Robust Safety for Autonomous Vehicles through Reconfigurable Networking

*Khalid Halba *Charif Mahmoudi **Edward Griffor
National Institute of Standards and Technology Gaithersburg, Maryland, USA
* Advanced Network Technologies Division / Information Technology Laboratory
** Smart Grid and Cyber-Physical Systems Program Office / Engineering Laboratory
khalid.halba@nist.gov charif.mahmoudi@nist.gov edward.griffor@nist.gov

Autonomous vehicles bring the promise of enhancing the consumer's experience in terms of comfort and convenience and, in particular, the safety of the autonomous vehicle. Safety functions in autonomous vehicles such as Automatic Emergency Braking and Lane Centering Assist rely on computation, information sharing, and the timely actuation of the safety functions. One opportunity to achieve robust autonomous vehicle safety is by enhancing the robustness of in-vehicle networking architectures that support built-in resiliency mechanisms. Software Defined Networking (SDN) is an advanced networking paradigm that allows fine-grained manipulation of routing tables and routing engines and the implementation of complex features such as failover, which is a mechanism of protecting in-vehicle networks from failure, and in which a standby link automatically takes over once the main link fails. In this paper, we leverage SDN network programmability features to enable resiliency in the autonomous vehicle realm. We demonstrate that a Software Defined In-Vehicle Networking (SDIVN) does not add overhead compared to Legacy In-Vehicle Networks (LIVNs) under non-failure conditions and we highlight its superiority in the case of a link failure and its timely delivery of messages. We verify the proposed architectures benefits using a simulation environment that we have developed and we validate our design choices through testing and simulations.

1 Introduction and Related Work

Safety is a key concern in the development of autonomous vehicles [7]. In fact, according to IBM [8], autonomous cars by 2025 are expected to be equipped with self-healing mechanisms that enable decreased human intervention and maintenance. Self-healing features such as network reconfiguration in the case of a link failure are not natively supported in LIVNs [25]. Attempts to add support for failover mechanisms in LIVNs such as the Controller Area Network (CAN) [16] will come with the price of extra complexity and a hefty development cost. A damaged in-vehicle bus or Electronic Control Unit (ECU) can cause functional exclusion and the consequences might be catastrophic when it comes to safety-critical features.

Introducing complex features such as failover is a design and cost challenge: Car manufacturers tend to introduce features that leverage existing components and design. This is not only a cost saving and risk minimization approach, but also saves the manufacturers significant research and development cost and time. With the advances in network technologies such as SDN and Time Triggered Ethernet (TTE), the introduction of new designs that support complex features without changing the underlying LIVN ECUs [21] [6] is now possible. The usual solution when existing LIVN buses cannot support new features is to re-architect, which introduces new risk and cost throughout the product line engineering.

Resiliency in LIVNs has been the subject of research efforts from industry and academia. In [16], Philips analyzed the different use cases associated with CAN bus failures and proposed Redundant Trans-

M. Gleirscher, S. Kugele, S. Linker (Eds.): 2nd International Workshop on Safe Control of Autonomous Vehicles (SCAV 2018)
EPTCS 269, 2018, pp. 48–58, doi:10.4204/EPTCS.269.5

