Priority Based Ethernet Handling in Real-Time End System with Ethernet Controller Filtering

Bjarne Johansson[1,2], Mats Rågberger[1], Thomas Nolte[2], Alessandro V. Papadopoulos[2]
[1] ABB Process Automation, Process Control Platform, Västerås, Sweden
[2] Mälardalen University, Västerås, Sweden
{bjarne.johansson, mats.ragberger}@se.abb.com,
{thomas.nolte, alessandro.papadopoulos}@mdu.se

***Abstract*—This work addresses the impact of best-effort traffic on network-dependent real-time functions in distributed control systems. Motivated by the increased Ethernet use in real-time dependent domains, such as the automation industry, a growth driven by Industry 4.0, interconnectivity desires, and data thirst. Ethernet allows different network-based functions to converge on one physical network infrastructure. In the automation domain, converged networks imply that functions with different criticality and real-time requirements coexist and share the same physical resources. The IEEE 60802 Time-Sensitive Networking profile for Industrial Automation targets the automation industry and addresses Ethernet network determinism on converged networks. However, the profile is still in the draft stage at the time of writing this paper. Meanwhile, Ethernet already provides attributes utilized by network equipment to prioritize time-critical communication. This paper shows that Ethernet Controller filtering with prioritized processing is a prominent solution for preserving real-time guarantees while supporting best-effort traffic. A solution capable of eliminating all best-effort traffic interference in the real-time application is exemplified and evaluated on a VxWorks system.**

I. Introduction

Distributed Control Systems (DCS) are transcending into the Industry 4.0 era, where data and information are valuable optimization-enabling assets. Data collection requires connectivity and communication, pushing the automation industry towards network-centric solutions, implying that the network, to some extent, replaces the controller as the information center of the control system. Following in the tracks of network-centric systems are increased interest for interconnectivity and interoperability, key concepts in the Open Process Automation™ Standard[1] (O-PAS). O-PAS prescribes OPC-UA[2] as the interoperable communication standard.

Upper bound end-to-end communication time of real-time traffic is a challenge for converged Operation Technology (OT) Ethernet networks when competing for network resources against low-priority, best-effort traffic induced by noncritical functions. A challenge addressed by Time Sensitive Networking (TSN) amendments to the IEEE 802.1Q Ethernet networking standard. The IEEE 802.1Qbv amendment for Scheduled Traffic (TSN-ST) brings forth short and bounded end-to-end communication time on converged networks using time-scheduled communication. TSN offers solutions but is still in the early stages of industry adoption. For example, the Industrial Automation IEEE 60802 TSN profile[3] is at the time of writing this paper still in the draft stage, and the support in industrial network equipment is scarce [1].

Ethernet networking does not require TSN to provide Quality of Service (QoS). The Priority Code Point (PCP), introduced in the late '90s in the IEEE 802.1D-1998 and later incorporated in IEEE 802.1Q, already offers that. PCP provides priority information utilizable by the OSI layer two network infrastructure (i.e., switches) to determine forwarding precedence.

In this paper, we address the challenge of preserving the correctness of network-dependent real-time functions in end systems when coexisting with non-real-time programs reliant on best-effort traffic. We illustrate the problem with a simulated DCN application running on VxWorks. A DCN application with OPC UA PubSub-based real-time communication dependencies. We identify prioritized frame processing aided by Ethernet Controller filtering as a prominent solution, which is the paper's contribution, together with the verification of the solution.

The paper is organized as follows. Section II presents the related work. Section III describes the filtered enabled prioritized frame processing, followed by a quantitative evaluation in Section IV, and a discussion in Section V. Section VI outlines conclusions and future work.

II. Related work

TSN combined with OPC UA is identified by Bruckner et al. [2] as the future of communication in the automation domain. TSN consists of multiple amendments to IEEE 802.1 [3] and Lo Bello et al. [4] provide an overview and discuss open issues from an automation mindset, where configuration ease is one of the highlighted challenges. As mentioned in the introduction and shown, for example, by Zhao et al. [5], TSN-ST provides low latency, deterministic end-to-end communication, but the scheduling problem is not a trivial problem [6], [7]. Hallmans et al. [1] highlight that industry TSN adoption is still

This work is funded by The Knowledge Foundation (KKS), projects ARRAY and SACSys, and by The Swedish Foundation for Strategic Research (SSF), project FuturAS.

