

PROCEEDINGS OF

# RTAutoSec 2025

---

Second Workshop on  
Real-Time Autonomous Systems Security

July 8<sup>th</sup>, 2025 in Brussels, Belgium

in conjunction with



The 37<sup>th</sup> Euromicro Conference on Real-Time Systems

July 8–11, 2025, Brussels, Belgium

*Editors:*  
Monowar Hasan  
Mohammad Hamad

## Contents

<b>Message from the Chairs</b>	<b>3</b>
<b>Keynote Talk</b>	<b>5</b>
<b>Invited Talk 1</b>	<b>19</b>
<b>Invited Talk 2</b>	<b>43</b>
<b>Session: RT-AutoSec Technical Session</b>	<b>57</b>
Defending Event-Triggered Systems against Out-of-Envelope Environments <i>Marcus Völz, Mohammad Ibrahim Alkoudsi, Azin Bayrami Asl, Kristin Kruger, Julio Mendonça,         Gerhard Fohler . . . . .</i>	<i>57</i>



## Message from the Chairs

Welcome to RT-AutoSec25, the second Workshop on Real-Time Autonomous Systems Security. We invite you to join us for engaging discussions and the exchange of ideas on system-level challenges in real-time and embedded autonomous systems. This year, RT-AutoSec is being held as a half-day workshop. We are excited to feature four presentations, including a keynote talk from Dr. Friedrich Wiemer of Robert Bosch GmbH, representing the perspective of industry on this critical topic.

We would like to extend our heartfelt thanks to you—the audience—for your participation. Your thoughtful questions and enthusiastic engagement are what continue to shape and strengthen RT-AutoSec. We hope you enjoy the workshop!

The Workshop Chairs,

Monowar Hasan

Washington State University, USA

Mohammad Hamad

Technical University of Munich, Germany



## Keynote Talk

### Foundational Security for Your Real-Time Autonomous System Networks

Dr. Friedrich Wiemer

*Senior Security Expert, Robert Bosch GmbH*

Real-time autonomous systems, ranging from vehicles to robots and industrial control systems, rely on robust and secure communication networks. This keynote explores current trends in securing these networks at the foundational Layer 2, addressing both modern and legacy protocols. We discuss the application of MACsec for Automotive Ethernet and introduce CANsec, a novel security solution designed for CAN XL with potential for adaptation to CAN FD. While these advances offer significant improvements, challenges remain, particularly in developing efficient group key agreement and authentication protocols suitable for resource-constrained real-time systems. This presentation highlights these open problems and motivates future research towards building robust, secure, and real-time capable communication networks for the next generation of autonomous systems.

**Dr. Friedrich Wiemer** is a Senior Security Expert at Bosch, specializing in secure communication for in-vehicle networks. His work focuses on driving pre-development activities, architectural design, and innovation in cross-domain ECUs. He contributes to several standardization efforts, including IEEE 802.1AE, OA TC17, and CiA 613-2. Friedrich holds a PhD in Mathematics from Ruhr-Universität Bochum, where his research focused on symmetric cryptography. He is also the CEO and co-founder of cryptosolutions GmbH and has published numerous papers on cryptography and security at leading conferences and journals.



# RTAutoSec

## Foundational Security for Your Real-Time Autonomous System Network

Friedrich Wiemer

Robert Bosch GmbH

XC-CE/ECS1

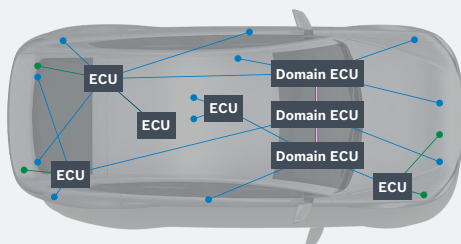
July 8<sup>th</sup>, 2025



### Introduction In-Vehicle Architectures

#### Domain-based Architecture

■ Ethernet ■ CAN ■ LIN

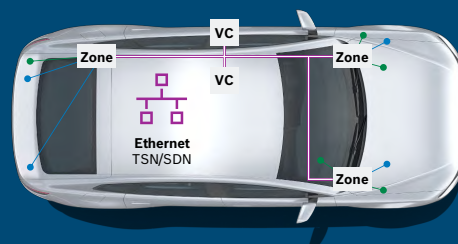


#### Disadvantages

- Vehicle consists of weakly coupled subsystems (domains)
- Deterministic communications between systems only exceptional with extra effort
- Specific individual solutions and software partitioning
- Limited systematic communication network design

#### Zonal-based Architecture

■ Ethernet TSN/SDN ■ CAN ■ LIN



#### Advantages

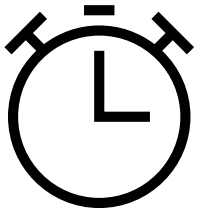
- One unified network with common time zone
- Synchronous actions in multiple ECUs with accuracy < 200 ns at vehicle level
- Systematic communication network design with simulation and validation

TSN Time Sensitive Network, SDN Software Defined Network, VC Vehicle Computer

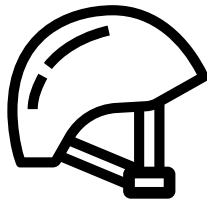


## Introduction

# Foundational requirements for Real-Time Autonomous Systems



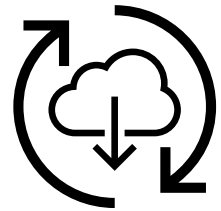
Hard real-time behavior



Safety critical



Immediate start up



Updateability

3

XC-CE/ECS1 | 25.06.2025

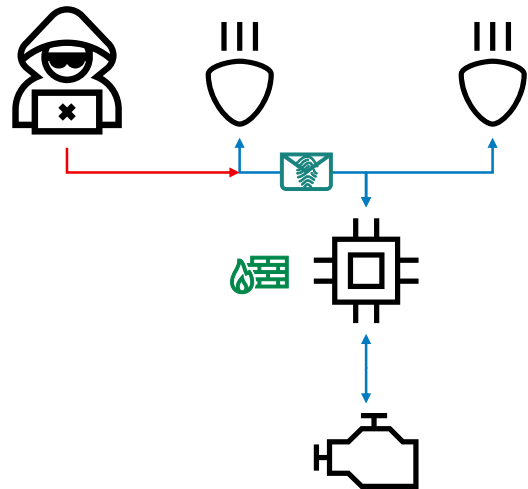
© Robert Bosch GmbH 2025. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.



## Introduction

# Why protect communication

- Attackers exploit most accessible parts
- Exploit chain:
  - Insufficient separation on architectural & network level
  - Insufficient authentication of messages
- Allows attacker to inject & spoof messages  
→ start engine / open doors → steal car
- Holistic security concept should
  - Separate communication with firewalls
  - Authenticate messages with security protocol
  - and more



4

XC-CE/ECS1 | 25.06.2025

© Robert Bosch GmbH 2025. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.





# Introduction

## How to protect communication

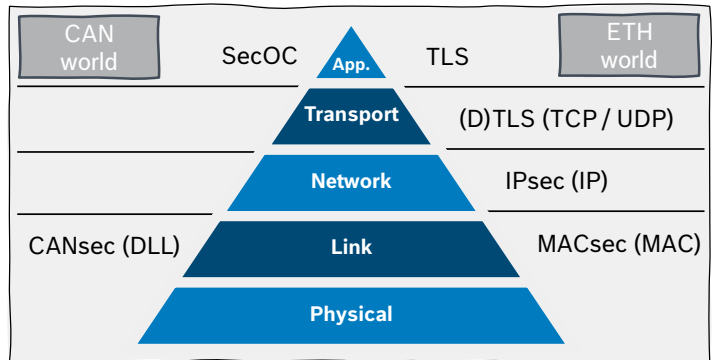
### ■ Mechanisms for secure in-vehicle communication:

- **SecOC** (CAN, FR, ETH),
- **(D)TLS** (ETH) for TCP (UDP)
- **IPsec** (ETH) for IP,
- **MACsec** (ETH) for MAC / LLC
- **CANsec** (CAN) on Layer 2

### ■ Security protocols typically consists of:

- Authentication
- Key Agreement
- Data protection

Communication and security protocols



Confidentiality / Authenticity  
 Standardized / Custom Implementations

Security protocols implemented in HW (PHY or MAC) gives us additional crypto performance.

5

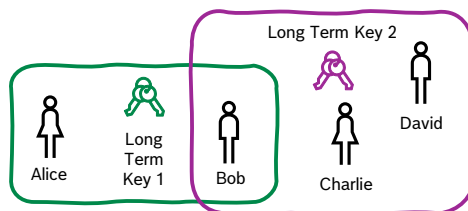
XC-CE/EC51 | 25.06.2025

© Robert Bosch GmbH 2025. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.



