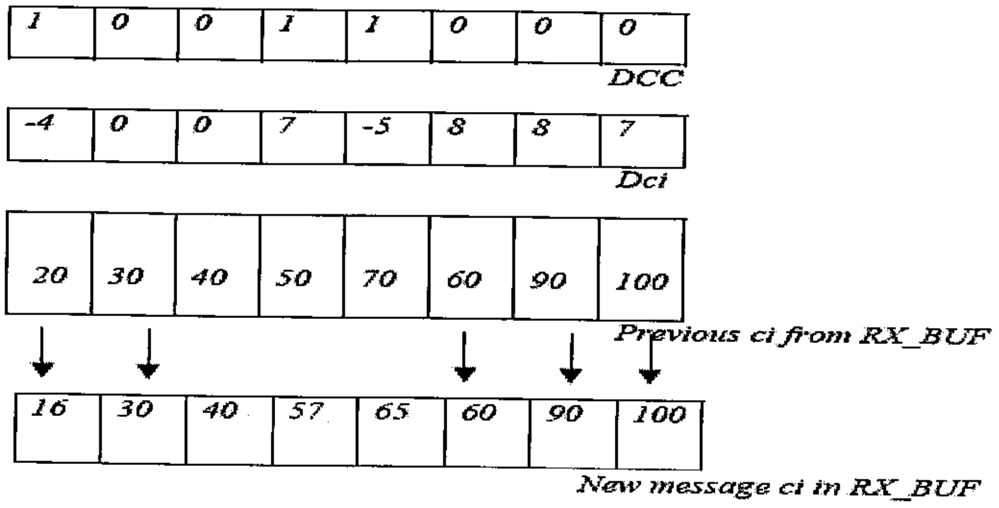


Fig3.2: Data-decompression process

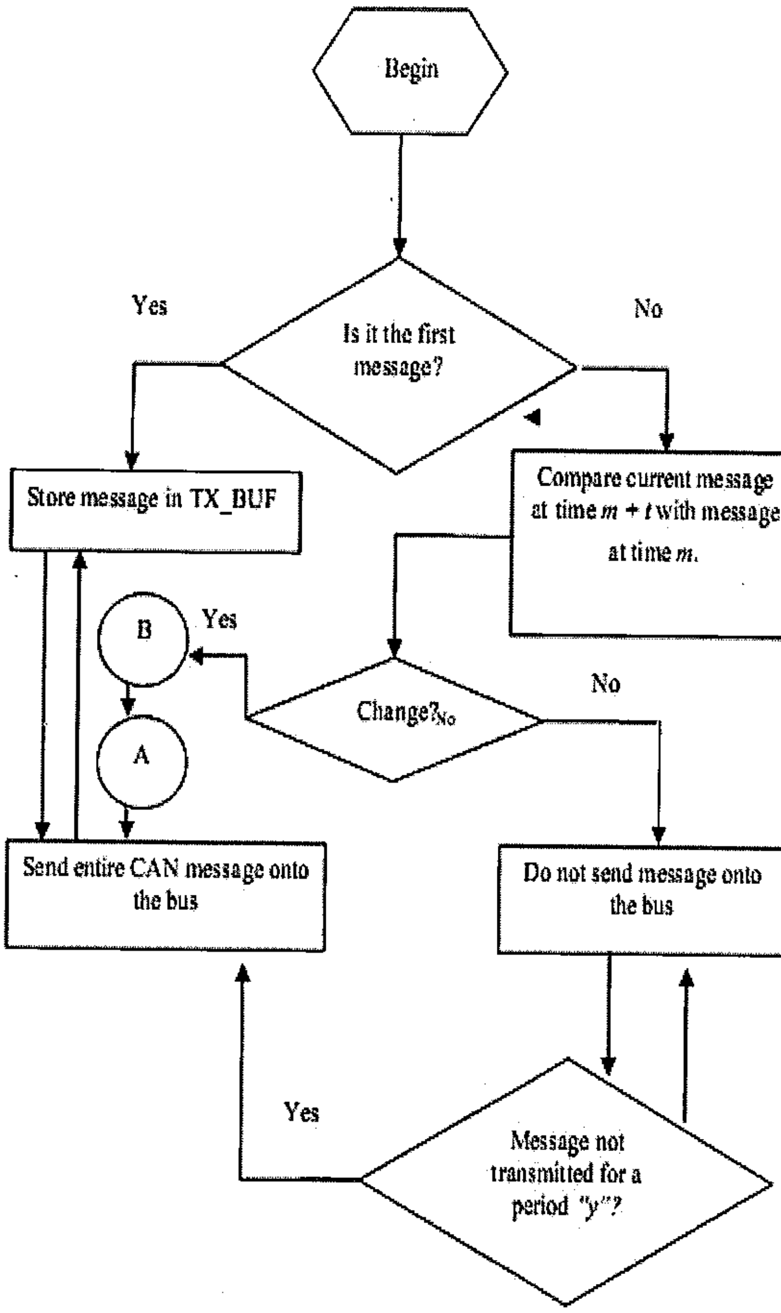

Begin
Is it the first message?
Yes
No
Store message in TX_BUF
Compare current message at time m + t with message at time m.
B
Yes
A
Change?No
No
Send entire CAN message onto the bus
Do not send message onto the bus
Yes
Message not transmitted for a period "y"?