[1]https://publications.opengroup.org/p190

[2]https://opcfoundation.org/

[3]https://1.ieee802.org/tsn/iec-ieee-60802/

low, and the support provided by industrial network equipment is still scarce, motivating why we do not consider TSN in the end system handling in this work.

Already in 2002, after the introduction of PCP, end system handling of prioritized traffic was addressed by Skeie et al. [8] with software-based priority queue to reduce latency, and the need for QoS in the end system highlighted by Thyrbom et al. [9]. Since then, many studies have been made on network stack performance on Linux. For example, Larsen et al. [10] studied TCP latency on Linux in a data center environment and found that the latency is around $10\mu s$ in point-to-point communication.

Beifuß et al. [11] measures latency and constructs a model to predict latency in the Linux network stack to find optimization knobs to turn. Describing four main optimization points: (i) copying between user- and kernel space, (ii) usage of preallocated buffer, (iii) polling to reduce interrupt frequency, and (iv) processing of batches instead of single frames. The performance study performed by Ramneek et al. [12] reaches a similar conclusion: buffer allocation and interrupt handling can be expensive in terms of CPU usage.

Priority inversion due to processing of low priority packets with high priority can potentially impact high priority real-time execution badly [13]. A challenge addressed by Lee et al. [14] and further improved by Blumschein et al. [15]. Both use an early software demuxer to classify packages and prioritize the handling of the incoming packet according to the priority of the receiving task. To further aid that approach, Behnke et al. propose a multi-queue network interface [16]. In contrast, our work focuses on unmodified network stacks and presents Ethernet Controller enabled filtering to enable processing priority matching the received frame's QoS.

III. Prioritization filtering

The Ethernet Controller (EC) also known as Media Access Controller (MAC) handles, with the Physical Layer (PHY), the transmission and reception of Ethernet Frames. When a frame is received, the EC stores the frame in a RxQueue and raises an interrupt to the CPU, and the OS Network stack process the frame further. However, as pointed out by earlier work, the network stacks do not, by default, treat the incoming frames according to priority, which may result in priority inversion or latency [14], [16].

ECs provide filtering options and the possibility to direct traffic to different Rx queues based on those filtering options. In this section, we base our EC examples on Intel I211[4], which has PCP filtering capabilities and two Rx queues. Other ECs, such as Intel I350, have eight Rx queues, allowing even better filtering granularity. Different ECs also support different filtering possibilities, from Ethernet header information to filtering on the higher protocol-layer information, typically IP-header information. We denote the configurable properties on which to take the filter decision, the filtering property (FP).

Figure 1 summarizes the idea, elaborated next. Based on the FP, the EC determines the frame priority. The EC queue high priority frames on the high priority Rx queue and raises the corresponding interrupt. The high priority ISR post the request to serve the incoming frame to the high priority job queue served by the high priority network task. Low-priority frames are handled similarly but follow the low-priority path. If the low priority task cannot read frames from the low priority Rx queue faster than they arrive, the low priority queue will eventually become full. It is then essential that the EC drop low priority frames that can't fit into the low priority queue to avoid filling the RxFIFO and causing high priority frame drops.

The principle described can also handle multiple network interfaces, where the filtering can ensure that the most suitable network task, priority vise, processes incoming frames.

A. QoS prioritization filtering on VxWorks

VxWorks[5] is a widely used and well-known commercial RTOS that has been around for more than 35 years and installed more than two billion times. The solution was also tested on Linux but left out due to page limitations.

Enabling priority packet filtering in VxWorks requires two things, enabling driver support and providing two network tasks with different priorities. The driver support added consists of (i) support for multiple Rx queues and (ii) filter configuration support. The filtering configuration of the EC also configures the EC to drop frames if the destination Rx Queue is full to prevent the RxFifo, from filling up and causing high priority frame drop due to the Rx Queue for low priority frames being full.

Two network tasks with different priorities are configured, denoted $NetTaskHP$ and $NetTaskLP$. The $NetTaskHP$ is the high-priority network task, given a priority of 20. The $NetTaskLP$ is the low priority network task and has priority 50. The priority values given here are the ones used in the evaluation system, described in section IV-C.