# Introduction

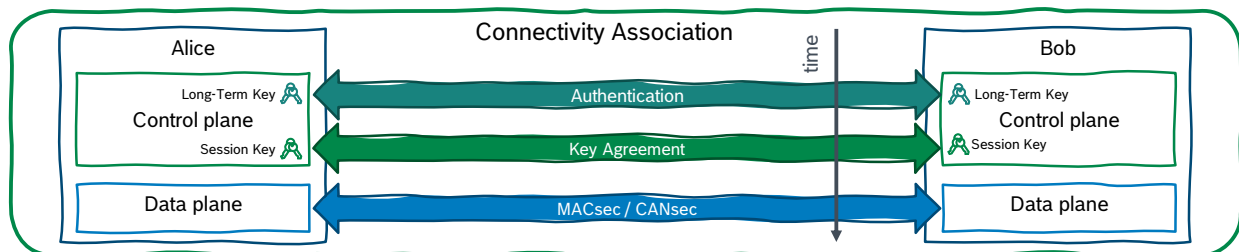
## Communication Architecture



Connectivity Associations

### Three consecutive phases

- Authentication of peers
  - Session key negotiation between peers
  - Secure communication
- } Control plane  
 } Data plane



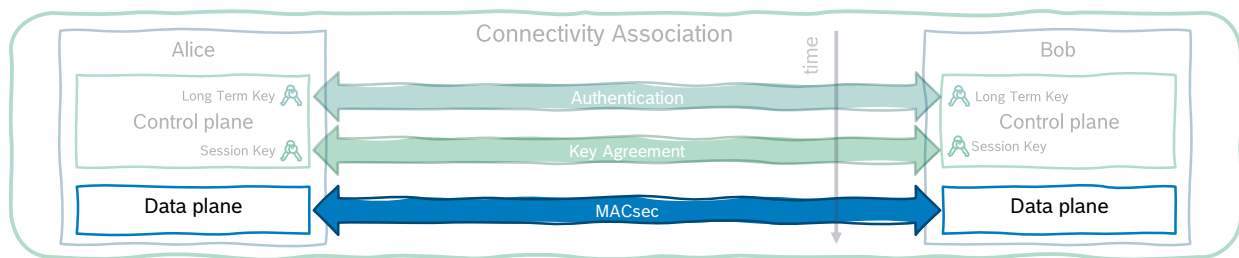
6

XC-CE/EC51 | 25.06.2025

© Robert Bosch GmbH 2025. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

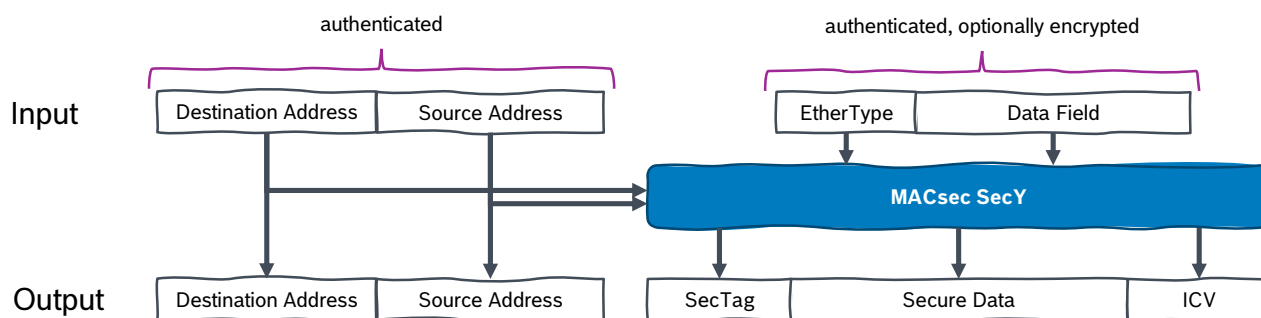


# MACsec



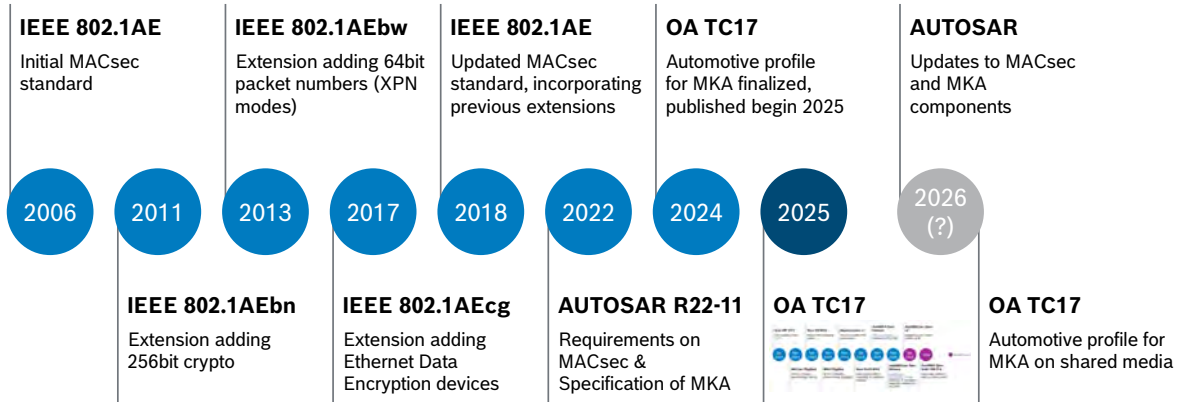
## MACsec High-Level View

Backup: MACsec Details



# MACsec

## Automotive MACsec relevant standards



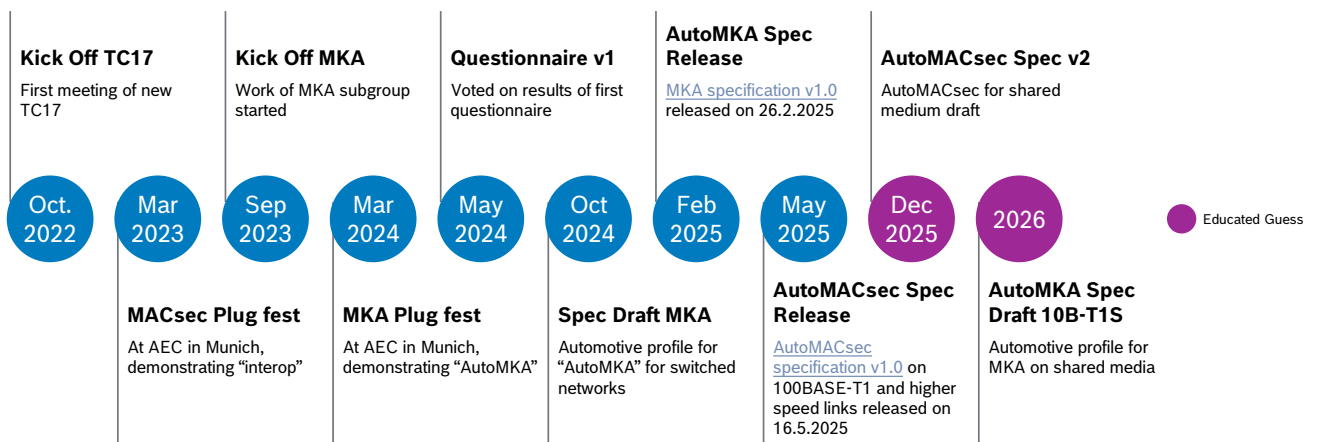
9

XC-CE/EC51 | 25.06.2025  
© Robert Bosch GmbH 2025. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.



