Meeting the Real-time Constraints with Standard Ethernet in an In-Vehicle Network

Youngwoo Lee[1] and KyoungSoo Park[1]

***Abstract*— Vehicular networks have traditionally focused on the real-time delivery of critical control messages for safe car operation. Unfortunately, the real-time requirements often cripple the development of flexible car applications by tying the application network stack to underlying physical networks. While popular real-time vehicular networks guarantee the timely delivery of prioritized messages, they often lack in bandwidth and flexibility, which limits the range of car network applications.**

In this work, we explore the idea of replacing the current vehicular network with standard switched Ethernet, the most popular LAN technology in computer networks. Ethernet is attractive in providing high bandwidth at a low cost with easy and flexible configuration. The most challenging part is to guarantee the real-time delivery of mission-critical messages. We first show that the *soft* message delivery latency of 10s to 100s milliseconds can be easily met in 100 Mbps switched Ethernet despite co-existence of high-bandwidth network applications. For meeting the *hard* delivery latency on the order of 100 microseconds for critical control messages, we propose limiting the path MTU to the destination node with priority queuing from IEEE 802.1Q. Our simulation shows that we can satisfy 100 microseconds of latency even in a rich set of vehicular applications without any modification of the application network stack.

I. Introduction

Modern vehicles typically have several tens of Electronic Control Units (ECUs) that are connected by in-vehicle networks. These ECUs execute various automotive functionalities such as controlling engines, brakes, and transmissions, as well as checking the status of the doors and seats. Recent trend shows that a modern vehicle has about several tens of million lines of source code for running the ECUs [1], and the amount of code is likely to increase as the user demand for convenience and safety grows. Such user demands would include feeding various sensing data to help driver's memory and vision, and to increase the user reaction speed, and car infortainment functionalities that can also interact with the world outside the vehicle.

Most popular in-vehicle networks, however, are mainly designed to support real-time communication, and they are not well-suited for high-bandwidth applications that transmit various sensing data. Moreover, these networks are too rigid in terms of adopting a new set of automotive applications or for changing the configuration of existing ECUs. It is very challenging to support new software installation and upgrade on existing in-vehicle networks [2].

To remedy the problem, many researchers have recently considered IEEE 802.3 Ethernet as the replacement of current in-vehicle networks. Several automotive OEMs have already designed a Ethernet-based network as the standard medium for automotive applications [3]. Ethernet provides a number of benefits compared with existing in-vehicle networks. First, Ethernet supports high-bandwidth data communication. Ethernet considered in vehicle networks typically supports 100 Mbps, which shows a significantly higher bandwidth than 1 Mbps Controller Area Network (CAN) or 10 Mbps Flexray. Second, Ethernet is flexible and easy to configure. It allows adding or removing a node on Ethernet without any re-configuration. Third, it simplifies the development of the networking stack of vehicular applications. Car applications that need to communicate with ones on a different network or using a different protocol would need an application-layer gateway to translate the protocols. This not only increases the development complexity but it also makes it difficult to debug and test new applications. Finally, many existing Internet applications already run well on Ethernet. If Ethernet is adopted as the in-vehicle network, many existing applications that use TCP/IP on Ethernet can be easily ported to car environments. This would make the user experience as close as possible to that of the Internet computing environments.

One challenge in using Ethernet as the universal in-vehicle network lies in meeting the real-time requirements of various time-critical car applications. Since standard Ethernet does not provide any quality-of-service (QoS) guarantee on the minimum delay or bandwidth, one should be careful not to miss any hard deadlines of certain message delivery. For example, some critical control message in a car needs to be delivered within a very small delay such as 100 μs [4]. Since a 1518-byte Ethernet message would take 122.08 μs to be forwarded in a 100 Mbps Ethernet switch, if a control message contends for the same destination switch port with a large Ethernet frame, it would miss the deadline if the control message happens to be scheduled after the large frame. While several research works have studied using Ethernet as an in-vehicle network [5], [6], [7], they conclude that it is challenging to meet the most stringent real-time constraint of a certain car message. Even the Ethernet protocols modified for real-time applications such as IEEE 802.1 Audio/Video Bridging (AVB) standard [7] and TTEthernet [8] are not suitable for delivery guarantee within 100 μs.