IV. Quantitative evaluation

A. Evaluation setup

The evaluation setup consists of the four nodes, listed in Table I. Table II lists the software. C1 runs the Evaluation

[4]https://cdrdv2.intel.com/v1/dl/getContent/333017

[5]https://www.windriver.com/products/vxworks

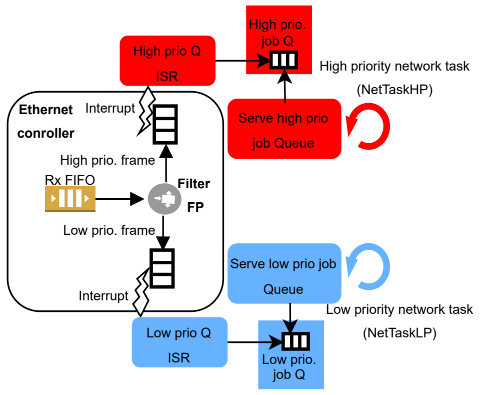

Fig. 1: Priority based filtering - an example with two priorities, high and low.

Application (EA), explained in Section IV-C. C3 and C4 are nodes addressing C1 with low-priority traffic. The high-priority data exchange between C1 and C2 is brokerless OPC UA PubSub over UDP[6]. The Switch is an OSI level two Ethernet switch configured to give precedence based on the PCP field in the Ethernet frame. The network consists of two virtual local area networks (VLAN), with VLAN ID (VID) 1 and 2, see Fig. 2.

B. Network configuration

C1 has two ECs connected to two RJ45 ports. An I211 is the Ethernet Port 1 (EP1) EC, and an I219 manages Ethernet Port 2 (EP2). EP1 connects to VID 1 and EP2 to VID 2. The OPC UA PubSub communication from C2 is the prioritized traffic. Ethernet frames carrying the PubSub UDP frames have the PCP field set to six, and the low priority traffic frames have PCP set to zero. The links are 1 Gbps full-duplex.

C. Evaluation application

The EA consists of two applications/subsystems, the real-time and non-real-time applications. The real-time application simulates a control application concerning CPU load, determinism, and dependency on input values.

The C1 CPU has four cores $\{P_1, P_2, P_3, P_4\}$. Each real-time task τ_i is a 4-tuple $\langle C_i, T_i, P_i, A_i \rangle$. C_i is the worst-case execution time, T_i is the period (and deadline), i.e., shortest inter-release time, P_i is the priority, where a lower value is a higher priority, and A_i is core affinity, $A_i \in \{0, P_1, P_2, P_3, P_4\}$ and $A_i = 0$ means no affinity, i.e., the task can be scheduled on all four cores.

The real-time application consists of two sets of tasks, $HP = \{\tau_1^{HP}, \tau_2^{HP}, \tau_3^{HP}, \tau_4^{HP}\}$ and $MP = \{\tau_1^{MP}, \tau_2^{MP}, \tau_3^{MP}, \tau_4^{MP}\}$. Each set contains as many tasks as there are cores, that is, four. *HP* contain the high priority tasks and *MP* the medium priority tasks.
$\forall_{\tau_i^{HP}} \forall_{\tau_i^{MP}}, \tau_i^{HP} \in HP, \tau_i^{MP} \in MP | \tau_i^{HP}.P > \tau_i^{MP}.P$
The application has two multiprocessor using modes, Partitioned Scheduling (PS) mode, where the real-time tasks are

[6]https://reference.opcfoundation.org/v105/Core/docs/Part14/

TABLE I: Hardware used.

Node	Hardware	Ethernet
C1	MSI Intel 2.4GHz I3-7100U 256GB RAM	EP1 I211 EP2 I219
C2	Lenovo Mini PC 2GHz Intel I7 I7-9700T 16 GB RAM	EP1 I219
C3,C4	Raspberry Pi 4B 1.5GHz ARM Cortex A72	EP1 Broadcom 2711
Switch	Zyxel GS1900-8	10 Gbps

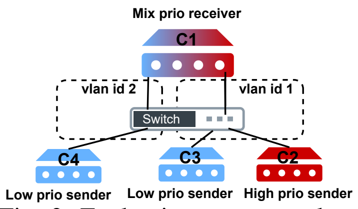

Fig. 2: Evaluation setup topology.

TABLE II: Software versions used.