# MACsec

## Open Alliance TC17 Automotive Profile for MACsec

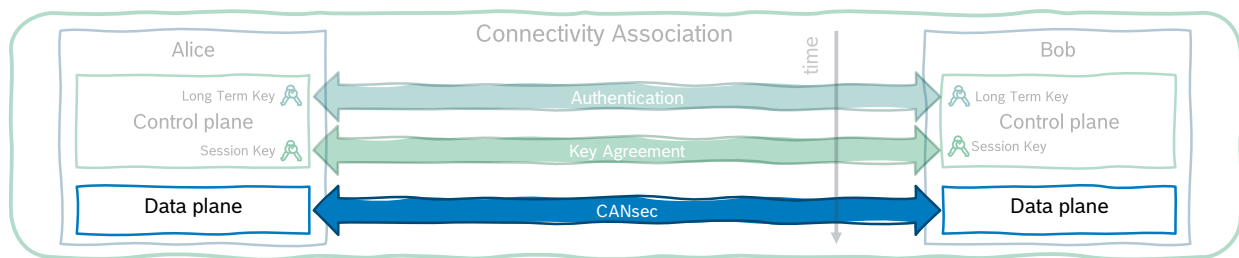


10

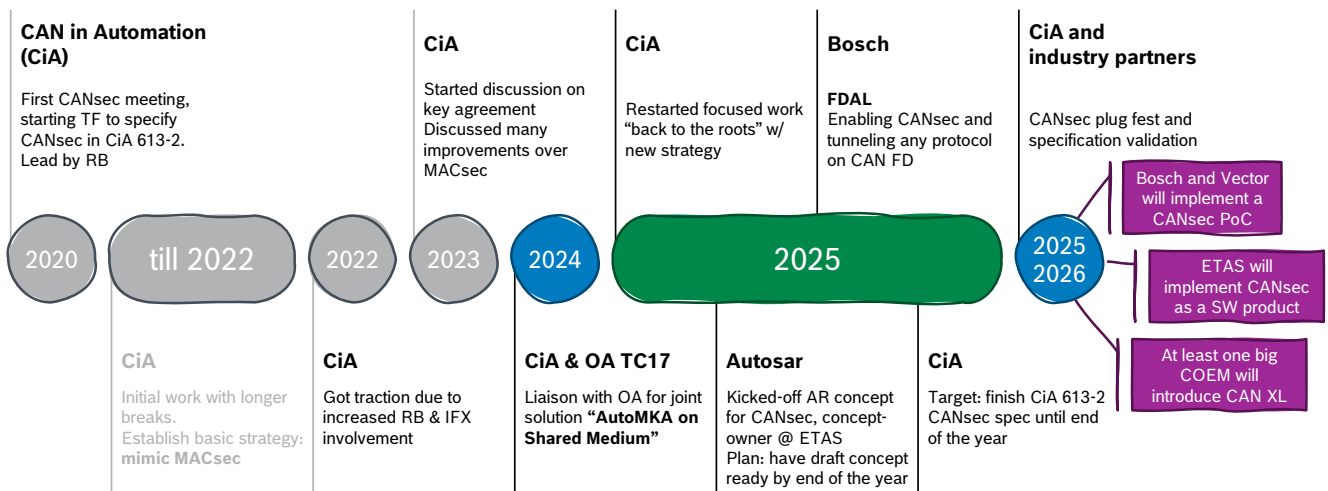
XC-CE/EC51 | 25.06.2025  
© Robert Bosch GmbH 2025. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.



# CANsec



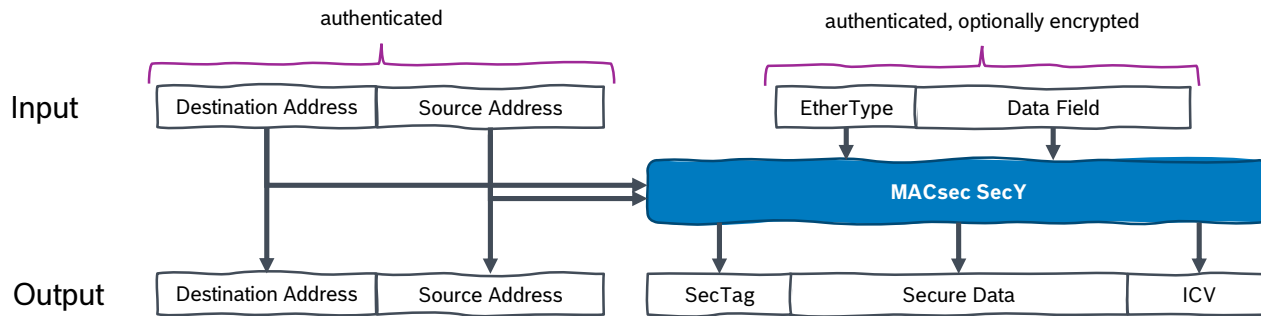
## CANsec Timeline



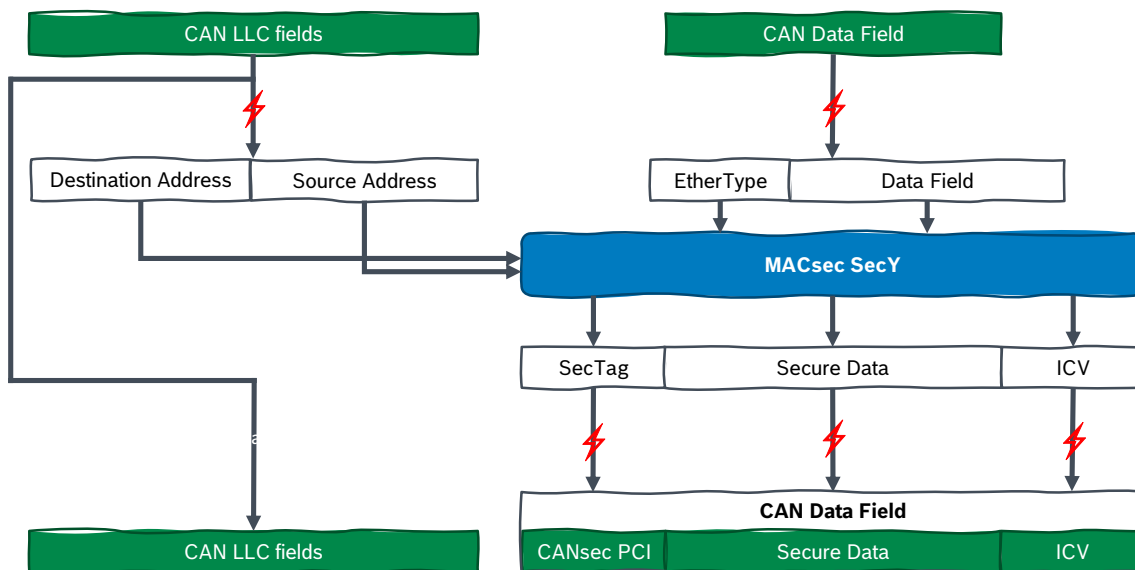
Supporters  
for new strategy: Reuse MACsec