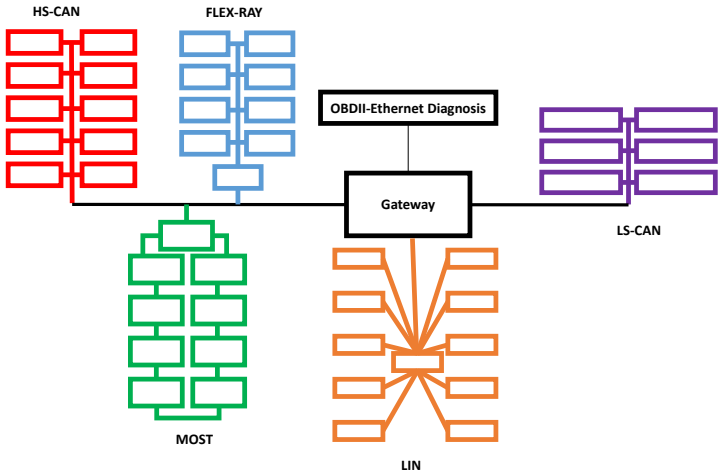

Figure 1: Legacy in-vehicle networks (LIVN) architecture

mission Technique to overcome link or transceiver failure. This technique relies simply on the replication of the ECU's components as well as the CAN bus itself. This allows the CAN system to use a second transceiver or bus if the first one fails. Applying the same technique to the other LIVNs such as LIN (Local Interconnect Network: a single wire network mainly used for non-critical purposes such as body control), Flex Ray (considered as the successor of the CAN bus as it handles more traffic and advanced control features), or MOST(Media Oriented Systems Transport : an optical bus designed to carry infotainment traffic) will lead to an increase in cost, weight, and environmental impact, all of which steer decision making in the automotive industry. Another study highlighted the risk associated with losing parts or most of an in-vehicle bus[12]: The authors categorized the attacks on the in-vehicle bus as a threat to the integrity and the availability of the vehicle's features. A native threat to the resiliency of LIVNs, especially the CAN bus, is its data transmission paradigm that relies on broadcasting data between components [25]. The fact that a CAN frame is received by all the ECUs constitutes a threat to the in-vehicle bus since attackers need only access to one ECU to be able to eavesdrop on all the messages circulating in the bus or inject false messages that could change the behavior of the vehicle. TTE represents a flavor of deterministic and real time Ethernet technologies. It was tested and proved a viable solution for automotive [22] and aeronautics [15] applications. TTE switches also supports fault tolerance [18], which makes it a perfect candidate for future autonomous vehicle's network backbone. SDN is a networking paradigm that leverages separation of control and data plane and network programmability to enable complex features such as fine grained forwarding [4], QoS [24], load balancing [2], and failover [1].

As opposed to the LIVN architecture described in Fig. 1, the architecture we propose enables Fast Failover [1] in the case of a link failure and message delivery in a timely manner regardless of data frequency or refresh rate, this will solve the lack of failover support by LIVNs. Our design has also the added benefit of leveraging unicast communications instead of broadcasts, reducing the risk of eavesdropping on automotive components [12]. Our contribution features SDN as an overlay on top of the CAN bus and the other LIVNs. This overlay will enable complex features that can stimulate innovation and help realize IBM's 2025 self-healing car vision.

In this work, we propose a SDIVN (defined above) with support of Fast Failover to enable autonomous vehicles self-healing in the case of a link failure. This design can scale to support more links

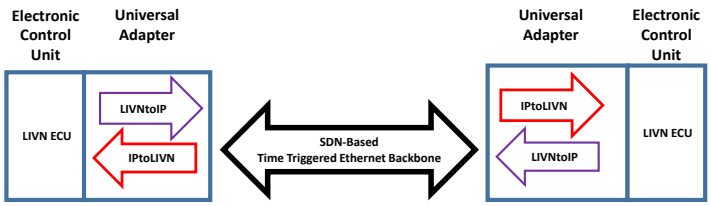

Figure 2: Each ECU is composed of an LIVN message Generator/Receiver and a Universal Adapter that Encapsulates/De-encapsulates ECUs generated/received messages

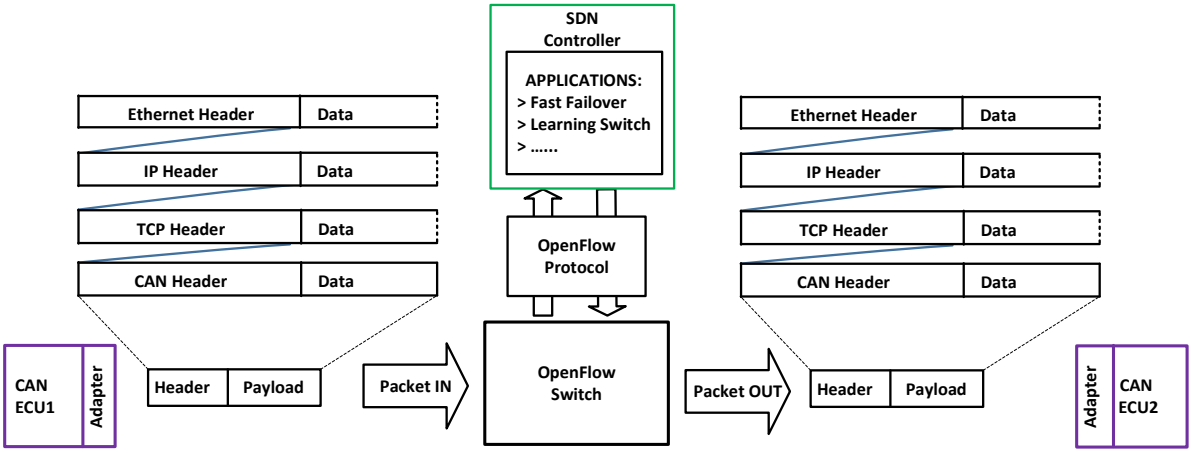

Figure 3: Software Defined In-Vehicle Network design and the journey of a CAN ECU message

