# Network protocols used in the automotive industry

Jon Bell[1]

Doc. ref. SD/TR/PRO/01
July 24, 2002

[1]email jpb@aber.ac.uk

## 1 Introduction

This report describes some of the different network protocols used in the automotive industry and then discusses similarities and differences between them.

The report is divided into three main sections. The first describes the Controller Area Network (CAN), developed by Robert Bosch. This was identified as the most important network system for the project. There are several other protocols being developed for use in the industry, however, especially for "drive by wire" applications as well as earlier protocols which might still be encountered. Some of these other protocols are described in the second section. The third section then considers similarities and differences between these protocols from the point of view of the project, so discusses issues of interest for modelling and simulating the networks.

The references consulted are listed in a bibliography. Much of the investigation for this report was done using the Web, so many of the sources are actually web-site urls rather than published papers. In the nature of the Web it is likely that some of these sources will become unavailable, but they have been checked during preparation of this report. Many of the web sources have been downloaded and saved or printed, so if a reader of this report cannot find a desired source, it might be worth getting in touch with the author.

## 2 Controller area Network

This section describes some of the features of the Controller Area Network of interest in modelling networks using the protocol and also briefly discusses these features from that point of view. It is therefore not a full description, the sources referred to should be consulted for further information.

Controller Area Network (CAN) is a network protocol developed by Robert Bosch GmbH for vehicle systems, but which is coming into use for linking distributed controllers, sensors etc in other fields. Bosch have published a specification [19]. The protocol has been adopted as a standard by the ISO, reference ISO11898. Any reference herein to "the spec" means the Bosch specification, not the ISO one, which I have not seen (it costs!).

CAN is a CSMA/CD protocol (some sources have CSMA/CR for similar protocols) that uses non-return to zero coding with bit stuffing. It supports speeds of up to 1Mb/s so is an SAE class C protocol, suitable for real time control applications. There is a brief explanation of the classes of protocols in section 3.1.

Messages are not addressed to intended recipients, but the sender's identifier is included, and this tells the receivers what data it contains so the receiver ignores it if it is not interested. Messages' identifiers give the priority of the message, so the priority of messages is decided at the design stage.

In [19] there are two standards for CAN 2.0, imaginatively called A and B. These differ in message format (see section 2.2), B has an extended message format, with a 29 bit identifier, as opposed to A's 11 bit one.

In basic can (not to be confused with CAN "A") each controller on the network is interrupted by every message on the bus. In full CAN, the CAN devices add filtration of the messages, and will only pass messages with specified identifiers on to its associated controller, so a controller is only interrupted by those messages the CAN terminal passes, that is those of interest to that controller.

The notes on CANbus in *Automobile Electrical and Electronic Systems* [5] draw attention to the difference between having a local intelligent control module (for example, for all functions located in the driver's door) and having the intelligence actually in the actuators, so control is distributed, and each actuator (and each sensor) is on the bus itself.

### 2.1 Network access, collision detection and resolution

Binary zero is represented by a "dominant" state in the bus and binary one by a recessive state, so a binary zero takes precedence over a one, so lower numbered message identifiers have priority over higher numbered ones.

CAN is a CSMA/CD protocol. If the network is idle, any node can send a message. If two messages are sent simultaneously, the node that sends a recessive bit, but detects a dominant bit stops transmitting, leaving the network free for the higher priority message. The higher priority message is not corrupted (so-called non-destructive bitwise arbitration). As this strategy resolves collisions and does not merely detect them (as is the case with other CSMA/CD protocols, such as Ethernet), some sources describe protocols with a similar collision strategy as "CSMA/CR", Carrier Sense Multiple Access/Collision Resolution. The identifier and RTR fields (see section 2.2) are used for collision arbitration. Therefore arbitration breaks down if two nodes can send data (as opposed to remote request) messages with the same identifier, as the clash will not be identified until later in the message, giving rise to a bit error. Each node must send data messages with a unique identifier. This has the side effect that if, say, all four road wheels had rotation sensors (for ABS and traction control) they would each need their own identifier, so they would have an order of priority. It seems to me not unreasonable to suggest that this could lead to conflicts in designing the system, which I do not propose to discuss here as it is outside the scope of the project.

This is supposed to guarantee message latency, but surely can only do so for messages of high priority? Clearly this guarantees the highest priority message access to the network once the current message transmission is complete. The second highest priority message is guaranteed access after that, provided the top priority message source doesn't broadcast continuously, so this is pretty much guaranteed. Surely, however, as one moves down the order of priority, eventually one is going to reach a point where a high priority source might be ready to transmit again while a low priority source is still waiting, so its latency is not guaranteed. Is this one reason why the SAE J2284-500 standard [23] is limited to 16 nodes? How well this limits message latency will inevitably depend on loading of the bus — how heavily used does it get, typically? It seems reasonable to suppose that rotation sensors (for wheel rotation, engine revs etc) cannot send data continually, as it takes time to find the speed, so in practice this might limit bus load sufficiently for latency to be OK. I have seen a reference to a paper on *Guaranteeing Message Latencies on CAN*, [26]. I have not yet found a copy.

