Enhancing Fault Detection in Time Sensitive Networks using Machine Learning

Nitin Desai and Sasikumar Punnekkat
Mälardalen University, Sweden
firstname.lastname@mdh.se

***Abstract*—Time sensitive networking (TSN) is gaining attention in industrial automation networks since it brings essential real-time capabilities to the Ethernet layer. Safety-critical real-time applications based on TSN require both timeliness as well as fault-tolerance guarantees. The TSN standard `802.1CB` introduces seamless redundancy mechanisms for time-sensitive data whereby each data frame is sequenced and duplicated across a redundant link to prevent single points of failure (most commonly, link failures). However, a major shortcoming of `802.1CB` is the lack of fault detection mechanisms which can result in unnecessary replications even under good link conditions - clearly inefficient in terms of bandwidth use. This paper proposes a machine learning-based intelligent configuration synthesis mechanism that enhances bandwidth utilization by replicating frames only when a link has a higher propensity for failure.**

***Index Terms*—Time sensitive networking, network configuration, machine learning, safety-critical systems, fault-tolerance, redundancy, fault-detection**

I. Introduction

Safety-critical systems in industrial automation need real-time guarantees from the Ethernet layer to deliver control data within a bounded time [1]. Consequently, network design for such systems has traditionally been conservative in the sense that requirements are developed with meticulous detail and the worst case network traffic is included in design prior to deployment. This provides operational guarantees by means of safety certification issued by agencies in accordance with applicable standards.

In addition to real-time guarantees such as bounded worst-case delays and jitter [2], redundancy measures must be incorporated in the network design to ensure that time-sensitive data meet their real-time requirements *in spite of* faulty network conditions. This fundamental goal has driven the design and development of networking systems for safety-critical applications. Among the foremost of such design paradigms is the Time-triggered (TT) network paradigm [3] which promises that time-critical control frames are delivered with bounded delays. Such a design paradigm has proven to be effective in a multitude of domains such as avionics, industrial automation and automotive all of which have stringent demands of determinism [4].

More recently, Time Sensitive Networking (TSN) has emerged as a front-runner and a competing technology to the well established field-bus standard that have been the staple of industrial and automotive networking [5]. TSN is a set of standards that provide real-time guarantees over standard Ethernet. Hence, current IEEE `802.1Q` standards [6] come integrated with TSN so that vendors can directly provide properties such as timeliness, fault-tolerance, reliability and availability to their networking products.

From a network layer perspective, in particular, it implies the co-existence of mixed traffic classes (standard best-effort Ethernet traffic alongside time-sensitive safety-critical control traffic). Additionally, thanks to the lowering cost of sensors and cameras, industrial automation and automotive applications (e.g., ADAS) are becoming bandwidth hungry. Such demands on the dual requirements of high bandwidth for multimedia data and deterministic, fault-tolerant delivery of safety-critical data can only be supported by a networking technology such as TSN Ethernet that can guarantee high reliability, fault-tolerance, and availability. Futuristic automotive and industrial network applications such as ADAS and smart factories envisage both best effort data traffic as well as time sensitive control traffic on the same network. Additionally, the widespread use of existing Ethernet technology is a strong driver to motivate the need for enhancing standard Switched Ethernet with TSN capabilities. From a safety perspective (at the network layer), introduction of such mechanisms requires traffic isolation as well as seamless redundancy. The TSN standard consists of a number of (sub)standards, some of which are - Timing and Synchronization for Time-Sensitive Applications (`802.1ASrev`), Stream Reservation Protocol (SRP) Enhancements and Performance Improvements (`802.1Qcc`), Enhancements for Scheduled Traffic (`802.1Qbv`) and Frame Replication and Elimination for Reliability (`802.1CB`).In this paper, we focus on provisions for seamless redundancy within TSN defined by `802.1CB` standard. The working principle of `802.1CB` is that a set of FRER or Frame Replication and Elimination for Reliability functions are instantiated on ports of switches and nodes. These functions replicate each time-sensitive frame belonging to a stream (see Section II) at various points (more precisely, on ports) in the network and eliminate the replicates at the receiver node to the extent that the higher layer protocols do not have any knowledge of this process thereby appearing seamless to the application.

