SmartNIC-based Load Management and Network Health Monitoring for Time Sensitive Applications

Kilian Holzinger*, Franz Biersack*, Henning Stubbe*, Angela Gonzalez Mariño‡, Abdoul Kane‡
Francesc Fons‡, Zhang Haigang§, Thomas Wild*, Andreas Herkersdorf†, Georg Carle†
Technical University of Munich, Huawei Technologies Düsseldorf GmbH
*firstname.lastname@tum.de, †lastname@tum.de, ‡firstname.lastname@huawei.com, §zanghaigang@huawei.com

***Abstract*—Time sensitive network applications, for example in Intra-Vehicular Networks, aim to give predictable end-to-end latency guarantees. As a consequence, processing resources of involved host systems remain partially unused, because they are reserved for rare worst cases. This circumstance provides the opportunity to reduce dimensioning overheads by managing the load on the nodes flexibly within the network. In our proposed approach, a SmartNIC involving an FPGA-based load balancer achieves dynamic routing of flows whilst preserving end-to-end latency guarantees. A flow-oriented online network measurement component continuously supervises network traffic with regards to compliance to flow specifications and constraints such as bounded one-way delay, absence of packet loss, and jitter. We use the supervisor to enhance forwarding decisions on the data plane. Initial evaluation yields a saving potential of around 30 %. We showcase quick dynamic reconfiguration of the FPGA when triggered by real-time measurement of the one-way delay using realistic automotive network traffic.**

***Index Terms*—SmartNIC, automotive, load balancing, monitoring, networks, vehicular networks, time sensitive networks**

I. Introduction

Currently, there is a trend towards increasingly complex time-sensitive distributed applications in many areas such as Advanced Driver's Assistance Systems (ADASs) or autonomous automotive systems in Intra-Vehicular Networks (IVNs). The latter requiring self-awareness, which includes supervising their run-time timing behavior and consecutive adaption of system configuration in case of deviations from modelled properties [1]. The complexity in this application domain comes from the number of flows with heterogeneous reliability, timing, and high bandwidth requirements as well as the interactions of the various system layers. Approaches to meet these mixed-critical requirements include over-provisioning of resources and introduction of redundancy so that in worst cases the functionality still is assured. On the other hand, computation resources remain idle during normal operation.

Here, a SmartNIC architecture is proposed to reduce this idle overhead by dynamically (re-)assigning network traffic to a pool of available local and remote processing resources, using a hardware-based load balancer, coupled with a network monitoring software. We refer to the load balancer as the Load Management Layer (LML). It performs a forwarding table lookup for every incoming packet, and uses the result, along with a set of locally stored host and flow state parameters, to perform a resource assignment algorithm in order to direct packets to local or remote processing resources. The flow states are obtained through passive per-packet network timing measurement component, referred to as Network Health Monitoring (NHM). Per-CPU core queue fill levels are made available to the LML. A set of Key Performance Indicators (KPIs) and threshold values capture many flow-specific constraints such as one-way delay, jitter, and target packet inter-arrival time. In case of detected impairments of flows, e.g., queue fill levels exceeding or falling below a certain threshold, the LML configuration is updated to take the current network and host state into account. Thus, the system can address a wide range of problems such as failures and timing violations. We showcase specifically how the measurement of experienced packet delay can be incorporated into load balancing decisions.

SmartNICs, based on a combination of ASICs, FPGAs, special purpose processing units, and software, are network interface cards (NICs) with additional functionality, and programmability [2]. For the architecture, we chose a hybrid approach consisting of a hardware-based data plane to achieve fast and deterministic behavior and software using the Data Plane Development Kit (DPDK) for the monitoring functionality. A prototype implementation of the hardware component is currently being developed using an FPGA-based development platform. An ASIC implementation can be considered for product-grade development. In addition, design space exploration is done using a network simulation environment. Usually time-sensitive application domains require custom-designed hardware, such as IVN gateways. So, we anticipate that saved processing resources on host systems overcompensate the increased complexity in the network.

In this publication, the following contributions are presented: (1) design, implementation, and evaluation of the NHM; (2) design, implementation, and evaluation of the LML; (3) assessment of NHM coupled with LML in an example IVN scenario. Artifacts are available online [3].