In this paper, we tackle the problem of meeting the end-to-end delay requirement in standard Ethernet for car applications. We first show that standard switched Ethernet

*This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2010-0028680)

[1]Y. Lee and K. Park are with the Department of Electrical Engineering, KAIST, Daejeon, South Korea `ywlee@ndsl.kaist.edu`, `kyoungsoo@ee.kaist.ac.kr`

with the IEEE 802.1Q support can satisfy most of *soft* end-to-end delay requirements (e.g., within 10-100 ms of delay requirement) in a practical car network topology. For time-critical messages that need a 100 μs or lower delay bound, we suggest limiting the maximum transmission unit (MTU) of the paths that lead to the destination ECU for the critical message. We show the overall network bootstrapping process that satisfies all deadline requirements in car applications, and show the evaluation results in a simulation setting.

II. Background

A. Existing In-vehicle Networks

1) Controller Area Network: Controller Area Network (CAN) [9] uses a broadcast bus to send and receive real-time control messages at the speed of up to 1 Mbps. Each message ranges from 1 to 8 bytes and has its own 11-bit message ID that is uniquely assigned within the bus. The main benefit of CAN is that it fully supports the real-time communication. When a collision happens on the bus, the message with a higher priority dominates the bus immediately.

Despite the popularity in an in-vehicle network, CAN has a few serious limitations. First, CAN is limited in bandwidth, which disallows any applications that require high bandwidth. Second, CAN configuration is complex and inflexible. Each CAN message should be registered first with the sending and receiving ECUs. This process could be complex and error-prone since an automobile may have hundreds of distinct messages in the bus [10].

2) FlexRay: FlexRay has been developed to address CAN's limited bandwidth capacity. It supports high data rates up to 10 Mbps and supports both time-triggered and even-triggered communication. The maximum payload size of FlexRay is 254 bytes, much larger than that of CAN (8 bytes).

FlexRay uses the time division multiple access (TDMA) scheme for shared bus access. In time-triggered communication, only one ECU node can transmit the data at any time. That is, each ECU is allocated its own time slot for dedicated bus access. FlexRay also provides event-triggered communication. Real-time messages are sent in the time-triggered manner, and the remaining non-critical messages are sent based on an event. This effectively helps improve the bus utilization while meeting the real-time requirements.

Like in CAN, FlexRay also uses a message ID for data communication, which requires each message ID to be registered before sending a new message type. While event-triggered messages make FlexRay more flexible than CAN, the message ID registration process increases the application development complexity.

B. Ethernet

Ethernet is the most popular LAN technology in computer networking. Besides the benefit of easy configuration and flexibility, its data rate has constantly increased from 10 Mbps to even 100 Gbps. Ethernet uses the carrier sense multiple access with collision detect (CDMA/CD) method to access a shared bus. An Ethernet node that wants to transmit a message waits until the shared bus is ready for use. If a message from one node collides with a message from another node, both nodes will stop transmitting and wait for a random amount of time with the exponential backoff algorithm before retransmission. Due to this, a message transfer could be delayed for a long time in case of many consecutive collisions.

However, the collision problem in a shared bus disappears in switched Ethernet since an Ethernet switch allows parallel transfers of messages as long as their paths are different. If a switch cannot forward the packets to a destination node as fast as it receives, the internal queue could be filled up. In this case, the switch takes two steps. First, it drops the incoming packets that enter after the queue is full. Second, the switch could send a *pause frame* to the sender, which halts the transmission from the sender temporarily [11]. With a pause frame, the switch could avoid packet drops, but it could inject a long delay.