### 2.2 Message format

The format of the message frames is to be found in detail in the Bosch spec [19] or in less detail in the Omegas web-site [16] and the Kvaser web-site [10]. The standard CAN (2.0A) frame has an 11 bit identifier, while an extended CAN (2.0B) frame has a 29 bit identifier, for compatibility with other protocols used in the US vehicle industry.

The standard identifier format allows for 2032 nodes on the network and the extended identifier allows more, but as the extra 18 identifier bits are needed for compatibility with other protocols, their use is restricted. The SAE J2284-500 standard [23] allows for any number of nodes between 2 and 16, inclusive, which doesn't seem many.

A 2.0A compliant device will flag an error if presented with a 2.0B message, unless it is "2.0B passive", when it will tolerate, but ignore, messages in 2.0B format. 2.0B devices are backward compatible, and can transmit and receive messages in either format. The "RTR" (transmission request) field which is set to 1 if the message is a request for information follows the identifier. As such a "remote request" frame uses the identifier of the source of the required data, this means that data takes precedence over a request for that data, but a request for high priority data takes precedence over lower priority data. These remote request frames are apparently rarely used. The identifier field and RTR field are used in collision resolution.

The data field can contain from zero to eight bytes, its length being stated by a 4 bit DLC field that immediately precedes the data field.

The data field is followed by a 15 bit cyclic redundancy check (CRC), a delimiter, acknowledgement field and end of frame and intermission fields. After these and a set idle time (which may be zero) another node can start transmission.

## 2.3 Error detection

There are 5 error detection mechanisms: -

1. Cyclic redundancy check. Each message contains a 15 bit CRC code computed by sender and checked by receivers, who will flag any errors. More in the spec [19].

2. Frame check. At certain points in the frame the current value is predefined.

3. Acknowledgement Error Check. If the transmitter determines an error has not been acknowledged, and ACK error is flagged.

4. Bit monitoring. A transmitter checks the network and flags a "bit error" if the value on the bus is not that sent. This does not happen during transmission of the identifier field, of course, as that is used for collision detection.

5. Bit stuffing. After five consecutive bits of the same value, a bit of the opposite value is added to the frame. This avoids errors arising from poor synchronisation of the network nodes, necessary because non return to zero encoding means there is no change in network voltage during a succession of bits of similar value.

If an error is detected, an error frame is sent, aborting the transmission.

Error confinement (which may be unique to CAN?) provides a mechanism for distinguishing between temporary and permanent errors. Each node has two error counters (for transmit and receive) which are incremented when errors are found. It is covered in more detail in the spec [19], but briefly each receive error increments its counter by one, and each transmit error increments its counter by 8. If either counter goes above 127 the node concerned goes into "error passive" mode. In this mode it can still transmit and receive messages, but is restricted in flagging errors. If a device's transmit error counter goes above 255, the device will go into "bus off" mode and will cease to be active. This condition might need to be modelled in simulating CANbus systems for FMEA. This seems to imply that we must allow for the modelling of repeat errors or for modelling the network as though the counter(s) had reached a level such that devices were going into "bus off" mode. A simpler approach, of course, would be simply to have "bus off" as a failure mode of the network terminal. This will allow the FMEA to test any mitigation strategy for this failure of the network.

Error detection is thorough. Omegas' material [16] suggests that the undetected error probability is $10^{-11}$. Of course, detected errors will result in loss or delay to messages, which effects might need modelling.

## 2.4 Bit timing and synchronisation

This is covered in the specification [19], of course, and there is an introduction to this in the Omegas material [16]. Briefly, a bit time consists of four non-overlapping segments, Sync-seg, Prop-seg, Phase-seg1 and Phase-seg2. An edge should lie within Sync-seg, while Prop-seg is used to compensate for delay times in the network. It is therefore the sum of twice the signal propagation time on the bus, the input comparator delay and the output driver delay, so is characteristic to the network. Phase-seg1 and Phase-seg2 are used to compensate for edge phase errors. They can be lengthened or shortened by resynchronisation. The sampling point is the boundary between Phase-seg1 and Phase-seg2. As non-return to zero encoding is used, there need not be an edge during Sync-seg, but bit stuffing ensures that there will be an edge after five edge-free Sync-segs.

There is a paper on *The Configuration of the CAN bit timing* [8] which describes the bit synchronisation algorithm and parameters to be considered in calculating the CAN bit time.

## 2.5 CAN in the ISO/OSI stack and higher level protocols

The scope of the CANbus protocol covers the physical and data link layers of the ISO/OSI model. The specification [19] refers to three levels in the CANbus protocol — physical layer, transfer layer and object layer. The physical layer is not defined in the Bosch spec, but is typically shielded or unshielded twisted pair. Idle state is both lines at +2.5 volts. A dominant bit reduces one line, known as CAN_L, to zero, while increasing the other line (CAN_H) to +5 volts while a recessive bit is close to the idle value, with CAN_L slightly above CAN_H, so is "over written" by a dominant bit. A standard for the physical layer of a 500 KBPS vehicle network is defined in SAE J2284-500 [23].