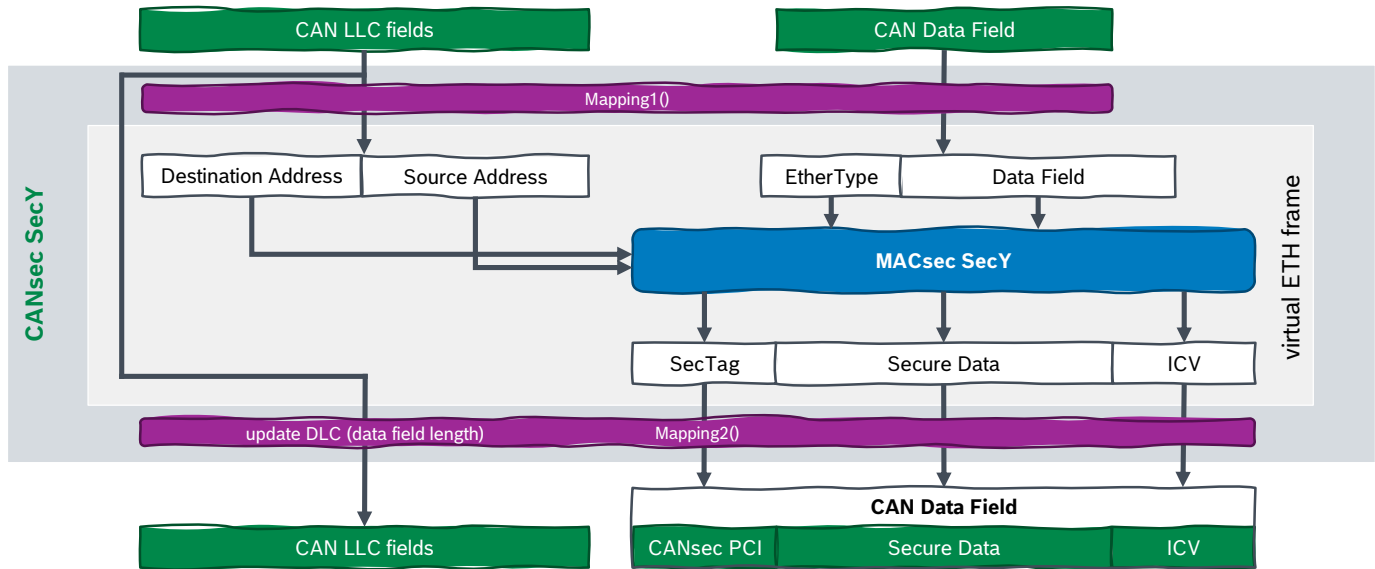
# MACsec High-Level View



# CANsec High-Level View



# CANsec High-Level View



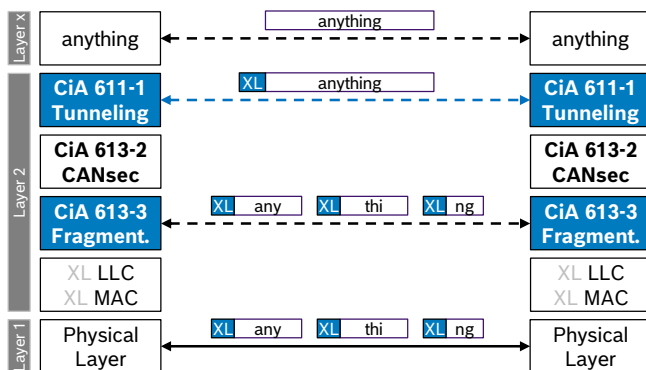
15 XC-CE/EC51 | 25.06.2025

© Robert Bosch GmbH 2025. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

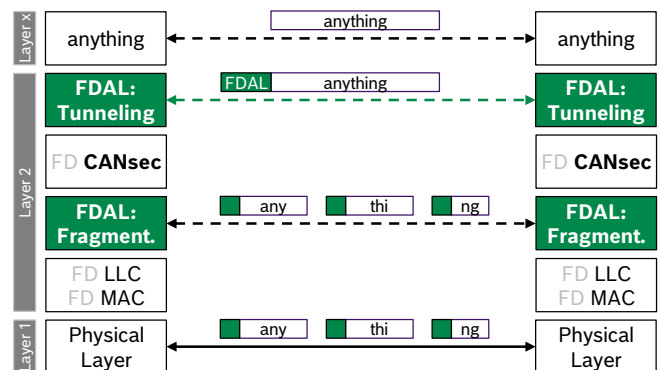


## CANsec for FD FD Adaptation Layer (FDAL)

### Existing: CAN XL solution



### New: enable “anything over FD”



#### Key property: Unified/Similar FD and XL solution

- FDAL Tunneling **is like** CiA 611-1 (CAN XL Tunneling)
- FDAL Fragment. **is like** CiA 613-3 (CAN XL Fragment.)
- FD CANsec **is like** CiA 613-2 (CAN XL CANsec)

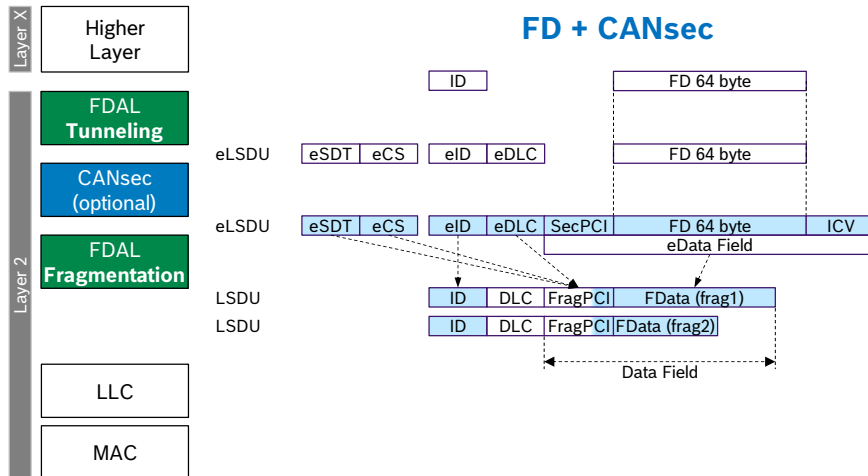
16 XC-CE/EC51 | 25.06.2025

© Robert Bosch GmbH 2025. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.



# CANsec for FD

## FDAL Example: FD tunneling

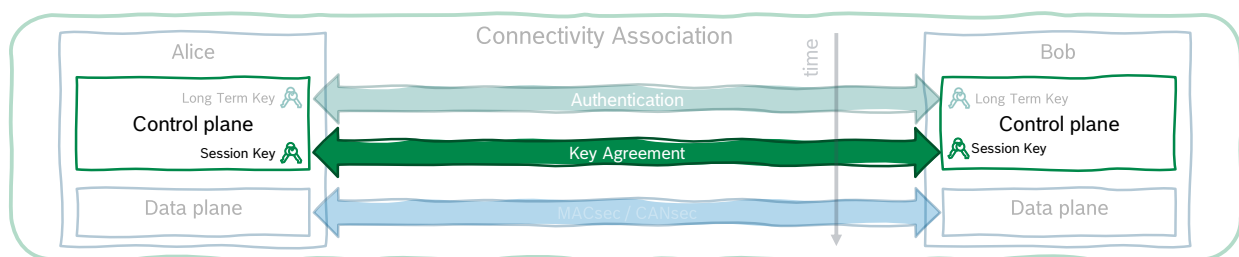


### FDAL Frag. PDU

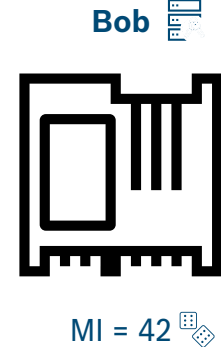
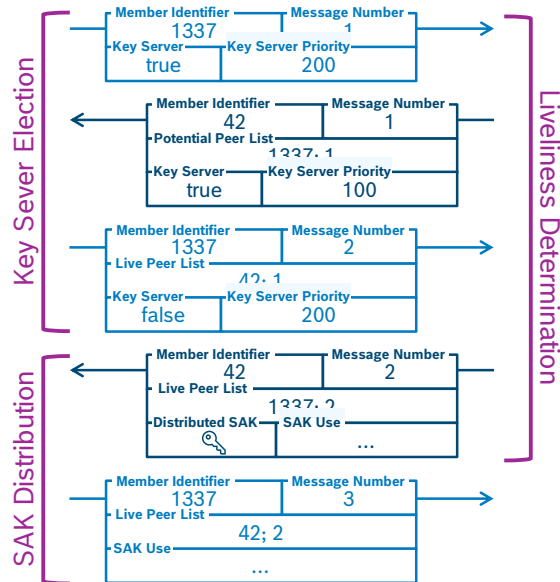
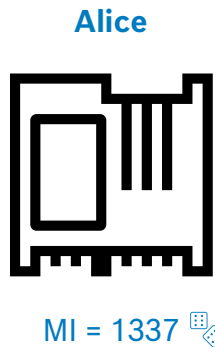
Byte 0				Byte 1	
eSDT	eCS	FF	LF	V	VDB
res.	FCNT				
FData					

Field	Name	Description
eSDT	Extended SDU Type	The payload type
eCS	Extended CANsec	Whether CANsec was applied
First Frame	First Frame	If this is the first fragment of an extended LSDU
Last Frame	Last Frame	If this is the last fragment of an extended LSDU
V	Version	FDAL version
VDB	Valid Data Bytes	To detect padding bytes in the CAN frame
FCNT	Fragment Counter	Detects missing frames
FData	Fragment Data	The actual fragmented payload

## Key Agreement for MACsec & CANsec



# MACsec Key Agreement (MKA) Overview



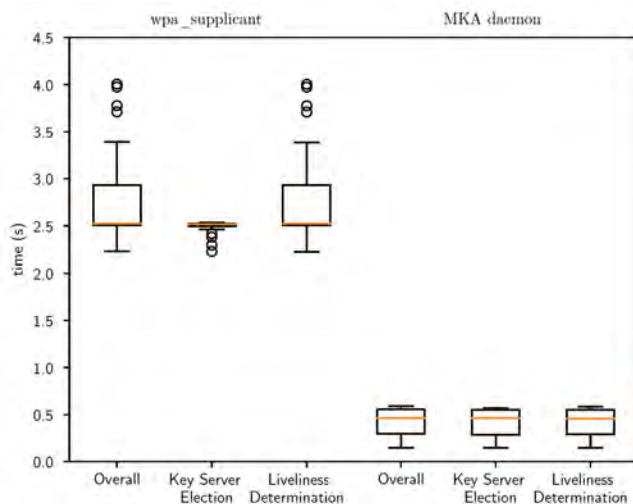
# MACsec Key Agreement (MKA) Automotive MKA for switched Ethernet



Automotive MKA Specification

Automotive MKA Specification for Automotive Ethernet and MACsec

[https://opensig.org/wp-content/uploads/2025/03/OA\\_MACsec\\_Automotive-MKA-v1.pdf](https://opensig.org/wp-content/uploads/2025/03/OA_MACsec_Automotive-MKA-v1.pdf)



**Pre-Selected Key Server**



**Immediate Responses  
and Reset Detection**



**Hello Time Ramp Up**



# MACsec Key Agreement (MKA)

## Automotive MKA for shared medium

- Open Alliance TC17 works on proposals for MKAv4
  - Optimized for shared medium / groups
- Already supported by MKA, but
  - AutoMKA optimizations lead to high bus load for bigger groups
  - Time-to-Key-Agreement (TTKA) is not good enough
- Currently two suggestions in discussion
- Timeline: submit to IEEE in July or November
- Proposal 1: Nonce Spaces
  - Extends Peer List Parameter Set
  - Key Server manages nodes nonce space
  - Nonce space is included in IV derivation