A. Motivation for fault-detection in TSN

Redundancy is an essential requirement to provide fault tolerance. Fault detection implies the process wherein a node or link fault in the network is detected and identified. Without loss of generality, we restrict ourselves to communication link

faults that can potentially cause link failures. We attempt to tackle a hitherto under-researched shortcoming - the lack of fault detection in `802.1CB` - by introducing an ML-based fault detection in order to instantiate FRER functions only on ports connecting links which have the highest probability of failure (a natural consequence of having faults). This is in contrast to the existing mechanism whereby frames are *arbitrarily* replicated and eliminated. This can be lead to inefficient bandwidth use, which is a premium commodity. Consequently, our approach aims to instantiate these functions at runtime based on the decisions provided by the ML-based fault detection. Faults must be detected before a link fails and redundancy established prior to failure. Furthermore, the execution of FRER functions must be maintained as long as the faulty state continues. It is essential to note however, that the specific ports where FRER functions are instantiated (*arbitrarily*) is part of the pre-deployment design phase.

B. Motivation for ML-based fault-detection

A typical TSN network has a vast number of parameters that must be configured for precise operation. These parameters can also be used to gather valuable information (either directly or inferentially) about link conditions. Software-defined networking paradigm is being increasingly used in TSN (refer to Section V) which makes it easy to introduce an ML-layer over the control plane. Additionally, ML algorithms have been used in network anomaly detection since the past decade [7] and are becoming increasingly powerful as well as accurate. The ML paradigm utilizes existing data to provide value addition to decision making, and in this specific case, to enable redundancy features for fault-tolerant behaviour while efficiently utilizing bandwidth. Furthermore, the complexity of configuration and scheduling schemes defy a deterministic and analytical solution to this problem as well as to implement it in real-time.

To the best of our knowledge, no previous works have focused on fault detection in TSN using ML-based methods. In this paper, we present an intelligent configuration synthesis approach that can enable us to design a process to detect faulty links and help in deciding when, where and how to replicate frames to avoid bandwidth wastage.

The remainder of the paper is structured as follows: Section II describes the system model. Section III introduces the notion of intelligent configuration synthesis and its motivation followed by Section IV which discusses the role of machine learning for fault detection in the specific TSN standard `802.1CB`. Section V refers the reader to recent related works on TSN as well as on machine learning in the networking domain. Finally, Section VI concludes the paper and outlines future work in this direction.

II. System model

A network is depicted as a directed acyclic graph (DAG), $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ with directed edges, $\mathcal{E}$ connecting the vertices, $\mathcal{V}$. A physical link between the vertices v_i, v_j is denoted by $(v_i, v_j), (v_j, v_i) \in \mathcal{E}$ where the first vertex in the pair description defines the source node and the second vertex defines the destination node. Nodes are either the source or destination of messages (end systems) or may forward messages to other nodes (switches). TSN standards define an entity called stream as a periodic flow of data transmission (the message) from one talker (the sender node) to one or more listeners (the receiver nodes) via intermediate nodes (switches). We denote the set of all streams in the network with $\mathcal{S}$. Similar to [8][9], we denote the route of a stream, $s_i \in \mathcal{S}$ from talker v_1 to listener v_n routed through intermediary nodes $v_2, v_3, ..., v_{n-1}$ as $\mathcal{R}_i = [(v_1, v_2), ..., (v_{n-1}, v_n)]$.

A stream $s_i \in \mathcal{S}$ is defined by the tuple $\langle C_i, T_i, L_i, J_i \rangle$ denoting the message size in bytes, the period, the maximum allowed end-to-end latency, and the maximum allowed jitter of the stream, respectively.