The transfer and object layers between them comprise all the services and functions of the ISO/OSI data link layer. These are discussed in the specification [19].

Various higher level protocols might be added to CAN itself. Kvaser [10] has some material on this, and Omegas [16] have some links on their web-site. Of these the Kvaser source is perhaps the more useful. I have also seen an article on higher level protocols [6] that gives an overview of the most important higher layer protocols, especially those used in industrial automation.

The Can in Automation (CiA) trade organisation [4] supports various higher level protocols: -

- CANopen
- DeviceNet
- CAL (CAN application layer)
- CAN Kingdom
- SDS (Smart Distributed System)

CiA is an organisation mainly interested in using CAN for industrial automation so it may well be that the protocols listed above are more common in that field than in the automotive sector.

In addition to these, Kvaser list J1939 and OSEK. J1939 is an SAE standard for a Truck and Bus Control and Communications Network that uses the CAN protocol and includes documentation (though not explicit definitions) for each layer in the ISO/OSI stack. There is a brief introduction to J1939 in section 3.5. OSEK is establishing standards for interfaces between hardware, network and software in the automotive field. There is an introduction in appendix A. In addition there is FNOS (Ford Network Operating System) that appears to be a Ford alternative to OSEK. There is an introduction in appendix B. Both OSEK and FNOS provide higher level protocols for use with CAN, though their scope is broader than that. There appears to be a good deal of overlap between OSEK and FNOS.

## 3 Other network protocols

This section will briefly introduce other protocols used in the automotive industry. Some of these are old ones, which appear to be falling out of use, and others are new ones being developed, many for so-called “drive by wire” applications. The protocols will be described in rather less detail than was CAN above. There is also a brief section on the SAE standards. These descriptions will be followed by a section that will discuss differences and similarities between these protocols, from a modelling point of view.

The protocols introduced here can be divided into two general classes, CSMA ones, such as CAN, and TDMA ones such as TTP/C. Some of the sources referred to for the TDMA protocols use the distinction ”asynchronous” and synchronous. This means that in the case of the TDMA protocols, all the nodes use a common notion of time, rather than being prompted to transmit by an external event (the terms “event driven” and “time driven” are also used). In fact CAN also uses synchronous transmission.

### 3.1 SAE standards

The Society of Automotive Engineers has defined three categories for in-vehicle networks:-

- Class A, low speed (less than 10Kb/s) for convenience features such as entertainment.
- Class B, medium speed (between 10 and 125Kb/s for general information transfer, such as emission data, instrumentation.
- Class C, high speed (greater than 125Kb/s) for real-time control such as traction control, brake by wire, etc.

CAN is class C, SAE J1850 (Ford SCP etc, see sections 3.2) is class B. It is, of course, not inconceivable that both protocols are used for different functions in the same vehicle. SAE standards exist for these categories. The document *Digital Networks in the Automotive Vehicle* [12] also lists a Class D for speeds greater than 1Mb/s. Although no (SAE) standards exist, such systems are apparently being referred to as “class D”.

The SAE has various standards for vehicle networks in these classes (i.e. of different speeds). They have adopted J1850 as the standard for class A and B networks. There is more on this standard below, section 3.2. CAN has been selected as the basis for J1939 -

a class C network for truck and bus applications, described briefly in section 3.5. There is also an SAE standard for high speed CAN (500 Kb/s), J2284-500 [23]. I have also seen a mention of a J2411 single wire CAN. These standards (and others) can be bought from the SAE web-site [21]. Some ISO standards are also available from this source.

## 3.2 SAE J1850

This is the SAE standard for Class A and Class B (slow and medium speed) networks. It is a combination of Ford's SCP (see below) and General Motors' Class 2 Protocol. These protocols differ, for example in operating at different speeds. The need for a standard was apparently driven by a need to interface with diagnostic equipment (for emission control testing). Fault codes have to be available through a diagnostic port through a standard protocol — J1850 or ISO 9141. OBD-11 requires the implementation of diagnostic tools for emission related systems.

As J1850 developed from two proprietary protocols, there are two alternative J1850 protocols, 41.6Kb/s with pulse width modulation and 10.6Kb/s with variable pulse width. I have found a paper on the latter, *Implementing the J1850 Protocol* [15]. Ford SCP seems to be the former.

J1850 (in both versions) is a CSMA/CR protocol, in which collisions are handled by arbitration, in much the same way as CAN, so the higher priority message is not corrupted by the collision. There is a bit on this in Intel's *Introduction to In-Vehicle Networking* [9]. In both versions the data field can be from 8 to 64 bits and both versions use a cyclic redundancy check.

SCP (which apparently stands for Standard Corporate Protocol) is Ford's version of SAE J1850. The Jaguar example circuits referred to in [1] appear to use this protocol, but it is apparently being superseded by CAN.