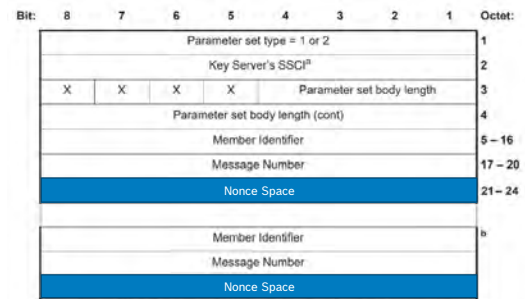
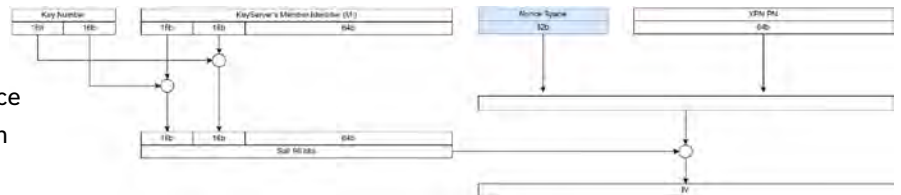
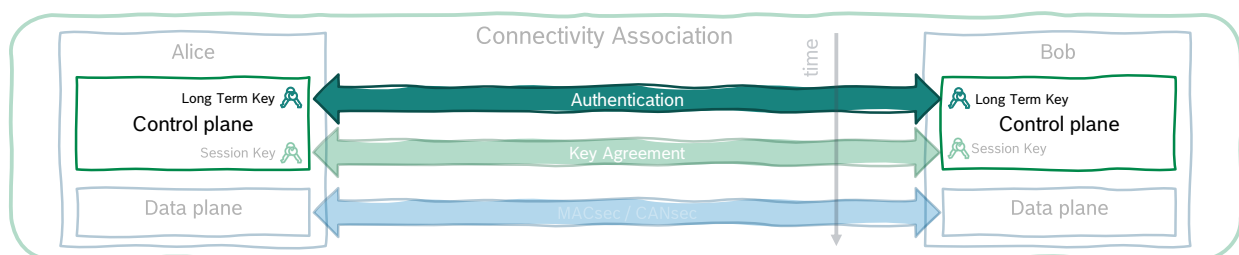


Figure 11-9—Live Peer List and Potential Peer List parameter sets



## Authentication



# Authentication

## Extensible Authentication Protocol



23

XC-CE/ECS1 | 25.06.2025  
© Robert Bosch GmbH 2025. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.



## Authentication

### EAP-TLS

- Best security properties
- Complex
- Slow
- Additional nodes (authentication server)
- Additional crypto (asymmetric & certificates)

### Pre-Shared-Keys

- Requires trusted environment for setup
- Most simple solution
- OEMs have proprietary solutions in place for key distribution in field (from SecOC)
- No additional crypto required
- Replacing ECUs is not straight forward
- No standardized way for pre-sharing-keys

Pre-shared keys look like 80's techn. – but IVNs are engineered networks and thus much simpler than complex, dynamic IT networks.

24

XC-CE/ECS1 | 25.06.2025  
© Robert Bosch GmbH 2025. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.



# Open Problems

## Efficient Group Key Agreements

- Meet very tight timing constraints for TTKA
- Efficiently implementable
  - Simple state-machine for HW-only impl.
  - based on symmetric crypto

## Efficient Key Injection Methods

- Efficiently inject keys in SW-less devices
- Example: MACsec on SW-less edge node
  - How to get keys in there?
  - Standardized approach?

## Efficient Authentication Protocols

- Efficiently implementable in constrained devices
- Optimized for engineered networks
  - Less dynamic
  - More known static configuration



## Wrap up Questions?



“Secure, or not secure,  
that is the question”

—freely adapted from Hamlet

## Efficient Group Key Agreements

- Meet very tight timing constraints for TTKA
- Efficiently implementable
  - Simple state-machine for HW-only impl.
  - based on symmetric crypto

## Efficient Authentication Protocols

- Efficiently implementable in constrained devices
- Optimized for engineered networks
  - Less dynamic
  - More known static configuration

## Efficient Key Injection Methods

- Efficiently inject keys in SW-less devices
- Example: MACsec on SW-less edge node
  - How to get keys in there?
  - Standardized approach?

## Invited Talk

### **Dynamic access rights management via trust of mutable attributes in realtime systems**

Prof. Andrea Saracino  
*Scuola Superiore Sant'Anna Pisa*

Abstract: This talk will discuss an application of the usage control paradigm to real time autonomous systems, discussing the relevance of real time scheduling for access control decision, and the effect on real time systems of delay introduced by policy evaluation procedures. This issue is exacerbated when decision depends from attributes whose attribute managers are not reliable, or might be unavailable at times. The talk will present a use case on a V2V application, discussing distributed trust and attribute freshness as probabilistic components of the risk of taking a decision when attribute values cannot be retrieved at real time.



RTAutoSec, 8 July 2025,  
Brussels, Belgium

# Dynamic access rights management via trust of mutable attributes in real- time systems

Andrea Saracino, Scuola Superiore  
Sant'Anna Pisa

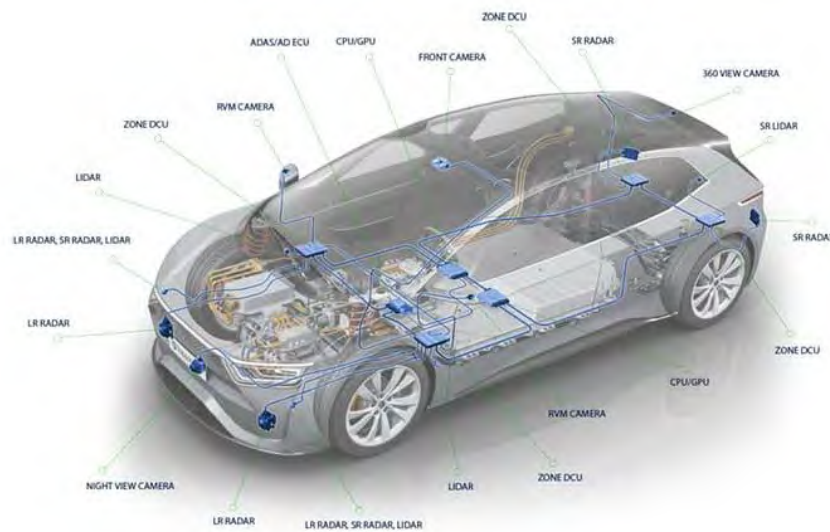


MEDIATE



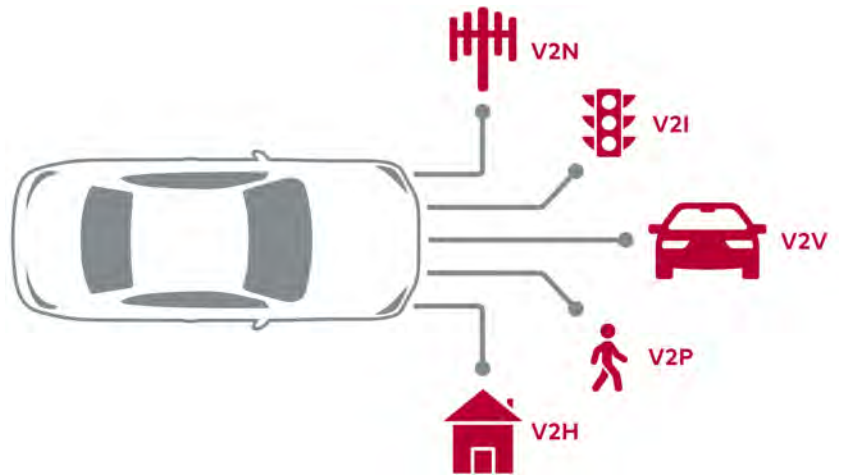
**Sant'Anna**  
Scuola Universitaria Superiore Pisa

## Autonomous Systems and Access Rights



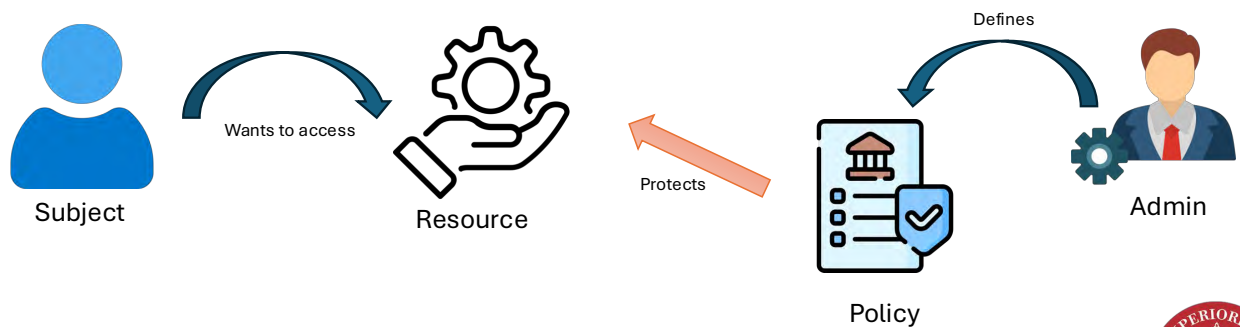
## V2X communication

- Possibility to control remotely car functionalities.
- Management of priorities for commands coming from different entities