Fig.3.3 : Adaptive data-reduction algorithm[15]

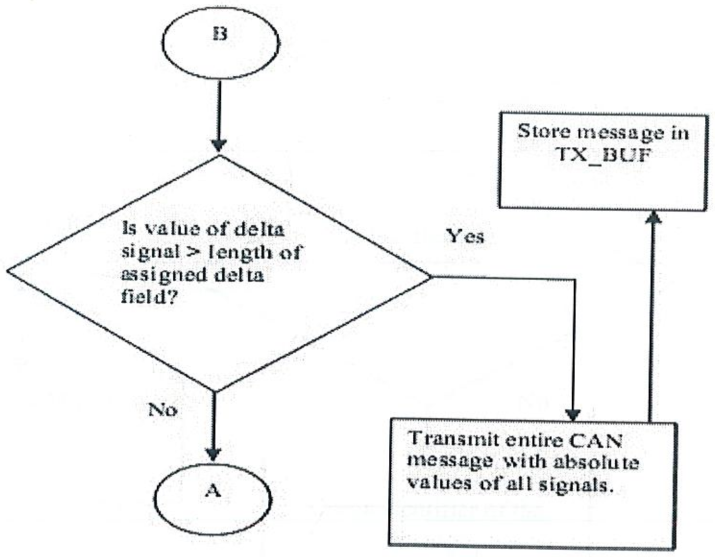

B
Is value of delta signal > length of assigned delta field?
Yes
No
Store message in TX_BUF
Transmit entire CAN message with absolute values of all signals.
A

Fig.3.4 : Adaptive data-reduction algorithm continued[15]

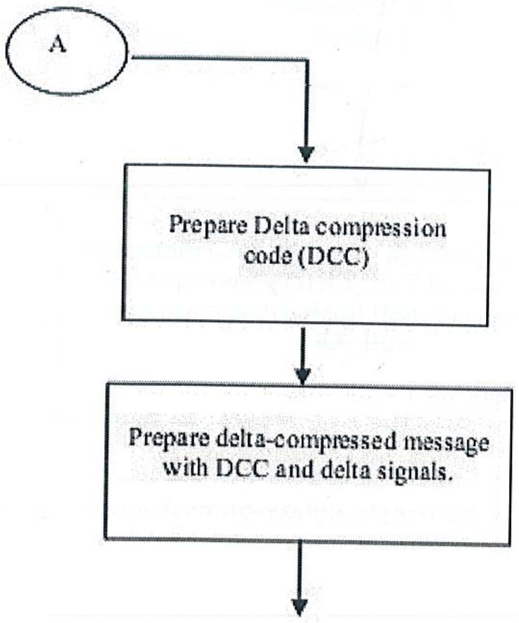

A
Prepare Delta compression code (DCC)
Prepare delta-compressed message with DCC and delta signals.

Fig.3.5 : Adaptive data-reduction algorithm continued [15]

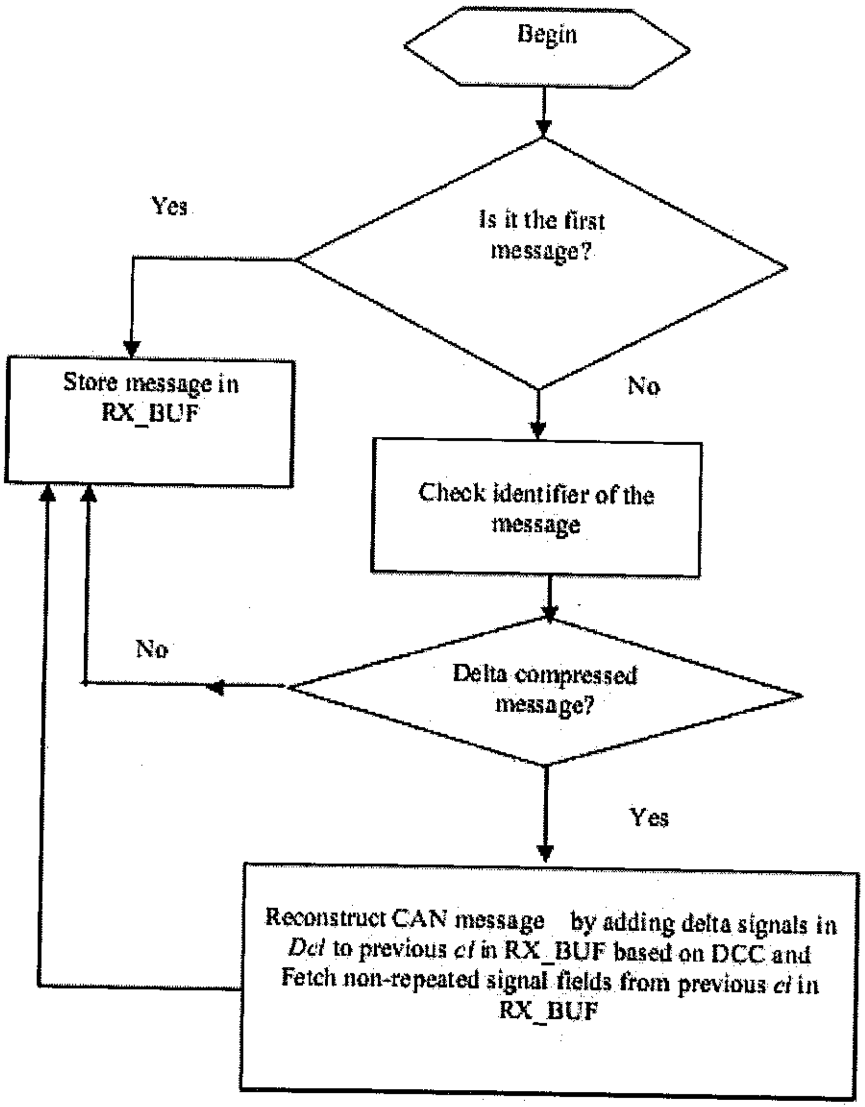

Begin
Is it the first message?
Yes
No
Store message in RX_BUF
Check identifier of the message
Delta compressed message?
No
Yes
Reconstruct CAN message by adding delta signals in *Dci* to previous *ci* in RX_BUF based on DCC and Fetch non-repeated signal fields from previous *ci* in RX_BUF