and ECUs, which will improve resilience, sustainability and interoperability. We discuss our contributions as follows: In Section 2 we discuss the elements of the design we propose, in Section 3 we leverage the design elements to build a simulation environment for validation. In Section 4, we stress the simulation environment through experimentation, results generation, and results discussion. Finally, we summarize the work we have done in this paper and we highlight current and future work in Section 5.

2 SDIVN Design Elements

In this section we present the SDIVN design that enables the implementation of advanced features within an automotive network. This design not only assures safety mechanisms but also interoperability between different ECU protocols and features. In this work we focus on the safety mechanisms and we leave the interoperability aspect for a future work. The SDIVN is composed of the following components: Electronic Control Units (ECUs) generate and receive data from off/on-board sensor arrays or from other ECUs. These ECUs are IP capable thanks to the universal adapters described in Fig. 2 that enable the communication across the SDN backbone by re-packing LIVN messages (such as CAN messages) to an Ethernet frames and vice-versa. These ECUs are connected to a TTE Backbone through Open Flow capable switches [13]. The choice of TTE as a backbone is due to the fact that it outperforms industrial automotive technologies such as Flex Ray [21] in terms of real time requirements, flexibility, and resiliency. The Open Flow switches are connected to the SDN controller which also runs additional

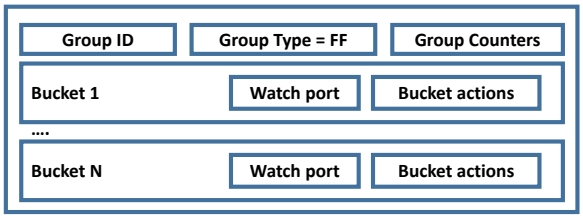

Figure 4: Fast Failover Group Elements

applications related to, for example, resiliency [20], load balancing [2], QoS [26], security [5], etc.

The process of generating data by an ECU includes re-packing it into an Ethernet frame by the adapter, processing it through the SDN backbone, subjecting it to the Open Flow rules within SDN switches, and delivering it to the destination ECUs. This process is described in Fig. 3.

The final element in our design is the Fast Failover application that runs in the SDN controller. Fast Failover [1] is a feature that allows link reconfiguration in the case of a port failure. This feature leverages Open Flow 1.3 Groups to enable port state monitoring and action upon port status change, using watch ports and action buckets. The SDN application that runs at the SDN controller populates SDN switches with flow tables and rules that help the network recover from link failure. Rules include Fast Failover Groups that implement mechanisms of path switching if a link is down. Fig. 4 is an illustration of Fast Failover Group components, the components of relevance are the buckets that contain a port status monitoring watch port; If the latter fails packets are processed according to actions defined in the bucket actions and are eventually sent to backup paths. More details about the dynamics of Fast Failover are explained in [14]. Tests have been run to assess this feature and results show that the time to recover is in the order of microseconds which is acceptable for the applications we intend to test.

3 Design Validation

In this section, we describe the resiliency validation design for both SDIVN and LIVN. An obstacle detection use case is leveraged to highlight how each design operates regarding the resiliency feature.

Obstacle detection in autonomous cars is a key stone property that requires reliable and complete perception of the environment of the vehicle in order to avoid high risk situations. Numerous works have investigated obstacle detection techniques and requirements in autonomous vehicles: In [17] the authors analyzed sonar image data properties in autonomous underwater vehicles and introduced a special sonar data compression technique that optimizes the obstacle avoidance process and enhances sonar data interpretation. In [3] the authors presented a survey on obstacle detection techniques that are based on stereo vision or 2D/3D technologies. For our work, we are rather interested in preserving critical functions such as obstacle detection during a bus failure by leveraging the SDIVN network programmability capability.