Name	Version	Comment
VxWorks	21.07	OS C1 and C2
Raspberry Pi OS	10	OS on C3 and C4
Open62541	1.0.1	OPC UA PubSub stack on C1 and C2

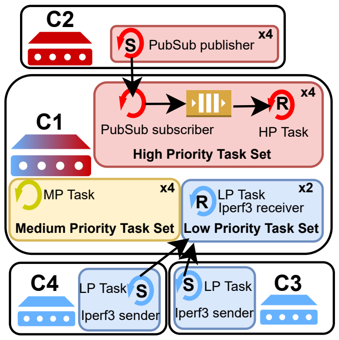

Fig. 3: Conceptual view of the evaluation application.

pinned to a specific core, using the task affinity property $\tau.A$.
$\forall_{\tau_i^{HP}} \forall_{\tau_i^{MP}}, \tau_i^{HP} \in HP, \tau_i^{MP} \in MP, i \in 1, \ldots, 4 | \tau_i^{HP}.A = P_i, \tau_i^{MP}.A = P_i$
The other mode is Global Scheduling (GS), where the scheduler is free to schedule the real-time tasks on all cores.
$\forall_{\tau_i^{HP}} \forall_{\tau_i^{MP}}, \tau_i^{HP} \in HP, \tau^{MP} \in MP, i \in 1, \ldots, 4 | \tau_i^{HP}.A = 0, \tau_i^{MP}.A = 0$
Fig. 3 gives a conceptual overview of the EA and system.

At each invocation, the *HP* task requires an updated value from C2 and consumes all previous values received. Values are exchanged with OPC UA PubSub, C2 is the publisher, and C1 is the subscriber. Each of the four sender tasks in C2 publishes an updated value every 5th ms. In C1, for each τ_i^{HP}, there is an event-driven OPC UA PubSub Subscriber task that shares the affinity and priority of the τ_i^{HP}, that stores the received values in a FIFO. A FIFO read by τ_i^{HP}, shown in Fig. 3.

The *MP* task does not have network dependency. An analogy is an application dependent on local I/O values. The *MP* tasks have lower priority since they can have a longer execution time than the *HP* task, allowing the *HP* task to preempt the *MP* tasks. Table III shows priorities and execution times, elaborated further in Section IV-D1.

The Low Priority (*LP*) application and tasks represent non-time-critical applications, dependent on data produced in C3 and C4. For example, reception of system maintenance files as preparation for a system upgrade, application change, and less critical process values. We use `Iperf3` [7] to emulate this application.

[7]https://iperf.fr/

TABLE III: Task parameters.

Name	Priority (P)	Period (T)	Exec. time (C)	CPU utilization
HP	20	10ms	1ms	10%

MP	40(HN), 50(EN)
	60(LN)
		20ms	0-16ms	0-80%

LP	100	Event driven	Comm. dep.	Comm. dep.
NetTask	50	Event driven	Comm. dep.	Net. dep.

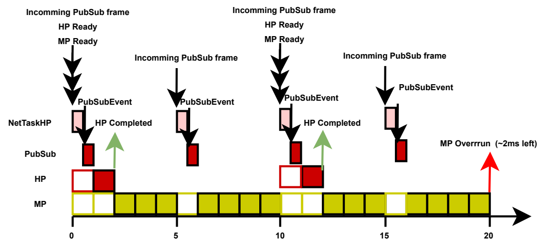

Fig. 4: Scheduling example of a *HP* and *MP* task.

D. *Evaluation variants*

By using different priorities, execution times, and best-effort traffic, we measure the correctness of the EA in terms of deadline misses (overruns) and missed high-priority values updates from C2.

1) Execution time: The *HP* tasks have a fixed execution time of one ms. We use different execution times for the *MP* tasks for two reasons. Firstly, to observe how *MP.C* impacts the NetTask and the *HP* dependencies on received values from C2. Secondly, the NetTask interference on *MP* when *MP.C* increases. Table III show the *MP* tasks C range and the corresponding utilization. Note that the execution time, C, specified is the CPU time the task requires to complete, i.e., the Worst-Case Execution Time (WCET) and execution time are always the same.

The *HP* and *MP* CPU utilization range is between 10% and 90%. OPC UA PubSub subscriber adds approximately four percent, in addition to the time shown in Table III. Fig. 4 shows that it is not possible to schedule *MP* tasks with $C = 16$ms and $T = 20$ms combined with *HP* tasks with $C = 1$ms and $T = 10$ms with the interference of a high priority NetTask, the NetTaskHP, and the PubSub task. The execution times illustrated for the PubSub and NetTaskHP are likely higher than in reality, but other high-priority executions are left out, such as scheduling overhead. A WCET analysis would give us the exact limits, but that is beyond the scope of this paper. The above is the motivation behind the upper limit of $MP.C = 16$ms; it is over the limit.

