P4 In-Network Source Protection for Sensor Failover

Steffen Lindner*, Marco Häberle*, Florian Heimgaertner*, Naresh Nayak†, Sebastian Schildt†,
Dennis Grewe†, Hans Loehr†, and Michael Menth*
* University of Tuebingen, Chair of Communication Networks, Tuebingen, Germany
Email: {steffen.lindner,marco.haeberle,florian.heimgaertner,menth}@uni-tuebingen.de,
† Corporate Sector Research and Advance Engineering, Robert Bosch GmbH, Renningen, Germany
Email: {naresh.nayak,sebastian.schildt,dennis.grewe,hans.loehr}@de.bosch.com

***Abstract*—Automated systems like industrial applications or autonomous cars heavily rely on sensor information. To increase reliability, several sensors may be used to provide identical data, e.g., temperatures or velocity. Applications exploiting this data may either use both data streams or rely on a single primary data stream until the primary stream fails. This increases the complexity of the application and is prone to errors. In this paper we present a prototype and mechanisms for in-network sensor failover. Our novel prototype detects the failure of a primary sensor and delivers in turn the data of a redundant sensor to the application.**

I. Introduction

Reliability is an important property of system critical infrastructure. Traditional resilience mechanisms protect against single link and single node failure, i.e. the connectivity can be restored if a single link or a single node fails. The detection of a link or node failure may require a few 10s of milliseconds so that traffic loss cannot be avoided. Loop-Free Alternates (LFAs) [1] are an example for such a mechanism. To respond faster to an incident, traffic can be transmitted redundantly on multiple paths. If there is an error on one path, the traffic on another path is still transmitted correctly. As traffic is transmitted redundantly, a substantially higher bandwidth is required and the resources of the network are not used in an optimal manner. 1+1 protection [2] is an example for a redundant protection mechanism. These network protection mechanisms protect only against network failures like link or node failures. However, they cannot help when the source node fails. The source can be protected by having several sources providing the same information. Possible use cases include sensors in industrial facilities, (autonomous) cars, or other publish/subscribe scenarios. In a publish subscribe environment, publishers offer data that can be accessed by subscribers. To compensate for the failure of a publisher, several publishers provide the same information. In case of an error, the information is provided by another publisher. To that end, the failure of a publisher must be detected and the network re-configured. Alternatively, an application can subscribe to several streams and independently perform error handling. These methods either require additional signaling effort, and thus time, or involve redundant implementations in different applications. In industrial and time-critical networks in general, the demands on data streams are usually stringent, not only in terms of bandwidth, latency and jitter, but also in terms of response time in the event of errors. While new paradigms and standardization efforts like time-sensitive networking (TSN) and deterministic networking (DetNet) provide a broad feature set to guarantee these requirements, they come with the need of specialized and costly hardware. With software defined networking, new mechanisms and protocols can be developed without requiring specialized hardware. Technologies such as P4 enable the data plane of a compatible switch to be programmed, paving the way for new prototypes and mechanisms. In this paper, we present two mechanisms that can be used to implement a fast failover for redundant sensor pairs in a network. In the error-free case, only data from the primary sensor is forwarded. If the primary sensor fails, the switch detects the missing data of the primary sensor and forwards the data of the redundant sensor. The presented mechanisms neither require additional signaling nor the reconfiguration of the network. Furthermore, the failover is transparent for the receiver. We present a P4-based implementation of the proposed mechanisms and evaluate them on the high-performance P4 switching ASIC Tofino.

Annex to ISBN 978-3-903176-28-7

The remainder of the paper is structured as follows. We discuss some related resilience mechanisms in Section II. Section III gives an overview of the data plane programming language P4. Afterwards, we introduce two mechanisms for in-network sensor failover in Section IV. Section V gives some insights regarding the P4-based implementation of the proposed mechanisms. We evaluate and discuss the mechanisms in Section VI. Finally, Section VII concludes the paper.

II. Related Work

In this section we present some related resilience mechanisms. Closely related to our case study is the area of publish/subscribe networks. In a publish/subscribe system, users can subscribe to publishers to receive information. To ensure reliability, several publishers can provide the same information. In our deployment scenario the user corresponds to the application and the sensors correspond to publishers.