For the SDIVN and LIVN the components behavior is described in Fig. 5 as follows: CAN1 ECU and CAN3 ECU generate different CAN messages with different frequencies, each message represents a specific automotive function. For illustration purposes, CAN3 represents a special hardware that detects obstacles in front of the car and sends messages to the Antilock Braking system (ABS) located at CAN4 at a high refresh rate of 1 message per 50 ms. The ABS system ensures the vehicle's stopping based on road conditions and information that comes from the obstacle detection ECU located at CAN3 ECU. CAN1 represents a sensor that measures the road condition and sends messages to an alarm system

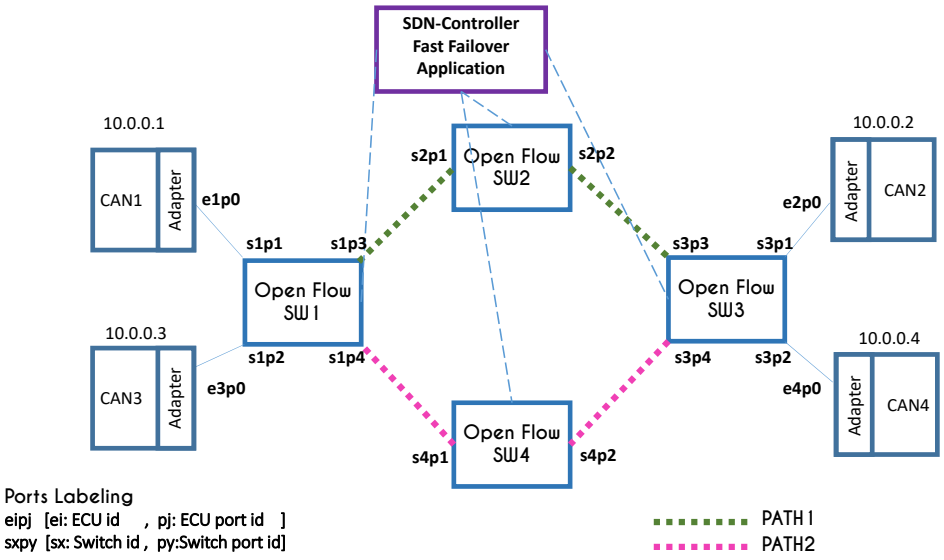

Figure 5: ECUs exchange in-vehicle messages and leverage the SDIVN

located at CAN2 ECU at a low refresh rate of a 1 message per 100ms. Frequencies used here may be different for different applications, they are used here for illustrating the use of SDIVN. We want to verify that under normal conditions (no link failure) the design guarantees data delivery and also ensures the integrity of the signals frequencies at the receiving ECUs. We also want to establish the benefit of the Fast Failover mechanism when a link fails.

3.1 SDIVN

We validate the proposed Software Defined In-Vehicle Network for link failover by the implementation of the different components described in the previous section and running a simulation of the proposed design. Fig. 5 illustrates the validation use case for the SDIVN.

We are aware that the size and design of the SDIVN in Fig. 5 is not representative of the wide spectrum of possible sizes and architectures. We are reducing the scope in purpose in order to make a proof of concept for the resiliency property.

We care about the integrity of the message frequency because as we approach an obstacle, if the frequency is altered at the reception, the detection or the expected actuation process might take longer than it should, and appropriate reaction to it in a timely manner might not be possible.

We have implemented the simulation environment using Mininet 2.2.1 installed on a patched Linux Kernel (4.11.0-rc8+) that supports CAN-Utils Package namespaces [14]. Additionally, we have built the protocol adapters based on TCP/IP Sockets (that ensure retransmission in the case of a packet loss) and CAN-Sockets [23]. For the SDN implementation, Floodlight [19] is our choice for the controller as it supports the implementation of Fast Failover. A representation of the software stack we have leveraged is described in Fig. 6

After running the simulation, the SDN controller populates the flow and group tables with necessary rules and actions to handle link failover Fig. 7 is a non-exhaustive perspective on the group table and flow table rules installed after running the application. The hosts are discoverable using the learning