To use Ethernet in the in-vehicle network, there are two things that need to be considered. The first thing is to ensure real-time communication. As mentioned above, switched Ethernet may drop or induce a long delay to a critical control packet when some other nodes send contend for the same destination. In addition, standard Ethernet does not enforce any QoS mechanisms. This implies that an important packet could have to wait until other packets in the queue are drained. IEEE 802.1Q helps the situation by provisioning a QoS prioritization scheme [12]. It requires slight modification of the original Ethernet frame by adding an additional 32-bit field in the Ethernet header. The prioritization has 8 levels from 0 (the lowest priority) to 7 (the highest priority). A packet with a priority level is handled differently by a switch with its priority queuing algorithm. One could exploit this feature to handle safety-critical packets to meet their hard deadline requirements. The second thing is to curb the electromagnetic emission. Ethernet has a high symbol rate of 125 MBaud and such a rate creates a high level of electromagnetic emissions in the critical FM radio band [6]. For this reason, OPENsig [3] selects a different 100 Mbps Ethernet PHY since it is difficult to apply the standard Ethernet PHY in an automobile.

III. Switched Ethernet in a Real-time Environment

We first test whether standard Ethernet can be used to deliver real-time messages in a realistic in-vehicle network. We also gauge the benefit of IEEE 802.1Q to see if it helps meet the deadlines of critical messages. For realistic simulations, we extract the topology and traffic information from the BMW research group [5]. The in-vehicle traffic includes various data types such as control data, driver assistance camera streaming, video and audio streaming, and bulk traffic data. The control data includes virtually all time-critical CAN and FlexRay messages, which has the highest real-time priority. The camera data is used to deliver the outside vision, which requires a large bandwidth with a medium real-time constraint. The audio and video streaming data are

Node Name	Frame Type	Length [byte]	Service Rate [ms]	Throughput	Destination	Deadline[ms]
CTRL1 - CTRL4	UDP	20	uniform(10, 100)	1.6 Kbps 16 Kbps	Head Unit	10 (CTRL1 - 100μs)
CAM1 - CAM3	UDP	786	0.25	25.1 Mbps	Head Hnit	45
FCAM	UDP	786	0.25	25.1 Mbps	PUCAM	45
Audio	UDP	1472	8.4	1.4 Mbps	AVSink	150
Video	UDP	1472	1	11.8 Mbps	AVSink	150
Bulk Traffic	TCP	1400	uniform(1,10)	1.12 Mbps - 11.2 Mbps	Head Unit	None

TABLE I. Traffic characteristics in our simulation.

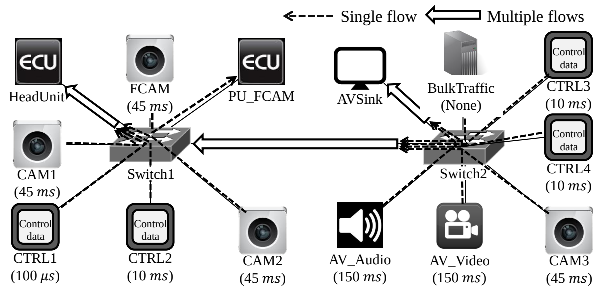

Fig. 1: Simulated car network topology with OPNET.

mainly for entertainment, and they have the lowest priority in delivery. The bulk traffic models several background TCP connections between source nodes and Head Unit. Head Unit is in charge of interpreting the delivered data. Figure 1 shows our simulation network topology and Table I presents the detailed traffic characteristics. Service rate of uniform(x, y) means that the next traffic period is chosen in uniform between x and y. We draw the deadline for each traffic type from previous literature [13].

We use OPNET [14] as a simulation tool, and use 3μs as the switch packet processing delay, which is reasonable for 100 Mbps Ethernet switches [15]. Each switch port is assumed to have 4 priority queues that can hold up to 100 packets, which reflects a real implementation of IEEE 802.1Q in currently-available switches. CTRL1 is in charge of the safety-critical traffic that has a 100 μs delay bound with the highest priority (3). We give the next highest priority (2) to other control traffic. Since the CAM data is used for a driver assistant application, we give a normal priority (1) to the CAM and FCAM traffic. All the other traffic has the lowest priority level (0). In our simulation, we use two priority queuing mechanisms. Packets with the highest priority queue are scheduled by strict priority queuing (SPQ) because they should be forwarded before any packets. Other priority queues are scheduled by weighted fair queuing (WFQ).