In [3], the authors propose an extension to the PURSUIT framework to introduce source recovery in information-centric networks (ICN). They introduce a new component, called resilience management (RM). To detect a component failure, each node exchanges link state updates (LSU) with its neighbors. If a node does not receive a LSU within a time limit, a link failures is assumed. The topology management (TM), another component in the PURSUIT framework, receives this information and updates the topology accordingly. Afterwards, affected distribution trees can be altered and the connectivity can be restored. If the failure affects a publisher, the distribution trees can be adjusted to use an alternative publisher. Hoefling et. al [4] propose a distributed load balancing mechanism for SeDAX, a publish/subscribe information-centric networking architecture. They ensure robustness by replicating content on multiple nodes. If the primary node fails, it is possible to switch directly to the second node in order to continue offering the content.

The data distribution service (DDS) [5] is a platform-independent standard for data-centric publish subscribe systems. It is designed for real-time systems with low latency and high robustness requirements. DDS facilitates that several publishers supply the same data while a subscriber always receives data from the so-called ”most-trusted” publisher. If the ”most-trusted” publisher fails, the application automatically uses the data of the next publisher. This failure mechanism requires an action from the application, provided by the DDS framework. Campelo et. al [6] propose an architecture for a fault-tolerant distributed industrial control system composed of several micro-controllers. The system can switch to an alternative micro-controller in case of a failure. Another architecture for safety-critical applications is described in [7].

III. P4 Foundations

We first give an overview of P4. Then we summarize basics of the P4 pipeline that are needed to understand the implementation of the P4 implementation.

A. P4 Overview

P4 is a programming language for protocol-independent packet processors [8]. It allows a flexible description of its processing pipeline, in particular the definition of arbitrary headers and packet parsers. P4 programs are compiled to so-called targets, e.g., the software switch BMv2 or switching ASICs. A compiled program offers the P4Runtime as an API so that P4 nodes can be re-configured by controllers during runtime.

B. P4 Pipeline

Figure 1 illustrates P4's abstract forwarding model. A user-programmable parser reads an incoming packet and stores its header information in P4-internal header fields. They are carried with the packet through the P4 pipeline, possibly with additional metadata.

The P4 abstract forwarding model is divided into two stages, the ingress and the egress pipeline, which are separated by the packet buffer. For modularity, the ingress and egress pipeline can be further subdivided by control blocks (CB). Match+action tables (MATs) allow for packet-specific processing. They have entries consisting of match fields and match types that map packets to actions and parameters. One action may be defined to be carried out if no table entry matches a packet (table miss).

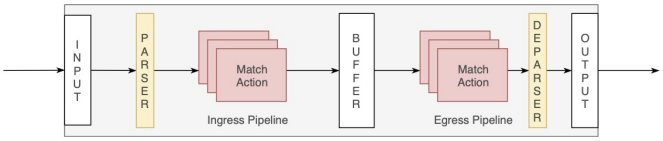


Fig. 1. P4 abstract forwarding model according to [8].

P4 offers in its core definition three match types: exact, lpm, and ternary. Exact implies that a packet header must contain the match field in the table entry, e.g. a given IPv4 address in the destination address field of an IP header. Lpm stands for longest prefix match which is well-known from standard IP forwarding. Ternary facilitates wildcard matches. A packet is processed at most once by the same MAT within the pipeline.

C. Registers

Information stored in metadata are only valid during a packets lifetime. To store information beyond the lifetime of a packet, P4 offers the ability to store information in so-called registers. Information stored in registers can be accessed during packet processing. We leverage registers to store data required for the protection mechanisms.

IV. In-Network Sensor Failover

In this section we first give an overview of the general context. Afterwards, we describe novel protection mechanisms for in-network sensor failover.

A. Overview

Figure 2 shows the concept of the proposed protection mechanisms. Two sensors periodically send data to an application over a network. To ensure reliability, both sensors send the same information, e.g., temperatures or velocity, but may send them with different periods.