Fig 3.6 : Data-decompression algorithm [15]

CHAPTER 4

SOLUTION APPROACH: 2

Application Of Huffman Coding For Data Reduction In CAN

Five data reduction techniques have been developed which can be applied in automotive environment. They are discussed as follows:

1) Simple Huffman coding
2) Arithmetic coding
3) Higher order arithmetic coding
4) Textual substitution coding
5) Command data stream reference coding

1) Simple Huffman Coding

Huffman coding works on the principle of assigning a shorter bit sequence to the characters having high frequency of occurrence, and a longer bit sequence to the characters having low frequency of occurrence so that the encoded string becomes shorter in length. The main limitation of Huffman coding is the requirement of keeping a copy of the probability table at each node in the automotive multiplexing system.

2) Arithmetic Coding

In this coding, after assigning a frequency of occurrence to each symbol, a range of real numbers is assigned to each symbol. The length of this range is equal to the probability of the symbol. For example, if the symbol has a probability 0.1, then the assigned range of numbers will be [0.0 to 0.1]. Following the arithmetic algorithm, a message consisting of a stream of symbols can be represented by a single floating-point number

3) Higher Order Arithmetic Coding

An extension of arithmetic coding is higher order arithmetic coding, in which the probability of each incoming symbol is calculated on the basis of the context in which the symbols were previously encountered. After determining these probabilities, encoding of arithmetic coding is used. A higher order arithmetic coding scheme requires a large amount of memory at each node.

4) Textual Substitution Coding

In a textual substitution algorithm, variable length strings of symbols are encoded into a single token. This token is used as an index to a phrase dictionary maintained at the receiving end.

5) Command Data Stream Coding

The above mentioned data-compression algorithm has been used to devise another algorithm, CDSR coding. The CDSR coding scheme is especially designed for automotive multiplexing application. In the CDSR scheme, a reference dictionary is maintained at each node in the multiplexing system. When a message is generated, the reference dictionary at the transmitting side is referred, and a token is generated instead of the actual message. This token indicates the position of the first symbol in the transmitted message and the message length. A copy of the message available at the receiving end is located with the help of the transmitted token. Kempf and Strenzal further investigated the application of common data-stream coding and proposed a communication protocol to overcome the drawback associated with it. Among all six data-compression algorithms, simple Huffman coding and common data-stream coding are two promising candidates for automotive multiplexing. The major drawback of all the DR techniques was that they could only be applied to text-data classes in automotive body electronics .

4.1 Huffman Coding

In 1951, David A. Huffman worked on the problem of finding the most efficient binary code. He hit upon the idea of using a frequency-sorted binary tree and quickly proved this method the most efficient.

Huffman coding is a statistical technique which attempts to reduce the amount of bits required to represent a string of symbols. The algorithm accomplishes its goals by allowing symbols to vary in length. Shorter codes are assigned to the most frequently used symbols, and longer codes to the symbols which appear less frequently in the string.

A set of symbols and their frequency of occurrence is given and we have to form a binary code (a set of codewords) with minimum expected codeword length .This is how the length of a message passed through CAN would be minimized.

Huffman coding uses a specific method for choosing the representation for each symbol, resulting in a prefix code (sometimes called "prefix-free codes", that is, the bit string representing some particular symbol is never a prefix of the bit string representing any other symbol) that expresses the most common source symbols using shorter strings of bits than are used for less common source symbols. Huffman was able to design the most efficient compression method of this type: no other mapping of individual source symbols to unique strings of bits will produce a smaller average output size when the actual symbol frequencies agree with those used to create the code. A method was later found to design a Huffman code in linear time if input probabilities (also known as *weights*) are sorted.

4.2 Algorithm for building a Huffman tree

Step 1. Create a parentless node for each symbol. Each node should include the symbol and its frequency of occurrence[Table.1].

Step 2. Select the two parentless nodes with the lowest frequencies.

Step 3. Create a new node which is the parent of the two lowest frequency nodes.

Step 4. Assign the new node a frequency equal to the sum of its children's probabilities.

Step 5. Repeat from Step 2 until there is only one parentless node left.

A Huffman Coding Tree is built from the observed frequencies of characters in a document.

• The document is scanned and the occurrence of each character is recorded.

• Next, a Binary Tree is built in which the external nodes store the character and the corresponding character frequency observed in the document

4.3 Building a Huffman Coding Tree

Consider the observed frequency of characters in a string that requires encoding:

Table I. Character Frequency Table

Character	C	D	E	F	K	L	U	Z
Frequency	32	42	120	24	7	42	37	2

The first step is to construct a Priority Queue and insert each frequency-character (key-element) pair into the queue.

Step 1: The queue is arranged in ascending order of frequency of the nodes.

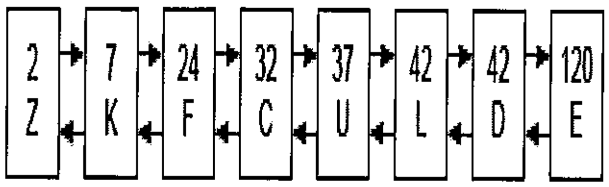


Fig: 4.1 Sorted, sequence-based, priority queue.