Remainder of this paper is structured as follows. Section II discusses related work. Afterward, Section III touches on IVNs and requirements. Section IV elaborates on the design of our approach. Finally, we evaluate our approach in Section V. Section VI concludes this paper.

II. Related Work

This work is related to publications addressing fail-over mechanisms, SmartNICs, and monitoring in time-sensitive

978-1-6654-0601-7/22/$31.00

domains as well as load balancing.

a) Fail-Over Mechanisms: [4] suggest a method for fast fail-over in Ethernet-based networks. Switches detect link or port failures and report them to a central controller. The exact functionality of the monitoring is not detailed. Based on the identified error mode, the controller instructs reconfiguration of data plane devices to re-establish network connectivity. The authors argue that the functionality should be implemented in a performance-optimized form. The evaluation of the approach yielded a reconfiguration latency of 50 ms using a software implementation as a demonstrator. A simulated hardware deployment achieved 1 ms.

A proposal by [5] discusses delegation of control plane functionalities of Time-Sensitive Networking (TSN) domains to local network elements. Nodes can detect failures or performance degradations of flows in the network. The receiving node can mitigate the problem by informing nodes on an alternative pre-computed backup path between sender and receiver. As a consequence, the considered Layer 3 routes are updated on nodes. The alternative configurations are distributed by a central controller as finite state machines, which take the error condition as input. The exact functionality of the failure detection is not described. In an experimental evaluation of the approach, the recovery time was 254 ms in the best case.

Table I summarizes the comparison of the considered fail-over mechanisms to this publication.

b) Critical Network Traffic Monitoring: Multiple publications address the problem of flow-level monitoring [6], [7].

[8] presents a conformity check of IVN traffic and a method to distribute monitoring workload across the network. Thus, a communication pattern can be assessed in respect to predefined constraints. Anomalies that alter the frequency of transmitted packets can be detected. The proposed method, however, does not directly address reliability and time requirements of flows.

[9] describes a real-time monitoring solution for time-sensitive event chains, however, the solution is not directly applicable to the network layer. The implementation is tightly integrated into the Robot Operating System which has a communication model based on message passing. The authors consider embedded Precision Time Protocol (PTP) synchronized timestamps for measuring propagation delays.

One-way delay is an important metric for time-sensitive communications. The problem of passively measuring it is addressed in [10], [11].

An initial design of the NHM was already presented in [12].

c) SmartNICs for Time Sensitive Domains: The vast majority of reported SmartNIC use-cases focus on data center applications. Still, there is desirable functionality of a SmartNIC presented for an IVN by [13]. The authors propose a solution, which uses an encapsulation protocol to transport additional state information, timestamps, and control data.

d) Load Management Layer (LML): Research regarding implementations of load management is mostly focused on data center networks [14]–[16]. Maglev [15] and Stratos [16] are techniques to evenly distribute traffic across a range of computing instances. Maglev is a software-based load balancer trying to match incoming packets to virtual IP addresses before forwarding them to a corresponding service back-end. Stratos is an orchestration layer for virtual middle-boxes. It collects metrics like the packet processing time to steer packets through a chain of middle-boxes. Both techniques are operating at bigger timescales while we are focusing on efficient resource utilization in the milli- and microsecond range.

Similar techniques are applied to distribute intra-node workloads such as Receive Side Scaling (RSS) [14] or extensions thereof [17]. Proposed solutions also include Shenango [18] or Shinjuku [19], both of which represent software-based solutions instead of a hardware-based mechanism located at the NIC. They are dedicating one [18] or more [19] CPU cores for the distribution of packets to worker CPU cores. [20] presents a combination of intra- and inter-node load balancing on which our solution is based. It distributes workloads across local CPU cores on the granularity of bursts of flow bundles, and can also offload flow bundle bursts to neighboring servers once all local resources are highly loaded.

III. Background and Requirements

IVNs interconnect applications of mixed-criticality. Some of them have real-time requirements, demanding low and deterministic application latencies. For these applications, a reliable packet-based communication is crucial to safety-relevant tasks. Currently, more and more vehicles are equipped with Ethernet-based on-board networks [21]. The special requirements of IVNs for time-critical and deterministic communication can be met using TSN.