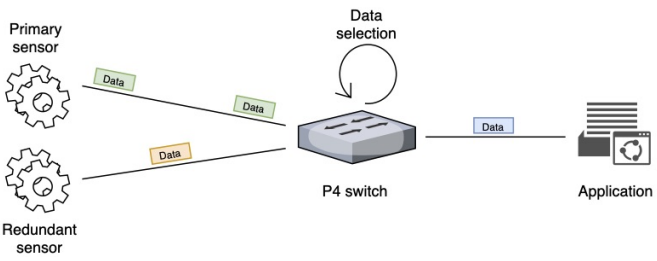
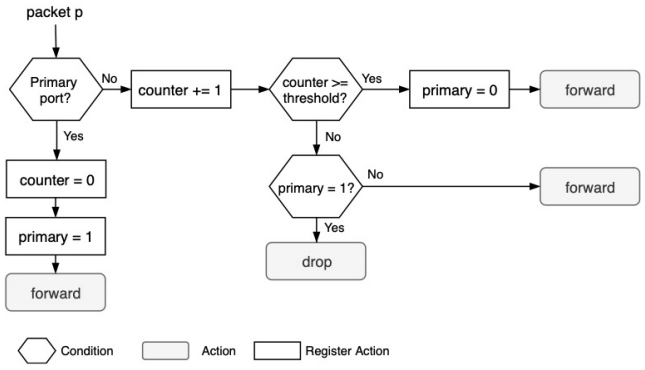
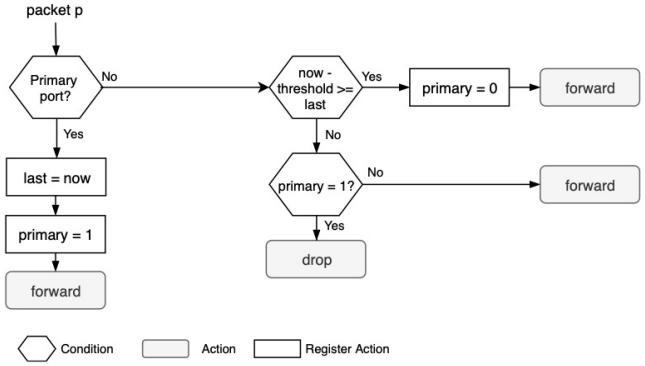


Fig. 2. Conceptual overview. Two redundant sensors provide information for a application.

An application may decide which data to use. The application might either use both data streams or only one data stream and, in case of an error, switch to the second data stream. However, this comes with an increase in program logic and developers have to deal with sensor failures. We propose to transfer the sensor failover to the network by leveraging programmable network devices, e.g. P4 switches. Two different modes of operation can be distinguished. By default, the primary sensor data is forwarded to the application. With the aid of periodic messages from the redundant sensor, the P4 switch can detect if the primary sensor fails. If a failure is detected, data from the redundant sensor can be forwarded to the application.



(a) Counter-Based protection mechanism.



(b) Timer-Based protection mechanism.

Fig. 3. Overview ot the operations of the counter-based and timer-based protection mechanism.

B. Mechanisms

To detect the failure of the primary sensor, we leverage the time dependencies between the two data streams of the sensors. We propose two mechanisms to detect sensor failures, *counter-based* and *timer-based* failover, respectively. Figure 3(a) and Figure 3(b) illustrate the operations of the two protection mechanisms. We refer to actions involving register access as *Register Actions*, and actions without this access solely as *Action*.

1) Counter-Based Failover: The first protection mechanism is based on a counter approach. A counter is increased for each arriving data portion from the redundant sensor. In simplest form, the counter is increased by one and stored in a register field leveraging a *Register Action*. For each arriving data portion of the primary sensor, the counter is set to zero. If the counter exceeds a certain threshold T_c, the switch forwards the data from the redundant sensor to the application. The threshold has to be selected in such a way that the dependencies of the two data streams are taken into account. For example, if the redundant sensor transmits data twice as fast as the primary sensor, a failure of the primary sensor can be detected by a threshold of two and an increase by one. In such a case, at most one packet of the primary sensor is lost. If the periods of the two sensors are not multiples of each other, the same effect can be achieved by scaling the threshold and the respective increase of the counter.

If the primary sensor transmits faster than the secondary sensor, it cannot be guaranteed that at most one packet from the primary sensor will be lost. Furthermore, a change in the sensor periods may result in undesired behaviour, which is further described in Section VI. This protection mechanism is implemented for the high-performance P4 switching ASIC Tofino and demonstrated in Section VI.

2) Timer-Based Failover: The counter based approach is reliable if the sensor periods are stable. If the sensor periods change during operation, the relation between the intermediate arrival times and the configured threshold is no longer correct. In addition, the currently stored value in the counter is no longer valid. As a consequence, the counter must be reset. However, this may cause a delayed switch-over to the redundant sensor. To overcome this problem, we propose to use the actual intermediate arrival times for the protection mechanism instead of the counter-based relation among the arrival times. The *timer-based* approach utilizes packet timestamps which can be accessed during packet processing. For each arriving data portion of the primary sensor, the packet timestamp is saved in a special register. Data from the redundant sensor is only forwarded, if the elapsed time since the last data portion of the primary sensor exceeds a certain threshold T_t. This more advanced mechanism is implemented for the BMv2 software switch and can be accessed at Github[1]. The timer-based mechanism can also be implemented for the Tofino. For simplicity, we only implemented this mechanism for the BMv2. We give some examples of its advantages over the counter-based mechanism in Section VI.