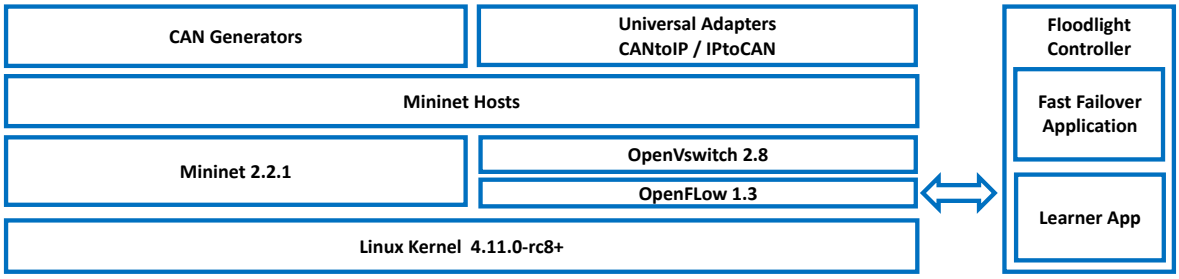

Figure 6: Developed and Open-Source components used in the validation experiment

Flow Table 0 for SW1

Table	SRC	DST	Instructions
0	Port1 or Port2	Port3 or Port4	Goto Group1
0	Port3 or Port4	ECU1 or ECU3	Forward to corresponding host

Flow Table 0 for SW3

Table	SRC	DST	Instructions
0	Port1 or Port2	Port3 or Port4	Goto Group1
0	Port3 or Port4	ECU2 or ECU4	Forward to corresponding ECU

Group 1 for SW1

Group ID	Group Type	Action Bucket
1	Fast Failover	If 3 is down 4 is up If 4 is down 3 is up

Group1 for SW3

Group ID	Group Type	Action Bucket
1	Fast Failover	If 3 is down 4 is up If 4 is down 3 is up

Figure 7: Open Flow tables and Group tables after the SDN controller runs the Fast Failover application

switch application that runs in the beginning of the process. The fast failover application operates as follows: One path can be used at a time, if PATH1 is enabled, PATH2 ports are disabled and vice versa, and ports from different paths can't be enabled at the same time. If there is a port failure on one path, the Fast Forwarding application disables ALL ports on that path and switches to another path (see Fig. 5 for illustration).The code for the application we have used for this setup is available in [10].

3.2 LIVN

For the CAN bus simulation, we have adopted the design in Fig. 8. This design can't support multiple paths for failover as the CAN bus is a broadcast medium and adding extra links to it will result in broadcast storms, resulting in replicate messages in the beginning of the data transfer, and a complete outage of the bus after few seconds. These are some of the limitations of the CAN bus.

4 Results Discussion

In this section we show the benefit of the Fast Failover resiliency mechanism in the SDIVN design we have proposed, and we estimate the Average Failover Cost per packet which will inform us about whether or not the impact of the Fast Failover mechanism is significant.

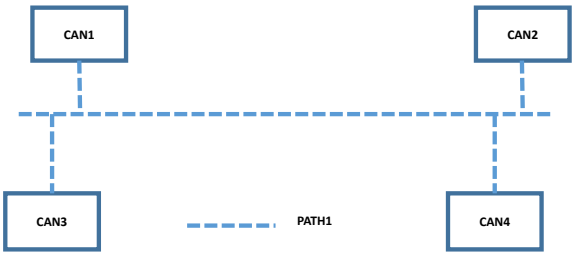

Figure 8: ECUs exchange CAN messages and leverage a broadcast medium to simulate a CAN bus

4.1 Results outcome for SDIVN and LIVN

We have run experiments described in the previous section for 60 seconds, and we simulate a link failure at t=30 seconds. Heuristic experiments show that the transition from one failed path to an up and running path in the case of SDIVN is almost instantaneous which is illustrated in Fig. 9. The figure shows how the SDIVN design we proposed preserved the function and its properties while the LIVN didn't. SDIVN allows the network to quickly recover from link failure and preserves message delivery and frequency. Another benefit of the SDIVN is its unicast nature, messages are sent to their specific destinations based on rules generated by the application running in the SDN controller, reducing the risk of bus outage or the replicate message caused by broadcasts. In the same figure we see that the LIVN is incapable of recovering from a link failure as it does not support resiliency mechanisms.

4.2 Timing Analysis

The main focus of this subsection is to estimate the delay cost of the Fast Failover mechanism. For this end, we describe the timing parameters that are involved when a packet is processed through the proposed SDIVN.