It is the faster version of J1850, so runs at 41.6Kb/s and uses pulse width modulation. It uses a two wire bus, unlike the General Motors J1850 protocol, which uses a single wire.

## 3.3 UBP

This protocol is mentioned in *Generic Network FMEA* [25]. It appears to be a proprietary UART based protocol intended for SAE class A applications. There are various such proprietary protocols, which are likely to be replaced by standard ones (such as SAE J1850 or CAN) in future. Smart Engineering Tools [22] build network simulation and analysis tools. They list several protocols under "UART" including UBP. A UART (Universal Asynchronous Receiver / Transmitter) is an integrated circuit used for serial communications.

One difference between UBP and other protocols is that it uses a checksum rather than a cyclic redundancy check (CRC), so undetected errors are more likely than in SCP or CAN. The risk is quoted as "low" in [25], rather than "extremely low".

We have been told that Ford's UBP is not used in the UK, as it interferes with the Radio 4 cricket commentary!

## 3.4 ISO 9141

This is an alternative standard to J1850 (see section 3.2) for protocols that must interface with a diagnostic port. While Ford's "domestic" products use J1850 (SCP), their international ones use ISO 9141. There is a protocol referred to as "Ford 9141" which appears to be distinct from Ford's UBP and based on ISO 9141. It is referred to in [30].

The NSI web-site [14] has an introduction (in French) to ISO-9141. These notes are based on Google's automatic translation. According to this site, it specifies "the characteristics of numerical exchange of information between the electronic control units embarked aboard road vehicles and suitable equipment of diagnosis." There are alternative configurations of the physical layer, one or two wire. It also specifies speed (5 baud for addresses and between 10 baud and 10 k baud for other transmissions), time intervals between key words and data transmission, message format and so on. Whether communication is point to point or multipoint is specified by individual manufacturers, so network access appears not to be specified in this standard.

## 3.5 J1939

J1939 is a high speed (Class C) network to support real time closed loop control functions between ECUs within a vehicle. Its documentation covers all layers in the ISO/OSI stack, so its scope is broader than, say CAN. J1931 does not necessarily formally define all layers. It uses CAN so network access and message format are consistent with it. The CAN 2.0B format is used with 29 bit message identifiers. The format of these 29 bits is defined in the standard and explained in the Kvaser tutorial, available from the Kvaser web-site [10]. The speed of J1939 is 250 Kb/s, so it is slower than J2284. The physical medium is intended to be shielded twisted pair. The standard can be purchased from the SAE [21].

## 3.6 TTP/C

TTP stands for Time Triggered Protocol. It is a deterministic protocol intended for SAE class C applications. It was developed by the Brite Euram Project "X-by-Wire" and ESPRIT OMI Project "TTA" at the Technical University of Vienna [24]. The specification has been transferred to TTTech [29] since the ending of these projects. There is a TTP group with a web-site [27] from which the specification can be ordered, for free. The companies listed on the web-site include VW/Audi and Honeywell. TTP/C can apparently manage higher data rates than CAN. The time triggered architecture is discussed in *Bus architectures for safety-critical embedded systems* [20].

TTP is "time triggered" as opposed to "event triggered" so all nodes on the network have a common concept of time, through roughly synchronised clocks. All activities are carried out at certain points in time, decided at system design time, rather than network activities being triggered by external events, as in a CSMA protocol such as CAN. As TTP is a TDMA protocol, latency is deterministic. There is a bus guardian that "guarantees" that no node can monopolise communication media outside its transmission slot, so the network should be safe from "babbling idiots".

The network appears to be peer-to-peer, as each node has its own controller and bus guardian. Therefore failure of a bus guardian will presumably only result in failure of that node, but does that not allow the node to become a babbling idiot?

There is a lower cost version, called TTP/A, for SAE class A applications. This version is also TDMA and is a master/slave architecture. It can be used for branching several sensors from a single TTP/C node. As TTP/A is intended for low cost systems, a standard UART and an 8-bit controller are sufficient for implementation.

## 3.7 LIN

LIN is an acronym for Local Interconnect Network and is a low cost field bus network intended to fit below CAN's functionality (i.e. for SAE class A applications?). I have found a paper comparing LIN with TTP/A, from the TTP forum [27], but on checking this paper appeared no longer to be available from there. The standard is described at [13]. The LIN consortium includes VW/Audi, Daimler-Chrysler and Motorola. Unlike TTP its development was driven by industry, rather than by academic institutions.

It is a single master/multiple slave architecture, so no need for arbitration. Speed is 20Kbit/s so while it is considered to be most appropriate for SAE class A applications, the speed is actually at the lower end of class B. As it is time triggered, message latency is guaranteed. Silicon implementation is cheap, based on common UART/SCI interface hardware. SCI stands for serial communications interface.

## 3.8 Volcano

Volcano might be described as "TTP on CAN" and the Volcano web-site [31] describes the protocol as CAN-based and deterministic. The protocol is used by Volvo on the S80 and V70 cars, and is coming into use on Volvo buses.