V. Implementation

In this section we describe the P4 based implementation of the protection mechanisms presented in Section IV. First, we will give an overview of the P4 pipeline. Afterwards, we will describe the important properties of the control block *Protect*, which implements the protection mechanisms. Finally, we give a rough overview of the control plane.

[1]Repository: https://github.com/uni-tue-kn/p4-source-protection

A. Overview

The implementation is based on a local Ethernet network and comprises local layer-2 switching and the applied protection mechanism. Figure 4 illustrates our implemented P4 pipeline.

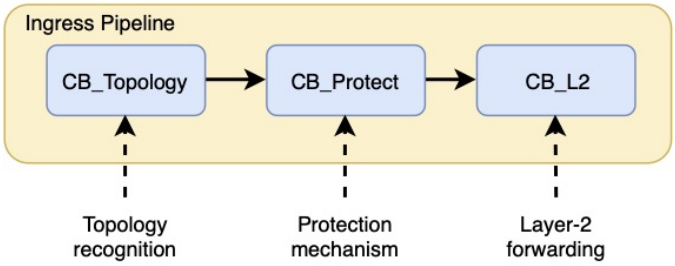


Fig. 4. Overview of the ingress pipeline.

The implementation solely requires the ingress part of the P4 pipeline. The P4 pipeline was introduced in Section III. The ingress pipeline is divided into three control blocks (CBs), named *CB_Topology*, *CB_Protect* and *CB_L2*. The control blocks *CB_Topology* and *CB_L2* are used for general network connectivity such as topology recognition and layer-2 forwarding. The protection mechanisms are implemented in the control block *CB_Protect*. Incoming packets traverse all three control blocks.

B. Control Block CB_Protect

The control block *CB_Protect* implements the two previously presented mechanisms. In order for the mechanisms to work, the switch must have access to several pieces of information. First and foremost, the switch must know the relation between the sensors and its physical interfaces, i.e. which interface corresponds to which sensor. Furthermore, the switch requires the configured threshold T_c or T_t. These information is dynamically provided with match+action tables (MATs). During packet processing, the information is made available by matching on these MATs and storing the required information in metadata fields. As soon as the period of the sensors changes, the contents of the MATs can be updated by the control plane. As a consequence, we can react dynamically to the changes. In addition to the interface and threshold information, the last timestamp of the primary sensor and the counter must be stored. We leverage registers that are available in the software switch BMv2 as well as in the Tofino.

C. Control Plane

The control plane is responsible for filling the different MATs with entries. To that end, it provides an interface for runtime changes and updates the MATs accordingly. It utilizes information of a proprietary topology detection mechanism to calculate the appropriate forwarding rules of the network to enable local layer-2 forwarding.

VI. Evaluation & Discussion

In this section we illustrate the functionality and effectiveness of the two introduced protection mechanisms. To accomplish this, we perform experiments in our testbed using our prototype. We first explain the general setup of our experiments. Afterwards, we introduce our evaluation metrics. Finally, we describe the experimental results and explain some theoretical examples.

A. Methodology

1) General Setup: The hardware testbed consists of three servers physically connected to a Tofino Edgecore Wedge 100BF-32X as shown in Figure 2. The servers are based on an Intel Xeon Scalable Gold 6134 (8x 3.2 GHz) and 4x 32 GB RAM. The Tofino Edgecore Wedge 100BF-32X is a high-performance P4 switch with 32 100G ports. Two servers thereby act as sensors, the third server mimics an application. Both sensors send data to the application with different periods p_0 and p_1.

2) Metric: We evaluate the arrival of the packets of the two sensors at the application. During operation we simulate a sensor failure by disconnecting the link between the primary sensor and the P4 switch. We demonstrate that the correct configuration of the mechanism is essential and show by a theoretical example that the *timer-based* approach is superior to the *counter-based* approach.