Table II shows our simulation results. Interestingly, switched Ethernet satisfies all soft end-to-end delay requirements even without the IEEE 802.1Q mechanism. However, this result is not too surprising since 100 Mbps switched Ethernet is sufficient to prevent any packet queuing, which leads to a small delay in packet forwarding. The link that shows the highest throughput (up to 86.6 Mbps) is between Switch1 and Head Unit that serves as the consumer of the most traffic. Other links mostly show low bandwidth consumption related with periodic control packets.

IEEE 802.1Q brings the benefit of limiting the maximum delay of the high priority traffic. The control traffic with prioritization has 30.4 to 83.1% smaller maximum end-to-end delays than the ones without it. Under prioritization, the CAM data packets get penalized since they compete for the same destination port with the higher-priority control traffic. However, since the bandwidth consumption by control packets is low, the increased latency of the CAM data does not affect its normal operation. The delays of FCAM and PU_FCAM (front camera streaming data and its consumer) do not get affected by prioritization since their traffic paths do not collide with other traffic.

The only problem with the table is that CTRL1 does not meet the deadline with or without prioritization. While prioritization greatly reduces the maximum delay to 130.02 μs from 214.89 μs, it is not enough to fall below the hard deadline of 100 μs. This is because standard Ethernet does not enforce preemptive scheduling. That is, even a single regular Ethernet frame is being forwarded when a CTRL1 packet just arrives, the CTRL1 packet could miss the deadline since it takes more than 100 μs to finish forwarding the previous packet. We consider the solution to this problem in the next section.

IV. Meeting the stringent real-time deadline with path MTU

In this section, we find a solution that satisfies the hard delay requirement of critical control messages with standard Ethernet. The basic idea is to limit the payload size of lower priority messages that compete for the same destination port with the safety-critical messages while still enforcing 802.1Q. We find that this combination promises a timely delivery of the messages even with the 100 μs delay bound. We note that this could reduce the effective bandwidth of lower priority packets, but since safety-critical messages typically require low bandwidth, it will not affect the bandwidth of other applications too much.

Before advancing the scheme, we assume that all nodes that exchange safety-critical messages are directly connected to the same Ethernet switch. While we could develop a method for a stacking switch scenario where a critical message traverses multiple switches, the analysis becomes much more complex with a small flexibility benefit. Given that the number of nodes that exchange safety-critical messages is typically small, we would rather choose to reduce the complexity than to allow an arbitrary topology of such nodes.

To satisfy the hard delay requirement, we limit the maximum transmission unit (MTU) of the packets that share the same destination. MTU is the maximum IP datagram size

Traffic	Without prioritization			With prioritization		
	Max.	Min.	Mean	Max.	Min.	Mean
CTRL1	214.89	14.84	64.80	130.02	14.84	45.07
CTRL2	242.42	14.84	66.55	132.80	14.84	45.47
CTRL3	339.39	23.76	91.37	256.87	23.76	76.31
CTRL4	329.18	23.76	118.71	252.78	23.76	76.02
CAM1	155.59	137.40	143.45	169.93	137.40	149.98
CAM2	216.75	137.40	207.42	231.08	137.40	213.77
CAM3	234.97	209.68	215.78	251.74	214.43	224.15
FCAM	137.40	137.40	137.40	137.40	137.40	137.40
Audio	248.70	247.16	247.16	250.14	247.16	247.16
Video	252.89	247.16	248.71	253.08	247.16	248.76
BulkTraffic	443.62	357.84	390.11	439.83	357.84	391.07

TABLE II. Simulation results (end-to-end delays, unit: μs).