In the second step, the two Items with the lowest key values are removed from the priority queue.

• A new Binary Tree is created with the lowest-key Item as the *left* external node, and the second

lowest-key Item as the *right* external node.

• The new Tree is then inserted back into the priority queue.

Step 2: The leftmost two nodes are combined to form one parent node

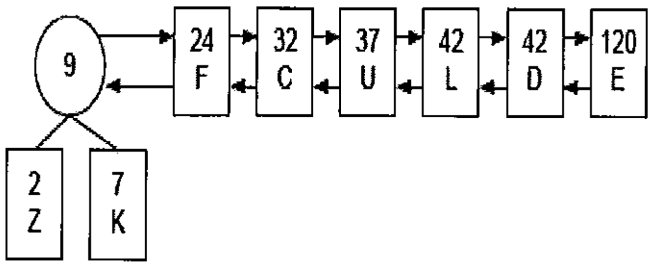


Fig 4.2:Binary tree (contd.)

The process is continued until only *one* node (the Binary Tree) is left in the priority queue.

Step 3: This process is repeated successively.

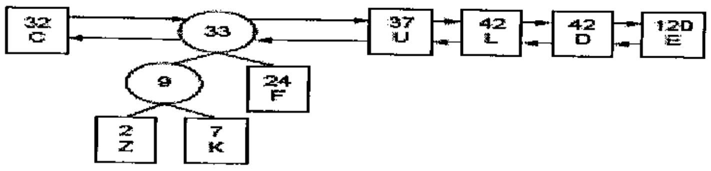


Fig.4.3: Binary tree (contd.)

Step 4: The step again is shown here.

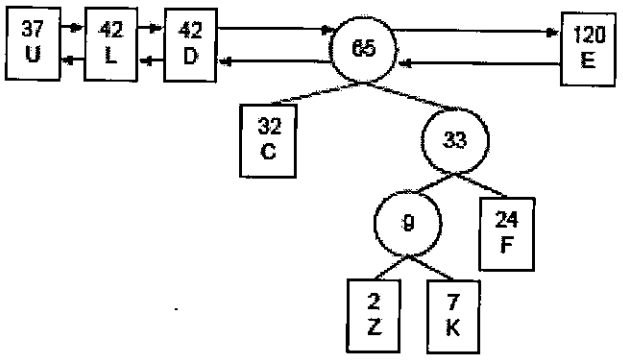


Fig.4.4: Binary tree (contd.)

Step 5: The binary tree is being formed here

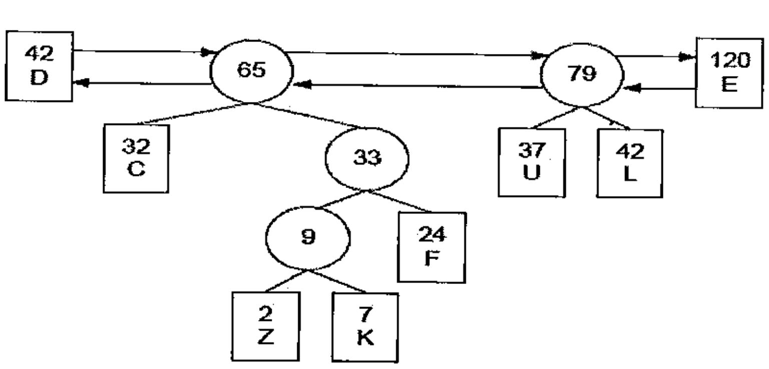


Fig.4.5: Binary tree (contd.)

Final tree, after $n = 8$ steps:

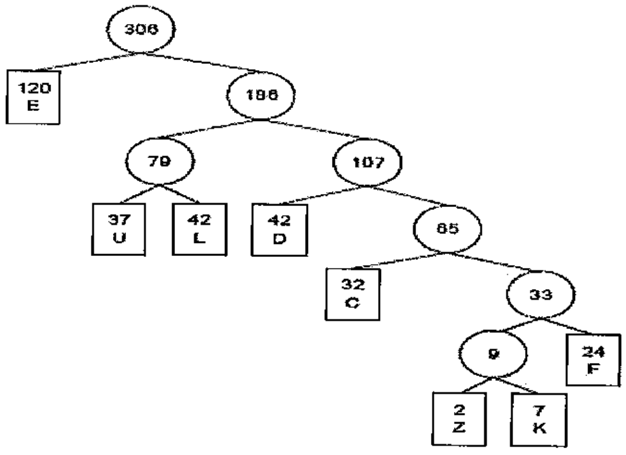


Fig.4.6: Final Tree Formed

4.4 Encoding

Once a Huffman code has been generated, data may be encoded simply by replacing each symbol with its code. Given a code (corresponding to some alphabet) and a message it is easy to encode the message. Just replace the characters by the codewords.

Method for encoding :

1) Perform a traversal of the tree to obtain new code words
2) Going left is a 0
3) Going right is a 1
4) Code word is only completed when a leaf node is reached

Create a lookup table storing the binary code corresponding to the path to each letter

Table II. Compression Table

Character	Frequency	Code	# bits
C	32	1110	4
D	42	110	3
E	120	0	1
F	24	11111	5
K	7	111101	6
L	42	101	3
U	37	100	3
Z	2	111100	6

Encode:

DECK

110ECK

1100CK

11001110K

110001110111101

ASCII representation would require 32 bits. Huffman encoding requires 15 bits.