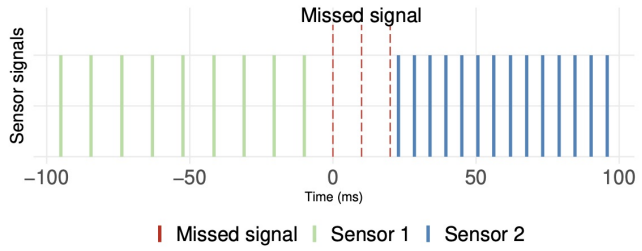
B. Counter-Based Protection

To illustrate the influence of the threshold on the counter-based mechanism, we consider the experiment as described in Section VI-A. Figure 5(a) reflects the result of an incorrect configuration of the *counter-based* mechanism. The primary sensor sends with a period of $p_1 = 10$ ms and the redundant sensor with a period of $p_2 = 5$ ms. Since the secondary sensor transmits twice as fast as the primary sensor, a failure of the primary sensor can be detected by a threshold of $T_c = 2$. In this experiment the threshold was falsely set to $T_c = 5$. Figure 5(a) shows the incoming data packets from the primary sensor (sensor 1) and the redundant sensor (sensor 2) at the application for this configuration.

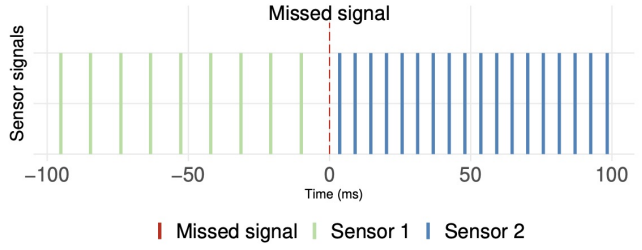
At time $t = 0$, the first packet of the primary sensor is lost. Subsequently, due to the wrong threshold, two additional packets of the primary sensor are lost before the system switches to the redundant sensor. In contrast, Figure 5(b) shows that with a properly tuned threshold, only one packet of the primary sensor is lost.

C. Timer-based Protection

With constant sensor periods, the *timer-based* mechanism performs as well as the *counter-based* mechanism. However, as soon as the periods change during operation, the *timer-based* approach is superior. Two strategies can be pursued for the *counter-based* approach. After a period change, the counter can either be reset or maintained. We now consider the case that the counter is reset. Figure 6(a) illustrates an example. Note that all transmitted signals are displayed. We



(a) $p_1 = 10$ ms, $p_2 = 5$ ms, $T_c = 5$.



(b) $p_1 = 10$ ms, $p_2 = 5$ ms, $T_c = 2$.

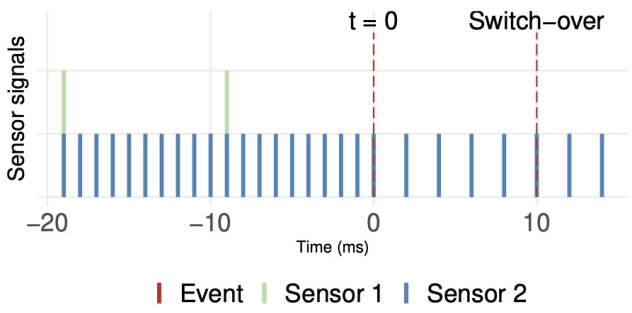
Fig. 5. Influence of threshold on sensor failover for the counter-based protection mechanism.

further assume that at $t = 0$ the primary sensor fails and that the counter is reset. At the same time the period of the redundant sensor changes from 1 ms to 2 ms. The *counter-based* approach forwards the data of the redundant sensor at time $t = 10$ ms, as all information before the period change is lost. In contrast to the *counter-based* mechanism, the timer based mechanism performs the switch-over as intended at $t = 4$ ms. Figure 6(b) visualizes the differences.

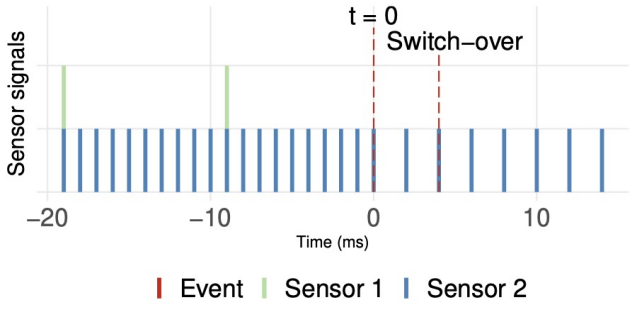
If the counter is maintained at a period change, the *counter-based* mechanism will still not behave correctly. Lets assume similar settings as in the previous example. The primary sensor transmits data with period $p_1 = 10$ ms, the secondary sensor with $p_2 = 1$ ms. At time $t = 0$ the primary sensor fails, the counter equals four and the period of the secondary sensor switches again to $p_2 = 2$ ms. Figure 7(a) illustrates this setup. As the counter has not been reset, the switch erroneously switches to the redundant sensor at time $t = 2$. Again, Figure 7(b) shows that the *timer-based* mechanism is not affected by this problem.