According to the *Volcano Communications Concept* [18], Volcano appears to be a technique in which the CAN network is integrated in such a way as to guarantee the latency of all the messages. It does this by specifying the latency and periodicity of messages at design time. This allows the maximum latencies to be calculated, so the system integrator (designer) can specify the network set up in such a way as to juggle these specifications to guarantee the specified parameters, by avoiding arbitration as far as possible. This seems to imply that the sending of network communication is time triggered rather than event triggered, so the description "TTP on CAN" seems a pretty good summing up.

This apparently means that network loadings can be considerably higher than using CAN conventionally, maybe 60% loading, whereas for latency of lower priority messages to be contained to reasonable limits, CAN loading may need to be around 10%.

## 3.9 Byteflight

Byteflight is a high speed, deterministic protocol developed by BMW and several semiconductor manufacturers for safety-critical automotive applications. There is a web-site on the protocol, from which the specification can be downloaded [3].

It is capable of speeds of up to 10Mbps gross, (better than 5 Mbps net) using an optical fibre physical layer to avoid electro-mechanical interference problems, in a star configuration. It has also been tested using a bus configuration, on twisted pair, but at lower speeds. The protocol combines time and priority controlled bus access, but claims collision free operation, so no arbitration loss. Latency is guaranteed for "a certain amount of high priority messages" and there is an analytical check for worst-case latency for high priority messages. There is flexible bus access for low priority messages, but latency cannot be guaranteed for these.

According to the description in [2], one node (it can be any) sends a periodic signal that marks the beginning of a "slot". In the current standard each slot has a duration of 250 microseconds. After transmission of the slot signal, each node starts a counter and can send a message when the counter reaches its own number. When a transmission is made, the counters pause for the duration of the message, so no slots are missed. If there is no message, there is a brief pause before the counters increment. Therefore there is a number of messages that can be certain of reaching their time in each slot, so can be sure of transmitting every 250 microseconds. Lower priority messages cannot be certain of transmitting in a given slot (or in any slot, in principle).

## 3.10 FlexRay

FlexRay is a protocol that combines time triggered and event triggered messaging. It is being developed by BMW and DaimlerChrysler with Philips and Motorola. It is capable of a net data rate of 5Mbps (10 Mbps gross). Information on the protocol is available on the Web [7]. The Requirements Specification can be downloaded from here. It is one of four protocols discussed in *Bus architectures for safety-critical embedded systems* [20].

Not surprisingly, in view of its developers, FlexRay has a certain amount in common with Byteflight. There is a signal indicating the beginning of a network slot, like Byteflight's, but this slot is divided (at design time) into static (time triggered) and dynamic (event triggered) portions. In the static part, each message source has its own slot, during which the network is idle if that source does not transmit. This is followed by the dynamic portion of the slot in which any node can transmit, using the Byteflight protocol, so it is still free of arbitration and transmission is priority based. The example in the FlexRay introductory presentation [22] shows the highest priorities having slots allocated in the static (time triggered) part, and lower priority sources in the dynamic (event triggered) part, so a source (id) has a slot in one or the other. This presumably reduces jitter for messages allocated slots in the static portion (as compared to Byteflight) as their timing is constant, unlike in Byteflight, where vacant slots are shortened.

## 3.11 TTCAN

This protocol is a session layer (from the ISO/OSI stack) extension to CANbus, currently being standardised by the ISO, which allows CAN to be used for time triggered messages, so increasing determinism, reliability, composability and synchronisation over CAN. The summary here is taken from Leen and Hefferman [11].

In TTCAN a specific node (the "time master") transmits a reference message, indicating the start of a time cycle. The message is recognised by other nodes (by its identifier). A time cycle is divided into a number of slots each of which can be assigned statically to a specific node, or to a group of nodes that compete for it by CAN arbitration, though without retransmission. Slots can also be designated as idle time to allow for expansion. A transmission must be started early in the time slot (during the so called Tx_Enable window) so as to avoid the message over-running its allocated slot. The only time retransmission is allowed is when two or more arbitration slots follow consecutively, so the message that lost arbitration can retransmit during the combined Tx_Enable window of the successive slots. A complete cycle can cover several successive transmissions of the reference message. This message includes a cycle_count value to indicate which row of the resulting "matrix cycle" has been reached. Each row can have its individual slots allocated differently from the other rows.

As there is now a master node that transmits the reference message, the protocol needs to ensure fault tolerant behaviour of the time master. If a time master fails, another potential time master takes over. Any one of eight nodes can be potential time master.

## 4 Comparison of the protocols

This section will discuss each common feature or difference of the various protocols in turn. Each section will contain some discussion of modelling issues that feature raises and a final summing up will conclude the discussion.

### 4.1 Broadcast messages

The most noticeable common feature of these protocols seems to be that messages are broadcast so all receivers receive the message and can act upon it if they are interested. This is necessary as some sensors will transmit data of interest to several systems, such as road wheel sensor data being of interest to ABS, traction control, instrumentation (speedometer) and possibly others. The only exception seems to be that some implementations of ISO 9141 use point to point communication. This does not seem an important exception from the point of view of the project (as these older standards appear to be being phased out) but if we ever wanted to model telecommunications systems we might need to model point to point communication.