Each stream is composed of frames which are the standard 802.1Q Ethernet frames[6]. We denote these by f, characterized by its period, length and priority:

$$\forall f_{s_i} \in F : f_{s_i} = \{T_{f_i}^{s_i}, len_{f_i}^{s_i}, prio_{f_i}^{s_i}\} \tag{1}$$

Consequent to the above model design, we have each link, $\mathcal{L}_i, i \in \mathcal{E}$, on which frames from multiple streams, s_i with varying periods, $T_{f_i}^{s_i}$ flow (from (1)).

A. Fault model

The system model is composed of two principal components, the nodes (talkers, listeners and switches) and the communication links. There are many types of faults that can affect the transmission and reception of messages within this system. This paper focuses on transient faults on communication links (edges) that connect switches to nodes and nodes to other nodes. It has been observed that in typical automotive Ethernet networks, the most common faults are transient faults in the wired communication links that occur at the rate of $10^2/(h * car)$ [10]. These could be due to bit flips caused by electromagnetic interference or other sources. Bit flips can cause frame loss since the cyclic redundancy check(CRC) at the receiver detects a corrupted frame and is discarded. The fault occurrence can follow a statistical probability distribution function, can be sporadic or even periodic. In this paper, we do not delve into these particularities.

Below are examples of intermittent faults that can occur on nodes as well as links.

- A stuck transmitter sending the same sequence numbers for multiple packets or the same packet in each transmission cycle;
- A corrupted link sending erroneous sequence numbers or application payload;
- Transient link failure causing erroneous (corrupted) frames to be delivered and normal ones after a random interval which is decided by the probability distribution of the fault.

A fault, ψ_i on each communication link, $\mathcal{L}_i$ is characterized by a tuple:

$$\forall \psi_i \in \Psi : \psi_i = \{\nu_{\psi_i}, T_{\psi_i}, \mathcal{L}_{\psi_i}\} \tag{2}$$

representing the frequency of occurrence of the fault ν_{ψ_i}, the duration for which the fault occurs T_{ψ_i} and the identity of the faulty link $\mathcal{L}_{\psi_i}$, respectively.

The faults result in packet losses $\mathcal{P}^i_{loss}$ for each link $\mathcal{L}_i \in \mathcal{E}$ which are observed at the receiver MAC layer by means of CRC checksum validation. We are interested in the number of packet losses within a specified observation window, $\mathcal{T}_{obs}$ on each link.

B. Problem formulation

We are given a networked system of end stations (talkers and listeners), switches and links. Specific set of streams, $s_i \in \mathcal{S}$ flow across the network with given attributes. The fault model describes the faults on each link that can result in packet losses. These are measured for a specified observation window, $\mathcal{T}_{obs}$ on each link to generate the necessary training data set which is then used to make decisions as to where the FRER functions need to be executed to effect the same redundancy benefits as with an arbitrary scheme but with the added advantage of bandwidth conservation.

C. Assumptions

1) Intermittent faults occur either randomly or with a given periodicity that can be defined in the analysis;
2) Only the communication links are assumed to experience faults; Nodes and switch faults are inconsequential to our problem i.e., we assume fault-free operation;
3) A ring topology is assumed which in itself is conducive to fault-tolerance. However, the topology aspects are not a limiting factor;
4) All streams, $s_i \in \mathcal{S}$ are assumed time-sensitive in that their delivery at the receiver must be guaranteed within a worst-case delay less than or equal the period as specified in the tuple $\langle C_i, T_i, L_i, J_i \rangle$;
5) All frames belonging to a stream have the same size, period and priorities;
6) The redundant link is assumed to be unaffected by faults.