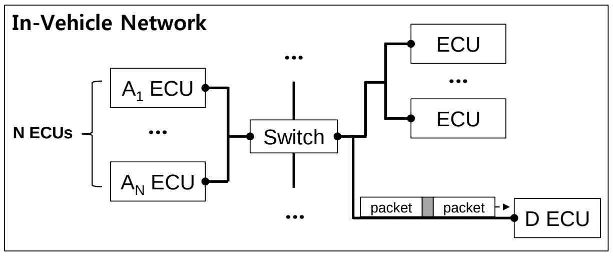

Fig. 2: Path MTU calculation.

that can be carried in a frame. The path MTU represents the smallest MTU of the path between a source and a destination. In computer networks, the path MTU can be obtained by a couple of MTU discovery packets and ICMP responses. However, we slightly modify the process to an end-to-end query and response, and the detail will be discussed in section VI.

Figure 2 shows one example. We assume that N ECUs, (A_1 to A_N), have safety-critical applications with the same destination ECU, D. Since safety-critical applications are typically control applications [4], these have control data whose size is small (e.g., less than 20B) according to BWM's control data analysis [5]. We start with the case with one ECU (N = 1) that sends a control packet to D with the hard delay requirement. The total end-to-end delay can be obtained by adding the packet transmission time (node to switch and switch to node) and the waiting time in the switch. The total delay from A_1 to D is

$$\{(8 * s/(10^8[bits/sec]))\} * 2 + f \tag{1}$$

seconds (if there is no packet in the switch queue) where *s* is the maximum control packet size and *f* is the processing time in the switch. Next, we calculate the maximum waiting time of the packet in the switch. In the worst case, the control packet has to wait for one full packet to be forwarded, and we can calculate the worst-case waiting time as

$$\frac{\{8 * (i + MTU + 28 + i)\}}{(10^8[bits/sec])} \tag{2}$$

seconds. Note that *i* is the 12-byte inter-frame gap (IFG), and 28 bytes are the Ethernet overhead (8 bytes of preamble, 14 bytes of the Ethernet header, 2 bytes of additional 802.1Q header, and 4 bytes for a CRC field). Ethernet dictates to have an IFG between Ethernet frames, and the expression above reflects the case when a control packet arrives when the previous packet waits for the IFG for its own transmission. For the worst case, we need to add the transmission time of the previous packet and the time for two IFGs.

If N ECUs simultaneously send their control packets with hard delay deadlines, expression (2) should be modified to wait for processing (N-1) previous packets with the same priority in the queue. We calculate the worst-case waiting time for a packet from A_1 as

$$\frac{\{8 * (2i + MTU + 28) + 8 * (s + i) * (N - 1)\}}{(10^8[bits/sec])} \tag{3}$$

seconds. With expressions (1) and (3), we get the maximum transmission time of the packet from A_1. It must be smaller than its own deadline, represented by D_1. Finally, we can get the MTU of the path.

$$MTU \le \frac{(10^8[bits/sec]) * (D_1 - f)}{8} - (s + i)(N + 1) - 28$$

This MTU guarantees that the packet from A_1 arrives within its own requirement delay regardless of other packets. We further discuss the effect of the path MTU in section V.

V. Evaluation and Discussion

In this section, we evaluate the effect of the path MTU by simulation and discuss our assumptions in the simulation.

A. The Effect of Path MTU

We simulate the in-vehicle network traffic again with the proposed path MTU. The CTRL1 traffic has the 100 μs delay requirement, and its destination is Head Unit. We set the MTU of the messages destined to Head Unit to 1012 bytes according to our algorithm, which affects only BulkTraffic packets in our case. The BulkTraffic packet payload size is set to 972 bytes excluding 40 bytes of the TCP/IP headers. Since the traffic throughput should be stable, we change the service rate of BulkTraffic to uniform (0.694, 6.94) [ms], instead of default service rate, uniform(1, 10) [ms].

Table III shows the end-to-end delays of all traffic. In comparison with Table II, we see that all traffic has smaller end-to-end delays with a smaller average delay. These results are understandable since the smaller MTU would reduce the message latency in the in-vehicle network. However, the effective link utilization is worse due to the increased frame overhead. This is because BulkTraffic now consumes more packet headers to send the same amount of data with the smaller MTU. Its maximum utilization of the link increases to 97.32% with the new MTU from 96.78% with the default MTU for the same amount of the traffic.