This is how we can make use of Huffman Coding Algorithm in serial bus communication like the CAN. Since it occupies less bits, the message bandwidth would be reduced which would help increase the system performance.

4.5 Decoding

To decode a bit stream (from the leftmost bit), start at the root node of the Tree:

1) Move to the left child if the bit is a "0".

2) Move to the right child if the bit is a "1".

3) When an external node is reached, the character at the node is sent to the decoded string.

4) The next bit is then decoded from the root of the tree.

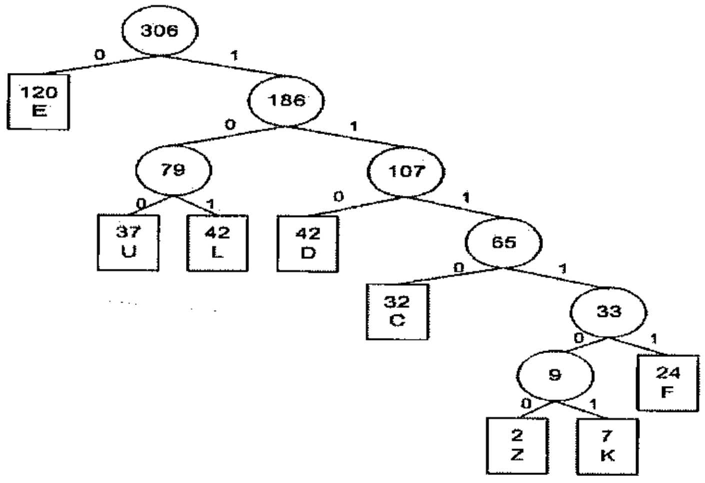


Fig 4.7: Decoding

Decode:

for example if we decode "FED"