III. Intelligent Configuration Synthesis

It is evident by a careful study of the `802.1CB` standard that there exist no fault detection mechanisms in order to detect when and where a fault occurs. The functions described in the standard merely replicate and eliminate frames belonging to streams based on a set of rules formulated prior to deployment (static) in order to ensure seamless redundancy. This can be wasteful of bandwidth since the streams that are correctly transmitted are also replicated with no apparent benefit. Hence, we motivate the need for an improved method to achieve bandwidth savings by adding intelligence on top of the redundancy mechanisms so that the functions described in the standard are instantiated only when there is a real and provable need for replication of critical frames.

However, there exists a function in the standard, the `LatentErrorTest`, which detects latent errors by monitoring the number of packets discarded by the `BaseRecoveryFunction`. However, this function does not specify when and where a fault occurs in the network.

Consequently, we see that a purely static approach to synthesizing a configuration cannot provide efficient resource utilization. In this paper, we sketch the details of a predictive error detection mechanism that monitors the network parameters at a specified rate (which can be varied) and takes as its input a subset of these parameters. An Artificial Intelligence (AI) engine processes these values and predicts the likelihood of a fault (location and instant of occurrence) even before it happens.

The outputs of the fault detection mechanism form the inputs to the configurator that will then re-configure the network parameters (numerical values to the *managed objects*) to reflect the changes that need to be made in light of these new fault conditions.

A. Fully Centralized Configuration of TSN

Three configuration models are detailed in the `802.1Qcc` standard [11] - the fully distributed model, the centralized network/distributed user model and the fully centralized model. From a practical perspective, a fully centralized model is considered since it has complete knowledge of the network parameters at any given time instant. The principal logical components of the fully centralized configuration model are Central user configuration (CUC) and central network configuration (CNC) as shown in Fig. 1.

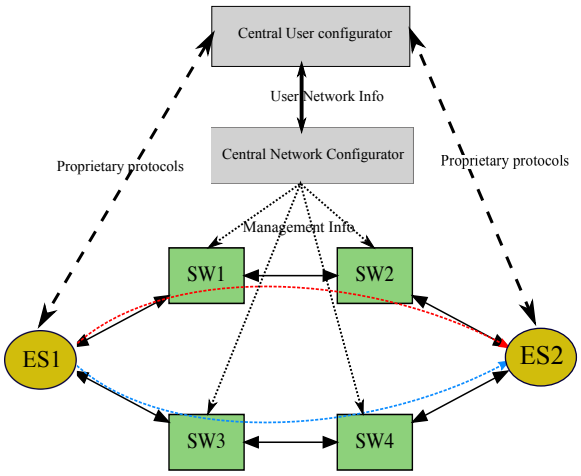

Fig. 1. Central configuration of TSN mechanisms [11]

The CUC is the interface between the user (typically a control engineer who has knowledge about the network requirements) and the CNC (described below). The CUC discovers end stations, retrieves end station capabilities and user requirements, and configures TSN features in end stations. Therefore clearly, as the name suggests, the CUC deals entirely with the end stations and user requirements.

The CNC has a complete view of the network topology and the capabilities of the switches and end-stations. This knowledge together with user requirements (such as deterministic latency and fault-tolerance) from the CUC to generate

a network-wide configuration that must guarantee said performance. The particularities of the standard are beyond the scope of this paper and hence interested readers are referred to [11]. The CNC communicates with the switches as well as extracts parameters using network management protocol such as RESTCONF [12] or NETCONF [13].

B. Redundancy features of 802.1CB

The primary functionality of 802.1CB is to provide seamless redundancy against single points of failure (SPoF) in the links. An arbitrary topology is shown in Fig. 2. Two streams are depicted using the red and blue dotted arrows respectively.

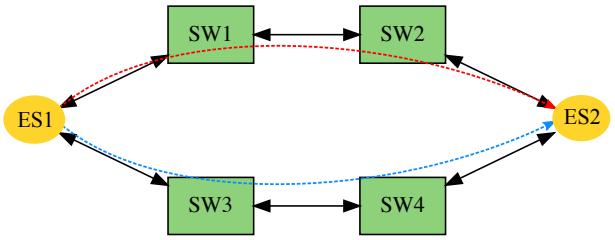

Fig. 2. A ring topology of switches (SW) and end systems (ES).