Figure 3 shows the impact of the controlling the path MTU. We find that all CTRL1 packets now satisfy the hard delay requirement, with the maximum end-to-end delay as 99.70 μs. This result validates that limiting the path MTU is effective in satisfying the real-time delay requirement.

Source node	Max.	Min.	Mean
CTRL1	99.70	14.84	43.54
CTRL2	100.41	14.84	43.77
CTRL3	189.43	23.76	73.21
CTRL4	185.27	23.76	73.46
CAM1	150.48	137.4	141.79
CAM2	215.75	137.4	206.87
CAM3	228.88	208.43	213.24
FCAM	137.40	137.40	137.40
Audio	250.15	247.16	247.17
Video	253.05	247.16	248.76
BulkTraffic	375.96	258.72	300.3

TABLE III. End-to-end delays in the limited MTU (unit: μs).

B. Discussion

In section IV, we assume that source and destination of automotive application traffic that has 100 μs delay requirement are connected by the same physical switch. While this assumption reduces the design flexibility of an in-vehicle network, we argue that our assumption is unavoidable to some degree for efficient communication.

We show our point by one example. Let's assume that the CTRL3 and CTRL1 traffic in Figure 1 have 100 μs and 10 ms of deadlines respectively instead of the values in Table I. Then, we need to set the MTU in the path from CTRL3 to Head Unit that crosses the switches. We calculate the path MTU of the links as in Section IV. In this topology, any message that comes from Switch2 to Head Unit should have the MTU as small as 424 bytes. It means that all traffic that travels between the two switches must limit their packet size. As we see in this example, even if there are only two switches between the source and the destination, the MTU could decrease rapidly, and it severely throttles the effective bandwidth of other applications. The applications that travel the links have only 28.3% of the default MTU and 41.9% of the MTU in a single-switch case. This result suggests that safety-critical applications that have hard end-to-end delay requirements are recommended to be clustered in one physical switch not to affect other applications.

C. Network topology for the soft requirement

We calculate the maximum number of Ethernet switch hops for the network topology of an application that has the soft end-to-end delay requirement in the range of 10 to 100 ms. As mentioned earlier, we assume that there is no MTU limitation at links between the switches, since the application with the hard delay requirement does not travel across the switches. Thus, the application with the soft requirement now has the highest priority in the paths that cross the switches. Using this fact, we calculate the maximum end-to-end delay of a control packet while it travels M consecutive switches. We assume that an application sends the control data that has the 10 ms deadline. We ignore the effects of other packets that have the same priority as the control data because the transmission delay of the control packet (e.g. 5.92 μs) is much smaller than 10 ms.

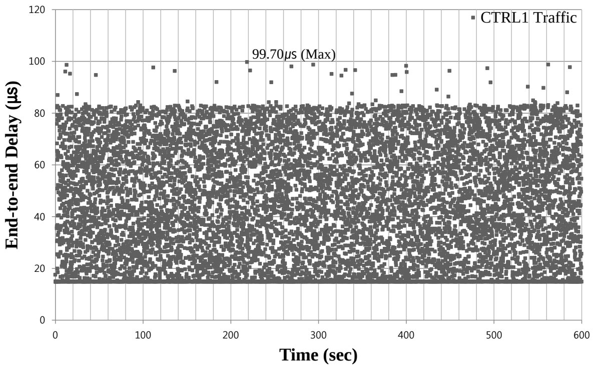

Fig. 3: End-to-end delays of the CTRL1 traffic.

$$(8{*}s/10^8[bits/sec]) * (M+1) + f * M < 10 * 10^{-3}[sec] + \{8 * (2i + S) * M/10^8[bits/sec]\}$$

If we solve the inequality above, we can find the maximum number of switches that can be connected for the control packet without violating the soft delay. Note that S is the maximum Ethernet frame size, 1518 bytes, plus 8 bytes of preamble. We find that an application that has the 10 ms soft delay can have as many as 76 switches between the sender and the receiver. Since an in-vehicle network has only several tens of ECUs, soft requirements could be easily met virtually in any topology. In other words, except for the hard end-to-end delay requirement, we rarely need to worry about other real-time requirements in any reasonable 100 Mbps switched Ethernet-based in-vehicle networks.