2) Priority: As shown in Table III we use three priority levels for *MP*. These are Higher than NetTask (HN) with priority 40, Equal to the NetTask (EN) with priority 50, and Lower than the NetTask (LN) with priority 60. The three levels are selected to show the interference between NetTask and real-time application tasks. *MP* priority HN, can cause *MP* execution to block network handling and delay the values communicated to *HP*. *MP* priority EN, can cause the *MP* execution time to affect the network handling, similar to HN, since VxWorks, by default, will not preempt the same priority. Finally, when *MP* is lower in priority than the NetTask, LN, the execution of the *MP* will not interfere with the network communication. However, network handling can block *MP* and cause *MP* to overrun.

3) Network traffic: The high priority network traffic consist of the variable exchange over OPC UA PubSub from C2 to C1. The different types of low priority, best-effort traffic are summarized in Table IV. With No *LP*. we mean no low-priority traffic. The 1Gbps link and the receiver processing possibilities limit the TCP bandwidth utilization. The UDP max bandwidth utilization is limited to 400Mbps, about 34 frames per millisecond and 40% of the available bandwidth.

TABLE IV: Low priority network communication types.

Abbreviation	Sender	Protocol	Bandwidth
No LP.	None	-	0
1 TCP	C3	TCP	$<$ 1Gbps
2 TCP	C3,C4	TCP	$<$ 1Gbps
1 UDP	C3	UDP	$<$ 400Mbps
2 UDP	C3,C4	UDP	$<$ 400Mbps

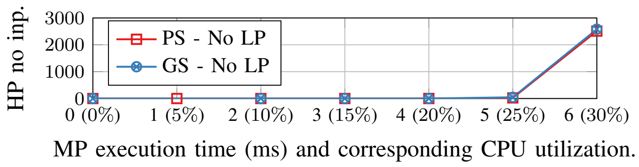

Fig. 5: *MP* tasks with higher priority than NetTask (HN) result in *HP* cycles without input updates.

4) Scheduling variants: The VxWorks scheduler is a priority-based preemptive scheduler that, by default, uses GS and does not pin tasks to cores, with some exceptions, such as the NetTask, that have a core affinity for performance reasons. The default affinity for the NetTask is P_1. The EA can run in two scheduling modes, PS and GS. In PS mode $\tau_1^{HP}.A = P_1, \tau_1^{MP}.A = P_1$, i.e., τ_1^{HP} and τ_1^{MP} are pinned to the core P_1, the same core as the NetTask. Their uncompromising coexistence with the NetTask on the core P_1 makes the EA in the PS mode more likely to encounter failures faster due to overuse of P_1. The affinity to P_1 prevents those tasks from utilizing the potentially available CPU time on other cores.

E. *VxWorks results – no filtering*

This section presents the results of running the variants discussed without prioritization filtering. The data comes from one-minute per variant runs. Fig. 5 shows that *MP* with HN priority, cause *HP* to lack input data when *MP.C* increases. *MP* execution blocks the NetTask, and C2 PubSub values do not reach *HP* during *MP.C*.

Fig. 6 shows the result when *MP* priority EN. The result follows the same pattern for EA in PS mode as for *MP* priority HN. For EA in GS mode, *HP* gets the data each cycle without low priority traffic. A *MP* task with priority EN, equal to NetTask priority, won't preempt and block the NetTask; the scheduler migrates the *MP* task to another available core if any. With low-priority traffic, *HP* lacks updates when *MP.C* increases.

With *MP* priority LN, *MP.C* does not affect *HP* reception of values. However, NetTask execution can prolong the *MP* response time. Fig. 7 shows when *MP* start to miss deadlines and overruns for the different types of low priority, best-effort traffic.

Again, we see that the EA in PS mode fails before the EA in GS mode. Notable is also that the UDP traffic is more of a challenge than TCP because the TCP flow control eases the burden on the receiver.