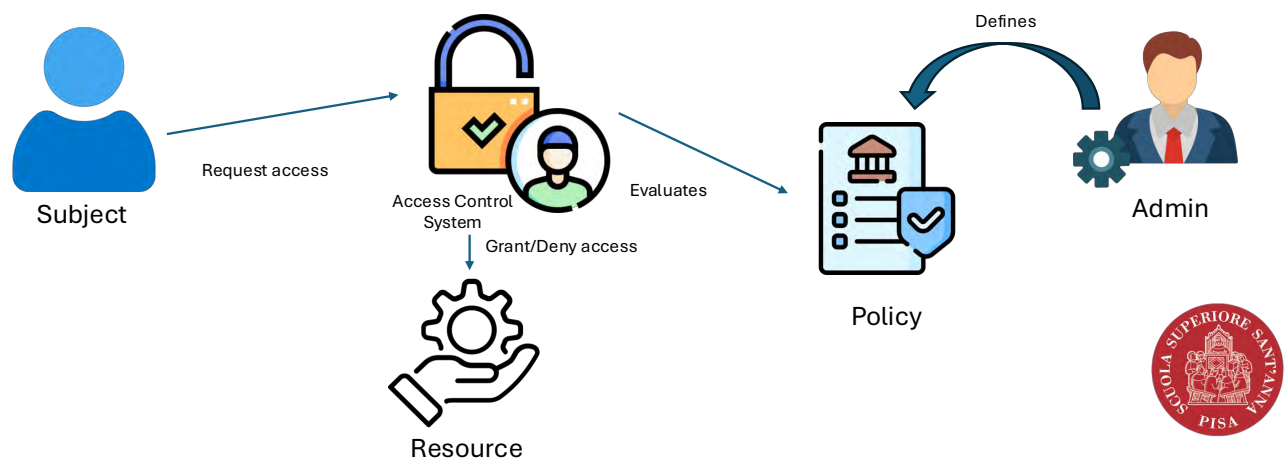
## Controlling Access Rights

- Using formal methods to assess if a *subject* can/cannot access a *resource* given a specific context.



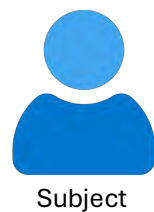
# Controlling Access Rights

- Using formal methods to assess if a *subject* can/cannot access a *resource* given a specific context.



## Attacker model

- Honest but curious



- If the system allows subject to get more privileges, he will take them



# Attribute Based Access Control

- Several Access Control models
  - What is used to take the PERMIT/DENY decision?
- Attribute Based Access Control (ABAC)
- Attributes:
  - Subject
    - Role
    - Age
    - Reputation
  - Resource
    - Recorded accesses
    - Owner
  - Context
    - Temperature
    - Time

RBAC  
BLP  
IBAC/ACL  
ABAC



## Relevance of real time to access control

- Decision on access has to be taken before specific deadlines
  - Degradation of user experience (overhead)
  - Decision arrives when not useful anymore
    - Missed deadline
- Decision time:
  - Time needed to assess a subject request against a policy and take a decision.
  - Might not be feasible depending on real time requirements of the system







## Access Control

- Once the Access to a resource is granted it cannot be revoked.
- Resource is released to the subject until he voluntarily close the access session.
- Designed for instant operations
  - Open a lock
  - Money transfer
  - Braking



## Lasting Operations

---

- Turning on the A/C
- Streaming a content
- Staying logged in a terminal
- Allowing fuel engine transmission (full hybrid car)
- What if access conditions change?



# Mutable Attributes

- Attributes whose value changes over time
  - Temperature
  - Geolocation
  - Number of people in room
  - Number of cars in a lane
  - ...
- What if decision was PERMIT under a certain condition and the access session is still open when mutable attributes change their value?

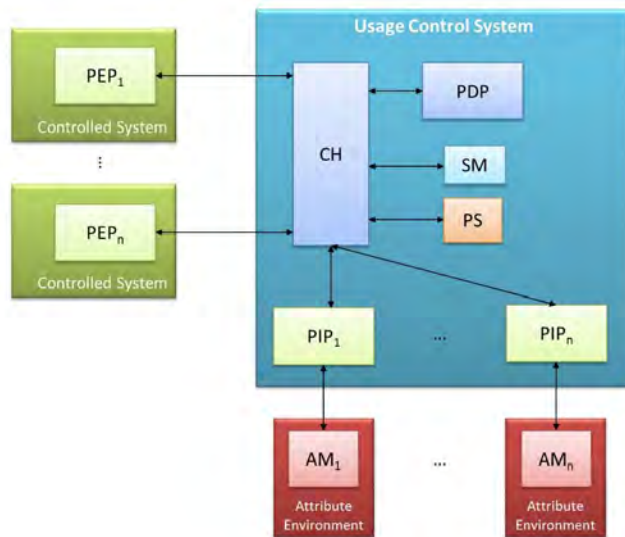


## Usage Control

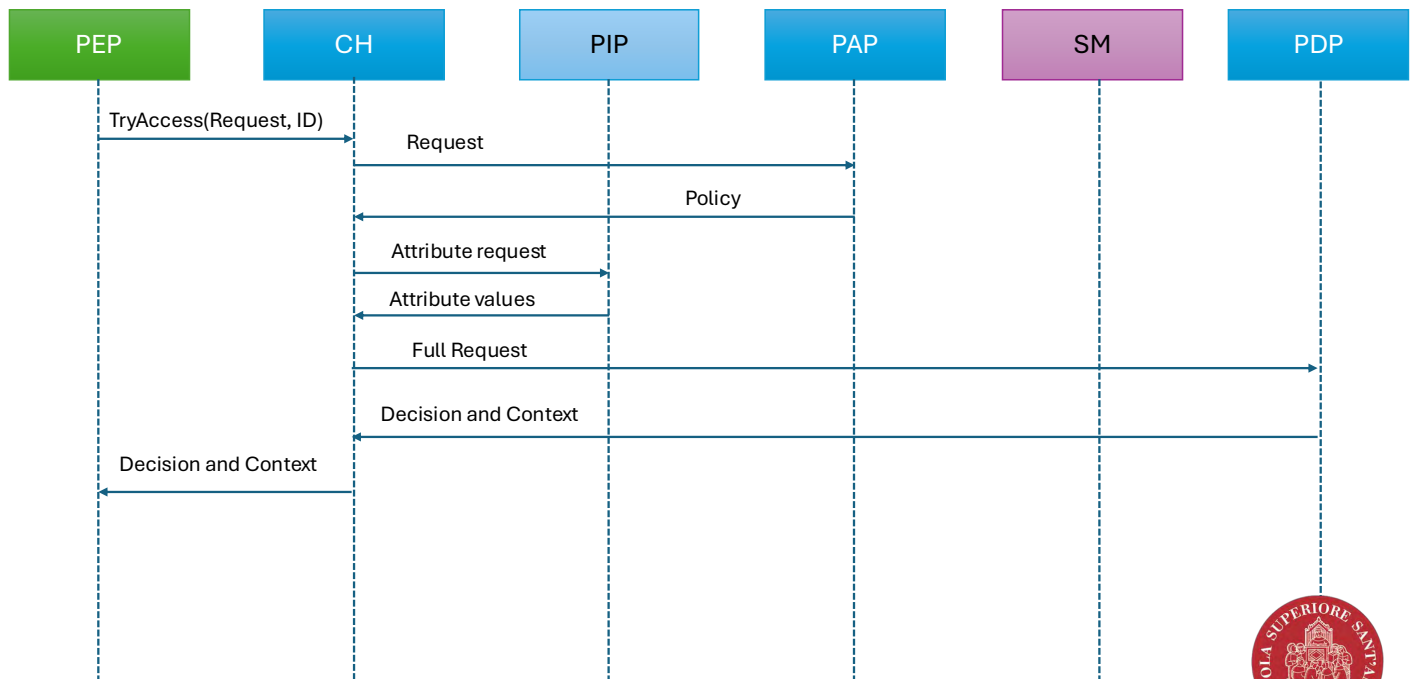
- Introduces the possibility to REVOKE a granted access when a session evaluated to PERMIT has a re-evaluation to DENY
- Continuous re-evaluation as mutable attribute change their value
- Retaining control on the resource