VI. Standard Ethernet in an in-vehicle network

In this section, we discuss the bootstrapping process and its related issues as Ethernet replaces conventional in-vehicle networks. During the bootstrapping, the IP addresses of ECUs are assigned automatically and the path MTU of each application is set to satisfy the real-time constraints of the system.

Figure 4 shows the bootstrapping process. We assume each in-vehicle network supports DHCP and DNS servers for the ECUs, and the applications use TCP/IP-based networking. Bootstrapping consists of following procedures.

1) Applications at each ECU register their delay requirements and DNS names.
2) Each ECU obtains its own IP address by DHCP in the in-vehicle network
3) Each application requests the IP address of its destination by DNS lookup from a DNS server for the in-vehicle network. The DNS server information is downloaded by DHCP.
4) Each ECU requests a destination MAC address by the IP address of the destination. (ARP)
5) Each application sender sends a path MTU query to its destination.
6) The destination ECU responds with the MTU, and the sender ECU maintains a table of (MTU, destination) pairs.

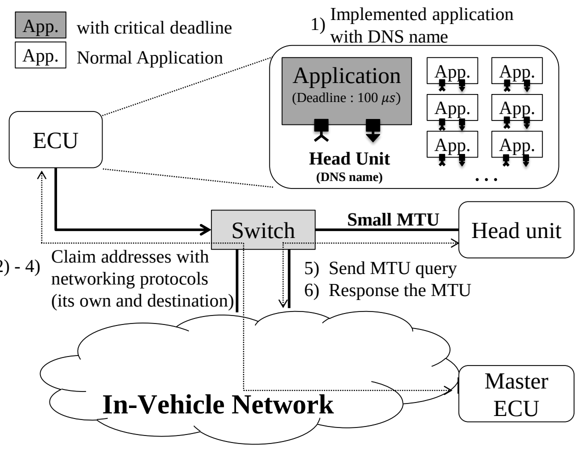
App. with critical deadline
App. Normal Application
1) Implemented application with DNS name
Application (Deadline : 100 μs)
Head Unit (DNS name)
App.
ECU
Switch
Small MTU
Head unit
2) - 4) Claim addresses with networking protocols (its own and destination)
5) Send MTU query
6) Response the MTU
In-Vehicle Network
Master ECU

Fig. 4: Bootstrapping process.

7) Bootstrapping is done.

What we suggest here is to migrate the in-vehicle network to a networking environment similar to a typical Ethernet-based LAN. We view each ECU as a separate node with its own domain name, and applications communicate with each other by IP addresses and port numbers. Like in a typical LAN environment, each ECU has IP and MAC addresses to communicate with other ECUs unlike conventional message ID-based communication in existing in-vehicle networks. We also encourage to write the networking code for car applications using domain names to ensure flexible management of the code. For this, we need a default DNS and DHCP server for each car network, and we believe we can run a lightweight version on a dedicated ECU which we call master ECU.

We note a few issues related to bootstrapping. First, an automobile does not need to repeat the entire bootstrapping process at every bootup. Most information can be cached across bootups and a full bootstrapping is needed only when the ECU/application configuration is updated. That is, the master ECU can check whether there is configuration update and skip IP address assignment and MTU setting if the configuration has not changed. Second, the bootstrapping process should finish in a short time. If a user starts her own automobile even after ECU or automotive application updates, she expects her car to move in a few 100 milliseconds. We are currently building a prototype car network using Ethernet, and plan to measure and optimize the delay of each step. Last, the bootstrapping process should be flexible enough to automatically allow a new configuration in the car network at the event of adding and removing ECUs or applications. This would help provide plug-and-play of various car components while guaranteeing real-time and safety requirements.