Fig. 8 shows the average time the *MP* tasks are in the ready state. Ready is the state a VxWorks task is in when

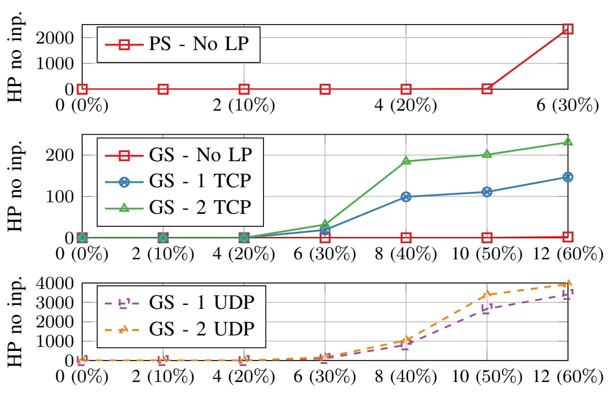

Fig. 6: *MP* tasks with equal priority to the NetTask (HN) result in *HP* cycles without input updates.

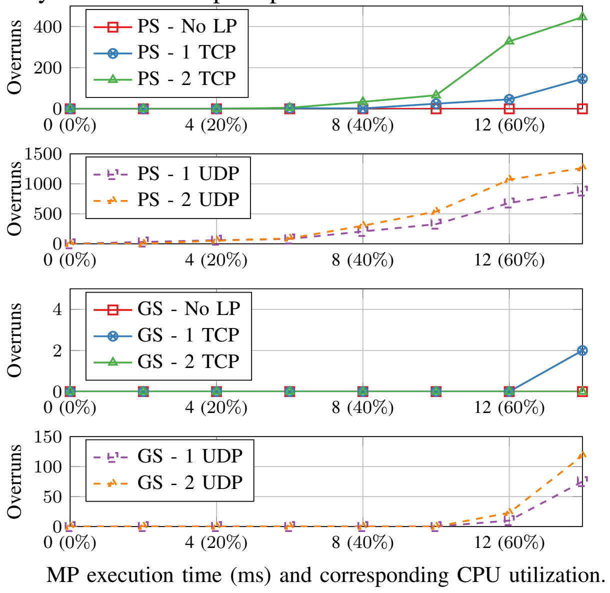

Fig. 7: NetTask lower priority than *MP* - *MP* misses deadlines (overruns).

it is ready to execute, but the CPU is busy executing higher priority tasks/interrupts. The NetTask interference on the *MP* task is higher in PS mode but limited to the *MP* task that shares the core with NetTask. The *MP* average ready time in PS mode when receiving low-priority TCP traffic is higher for lower *MP.C*. Due to *LP* tasks (`iperf3`) getting more execution time, resulting in a larger TCP flow control window. Except for that TCP variant, UDP traffic causes the highest interference on the *MP* tasks.

F. Evaluation system – with filtering

We apply the prioritization filtering mechanism described in Section III-A on VxWorks running in C1. Table V shows the task priorities when using filtering. NetTaskHP handles the high-priority traffic and NetTaskLP processes the low-priority traffic.

G. VxWorks result – with filtering

NetTaskHP handles the high-priority network traffic, the data that *HP* tasks depend on, and NetTaskHP has a higher

TABLE V: Task parameters.

Name	Priority (P)	Period (T)	Exec. time (C)	CPU utilization
HP	20	10ms	1ms	10%
MP	40	20ms	0-16ms	0-80%
LP	100	Event driven	Event driven	Comm. dep.
NetTaskHP	20	Event driven	HP Comm. dep.	HP comm. dep.
NetTaskLP	50	Event driven	LP Comm. dep.	LP comm. dep.

priority than the *MP* tasks. Hence *MP* execution does not cause lost / late inputs for *HP*.

Fig. 9 shows that with priority filtering of the incoming frames, the *MP* deadline missing limit is higher; it now occurs at 16ms (80% utilization), the upper limit of what is feasible. Fig. 10 shows that the time the *MP* task is in a ready state is not affected by the low priority traffic. Hence, with the help of filtering in the EC, a network-dependent real-time application can be free from interference from less critical, best effort, and low priority network traffic.

V. DISCUSSION

Section IV-G shows that prioritization filtering can eliminate best-effort traffic impact on the real-time functions. Even though the EA is a simulated application designed to show the priority inversion problem that emerges when handling incoming traffic with different criticality, end-systems on converged networks benefit from eliminating the priority inversion problem. The elimination of priority inversion due to best-effort traffic interference on critical real-time task increase the dependability of the system. How much depends on properties like CPU utilization, communication patterns, etc., domain and solution-specific properties. Other potential benefits are reduced latency and decreased probability of dropping high priority frames due to full queues.