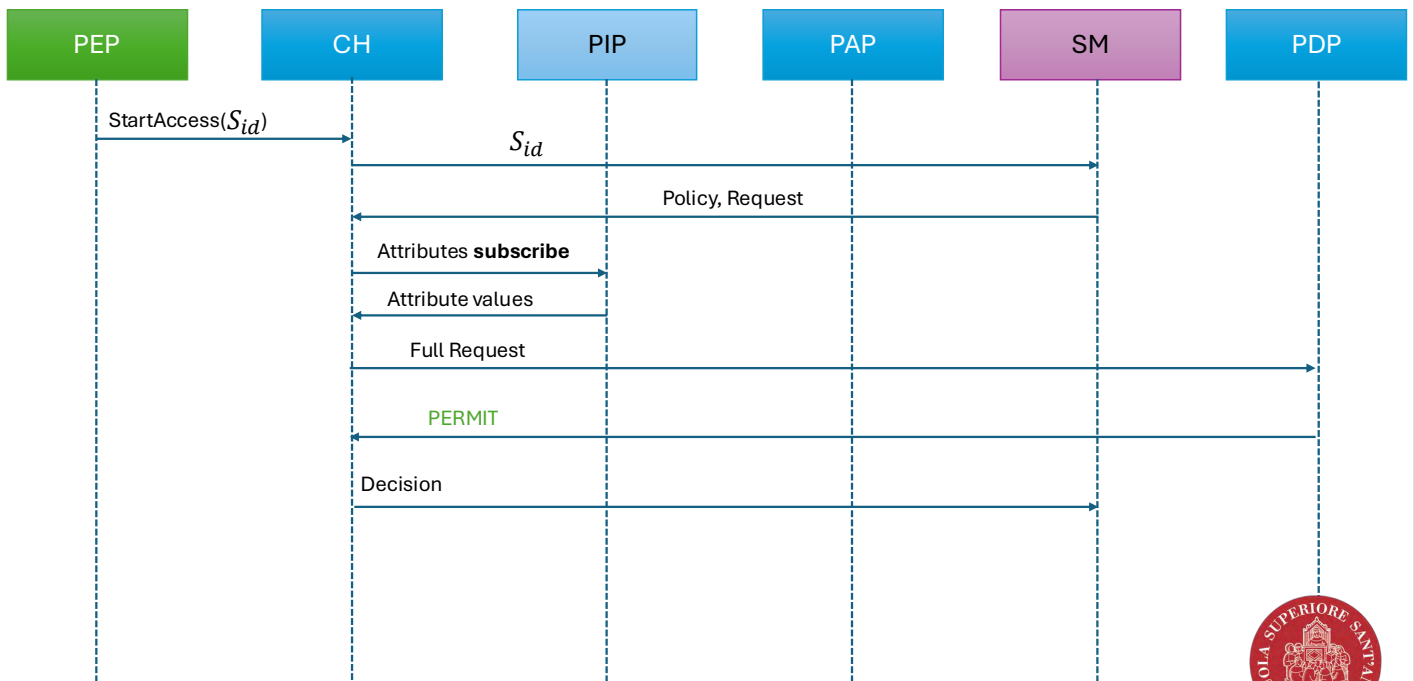
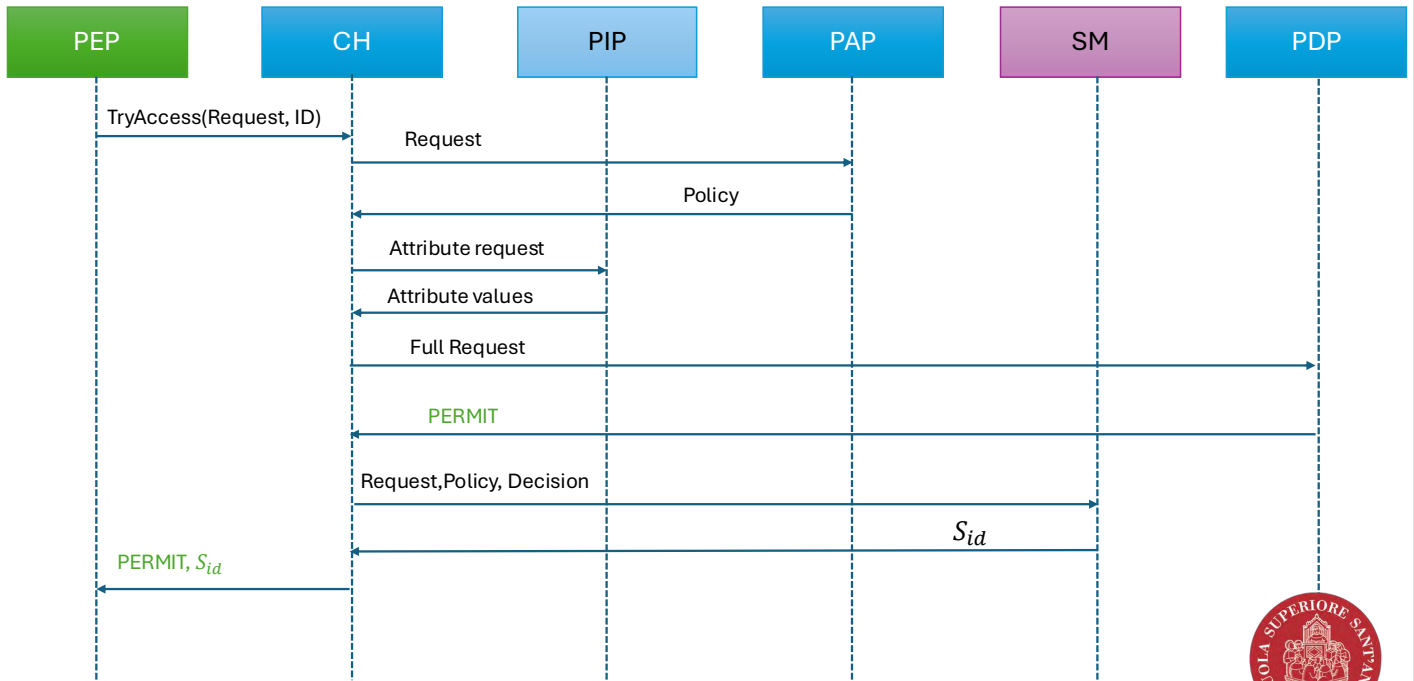