C. List of seamless redundancy functions

- Stream splitting
- Sequence generating function - *SGen*
- Stream identification function
- Sequence encode and decode function
- Sequence recovery function - *SRec*
- Stream forwarding function

Out of these, only two functions - *SGen* and *SRec* - are central to our problem. The *SGen* function generates ordered sequence numbers for each frame belonging to a stream that is replicated across multiple routes while the *SRec* function eliminates the duplicate frames identified by the sequence numbers.

IV. ML-based fault detection for 802.1CB

We focus on transient faults induced by the wired communication links. The monitor inspects network traffic using the following parameters to make inferences about anomalies. Hence, the problem we tackle here is largely one of anomaly detection in the networking domain with specific application to time-sensitive traffic.

A. Link quality measure

A link could be degraded by soft errors due to electromagnetic interference or other sources which can occur randomly or periodically. A drop in link quality manifests itself as a degradation of the transmitted data leading to corrupted frames and eventually dropped frames (owing to mismatch in cyclic redundancy check sequence at the receiver). As soon as such a degradation is detected (inferred from dropped frames at the receiver port), the fault detection mechanism identifies the link in question, the precise time of occurrence, the duration of the link quality drop (reflecting the number of packet losses) and the frequency of the link degradation (assuming a large observation window).

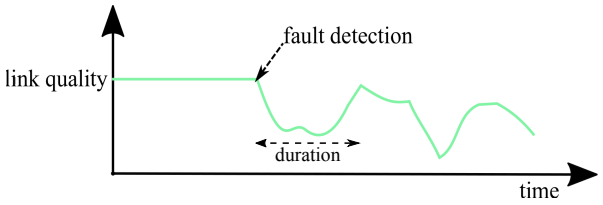

Fig. 3. Link quality as inferred from packet losses at the MAC layer.

B. Training data for fault detection

An anomaly in our system model is a condition wherein the pattern of linkQuality changes too differently (large variance) from expected patterns. To enable such a pattern to be recognized in advance, the packet losses are observed over a large duration (increasing $\mathcal{T}_{obs}$). This constitutes the training data collection process.

Link_ID	Stream_ID	Frame_instance	Timestamp	LinkQuality	PacketLoss

Fig. 4. Parameters for the training data set.

The link, stream and frame parameters in Fig. 4 are readily obtained from the *managed objects* in the TSN standard 802.1Qcc. The rest of the parameters of the data set are from monitoring the MAC layer at the receiver port. We assume a fixed sampling interval during the entire data collection stage.

C. Data processing

The training data is a collection of packet loss events with accurate timestamps for each link, stream and frame belonging to a stream in the network. Depending on the sampling rate and the value of the observation window $\mathcal{T}_{obs}$, the dimensionality of the data set can quickly become unmanageably huge as is common in almost all network measurement data sets. Towards this end, we wish to employ *principal component analysis (PCA)*, a well known procedure for reducing the dimensionality of the variable space by representing it with a few orthogonal (uncorrelated) variables that capture most of its variability. This has been used previously for anomaly detection in the networking domain [14].

V. Related work

Over the past decade, interest in TSN and its applicability to real-time safety-critical systems has been growing both in academia and industry. A large chunk of research has been devoted to generating real-time schedules based on satisfiability modulo theorems (SMT) [15][8][9][16][2]. Recently, however as more features have been added to the TSN standards, the complexity of configuring the network has emerged as a key driver for academic research. In [17], the authors describe a configuration agent that generates a feasible schedule based on information such as traffic patterns. A more recent work by

the authors [18] develops a runtime reconfiguration algorithm for centralized monitoring of the TSN network. A particular challenge has been to introduce flexibility in the configuration process while maintaining guaranteed safety performance. [19] [20] discuss fault tolerance and safety aspects of TSN as a key enabler for Fog computing [21]. [22] gives a very good overview of various challenges for TSN in the industrial automation context.