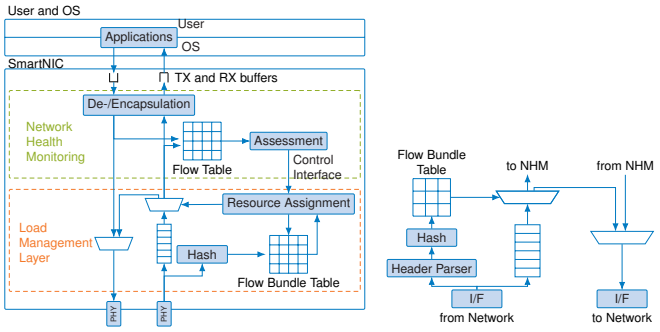
To reflect peculiarities of IVNs and improve load distribution and network performance the SmartNIC solution needs to fulfil a set of requirements which are listed in Table I. They are used as guidance for the design in Section IV and as comparison criteria to related work.

IV. Design

Modern IVNs employ a zonal model, which creates a mesh-like topology where each node is connected to several neighboring nodes [24]. The SmartNIC is connected to nodes, also referred to as host systems, and switches which integrate

	[4]	[5]	this
R1: ***Integration into IVN TSN data plane*** **[21]**	✓	✓	✓
R2: ***Reduction of end-to-end application latency***	×	×	✓
R3: ***Recovery of safety-relevant functions within 100 ms compliant with AVNU requirements*** **[22]**	✓	×	✓
R4: ***Fine-grained monitoring of flow requirements*** **to enable self-awareness of the network stack [1]**	×	×	✓
R5: ***Low overhead*** **of additionally required resources**	✓	✓	✓
R6: ***Simple integration into existing architectures***	✓	✓	✓
R7: ***Realization of functionality on the data-link layer*** **for IVN protocols such as [23]**	✓	×	✓
R8: ***Integration of cross-layer concepts*** **[1], e.g., network changes are caused by host system load**	×	×	✓
R9: ***Fail-over without direct control plane interference*****, as requests towards control plane entail delay**	×	✓	✓
R10: ***Evaluation w.r.t functionality and performance***	✓	✓	✓

Table I: Comparison to related work and requirements



(a) Functional BD of the SmartNIC (b) BD of the LML

Figure 1: Block diagrams (BD)

into the zonal topology. TSN support is enabled by the networking stack of the host system and compatible switches (**R1**). Both, the monitoring and load balancing functionality is meant to be deployed on a SmartNIC device, and, as such, easy to integrate into existing architectures (**R6**). Although there are SmartNICs available enabling deployment of DPDK applications, e.g., [25], current NHM prototype is running on the host system. The LML is developed on an FPGA platform. Figure 1a depicts the functional architecture of the SmartNIC. The device has one duplex network connection.

We consider all zonal compute nodes in the IVN to be equipped with a SmartNIC. The feature set is split between the LML and NHM. The SmartNIC's LML component either forwards incoming packets to the host or offloads them to another destination, contributing to fast recovery of functionality (**R3**) and lower end-to-end application latency (**R2**). There is no direct interaction with a control plane as it would entail additional latency (**R9**). It parses the header fields of every packet and then accesses a *Flow Bundle Table*, where the current resource assignment is stored. A subsequent *Resource Assignment* algorithm decides whether to keep the current assignment or to redirect packets before updating the table. In order to choose an alternative processing resource, the current load state of every CPU core is stored in the LML and can be updated via the *Control Interface* between the NHM and the LML. The mechanism operates on Layer 2 (**R7**).

The NHM passively monitors flows with the objective to detect violations of timing and reliability requirements, e.g., increased delay, using KPIs (**R4**). There are rules to transform flow KPIs to dynamic threshold values of node utilization. In case the node-local resources are too busy to meet end-to-end deadlines of the underlying network application, the LML redirects the flow, based on flow match, queue utilization, and the dynamic threshold value, to an other node where likely the deadline still can be met. As a result, the network becomes self-aware and incorporates cross-layer concepts, in this case, the host load state (**R8**).