Broadcast messages immediately introduce a problem in using SDL to model network message passing, as an SDL "signal" is sent to a specific recipient, though it may be broadcast in the sense of being sent by all available paths, in much the way that an Ethernet message is sent to every node on the network, but is addressed to a specific receiver. In CAN, for example, the transmitter will not know which nodes are interested in the message, it broadcasts it to all and sundry, similarly to a radio broadcast, the message being identified by its source, not its recipient. An SDL signal can have its sender identified.

Of course, in the context of a subsystem, we might be able to dodge this issue, by simply modelling the message being sent from the subsystem's (only) sender to its (only) receiver, but this fails to model the protocol, and also fails to address situations where the system has several receivers of the same message, such as separate ECUs for each lamp cluster.

### 4.2 Undetected errors

According to *Comparison CAN vs. Byteflight vs. TTP/C* [28], CAN, Byteflight and TTP/C all use a CRC and all have a hamming distance of 6, so the probability of an undetected error is more or less equal in each case. I imagine that the same applies to Volcano, being based on CAN. The probability is not equal as it will be affected by the length of the message, there being a greater probability of six bits being corrupted in a long message than a short one.

This makes the probability of an undetected error vanishingly small, but of course if a fault in a sensor, say, causes it to send an incorrect reading, this will be transmitted accurately and so will lead to incorrect behaviour. This is, of course, not a fault with the network.

Some older protocols (such as UBP) use a checksum rather than a CRC, which makes an undetected error less unlikely. The probability is still low and, in view of these protocols' obsolescence the difference can probably be ignored.

Of course a detected error will lead to the message being retransmitted which will result in a delay in its correct reception, especially if it needs to contend for network access again (as in CAN). Late messages clearly need modelling.

CAN's error containment (see section 2.3) might result in a node being rendered inactive (so-called "bus off" mode). This might result in a fault mitigation strategy being invoked by the receiver. This feature should be modelled, but arguably need not be distinguished from messages being lost for any other reason (the message not transmitted or message not received faults listed in *Generic Network FMEA* [25].

## 4.3 CSMA and TDMA

The most obvious way of grouping the various protocols is by access control. CAN is CSMA/CD, so messages will be interrupted if they lose arbitration, while TTP/C and Volcano are TDMA, so there will be no collisions to resolve, if the network is working correctly.

Modelling interruptions in CAN looks to be potentially the most difficult problem to solve in modelling these networks, as it non deterministic. It seems plausible to argue that modelling messages coming from different concurrently running sources is impossible on a single processor computer, as even running concurrent processes means that the models of some sources will stop while another process has the CPU, so Java threads are not an ideal solution. Another approach might be to model late messages statistically. This is possible, once the source priorities are known. If the frequency with which each source sends (or tries to send) a message is also known, then the network loading can also be established and the statistical likelihood of an interrupt can be arrived at. Volcano uses a similar calculation as a basis for setting the sources' priorities and periodicities such that collisions are avoided. Another, simple, approach would be simply to model individual messages as arriving late, allowing the user to determine which messages are modelled as delayed when setting up the FMEA scenario. This will result in long scenarios if it is to be comprehensive.

Another problem that arises from modelling collisions, whether by modelling different terminals' behaviours or statistically, is the possibility that the configuration of the network is not known at the time of running the simulation. It seems possible that this might arise if a simulation of a subsystem is to be undertaken early in the design lifecycle of the vehicle. There seems to be no useful way of avoiding this difficulty.

As the TDMA protocols are more deterministic, it seems reasonable to suppose that they will be easier to model. Therefore it should be possible to adapt a modelling approach that is capable of simulating a CSMA protocol for a TDMA one. The main difficulty seems to be the need to model time in some suitable way, but if we are to model message delays, we must model time anyway. There is a separate section on message delays, section 4.4.

On the other hand, as a TDMA protocol needs some global concept of time, they introduce a new class of failures concerning loss of this idea, for example a TTCAN "time master" failing to send a time reference message. All such protocols will have some fault tolerance in this area, but it might well be that these behaviours need modelling.

## 4.4 Message delays

The need to model late messages is common to all protocols though a TDMA protocol network that is working correctly should not have any. Clearly if the specification for frequencies of and acceptable waiting times for messages is unattainable, then message delays will be apparent, but I imagine that this will be dealt with by the network specification people, not the electrical engineers, and is capable of being modelled using specialist tools.

The possibility of a fault in a network component leading to unacceptable message delays does seem to be something we might be expected to model, however.