TSN networks are increasingly becoming software-defined [23][24][25] and are indeed compatible with the centralized configuration model presented in `802.1Qcc`[11]. Such a software paradigm enable ML algorithms like anomaly detection and condition monitoring to be easily deployed to provide runtime diagnosis - a key element in any decision-based control process [26]. Industry driven research such as [27] [28] provide a stimulating discussion on the general topic of AI-based safety. A relevant paper to the topic of fault detection and recovery is [29] where the authors discuss an architecture that integrates multiple and diverse technologies, as hypervisors, run-time monitoring, redundancy with diversity, predictive fault detection, fault recovery, and predictable resource management. Seshia *et al.* [30] discuss verification of artificial intelligence. The authors explain the challenges of verifying AI systems using formal methods approach which have so far worked well with systems with specified behaviour. a principle aspect of which is system modelling.

VI. Conclusion

In this paper, we have provided a feasible path towards introducing fault detection capabilities into an important TSN standard, `802.1CB` for seamless redundancy. The parameters needed to create the dataset are available from the standard-defined *managed objects*. Additionally, our approach uses network parameters that can easily be extracted through tools such as a packet sniffer like wireshark and used in established anomaly detection algorithms to help detect accurately the links where faults and failures are most likely to occur.

As future work, a simulation of the proposed scheme will be undertaken by integrating this with existing TSN-based simulators implemented in OMNET++ network simulator. Additionally, an important aspect is to be able to provide guarantees for the correct operation of such ML-based schema in traditional safety-critical networked systems through model checking. We therefore envisage that such a scheme can, after due verification and validation, be inducted into the standard as an enhancement.

Acknowledgment

The reported research is funded by the EU Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 764785, FORA—Fog Computing for Robotics and Industrial Automation and by the Swedish Foundation for Strategic Research (SSF) via the Future Factories in the Cloud (FiC) project.

References

[1] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0," *IEEE Industrial Electronics Magazine*, vol. 11, pp. 17–27, March 2017.

[2] W. Steiner, "Synthesis of static communication schedules for mixed-criticality systems," in *IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, ISORCW '11, (Washington, DC, USA), pp. 11–18, IEEE Computer Society, 2011.

[3] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE*, vol. 91, pp. 112–126, Jan 2003.

[4] S. Craciunas and R. Oliver, "Combined task- and network-level scheduling for distributed time-triggered systems," *Real-Time Systems Journal*, vol. 52, Mar. 2016.

[5] N. Finn, "Introduction to Time-Sensitive Networking," *IEEE Communications Standards Magazine*, vol. 2, pp. 22–28, JUNE 2018.

[6] "IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks - Redline," *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014) - Redline*, pp. 1–3654, July 2018.

[7] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE Symposium on Security and Privacy*, pp. 305–316, May 2010.

[8] W. Steiner, "An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks," in *IEEE Real-Time Systems Symposium*, Nov 2010.

[9] R. Serna Oliver, S. S. Craciunas, and W. Steiner, "IEEE 802.1Qbv Gate Control List Synthesis Using Array Theory Encoding," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 13–24, April 2018.

[10] R. Ernst, "Automotive ethernet: Opportunities and pitfalls," Emerging Technologies for Factory Automation (ETFA) conference, 2016 Keynote. https://www.etfa2016.org/images/keynote/keynote-ernst.pdf.

[11] "IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks – Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements," *IEEE Std 802.1Qcc-2018 (Amendment to IEEE Std 802.1Q-2018 as amended by IEEE Std 802.1Qcp-2018)*, pp. 1–208, Oct 2018.

[12] M. Jethanandani, "Yang, netconf, restconf: What is this all about and how is it used for multi-layer networks," in *2017 Optical Fiber Communications Conference and Exhibition (OFC)*, pp. 1–65, March 2017.