VII. Conclusion

In this paper, we have presented the possibility of replacing conventional in-vehicle networks with standard Ethernet. We have proposed a simple yet effective scheme that satisfies even the hard real-time delay constraint of a few 100 μs for time-critical car applications. The key idea is to limit the MTU of the messages that have the same destination port as the time-critical control message while benefiting from IEEE 802.1Q. Unlike previous approaches, our scheme does not require any modification of the Ethernet protocol itself, and we expect it would greatly accommodate easy and flexible car application development without typing the application logic to any specifics of underlying physical networks. We have also shown how we bootstrap the Ethernet-based car network that allows flexible communication based on the IP address. We believe the new car network would present a rich development environment that could produce many interesting new car applications that sometimes require high bandwidth without worrying about the real-time deadline requirements in a car.

References

[1] "This Car Runs on Code," 2009. [Online]. Available: http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code/
[2] R. Anthony, A. Rettberg, D. Chen, I. Jahnich, G. de Boer, and C. Ekelin, "Towards a Dynamically Reconfigurable Automotive Control System Architecture," in *Embedded System Design: Topics, Techniques and Trends*, ser. IFIP Advances in Information and Communication Technology, A. Rettberg, M. Zanella, R. Dmer, A. Gerstlauer, and F. Rammig, Eds. Springer Boston, 2007, vol. 231, pp. 71–84.
[3] *OPEN Alliance SIG*, http://opensig.org/about.php, OPEN Alliance Std.
[4] Y. Kim and M. Nakamura, "Automotive ethernet network requirements."
[5] H.-T. Lim, K. Weckemann, and D. Herrscher, "performance study of an in-car switched ethernet network without prioritization," in *Proceedings of the Third international conference on Communication technologies for vehicles*, ser. Nets4Cars/Nets4Trains'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 165–175. [Online]. Available: http://dl.acm.org/citation.cfm?id=1987310.1987328
[6] P. Hank, T. Suermann, and S. Mller, "Automotive ethernet, a holistic approach for a next generation in-vehicle networking standard," in *Advanced Microsystems for Automotive Applications 2012*, G. Meyer, Ed. Springer Berlin Heidelberg, 2012, pp. 79–89. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-29673-4_8
[7] M. Rahmani, R. Steffen, K. Tappayuthpijarn, E. Steinbach, and G. Giordano, "Performance analysis of different network topologies for in-vehicle audio and video communication," in *Telecommunication Networking Workshop on QoS in Multiservice IP Networks, 2008. IT-NEWS 2008. 4th International*, feb. 2008, pp. 179 –184.
[8] E. MATZOL, "Ethernet in automotive networks."
[9] O. I. de Normalización, *ISO 11898 : Road Vehicles : Interchange of Digital Information : Controller Area Network (CAN) for High-speed Communication*. ISO, 1993. [Online]. Available: http://books.google.co.kr/books?id=b506cgAACAAJ
[10] R. Davis, A. Burns, R. Bril, and J. Lukkien, "Controller area network (can) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, pp. 239–272, 2007. [Online]. Available: http://dx.doi.org/10.1007/s11241-007-9012-7
[11] L. Martini, E. Rosen, N. El-Aawar, and G. Heron, "Encapsulation methods for transport of ethernet over mpls networks," *RFC4448, April*, 2006.
[12] N. Ek, "Ieee 802.1 p, q-qos on the mac level," *Apr*, vol. 24, pp. 0003–0006, 1999.
[13] H.-T. Lim, D. Herrscher, and F. Chaari, "Performance comparison of ieee 802.1q and ieee 802.1 avb in an ethernet-based in-vehicle network," in *Computing Technology and Information Management (ICCM), 2012 8th International Conference on*, vol. 1, april 2012, pp. 1 –6.
[14] OPNET, http://www.opnet.com.
[15] M. Rahmani, K. Tappayuthpijarn, B. Krebs, E. Steinbach, and R. Bogenberger, "Traffic shaping for resource-efficient in-vehicle communication," *Industrial Informatics, IEEE Transactions on*, vol. 5, no. 4, pp. 414 –428, nov. 2009.