111110110

F0110

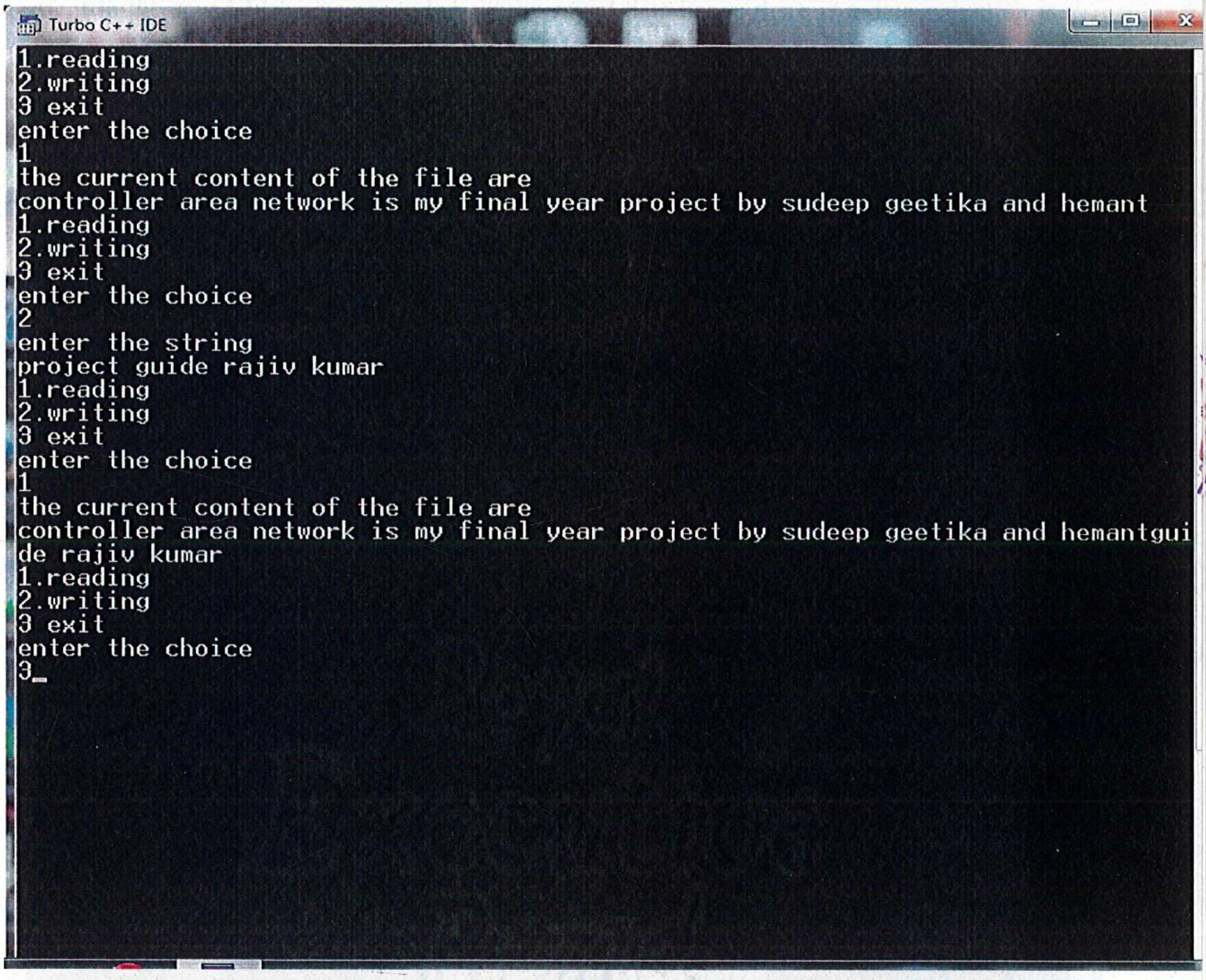
F E 110

F E D

CHAPTER 5

CONCLUSION AND RESULTS

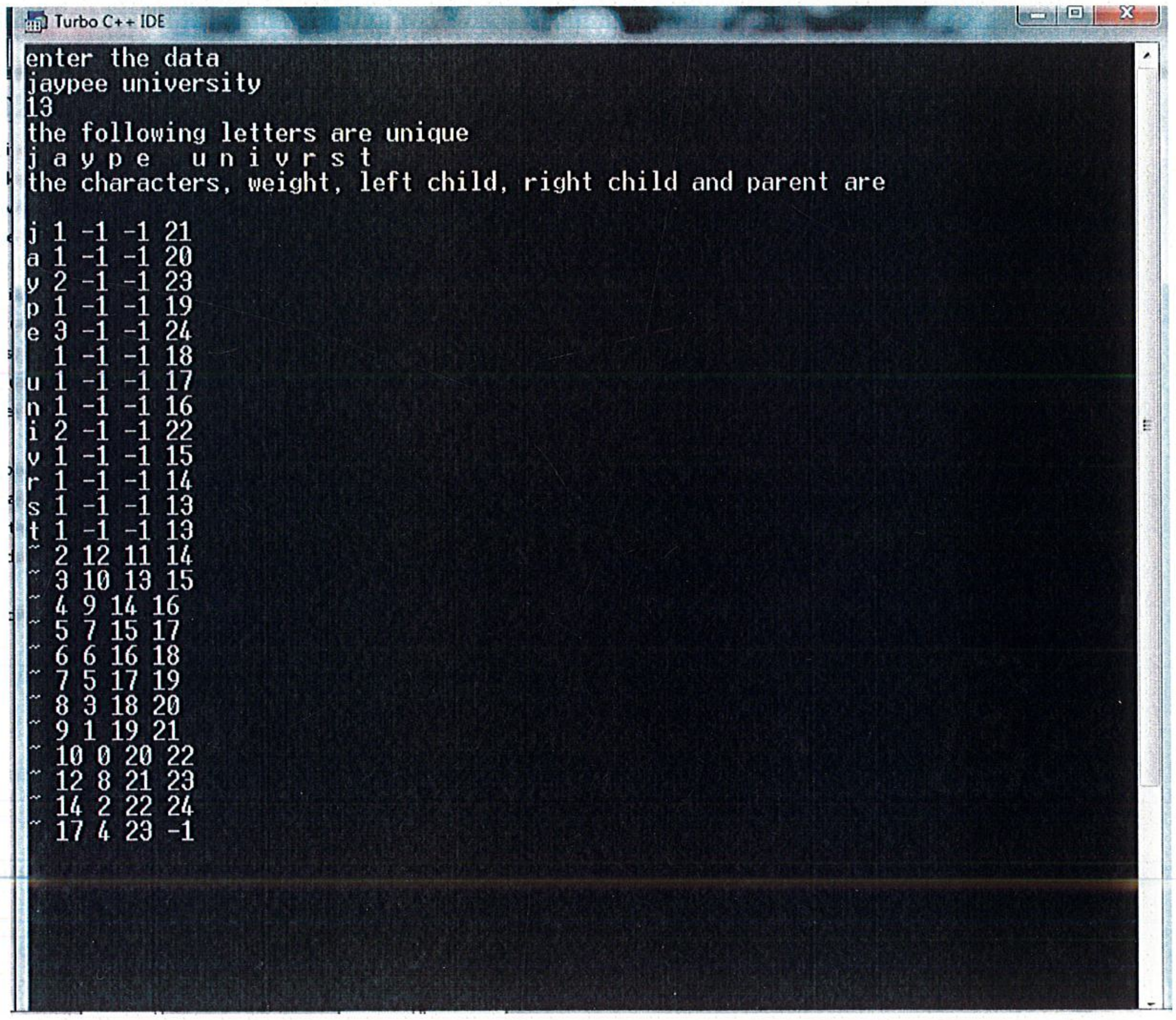
Output For Adaptive Data Reduction In CAN



```
1.reading
2.writing
3 exit
enter the choice
1
the current content of the file are
controller area network is my final year project by sudeep geetika and hemant
1.reading
2.writing
3 exit
enter the choice
2
enter the string
project guide rajiv kumar
1.reading
2.writing
3 exit
enter the choice
1
the current content of the file are
controller area network is my final year project by sudeep geetika and hemantgui
de rajiv kumar
1.reading
2.writing
3 exit
enter the choice
3_
```

Here first we saved a string in a file named "record.txt" this file read " controller area network is my final year project by sudeep geetika and hemant ".The switch option gives us options whether to read the file, write in the file or to exit . If we choose the read option then the contents of the file will be displayed on the screen . Then on write option we want to feed another message to the saved string now we will enter the string to be saved let it be "project guide rajiv kumar" .Here in normal case whole string would have been get saved in the "record.txt" but now project is already there in the saved string so only "guide rajiv kumar" gets appended at the end of saved data and the saved data gets edited now the whole string along with new appended data gets saved in "record.txt" i.e. only the strings which are not present in the saved file gets appended in the saved data hence there is a data reduction is being performed here.