The realization of the forwarding and load balancing component in hardware introduces only a low and deterministic additional latency. The monitoring operates passively and does not directly interfere with by-passing traffic (**R5**). The evaluation results, obtained with simulation and measurements, are shown in Section V (**R10**).

A. Load Management Layer (LML)

The LML is a hardware-based data plane component capable of determining an assignment to available computing resources at line-rate for every packet entering the node. The most important components are depicted in Figure 1b.

Upon entering the LML, packets are first parsed to extract relevant data from protocol header fields. These header fields are then used to determine the flow the packet belongs to. Individual flows are defined by their IP destination and source address, the Layer 4 destinations and source ports as well as the protocol type. In addition, a hash value is calculated over these extracted header fields.

For each assignment, an internal forwarding table is consulted. A given number of Least Significant Bits (LSBs) of the aforementioned hash result acts as an index for the table. As a result, this mechanism maps the lookup for multiple different flows to a common table entry, effectively bundling flows and performing resource assignments as well as offloading decisions on a per flow bundle basis.

The lookup yields the current assignment of the corresponding flow bundle to either one of the local CPU cores or a neighboring node in the network. As long as local CPU cores within the compute node provide sufficient capacity, packets belonging to a given flow bundle maintain their assignment.

Due to load imbalances and peaks in computing demand, queues of packets yet to be processed can fill up and may result in queue lengths exceeding threshold values. To prevent processing resources from becoming overloaded and, hence, packets from suffering high queuing latencies, the current load state (packet queue threshold exceeded or not) of every core on the host system is communicated to the LML via the *Control Interface* between the NHM and LML. If local CPU cores show the tendency to become highly loaded, they will be marked as such in the LML. Packets belonging to flow bundles which, according to the table lookup, shall be forwarded to highly loaded CPU cores can then proactively be reassigned to less loaded CPU cores. Should all local CPU cores be highly loaded, flow bundles can also be offloaded to a neighbor node in the network, given that these nodes provide the functionality the affected flows require. Based on the hash result of a packet, a resource assignment algorithm will first search for an eligible local CPU core, or pick a neighboring node if all cores are highly loaded.

To prevent packet reordering, offloading or reassignment cannot take place for every packet. Sending two subsequent packets of the same flow bundle along two different paths might expose them to different forwarding delays and, hence, lead to packets arriving at their destination in a different order than they left their source. A way to mitigate this problem, as shown by [17], is to allow changes in the path packets are sent along only if no packet of the corresponding flow bundle has been observed for a given time. [17] define a timeout value δ, and consider all packets of the same flow whose Intra-Arrival

Time (IAT) is smaller than δ to belong to the same packet burst, or "flowlet". We set δ to the largest difference in latency two different paths packets can take and allow reassigning or offloading flow bundles only if this timeout value has expired for the corresponding bundle. To enable this, every entry of the flow bundle table also holds the time of when the last packet of the same flow bundle arrived. After each lookup, the corresponding table entry is updated with most recent packet arrival time and currently assigned processing resource.

In case of offloading, a packet is sent to a neighboring node, depending on the hash value calculated after parsing its header. It is then forwarded via the egress network interface.

B. Network Health Monitoring (NHM)

Flows in IVNs have end-to-end timing requirements such as one-way delay, which stem from the underlying applications. While deterministic network configurations are available with TSN, there are complex interactions of many involved layers, e.g., software stacks, to be considered. As an additional way to improve network performance and to react to unforeseen impairments we add a monitoring component to the automotive SmartNIC and use it to control the LML.

The NHM component supervises network traffic on a per-packet and per-flow granularity. It is able to calculate flow-specific KPIs which capture the current behavior of a flow w.r.t a specification or common metrics of interest, e.g., one-way delay. KPI-specific threshold values are used to assess whether the flow is performing in compliance with the requirements. In the case of an IVN, we are able to assess safety relevant properties of network applications such as packet-loss, one-way delay, and IAT in real-time. In this publication, we focus on the one-way delay metric to use the flow assessment result in conjunction with the load balancer.