We define the total propagation delay T_pd as the sum of the propagation delays of the N links $P(i)$ connecting sending and receiving endpoints.

$$T_pd = \sum_{i=1}^{N} P(i) \tag{1}$$

We define the Transfer Time T_p for 1 packet from a sender to a receiver as the difference in time between the reception time T_r of the packet at the receiving ECU and the sending time T_s at the generating ECU, as expressed in (2).

$$T_p = T_r - T_s \tag{2}$$

T_p can also be expressed as follows (3)

$$T_p = T_pd + ED_d + F_d + C_d + FO_d \tag{3}$$

Where T_pd is the total propagation delay along the path between the two ECUs, ED_d is the time it takes to encapsulate and decapsulate the packet at the adapter, F_d is the forwarding time at the Open Flow

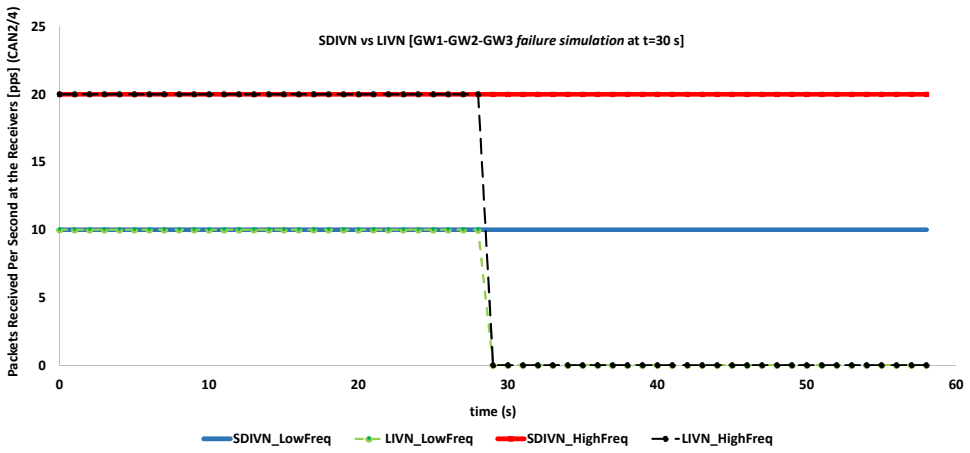

Figure 9: Fast Failover results in both cases: SDIVN instantly switches to PATH2 when PATH1 fails. LIVN fails to maintain function since it does not have failover mechanisms

switches, C_d is the processing time at the SDN controller, and FO_d is the failover time in the case of a link failure

The encapsulation time and decapsulation time ED_d is the time it takes the universal adapter at the ECU's edge to Encapsulate E_d and Decapsulate D_d the packet as expressed in (4)

$$ED_d = E_d + D_d. \tag{4}$$

The failover time FO_d is the time it takes the system to detect failure DF_d, reconfigure the network RN_d, and retransmit the packet RP_d if it was lost during failure, as expressed in (5)

$$FO_d = DF_d + RN_d + RP_d \tag{5}$$

We define the Transfer Time for all packets TT_d as the sum of the Transfer Time of all received packets $T_p(i)$. K is the total number of received packets, as expressed in (6)

$$TT_d = \sum_{i=1}^{K} T_p(i) \tag{6}$$

We define the average packet transfer time ATT as follows (7)

$$ATT = TT_d/K. \tag{7}$$

Under normal conditions (no link failure), $ATT = ATTN_p$, and if a link failure occurs during the transmission process, ATT=$ATTF_p$. The relationship between $ATTF_p$ and $ATTN_p$ is expressed in (8)

$$AFCP_p = ATTF_p - ATTN_p \tag{8}$$

For a single link failover we calculate the Average Failover Cost Per Packet $AFCP_p$ for both refresh rates (50 ms and 100 ms). We can express $AFCP_p$ as the difference in time between the Average

	Refresh Rate = 50ms	Refresh Rate = 100ms
ATTFp	50.404777 ms	50.337497 ms
ATTNp	50.403160 ms	50.336002 ms

Figure 10: $ATTN_p$ and $ATTF_p$ for both frequencies.

Packet Transfer Time during normal conditions $ATTN_p$ and the Average Packet Transfer Time during link failover $ATTF_p$. This calculation will give us insights on whether or not Fast Failover has a negative impact on the functionality.

We calculate the $AFCP_p$ based on the $ATTF_p$ and $ATTN_p$ values we have measured and logged in Fig. 10

$AFCPp$**[50 ms]** = $ATTFp$ - $ATTNp$ = 1.617 us = 0.00320% of the $ATTNp$[50 ms].

$AFCPp$**[100 ms]** = $ATTFp$ - $ATTNp$ = 1.495 us = 0.00297% of the $ATTNp$[100 ms].