Lateness can only be measured either in very vague terms such as "The fault leads to the message arriving late, and the resulting delay in reacting to a wheel locking might lead to an uncontrollable skid" or in terms of failing to meet the specified message frequency and that the data is older than it should be - for example, "the delay in message reception meant that the time between successive messages was 60 ms instead of 50, and the transmitted datum was produced 35 ms before reception, instead of 25." Clearly this requires a quantitative modelling of time, and sufficiently detailed knowledge of the network and its terminals to know how much a message might be delayed by, say, being overridden on first transmission. It might be possible to use a qualitative model of time like that used in Statebuilder to approximate this. I imagine it might be possible to model a delay in terms of, for example, a message taking seconds instead of milliseconds, but this seems rather coarse grained. The idea of modelling delays also entails some linking of input to output in the functional labelling, so the connection between, say pressing the dip switch and the headlamps dipping is known, so the delay in achieving the required function can be established.

### 4.5 Tool support

All the interesting networks appear to have tool support available. CAN has CANOE, TTP/C can be modelled using Matlab Simulink and there appear to be tools available for Byteflight.

It is worth noting that CANOE does not model the network, it allows a computer to be added to a real network for monitoring and analysis. This limits its usefulness to late in the design lifecycle when a network has been built with real hardware, distinct from the idea of using simulation for design analysis early in the design lifecycle.

There might be some value in devising a protocol to enable a bridge to any of these tools to be created, alongside the strand of devising our own modelling method, applicable to any network and usable early in the design lifecycle. It seems possible that these approaches might be useful alongside one another, as a network becomes more closely specified, then the specialist tool might have advantages in terms of detail of simulation over a generic approach. There seem to me to be parallels between this idea and using SABER as the simulator in AQQA, later in the design lifecycle.

## 5 Conclusion

It is, I think, apparent that while CAN might be the network protocol worth most consideration in SoftFMEA, there are others of sufficient importance that whatever approach we take to modelling network components should be capable of easy adaptation for these other protocols. The various time triggered protocols being developed with drive by wire applications in view are good examples.

As a general point, it seems reasonable to suggest that making our modelling approach adaptable to different protocols will tend to militate against detailed modelling of any one protocol. The question of how much detail we should model a network and its protocol is one which needs further consideration. There seem to be problems in actually modelling collisions in CAN, there are difficulties modelling a truly concurrent system on a single processor computer and the idea of modelling a succession of network messages sits a little uneasily with the idea of running a simulation until it reaches a steady state, there arguably being no steady state in the network itself.

One plausible and simple approach, which has the benefit of adaptability to different protocols is to treat the network at large as an invisible component with its own failure modes. Clearly the failure modes of the network will differ according to the protocol. This component would sit alongside the explicitly drawn network connections between the various ECUs in the system being modelled. It seems entirely possible that such an approach can model the generic faults listed in *Generic Network FMEA* [25]. This does not remove the need for connecting input and output in functional labelling nor the need to devise a suitable modelling of time to allow delays to be captured.

This approach will not give any guide to the probability of message delay, and treating message delays and non-arrival as a failure mode of the network is different from the failure modes AutoSteve currently uses as not all messages will be delayed, the failure mode will be intermittent. The user could specify which messages are delayed, as part of the FMEA scenario, but this will make it difficult to create a comprehensive scenario. Maybe for purposes of FMEA simply treating all messages as being delayed is a possible approach, especially as an event in a scenario will frequently only entail one message being sent.

This simple method does have the advantage of being appropriate for the faults listed in *Generic Network FMEA* and being easily adaptable to other protocols, by merely specifying a slightly different set of failures. There is perhaps room for discussion on any need for a more sophisticated approach.

## A OSEK

The aim of the OSEK project is to develop a standard interface for the combination of hardware, network protocols and application software for use in automotive systems. The aim of the interfaces is, of course, abstraction away from the network and hardware, encouraging reusability of components.

OSEK and VDX were independent projects for developing open system architectures for automotive systems. OSEK was developed by a number of German companies, and VDX by French ones. The French manufacturers joined OSEK in 1994, resulting in an OSEK/VDX standard. *Digital Networks in the Automotive Vehicle* [12] has a section devoted to OSEK/VDX and the OSEK/VDX people have a web-site [17].

*OSEK COM* specifies standard protocols and interfaces approximating to various levels of the ISO/OSI stack. These include a Device Driver interface between the OSEK Data Link layer and the bus hardware, standard Network Layer protocols and a standard API in an Interaction Layer that approximates (together with the OSEK Network Management spec) to the roles of the Transport, Session and Presentation layers in the ISO/OSI stack. There is a specification (OSEK COM 2.2.2) available from [17]. It runs to 185 pages. Its introduction summarises the aim of the standard as being to "agree on interfaces and protocols for in-vehicle communication". This is intended to cover both communication between ECUs and within an ECU, allowing different vendors' products to be used.

The network management layer is covered by the specification NM 2.5.1, also available from [17]. In summary, the Network Management system "provides standardised features which ensure the functionality of inter-networking by standardised interfaces". Its role is to ensure the safety and reliability of a network between ECUs.