Main logical functions of the NHM are depicted in the *Network Health Monitoring* component of Figure 1a. In general, when a new packet is received from the network and the LML, an high-precision receive timestamp is taken using capabilities of the network hardware. Then a flow identifier is calculated, e.g., based on a TSN stream identifier or a tuple consisting of IP addresses and transport protocol ports. The implementation is parallelized using RSS [26]. The *Flow Table*, a concurrent hash table, stores state information for each flow. Where the flow identifier acts as a key and the value is the flow state needed to calculate stateful KPIs. The flow state is updated on each packet. Afterwards an *Assessment* logic is executed which accesses the state information of each flow and calculates a set of KPIs. Using a rule-based decision algorithm the LML configuration is updated through the *Control Interface*.

One relevant KPI, for example, is the one-way delay of a packet which is the difference between send and receive time. The receive time is captured by the NHM. We consider three ways to obtain the send time of a packet: (1) In IVNs periodic flows are prevalent. Those are flows which have a cyclic sending pattern, as it corresponds, e.g., to a sensor value update. We can leverage observed changes in IAT and, thus, calculate changes in delay. (2) Send time can be obtained through parsing timestamps embedded in application protocols such as Audio Video Transport Protocol (AVTP) or Real-Time Transport Protocol (RTP) which are common in IVNs. (3) Send timestamps can be embedded into custom encapsulation protocols. This approach has been discussed, e.g., in conjunction with Network Service Headers (NSH) [27]. In this case, the SmartNIC performs the *De-/Encapsulation* transparently for the host system.

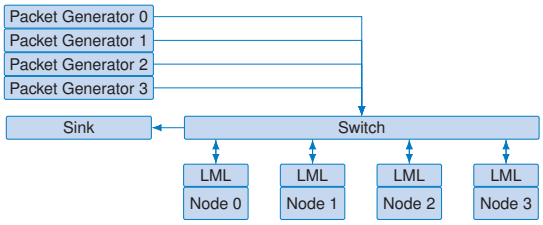


Figure 2: Network simulation model

Options (2) and (3) require synchronized clocks to yield valuable results which can be achieved using PTP. A viable approach, since the TSN profile for IVNs mandates its use anyway [28] and reportedly the synchronization error in many cases is below 1 µs [29]. Using embedded transmit timestamps for one-way delay measurements has been described in [10].

V. Results and Discussion

In this section the results of the initial evaluation of the simulation model and the prototype are presented. At first the suitability of the LML to save processing resources is shown using a simulative approach. In this case, NHM is inactive. Internet backbone traffic from CAIDA [30] was used to show the effect with a high number of flows and packet rates.

An example one-way delay measurement of a flow in an IVN using the NHM prototype is presented next. Finally the behavior of the FPGA implementation is shown when control messages are sent by the NHM. For those measurements traffic traces of individual flows were generated synthetically using scripts. IVN flow properties can thus be obtained which were described in [24], [31]. Here, a flow is generated resembling an automotive camera system generating 60 packet bursts per second, corresponding to camera frames. We set the tolerable one-way delay to 10 ms, lower than requirements in the draft TSN automotive profile [32].

A. Load Management Layer (LML)

We evaluate the performance of the LML using a C++-based network simulation model as shown in Figure 2. The modelled IVN consists of four compute nodes. As mentioned, each compute node's SmartNIC entails a switch to establish a direct connection among compute nodes. Hence, for our simulation, we employ the "one big switch" abstraction, as these switches can be considered to be a single switch.

In our model, four packet generators, each replaying a PCAP-based packet trace of 15 s at 10 Gbit/s, inject the packets to be processed. These packets correspond to traffic emitted by various sensors and actuators in the modelled IVN. The

# instructions per	packet header	payload byte
90 % traffic	4500	0
10 % traffic	1250	50

Table II: Distribution of required instructions per packet (IPP)