4.3 Summary of Results

The Average Failover Cost Per Packet $AFCPp$ value in both cases (50 ms and 100 ms refresh rate) is negligible compared to the average transfer time per packet $ATTN$. We conclude that the Fast Failover mechanism does not cause a significant delay overhead when a link failure occurs in this example. However, it guarantees timely message delivery and message frequency integrity.

5 Conclusion

In this work, we presented a new design for in-vehicle networks that allows recovery from link failure by leveraging the network programmability provided by SDN. Since automotive networks are size determined, engineers could use SDIVN to design failover paths for a functionality that they judge to be critical. We have demonstrated through simulation that this mechanism is so efficient that the time to recover is insignificant for the frequencies and network size we have tested, and we believe that the positive results perceived during experiments we ran will hold for larger networks. In-Vehicle network components and links will continue to evolve in terms of hardware performance and software optimisation to accomodate the bandwidth and delay requirements.

For Future work, we are exploring other interesting areas such as the impact of interoperability between different ECUs. We believe that interoperability between heterogeneous data sources within an SDIVN will stimulate the innovation of a new spectrum of advanced autonomous vehicle features. We are also conducting a statistical analysis based on the fractional factorial design [9], the goal is to understand which parameters (network size, bus speed, bus degradation, bus delay ...) have the most influence on certain autonomous vehicle properties of interest(resiliency, interoperability, scalability ...). Another area of interest is Software Defined Wireless In-Vehicle Networks (SDWIVN), for which we built a testbed and conducted promising experiments [11], such design choice has the potential of bringing novel contributions to energy saving techniques and environmental impact improvement.

NIST Disclaimer

Any mention of commercial products or organizations is for informational purposes only; it is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the products identified are necessarily the best available for the purpose. The identification of any commercial product or trade name does not imply endorsement or recommendation by the National Institute of Standards and Technology, nor is it intended to imply that the materials or equipment identified are necessarily the best available for the purpose. Certain commercial entities, equipment, or materials may be identified in this document in order to describe an experimental procedure or concept adequately. Such identification is not intended to imply recommendation or endorsement by NIST, nor is it intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose.

References

[1] Neda Beheshti & Ying Zhang (2012): *Fast failover for control traffic in software-defined networks*. In: *Global Communications Conference (GLOBECOM), 2012 IEEE*, IEEE, pp. 2665–2670, doi:10.1109/GLOCOM.2012.6503519.

[2] Mikhail Belyaev & Svetlana Gaivoronski (2014): *Towards load balancing in SDN-networks during DDoS-attacks*. In: *Science and Technology Conference (Modern Networking Technologies)(MoNeTeC), 2014 First International*, IEEE, pp. 1–6, doi:10.1109/MoNeTeC.2014.6995578.

[3] Nicola Bernini, Massimo Bertozzi, Luca Castangia, Marco Patander & Mario Sabbatelli (2014): *Real-time obstacle detection using stereo vision for autonomous ground vehicles: A survey*. In: *Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on*, IEEE, pp. 873–878, doi:10.1109/ITSC.2014.6957799.

[4] Andrea Bianco, Robert Birke, Luca Giraudo & Manuel Palacin (2010): *Openflow switching: Data plane performance*. In: *Communications (ICC), 2010 IEEE International Conference on*, IEEE, pp. 1–5, doi:10.1109/ICC.2010.5502016.

[5] Marc C Dacier, Hartmut König, Radoslaw Cwalinski, Frank Kargl & Sven Dietrich (2017): *Security Challenges and Opportunities of Software-Defined Networking*. *IEEE Security & Privacy* 15(2), pp. 96–100, doi:10.1109/MSP.2017.46.

[6] Jia Lei Du & Matthias Herlich (2016): *Software-defined Networking for Real-time Ethernet.* In: *ICINCO (2)*, pp. 584–589.

[7] Pedro Fernandes & Urbano Nunes (2012): *Platooning with IVC-enabled autonomous vehicles: Strategies to mitigate communication delays, improve safety and traffic flow*. *IEEE Transactions on Intelligent Transportation Systems* 13(1), pp. 91–106, doi:10.1109/TITS.2011.2179936.

[8] Debanjan Ghosh, Raj Sharman, H Raghav Rao & Shambhu Upadhyaya (2007): *Self-healing systemssurvey and synthesis*. *Decision Support Systems* 42(4), pp. 2164–2185, doi:10.1016/j.dss.2006.06.011.