Output of Huffman Coding



```
enter the data
jaypee university
13
the following letters are unique
j a y p e   u n i v r s t
the characters, weight, left child, right child and parent are

j 1 -1 -1 21
a 1 -1 -1 20
y 2 -1 -1 23
p 1 -1 -1 19
e 3 -1 -1 24
  1 -1 -1 18
u 1 -1 -1 17
n 1 -1 -1 16
i 2 -1 -1 22
v 1 -1 -1 15
r 1 -1 -1 14
s 1 -1 -1 13
t 1 -1 -1 13
~ 2 12 11 14
~ 3 10 13 15
~ 4 9 14 16
~ 5 7 15 17
~ 6 6 16 18
~ 7 5 17 19
~ 8 3 18 20
~ 9 1 19 21
~ 10 0 20 22
~ 12 8 21 23
~ 14 2 22 24
~ 17 4 23 -1
```

This is the output of C code for Huffman coding here a string "jaypee university of information technology " is entered . The code finds out the number of occurrence of each letters and thus saves the number of occurrences for unique letters, it performs Huffman algorithm for making tree and a binary tree is being made following the algorithm

In Huffman coding, the assignment of codewords to source messages is based on the probabilities with which the source messages appear in the message ensemble. Messages which appear more frequently are represented by short codewords; messages with smaller probabilities map to longer codewords. This is how it can be used for data reduction in a serial bus system like CAN.

Have we made things any better by using Huffman coding ?

While discussing Huffman coding we took a string having 306 letters

Character	Frequency	Code	# bits
C	32	1110	4
D	42	110	3
E	120	0	1
F	24	11111	5
K	7	111101	6
L	42	101	3
U	37	100	3
Z	2	111100	6

Bits to encode the text = summation of (frequency*bits)

=32*4 + 42*3 + 120*1 + 24*5 +7*6 + 42*3 + 37*3 + 2*6

=128+126+120+120+42+126+111+12

=785

ASCII would take 8 * 306 = 2448 bits

Hence we reduced the data by 16663 bits

This is how the use of Huffman Coding can be done for reducing the number of bits occupied by a message passed through CAN thus reducing the bandwidth occupied by a message and improving system performance.

APPENDIX: A

A PROGRAM IN 'C' FOR ADAPTIVE DATA REDUCTION IN CAN :

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
#include<stdio.h>
#include<process.h>
#include<string.h>
void main()
{
         clrscr();
        int ch,flag=1;
        char arr[100],check[100],file_data[100];
        while(1)
        {
                cout<<"1.reading"<<endl;
                cout<<"2.writing"<<endl;
                cout<<"3 exit"<<endl;
                cout<<"enter the choice"<<endl;
                cin>>ch;
                switch(ch)
                {
                        case 1:
                                ifstream fin1("record.txt");
                                if(!fin1)
                                {
                                        cout<<"file cannot be opened"<<endl;
                                        cout<<"press any key to exit"<<endl;
                                        getch();
                                        exit(1);
```

```
          }
          cout<<"the current content of the file are"<<endl;
          while(!fin1.eof())
          {
                    fin1.getline(file_data,100);
                    cout<<file_data<<" ";
          }
          fin1.close();
          cout<<"\n";
          break;
case 2: cout<<"enter the string"<<endl;
          gets(arr);
          int len=strlen(arr);
          arr[len]=' ';
          arr[len+1]='\0';
          int i=0,j=0;
          while(i<=len+1)
          {

                    if(arr[i]!=' ')
                    {
                              check[j]=arr[i];
                              i++;
                              j++;
                    }
                    else if(arr[i]==' '||arr[i]=='\n'||arr[i]=='\0')
                    {
                              check[j]='\0';
                              j=0;
                              flag=1;
                              i++;
```

```
ifstream fin("record.txt");
if(!fin)
{
        cout<<"file could not be opened"<<endl;
        cout<<"press any key to exit"<<endl;
        getch();
        exit(0);
}
while(!fin.eof())
{
    fin.getline(file_data,100);
    if(strcmp(check,file_data)==0)
    {
            flag=0;
            break;
    }
}
fin.close();
if(flag!=0)
{
        ofstream fout;
        fout.open("record.txt",ios::app);
        if(!fout)
        {
                cout<<"file cannot be
                    opened"<<endl;
                cout<<"!! press any key to
                        exit!!!";
                getch();
                exit(0);
        }
```

```
                                        fout<<check<<"\n";
                                        fout.close();
                                    }
                                }