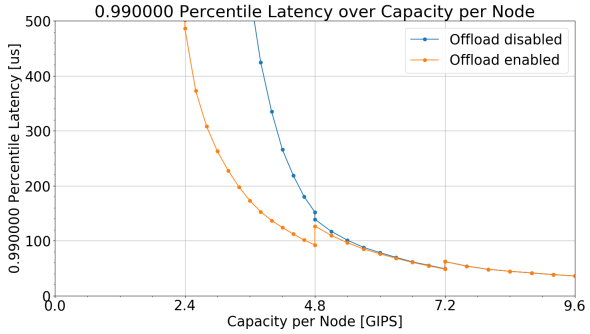


Figure 3: 99th percentile achievable end-to-end latencies

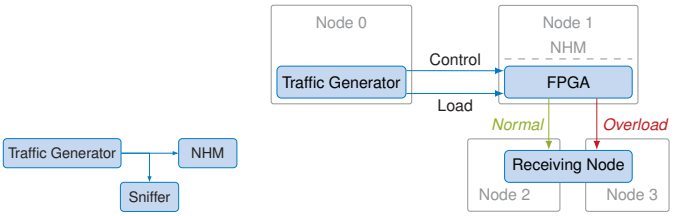
switching layer then distributes the flows across the compute nodes, where they will be processed.

For our setup, we assume that each compute node hosts a multi-core CPU running at a nominal clock frequency of 2.4 GHz. To save power, the frequency can be throttled down in 0.1 GHz steps. All CPU cores, however, are always running at the same clock frequency. We further assume an IPC (finished instructions per clock cycle) value for each CPU of 1, and that a given number of instructions per packet (IPP) must be performed. The distribution of IPPs to perform is shown in Table II and based on values from [33]. Combining variable numbers of available CPU cores with variable CPU processing frequencies yields a range of different processing capacities per node (Giga Instruction Per Second, GIPS) [20]. To acquire statistics, e.g., latency values, within the simulation framework, processed packets are sent to a sink module.

For comparison, we evaluate the 99th percentile of achievable end-to-end latency in two cases: In the first case, the LML of every SmartNIC is restricted to intra-node load balancing as described above. In the second case, the LML of every SmartNIC in addition is capable of offloading workloads to other nodes. Results are depicted in Figure 3. There, the blue curve represents the former case, while the orange curve represents the latter. For lower processing capacities, e.g., 2.4 GIPS to 4.8 GIPS, our approach can provide valuable resource savings. For example, when targeting a 99th percentile end-to-end latency of 300 µs, around 4.2 GIPS are necessary without offloading. In contrast, with offloading enabled, only 2.9 GIPS are required. In other words, around 30 % of compute resources can be saved for this configuration. Such savings decline towards higher processing capacities, since more CPU cores, running at high frequencies, are available. The reassignments of flows to other compute nodes become less frequent.

B. Network Health Monitoring (NHM)

For the evaluation of the NHM, we used the setup depicted in Figure 4a, inspired by [34]. It consists of three machines.



(a) NHM (b) Intra-SmartNIC functionality

Figure 4: Measurement setup used in evaluation experiments; signals are split using fiber taps

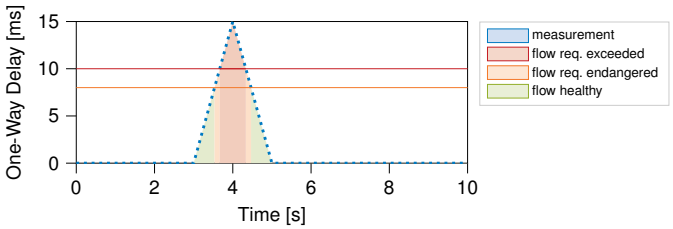


Figure 5: Assessed network delay by NHM; measurement error is not visible in plotted resolution.

NHM is the device under test. The *Traffic Generator* produces packets of an IVN camera flow. A *Sniffer* node is connected via passive fiber taps to record all traffic on the wire with high precision using hardware timestamping capabilities of Intel X552 NICs. The *Sniffer* is used to serve as a ground truth of the scheduled packet dispatch times at the traffic generator and the measured arrival timestamps at the NHM component.

The traffic is generated and provided to the NHM. An artificial increase and decrease of delay in the network was modelled in the generated traffic, which could be an effect from cross traffic in a real network. Here, case 3 from Section IV-B was used, so timestamps were embedded in an encapsulation protocol and the one-way delay can be obtained by comparing send and receive times. The implementation is capable to use hardware receive timestamps on the NIC with a resolution of 12.5 ns or to use a software fallback, which yielded a precision of about ±2 µs compared to timestamps taken at the *Sniffer*. Figure 5 shows the measured one-way delay for each packet over the experiment time. It matches the modelled properties of the flow in the plotted resolution and, thus, the results are viable to use for load balancing improvements. The signal colors mark warning levels which will affect decisions of the data plane configuration.