[9] Richard F Gunst & Robert L Mason (2009): *Fractional factorial design*. *Wiley Interdisciplinary Reviews: Computational Statistics* 1(2), pp. 234–244, doi:10.1002/wics.27.

[10] Khalid HALBA & Charif MAHMOUDI: *can-2-ip-wrapper*. Available at `https://github.com/usnistgov/Intra-Vehicular-Networks/tree/master/can-2-ip-wrapper`.

[11] Khalid HALBA & Charif MAHMOUDI: *Software Defined Wireless In-Vehicle Network TestBed*. Available at `https://github.com/usnistgov/Intra-Vehicular-Networks/blob/master/infra-setup.md`.

[12] António Lima, Francisco Rocha, Marcus Völp & Paulo Esteves-Verissimo (2016): *Towards Safe and Secure Autonomous and Cooperative Vehicle Ecosystems*. In: *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy*, ACM, pp. 59–70, doi:10.1145/2994487.2994489.

[13] Thomas A Limoncelli (2012): *Openflow: a radical new idea in networking*. Queue 10(6), p. 40, doi:10.1145/2246036.2305856.

[14] Ying-Dar Lin, Hung-Yi Teng, Chia-Rong Hsu, Chun-Chieh Liao & Yuan-Cheng Lai (2016): *Fast failover and switchover for link failures and congestion in software defined networks*. In: Communications (ICC), 2016 IEEE International Conference on, IEEE, pp. 1–6, doi:10.1109/ICC.2016.7510886.

[15] Andrew Loveless (2015): *TTEthernet for Integrated Spacecraft Networks*.

[16] Philips-Electronics (1991): *CAN Bus Failure Management Using the P8xC592 Microcontroller*. Technical Report.

[17] Robert L Pietsch (1998): *Autonomous vehicle obstacle detection and tracking*. In: Underwater Technology, 1998. Proceedings of the 1998 International Symposium on, IEEE, pp. 198–202, doi:10.1109/UT.1998.670091.

[18] John Rushby (2001): *Bus architectures for safety-critical embedded systems*. In: Embedded Software, Springer, pp. 306–323, doi:10.1007/3-540-45449-7_22.

[19] Alexander Shalimov, Dmitry Zuikov, Daria Zimarina, Vasily Pashkov & Ruslan Smeliansky (2013): *Advanced study of SDN/OpenFlow controllers*. In: Proceedings of the 9th central & eastern european software engineering conference in russia, ACM, p. 1, doi:10.7125/APAN.35.2.

[20] Anderson Santos da Silva, Paul Smith, Andreas Mauthe & Alberto Schaeffer-Filho (2015): *Resilience support in software-defined networking: A survey*. Computer Networks 92, pp. 189–207, doi:10.1016/j.comnet.2015.09.012.

[21] Till Steinbach, Franz Korf & Thomas Schmidt (2011): *Real-time Ethernet for Automotive Applications: A Solution for Future In-Car Networks*. doi:10.1109/ICCE-Berlin.2011.6031843.

[22] Till Steinbach, Franz Korf & Thomas C Schmidt (2010): *Comparing time-triggered Ethernet with FlexRay: An evaluation of competing approaches to real-time for in-vehicle networks*. In: Factory Communication Systems (WFCS), 2010 8th IEEE International Workshop on, IEEE, pp. 199–202, doi:10.1109/WFCS.2010.5548606.

[23] CAN-Utils Team: *CAN-Utils package*. Available at `https://github.com/usnistgov/Intra-Vehicular-Networks/tree/master/can-2-ip-wrapper`.

[24] Slavica Tomovic, Neeli Prasad & Igor Radusinovic (2014): *SDN control framework for QoS provisioning*. In: Telecommunications Forum Telfor (TELFOR), 2014 22nd, IEEE, pp. 111–114, doi:10.1109/TELFOR.2014.7034369.

[25] Shane Tuohy, Martin Glavin, Ciarán Hughes, Edward Jones, Mohan Trivedi & Liam Kilmartin (2015): *Intra-vehicle networks: A review*. IEEE Transactions on Intelligent Transportation Systems 16(2), pp. 534–545, doi:10.1109/TITS.2014.2320605.

[26] Jennifer M Yates (2017): *Managing service quality in a software defined network*. In: Optical Fiber Communication Conference, Optical Society of America, pp. M2H–4, doi:10.1364/OFC.2017.M2H.4.