[13] D. Valencic and V. Mateljan, "Implementation of netconf protocol," in *2019 42nd International Convention on Information and Communication Technology, Electronics*

and Microelectronics (MIPRO), pp. 421–430, May 2019.

[14] C. Callegari, L. Gazzarrini, S. Giordano, M. Pagano, and T. Pepe, "A Novel PCA-Based Network Anomaly Detection," in *2011 IEEE International Conference on Communications (ICC)*, pp. 1–5, June 2011.

[15] W. Steiner, "Synthesis of static communication schedules for mixed-criticality systems," in *2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, pp. 11–18, March 2011.

[16] S. S. Craciunas, R. Serna Oliver, and W. Steiner, "Formal Scheduling Constraints for Time-Sensitive Networks," *arXiv e-prints*, p. arXiv:1712.02246, Dec 2017.

[17] M. Gutiérrez, W. Steiner, R. Dobrin, and S. Punnekkat, "A configuration agent based on the time-triggered paradigm for real-time networks," in *2015 IEEE World Conference on Factory Communication Systems (WFCS)*, pp. 1–4, May 2015.

[18] M. L. Raagaard, P. Pop, M. Gutiérrez, and W. Steiner, "Runtime reconfiguration of time-sensitive networking (tsn) schedules for fog computing," in *2017 IEEE Fog World Congress (FWC)*, pp. 1–6, Oct 2017.

[19] N. Desai and S. Punnekkat, "Safety of fog-based industrial automation systems," in *Proceedings of the Workshop on Fog Computing and the IoT*, IoT-Fog '19, (New York, NY, USA), pp. 6–10, ACM, 2019.

[20] R. Dobrin, N. Desai, and S. Punnekkat, "On fault-tolerant scheduling of time sensitive networks," in *4th International Workshop on Security and Dependability of Critical Embedded Real-Time Systems (CERTS 2019)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[21] P. Pop, M. L. Raagaard, M. Gutierrez, and W. Steiner, "Enabling fog computing for industrial automation through time-sensitive networking (tsn)," *IEEE Communications Standards Magazine*, vol. 2, pp. 55–61, JUNE 2018.

[22] L. Lo Bello and W. Steiner, "A perspective on ieee time-sensitive networking for industrial communication and automation systems," *Proceedings of the IEEE*, vol. 107, pp. 1094–1120, June 2019.

[23] N. G. Nayak, F. Dürr, and K. Rothermel, "Time-sensitive software-defined network (tssdn) for real-time applications," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, RTNS '16, (New York, NY, USA), pp. 193–202, ACM, 2016.

[24] T. Gerhard, T. Kobzan, I. Blöcher, and M. Hendel, "Software-defined Flow Reservation: Configuring IEEE 802.1Q Time-Sensitive Networks by the Use of Software-Defined Networking," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 216–223, Sep. 2019.

[25] T. Hackel, P. Meyer, F. Korf, and T. C. Schmidt, "Software-defined networks supporting time-sensitive in-vehicular communication," in *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, pp. 1–5, April 2019.

[26] S. Jha, S. Banerjee, T. Tsai, S. K. S. Hari, M. B. Sullivan, Z. T. Kalbarczyk, S. W. Keckler, and R. K. Iyer, "ML-Based Fault Injection for Autonomous Vehicles: A Case for Bayesian Fault Injection," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 112–124, June 2019.

[27] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete Problems in AI Safety," 2016.

[28] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," 2016.

[29] A. Biondi, F. Nesti, G. Cicero, D. Casini, and G. Buttazzo, "A safe, secure, and predictable software architecture for deep learning in safety-critical systems," *IEEE Embedded Systems Letters*, pp. 1–1, 2019.

[30] S. A. Seshia, D. Sadigh, and S. S. Sastry, "Towards verified artificial intelligence https://arxiv.org/abs/1606.08514," 2016.