C. Intra-SmartNIC Functionality

Previous results highlight the benefits of the SmartNIC's individual components for IVN. But the feasibility of combining these components appropriately remains to be shown and, hence, is tackled subsequently. That is, we adapt the setup described in Figure 4a to Figure 4b. Plus, the *FPGA* is programmed to update its state based on control messages. Which means, depending on the configuration, received packets are forwarded via different interfaces. Note that, while commonly emitted by the *NHM*, in this setup, *Control* information is

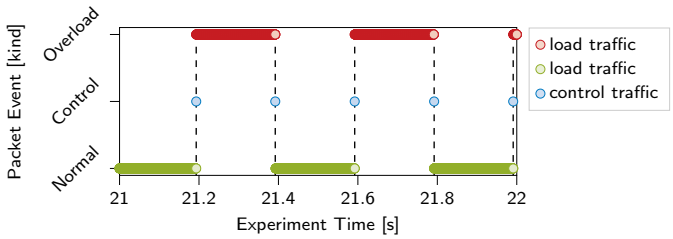


Figure 6: FPGA changes its state as instructed; communication between FPGA and NHM works

sent along with the automotive camera traffic as actual *Load* by the *Traffic Generator*, to ensure precise timing. Once forwarded by the FPGA, traffic is received by a *Receiving Node* representing the SmartNIC host system for the *Normal* operation state and a neighbor node for the *Overload* operation state. During the experiment, packets are recorded by the *Receiving Node*. Initial evaluation, cf. Figure 6, shows that the FPGA immediately adheres to control message instructions. Its state is updated and subsequent packets are processed following the new rules. Additional experiments, not included due to space constraints, indicate that the FPGA does not alter the shape of forwarded traffic in throughput or jitter. Jointly, results imply that NHM and FPGA can be combined and, thus, benefit from enhancements of the individual components.

VI. Conclusion

A set of requirements for load management and network monitoring for time-sensitive IVNs was defined, which are not met by identified related work. Thus, a novel concept of a custom SmartNIC solution was elaborated and evaluated. It can be deployed in TSN environments and assures vehicle safety by supervising time-critical flow properties and recovers or mitigates requirement violations in real-time. The two main components are realized as prototypes: The NHM as software and the LML as an FPGA implementation. The monitoring is able to assess network traffic in real-time w.r.t. many relevant KPIs. It is combined with a load balancer which helps to reduce end-to-end latency, as shown using network simulation. Using one-way delay as an example flow requirement, the feasibility of the approach to trigger configuration changes in the FPGA was assessed. As a consequence, available compute resources can be better utilized and, thanks to a more dynamic workload distribution, less over-provisioning is required.

References

[1] Schlatow et al., "Self-awareness in autonomous automotive systems," in *Design, Automation & Test in Europe Conference & Exhibition*, 2017.

[2] Deierling, "What Is a SmartNIC," 2018. [Online]. Available: https://blog.mellanox.com/2018/08/defining-smartnic/

[3] "Prototype, Measurement, and Result Artifacts." [Online]. Available: https://github.com/tumi8/NOMS-ITAVT22

[4] Kostrzewa et al., "Fast Failover in Ethernet-Based Automotive Networks," in *23rd International Symposium on Real-Time Distributed Computing*. IEEE, 2020.

[5] Sambo et al., "Enabling Delegation of Control Plane Functionalities for Time Sensitive Networks," *IEEE Access*, 2021.

[6] Zhang et al., "Flowatcher-DPDK: Lightweight line-rate flow-level monitoring in software," *IEEE Transactions on Network and Service Management*, 2019.

[7] Emmerich et al., "Efficient dynamic flow tracking for packet analyzers," in *7th International Conference on Cloud Networking*. IEEE, 2018.

[8] Waszecki et al., "Automotive electrical and electronic architecture security via distributed in-vehicle traffic monitoring," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[9] Peeck et al., "Online latency monitoring of time-sensitive event chains in ROS2," 2021.