## B FNOS

According to [30] this appears to be an acronym for Ford Network Operating System. This provides an architecture for interaction between ECUs and CAN. FNOS contains seven layers some of which seem to fulfil similar roles to the higher layers in the ISO/OSI stack, though the layers do differ. There is clearly some relationship between FNOS and OSEK. They cover similar ground and OSEK standards are used in some layers of FNOS. I shall describe each layer in one or two sentences. There is a slightly fuller description in [30].

**Communication Layer** Provides a (mostly) hardware independent interface between higher layers and CAN, so it transmits and receives CAN messages, and handles CAN message overrun and error handling as well as CAN hardware initialisation and wake-up handling. The implementation will be based on OSEK COM 2.2.1 specification.

**Transport Layer** Provides a standard method of transmitting data which needs more than one CAN message. Not used in normal communications, but required for diagnostics, such as reporting of fault codes.

**Interaction layer** Separates CAN message level details from the message's specific parameters (contents?). This feature handles the packaging of data to be sent into a CAN message. The ECU simply writes and reads parameters.

**Gateway** Provides the capability to transfer signals from one CAN network to another.

**Network Management** Handles network initialisation, sleep and wake-up. It also has functions to handle network configurations. Typically only needed by networks that require sleep/wake-up functionality. FNOS implementation of Network Management layer will be based on OSEK NM 2.5.1.

**Diagnostics** (part 1) Provides a consistent method for evaluating and in some cases responding to diagnostic requests.

**Bootloader** Provides a consistent method for interfacing with a device to begin programming or reprogramming. This was being worked on by Ford and Volvo at the time of release of [30].

## References

[1] Jon Bell. Systems with telematic components. SoftFMEA document ref. SD/TR/02, 2002.

[2] J Berwanger, M Peller, and R Griessbach. *Byteflight — a new high-performance data bus system for safety-related applications.* BMW, 2000. Available from http://www.byteflight.com.

[3] The byteflight website is at http://www.byteflight.com.

[4] CAN in Automation has a website at http://www.can-cia.de/.

[5] Tom Denton. *Automobile electrical and electronic systems.* Butterworth-Heinemann, 2000.

[6] K Etschberger. CAN-based higher layer protocols and profiles. from Web at http://www.stzp.de/papers/icc97/icc97_e.html.

[7] There is a FlexRay website at http://www.flexray-group.com/.

[8] Florian Hartwich and Armin Bassemir. The configuration of the CAN bit timing. In *6th International CAN Conference*, 1999.

[9] Intel. *Introduction to in-vehicle networking.* Available from http://developer.intel.com/design/auto/Autolxbk.HTM.

[10] Kvaser's website is at http://www.kvaser.com/can/.

[11] G Leen and D Hefferman. Time-triggered controller area network. *Computing and Control Engineering Journal*, 12(6):245–256, 2001.

[12] G Leen, D Hefferman, and A Dunne. Digital networks in the automotive vehicle. *Computing and Control Engineering Journal*, 10(6):257–266, 1999.

[13] There is a website on LIN at http://www.lin-subbus.org/.

[14] NSI website (in French) is at http://www.nsi.fr/iso9141.html.

[15] D John Oliver. *Implementing the J1850 protocol.* Obtainable from http://developer.intel.com/design/intarch/papers/j1850_wp.htm.

[16] Omegas' website was at http://www.omegas.co.uk/CAN/, but it failed on checking these references. There are paper copies of some of their materials in the project document store.

[17] The OSEK website is at http://www.osek-vdx.org/.

[18] A Rajnak, K Tindell, and L Casparsson. *Volcano Communications Concept*. Volcano Communications Technologies, 1998. Available from the Volcano website, http://www.vct.se/.

[19] Robert Bosch GmbH. *CAN specification version 2.0*, 1991. Bosch CAN website is at http://www.can.bosch.com/.

[20] J Rushby. Bus architectures for safety-critical embedded systems. In *EMSOFT 2001: First workshop on embedded systems*, volume 2211 of *Lecture Notes in Computer Science*, pages 306–323. Springer-Verlag, 2001.

[21] The SAE url for standards is at http://www.sae.org/technicalcommittees/.

[22] Smart Engineering Tools inc. is at http://www.smtools.com/index.html.

[23] Society of Automotive Engineers. *High speed CAN for vehicle applications at 500 Kbps*. Document number SAE J2284/500
SAE website at http://www.sae.org.

[24] The Technical University of Vienna's TTA project page is at http://www.vmars.tuwien.ac.at/projects/tta/ in English.

[25] Timothy J Thomas. *Generic Network FMEA*. Ford Motor Company, 2001.

[26] K Tindell and A Burns. Guaranteeing message latencies in CAN. In *Proceedings 1st International CAN Conference*, 1994.

[27] The TTP Group website is at http://www.ttpgroup.org/index.html.

[28] TTTech Computerthchnik AG. *Comparison CAN vs Byteflight vs TTP/C.*

[29] TTTech's website is at http://www.tttech.com/.

[30] Vector CANtech. *Generic statement of work core multiplex technology.* Available from http://www.vector-cantech.com/OEM/ford/fordmux.htm.

[31] The Volcano website is at http://www.vct.se/.