High NetTaskHP priority poses a potential risk. For example, a Denial of Service (DoS) attacker could generate

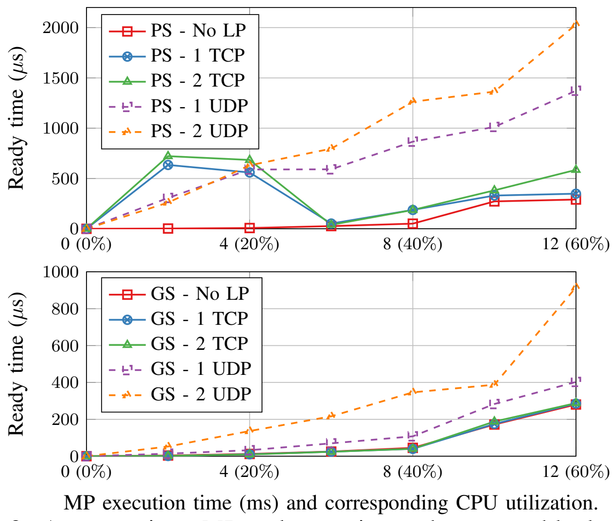

Fig. 8: Average time *MP* tasks are in ready state - blocked for execution by higher priority tasks. The difference between No *LP* and the graphs for *LP* traffic illustrates the *LP* traffic impact on *MP*

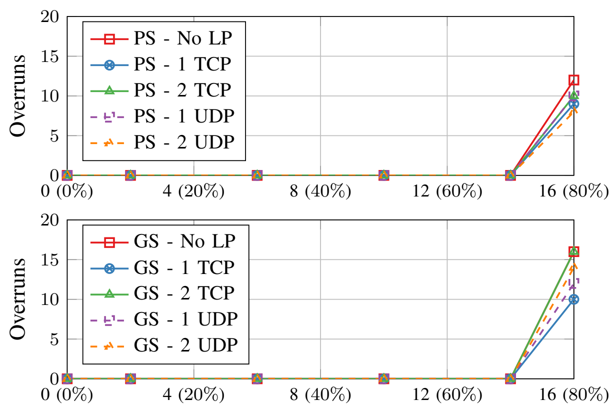

Fig. 9: *MP* deadline misses with packet filtering enabled.

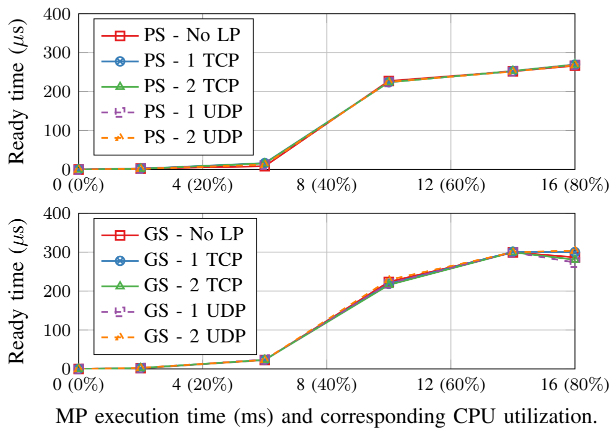

Fig. 10: Average pended time for all medium priority tasks - with packet filtering.

high-priority traffic that starves out other high-priority execution. However, if real-time functions are network-dependent, network handling is likely to have high priority. If that is the case, filtering might reduce the DoS attack surface since prioritization directs low priority, best-effort traffic to lower priority processing. A potential hardening strategy could be to limit the nodes trusted for high-priority processing and filter not only on a QoS property but also on node identities. Such as MAC- or IP-addresses. However, cybersecurity is a vast topic on its own. We realize that the challenges and potential future work could further evaluate how to use the Ethernet Controllers for security purposes.

The traffic load used can be discussed; 400 Mbps UDP traffic might be much. However, consider that we only used two Ethernet ports and two clients. A modern IPC, such as the APC 910 from B&R[8], has support for six and more one Gbps ports served by a similar CPU as the one in C1, a quad-core Intel I3.

VI. Conclusion and Future work

In this paper, we identified hardware-aided filtering of incoming frames to network processing with appropriate priority as a prominent solution. We described the steps needed to realize priority processing filtering on VxWorks. Finally, we evaluated the solution on VxWorks using a simulated controller application consisting of several real-time and non-real-time tasks with different priorities and network dependencies. The results show that prioritization filtering eliminates the best-effort traffic impact on the application's real-time functionality.