VII. Conclusion

Sensors in critical systems must provide applications with redundant information. Applications have to decide how to handle the different data streams and how to react to the failure of one of the data streams. This is not only prone to errors but also leads to duplication in application logic. In this paper we proposed two mechanisms to move the detection of sensor errors to the network to make the redundant data transmission transparent for the application. We have shown the disadvantages of a simple counter-based mechanism and presented a more complex timer-based mechanism for the BMv2.

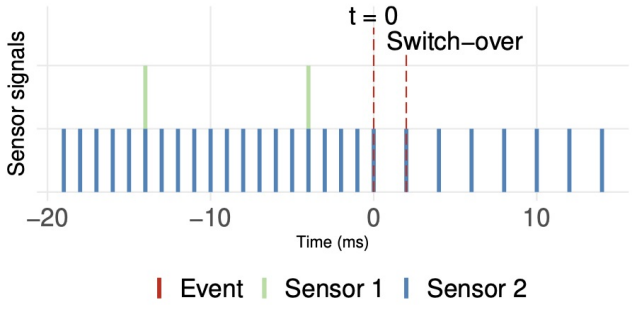


(a) Counter-based mechanism with reset.

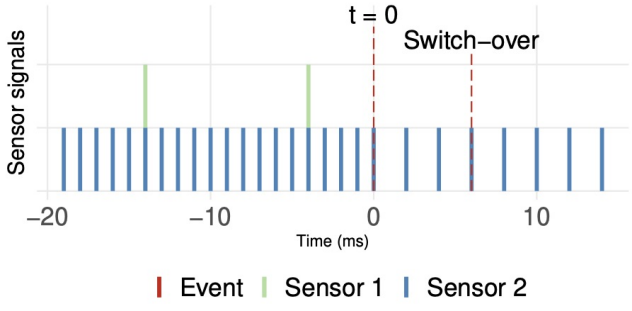


(b) Timer-based mechanism.

Fig. 6. Different behaviour of the two mechanisms during a period change. The *counter-based* mechanism requires more time for the switch-over than the *timer-based* mechanism.



(a) Counter-based mechanism without reset.



(b) Timer-based mechanism.

Fig. 7. Different behaviour of the two mechanisms during a period change. The *counter-based* mechanism erroneously forwards data from the secondary sensor.

REFERENCES

[1] A. Atlas and A. Zinin, "RFC5286: Basic Specification for IP Fast Reroute: Loop-Free Alternates ," Sep. 2008.

[2] "ITU-T Recommendation G.7712/Y.1703 (2010), Internet protocol aspects – Operation, administration and maintenance," ITU, Sep. 2010.

[3] M. F. Al-Naday, M. J. Reed, D. Trossen, and K. Yang, "Information resilience: source recovery in an information-centric network," *IEEE Network*, vol. 28, no. 3, pp. 36–42, 2014.

[4] M. Hoefling, C. G. Mills, and M. Menth, "Distributed load balancing for the resilient publish/subscribe overlay in sedax," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 147–160, 2017.

[5] M. Ryll and S. Ratchev, "Towards a publish / subscribe control architecture for precision assembly with the data distribution service," in *Micro-Assembly Technologies and Applications*, S. Ratchev and S. Koelemeijer, Eds. Boston, MA: Springer US, 2008, pp. 359–369.

[6] J. Campelo, F. Rodriguez, A. Rubio, R. Ors, P. Gil, L. Lemus, J. Busquets, J. Albaladejo, and J. Serrano, "Distributed industrial control systems: a fault-tolerant architecture," *Microprocessors and Microsystems*, vol. 23, no. 2, pp. 103 – 112, 1999.

[7] B. Rostamzadeh, H. Lonn, R. Snedsbol, and J. Torin, "Dacapo: a distributed computer architecture for safety-critical control applications," in *Proceedings of the Intelligent Vehicles '95. Symposium*, 1995.

[8] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-Independent Packet Processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, 2014.