[10] Shin et al., "Clock synchronization for one-way delay measurement: A survey," in *Advanced Communication and Networking*, 2011.

[11] Chefrour, "One-way delay measurement from traditional networks to SDN: A survey," *ACM Comput. Surv.*, 2021.

[12] Holzinger et al., "Precise real-time monitoring of time-critical flows," in *Proc. of the 17th International Conference on Emerging Networking EXperiments and Technologies*, 2021.

[13] Shreejith et al., "Smart network interfaces for advanced automotive applications," *IEEE Micro*, 2018.

[14] Niu et al., "PostMan: Rapidly Mitigating Bursty Traffic via On-demand Offloading of Packet Processing," *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[15] Eisenbud et al., "Maglev: A fast and reliable software network load balancer," in *Proc. 13th USENIX Symp. on Networked Systems Design and Implementation*, 2016.

[16] Gember et al., "Stratos: A network-aware orchestration layer for virtual middleboxes in clouds," *arXiv:1305.0209*, 2013.

[17] Kandula et al., "Dynamic load balancing without packet reordering," *SIGCOMM Comput. Commun. Rev.*, 2007.

[18] Ousterhout et al., "Shenango: Achieving high CPU efficiency for latency-sensitive datacenter workloads," in *Proc. 16th USENIX Symp. on Networked Systems Design and Implementation*, 2019.

[19] Kaffes et al., "Shinjuku: Preemptive scheduling for μsecond-scale tail latency," in *Proc. 16th USENIX Symp. on Networked Systems Design and Implementation*, 2019.

[20] Oeldemann et al., "Inter-Server RSS: Extending Receive Side Scaling for Inter-Server Workload Distribution," in *28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*.

[21] Bello et al., "Recent Advances and Trends in On-Board Embedded and Networked Automotive Systems," *IEEE Transactions on Industrial Informatics*, 2019.

[22] Takeuchi et al., "Requirements for automotive AVB system profiles," *Whitepaper, Knorrstrasse*, 2011.

[23] "IEEE Standard for a Transport Protocol for Time-Sensitive Applications in Bridged Local Area Networks," *IEEE Std 1722-2016 (Revision of IEEE Std 1722-2011)*, 2016.

[24] Pannell et al., "Use cases - IEEE P802.1DG V0.4," https://www.ieee802.org/1/files/public/docs2019/dg-pannell-automotive-use-cases-0919-v04.pdf, 2019.

[25] DPDK Contributors. Data Plane Development Kit – Mellanox BlueField Board Support Packages. [Online]. Available: http://doc.dpdk.org/guides/platform/bluefield.html

[26] "Receive Side Scaling (RSS)." [Online]. Available: https://docs.microsoft.com/en-us/windows-hardware/drivers/network/ndis-receive-side-scaling2

[27] P. Quinn, U. Elzur, and C. Pignataro, "Network Service Header (NSH)," RFC 8300, 2018. [Online]. Available: https://tools.ietf.org/html/rfc8300

[28] (2021) P802.1DG – TSN profile for automotive in-vehicle ethernet communications. [Online]. Available: https://1.ieee802.org/tsn/802-1dg/

[29] "IEEE standard for a precision clock synchronization protocol for networked measurement and control systems - redline," *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002) - Redline*, 2008.

[30] "The CAIDA UCSD Anonymized Internet Traces - 2019-01-17 nyc (dirA)." [Online]. Available: http://www.caida.org/data/passive/passive_dataset.xml

[31] Migge et al., "Insights on the Performance and Configuration of AVB and TSN in Automotive Ethernet Networks," *Embedded Real-Time Software and Systems*, 2018.

[32] Gopal. (2021) Text Proposal for section 6: In-vehicle network topology, 9: Traffic Separation and 11: Latency and congestion loss in IEEE 802.1DG/D1.3. [Online]. Available: https://www.ieee802.org/1/files/public/docs2021/dg-gopal-TrafficClass-text-1021-v02.pdf

[33] Ramaswamy et al., "Analysis of network processing workloads," *Journal of Systems Architecture*, 2009.

[34] Gallenmüller et al., "5G QoS: Impact of Security Functions on Latency," in *IEEE/IFIP Network Operations and Management Symposium*, 2020.