Relevant future work is to evaluate this approach on Linux combined with virtualization and prioritization handling in converged virtual networks. Another natural extension of this work is to take a holistic approach that incorporates outgoing traffic since outgoing traffic also requires network task processing.

[8]https://www.br-automation.com/en/products/industrial-pcs/automation-pc-910/

References

[1] D. Hallmans, M. Ashjaei, and T. Nolte, "Analysis of the TSN standards for utilization in long-life industrial distributed control systems," in *IEEE Int. Conf. Emerg. Tech. & Fact. Autom. (ETFA)*, pp. 190–197, 2020.

[2] D. Bruckner, M.-P. Stănică, R. Blair, S. Schriegel, S. Kehrer, M. Seewald, and T. Sauter, "An introduction to opc ua tsn for industrial communication systems," *Proc. IEEE*, vol. 107, no. 6, pp. 1121–1131, 2019.

[3] N. Finn, "Introduction to time-sensitive networking," *IEEE Comm. Stand. Mag.*, vol. 2, no. 2, pp. 22–28, 2018.

[4] L. Lo Bello and W. Steiner, "A perspective on IEEE time-sensitive networking for industrial communication and automation systems," *Proc. IEEE*, vol. 107, no. 6, pp. 1094–1120, 2019.

[5] L. Zhao, P. Pop, and S. S. Craciunas, "Worst-case latency analysis for IEEE 802.1Qbv time sensitive networks using network calculus," *IEEE Access*, vol. 6, pp. 41803–41815, 2018.

[6] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks," in *Int. Conf. Real-Time Networks and Systems*, pp. 183–192, 2016.

[7] P. Pop, M. L. Raagaard, M. Gutierrez, and W. Steiner, "Enabling fog computing for industrial automation through time-sensitive networking (tsn)," *IEEE Comm. Stand. Mag.*, vol. 2, no. 2, pp. 55–61, 2018.

[8] T. Skeie, S. Johannessen, and O. Holmeide, "The road to an end-to-end deterministic ethernet," in *IEEE Int. Workshop on Factory Communication Systems*, pp. 3–9, 2002.

[9] L. Thrybom and G. Prytz, "QoS in switched industrial ethernet," in *IEEE Conf. Emerg. Tech. & Fact. Autom. (ETFA)*, pp. 1–8, 2009.

[10] S. Larsen, P. Sarangam, R. Huggahalli, and S. Kulkarni, "Architectural breakdown of end-to-end latency in a TCP/IP network," *Int. journal of parallel programming*, vol. 37, no. 6, pp. 556–571, 2009.

[11] A. Beifuß, D. Raumer, P. Emmerich, T. M. Runge, F. Wohlfart, B. E. Wolfinger, and G. Carle, "A study of networking software induced latency," in *Int. Conf. and Workshops on Networked Systems (NetSys)*, pp. 1–8, 2015.

[12] Ramneek, S.-J. Cha, S. H. Jeon, Y. J. Jeong, J. M. Kim, and S. Jung, "Analysis of Linux kernel packet processing on manycore systems," in *IEEE Region 10 Conf. TENCON*, pp. 2276–2280, 2018.

[13] I. Behnke, L. Pirl, L. Thamsen, R. Danicki, A. Polze, and O. Kao, "Interrupting real-time iot tasks: How bad can it be to connect your critical embedded system to the internet?," in *2020 IEEE 39th International Performance Computing and Communications Conference (IPCCC)*, pp. 1–6, 2020.

[14] M. Lee, H. Kim, and I. Shin, "Priority-based network interrupt scheduling for predictable real-time support," *Journal of Computing Science and Engineering*, vol. 9, no. 2, pp. 108–117, 2015.

[15] C. Blumschein, I. Behnke, L. Thamsen, and O. Kao, "Differentiating network flows for priority-aware scheduling of incoming packets in real-time iot systems," in *25th IEEE International Symposium on Real-Time Distributed Computing (ISORC)*, 2022.

[16] I. Behnke, P. Wiesner, R. Danicki, and L. Thamsen, "A priority-aware multiqueue nic design," in *In Proceedings ofthe 35th Annual ACM Symposium on Applied Computing (SAC)*, 2022.