                            }
                            break;
                        case 3: cout<<"press any key to exit..."<<endl;
                            exit(0);
                        default:cout<<"wrong choice"<<endl;
                            break;
                    }
                }}
```

APPENDIX: B

A PROGRAM IN 'C' FOR HUFFMAN CODING :

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#include<string.h>
class basic
{
   public:
          char ch;
          int weight;
};
class huff_tree
{
          public:
            basic asdf;
            int parent,leftchild,rightchild;
};
void select(huff_tree *ht,int n,int *s8,int *s9)
{
          int min1=9999;
          int min2=9999;
          int pos1,pos2;
          for(int i=0;i<n;i++)
          {
                    if(ht[i].parent==-1)
                    {
                              if(ht[i].asdf.weight<=min1)
                              {
```

```
                        pos2=pos1;
                        min2=min1;
                        pos1=i;
                        min1=ht[i].asdf.weight;
                    }
                    else
                    {
                        pos2=i;
                        min2=ht[i].asdf.weight;
                    }
                }
        }
        *s8=pos1;
        *s9=pos2;
}
void createtree(huff_tree *ht,int count,basic *obj)
{
        int m=(2*count)-1;
        int s1,s2;
        for(int i=0;i<m;i++)
        {
                if(i<count)
                {
                        ht[i].asdf.ch=obj[i].ch;
                        ht[i].asdf.weight=obj[i].weight;
                }
                else
                {
                        ht[i].asdf.ch='~';
                        ht[i].asdf.weight=-1;
                }
                ht[i].parent=-1;
                ht[i].leftchild=-1;
                ht[i].rightchild=-1;
        }
    /*  for(i=0;i<m;i++)
        {
                cout<<"\n"<<ht[i].asdf.ch<<" "<<ht[i].asdf.weight<<" ";
        } */
        for(i=count;i<m;i++)
        {
                select(ht,i,&s1,&s2);
                ht[i].asdf.weight=ht[s1].asdf.weight+ht[s2].asdf.weight;
            //  ht[i].asdf.ch=ht[s1].asdf.ch+ht[s2].asdf.ch;
                ht[i].leftchild=s1;
                ht[i].rightchild=s2;
                ht[s1].parent=i;
                ht[s2].parent=i;
        }
        cout<<"the characters, weight, left child, right child and parent are"<<endl;
        for(i=0;i<m;i++)
        {
                cout<<"\n"<<ht[i].asdf.ch<<" "<<ht[i].asdf.weight<<" "<<ht[i].leftchild<<"
"<<ht[i].rightchild<<" "<<ht[i].parent;
        }
}
void huffman_code(huff_tree *ht,int count)
{


}
void setfreq(basic *obj,int count,char *data)
{
        for(int i=0;i<count;i++)
        {
                int j=0;
                int count=0;
                while(j<strlen(data))
                {
                        if(obj[i].ch==data[j])
                        {
                                count++;
                        }
                        j++;
                }
                obj[i].weight=count;
        }
        cout<<endl;
    //  cout<<"the character with weight are as follows"<<endl;
    /*  for(i=0;i<count;i++)
        {
                cout<<obj[i].ch<<" "<<obj[i].weight<<"\n";
        } */
}
void setdata(basic *obj,int count,char *data)
{
        int i,j,k,flag=0;
        obj[0].ch=data[0];
        for(i=1,k=1;data[i]!='\0';i++)
        {
                flag=0;
                for(j=i-1;j>=0;j--)
                {
                        if(obj[j].ch==data[i])
                        {
                                flag=1;
                                break;
                        }
                }
                if(flag!=1)
                {
                        obj[k].ch=data[i];
                        k++;
                }
        }
        cout<<"\n";
        cout<<"the following letters are unique "<<endl;
        for(i=0;i<count;i++)
        {
                cout<<obj[i].ch<<" ";
        }
}
int check(char *data)
{
        int count=0,flag=1;
        for(int i=0;data[i]!='\0';i++)
        {
                flag=1;
                for(int j=i-1;j>=0;j--)
                {
                        if(data[i]==data[j])
                        {
                                flag=0;
                                break;
                        }
                }
                if(flag!=0)
                {
                        count++;

                }
        }
        cout<<count;
        return count;
}
void main()
{
        clrscr();
        char data[100];
        cout<<"enter the data"<<endl;
        gets(data);
        int count=check(data);
        basic *obj=new basic[count];
        setdata(obj,count,data);
        setfreq(obj,count,data);
        int num=(2*count)-1;
        huff_tree *ht=new huff_tree[num];
        createtree(ht,count,obj);
        huffman_code(ht,count);
        getch();
}
```

REFERENCES

1. Controller Area Network (CAN) : Univ.-Prof. Dr. Thomas Strang, Dipl.-Inform. Matthias Rockl
2. Review of Researches in Controller Area Networks : Evolution and Applications : Wei Lun Ng , Chee Kyun Ng, Borhanuddin Mohd. Ali, Nor Kamariah Noordin, and Fakhrul Zaman Rokhani
3. A Unified Approach to High-Gain Adaptive Controllers : Ian A. Gravagne ,John M.Davis,Jeffrey J. DaCunha
4. Controller Area Network (CAN) Basics : Author: Keith Pazul Microchip Technology Inc.
5. CANOPEN – Higher layer protocol based on CAN supports device profiles for I/O modules ,motion control : Wilfred Voss esd electronics, Inc. Hatfield, MA
6. Controller Area Network (CAN) : EECS 461, Fall 2007_ J. A. Cook J. S. Freudenberg
7. A Neural Network Approach for Controller Area Network Message Scheduling Control : Chuan Ku Lin, Hao-Wei Yen, Mu-Song Chen, and Chi-Pan Hwang
8. CAN : esd gmbh Vahrenwalder Str. 205 D-30165 Hannover
9. NI-CAN Hardware and Software Manual
10. Local Interconnect Network (LIN) – Packaging and Scheduling : Magnus Ahlmark Malardalen Real-Time Research Centre (MRTC);Department of Computer Engineering, Malardalen University (MDH)
11. Bandwidth Reduction for Controller Area Networks Using Adaptive Sampling : Ian A. Gravagne; John M. Davist Jeffrey J. Dacunhat, Robert J. Marks
12. Huffman Coding and Compression of Data –Lawrence M. Brown
13. Huffman Coding -Patrick J. Van Fleet University of South Florida and University of St. ThomasSACNAS 2009
14. CAN Introduction and Primer : by Robert boys
15. An Adaptive Data-Reduction Protocol for the Future In-Vehicle Networks : Praveen R. Ramteke and Syed Masud Mahmud (Electrical and Computer Engineering Department, Wayne State University)

14.An Adaptive Data Reduction protocol for future in – vehicle networks : Praveen Kumar

Ramesh Ramteke

15.Atmel Microcontrollers for Controller Area Network (CAN) : *By Michel Passemard, Industrial Control Business Development Director*

16. www.wikipedia.org

17. www.can-cia.org

18. http://ecee.colorado.edu/

19. www.springer.com