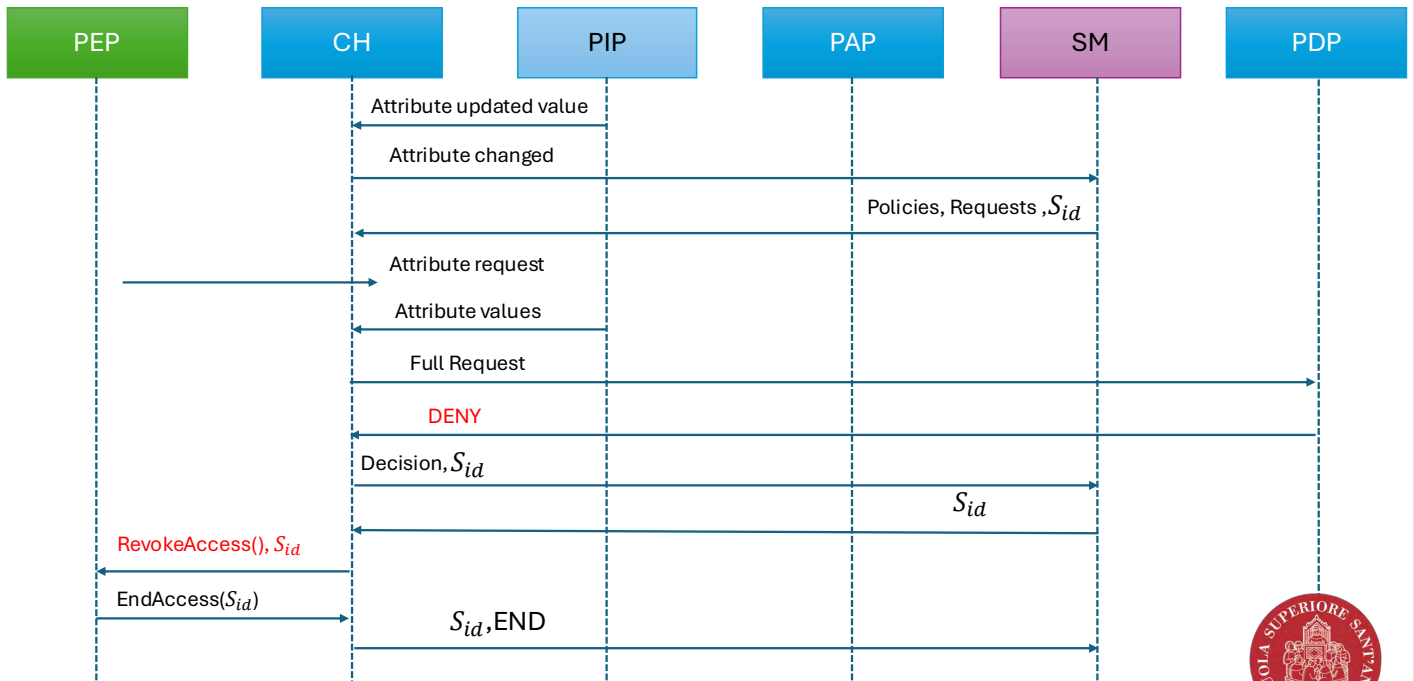
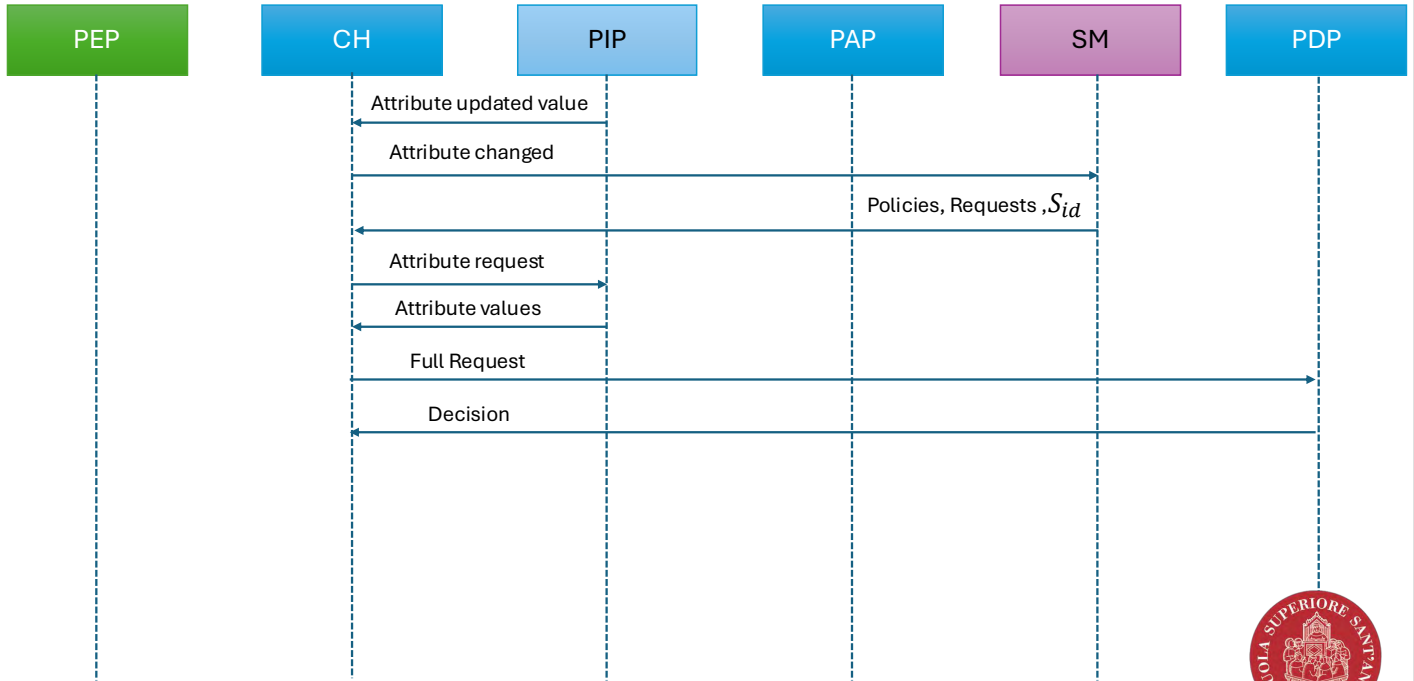
# The Usage Control Framework



- Policy Enforcement Point [PEP]: Manages the access to the resource being able to grant access and block access to it
- Policy Decision Point [PDP]: Evaluates policy VS request and make decision
- Attribute Manager [AM]: External source of attribute values.
- Policy Information Point [PIP]: Retrieves/updates attribute values on Ams
- Policy Administration Point [PAP/PS]: Stores and retrieves policies
- Session Manager [SM]: keeps track of active access sessions.
- Context Handler [CH]: Switches messages among other components.







# Reliability of mutable attributes value

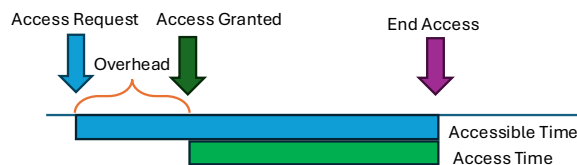
- Lack of subscription possibility
  - Polling interval delay
  - Attribute Manager not available
    - Missing connection
    - Excessive workload
    - DoS attacks
- What happens if attribute value cannot be retrieved?
  - Evaluation is Indeterminate:
    - Request cannot be evaluated

**MISSING**



## Measuring Performance of Access Control

- Delay introduced by the access request evaluation: Overhead
- Different factors:
  - Policy evaluation
  - Attribute retrieval
  - Network time
  - Request composition
- Effect on user experience:
  - **Access Delay:** Time where the client had the right to access the resource but is denied access to it because of needed evaluation.

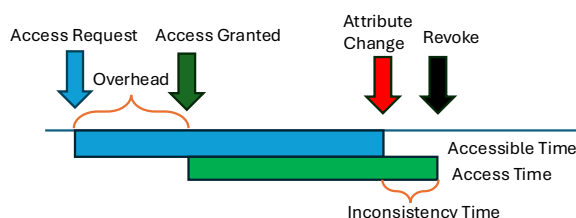






## Measuring Performance of Usage Control

- **Inconsistency Time:** amount of time in which the client would **have not** the right to access a specific resource, but he is still accessing it.
- Await for revocation.
- Delay introduced by different elements:
  - Attribute notification time
  - Policy reevaluation time
  - Session retrieval time
  - Network communication



## Handling missing attribute values

- Using last available value to take the decision
  - Caching
  - Reliability depending from freshness of the attribute
    - More time passed since last caching, higher the probability the value changed.

$$p(\tau \leq t) = \int_0^t f_\tau(t) dt = 1 - e^{-\lambda_i t} \quad \text{Probability that value changed}$$

$$E[F(A_i, t)] = 0 \cdot (1 - e^{-\lambda_i t}) + 1 \cdot e^{-\lambda_i t} = e^{-\lambda_i t} \quad t \in (0, T) \quad \text{Expected Freshness}$$

- Defining a threshold for expected freshness depending from the use case



# How to increase freshness?

- Last caching time: last evaluation on attribute done by UCS
- Periodic polling
  - Consumes resources
  - Does not work if AM not reachable
- Relying on more agents to cache attribute values and their freshness.



## Proposed Approach

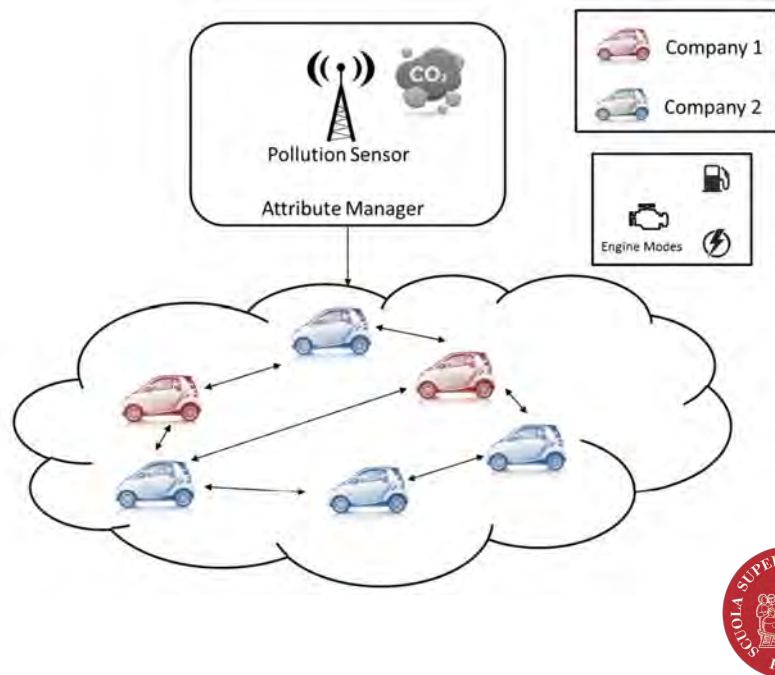
- Collaborative, distributed and decentralized model where different Usage Control systems cooperate to increase the availability of attributes provided by faulty Attribute Managers
  - Maps well IoT Systems
  - Does not require a centralized root of trust
  - Can be used in multi-stakeholder scenarios
    - Could be used to map V2X systems





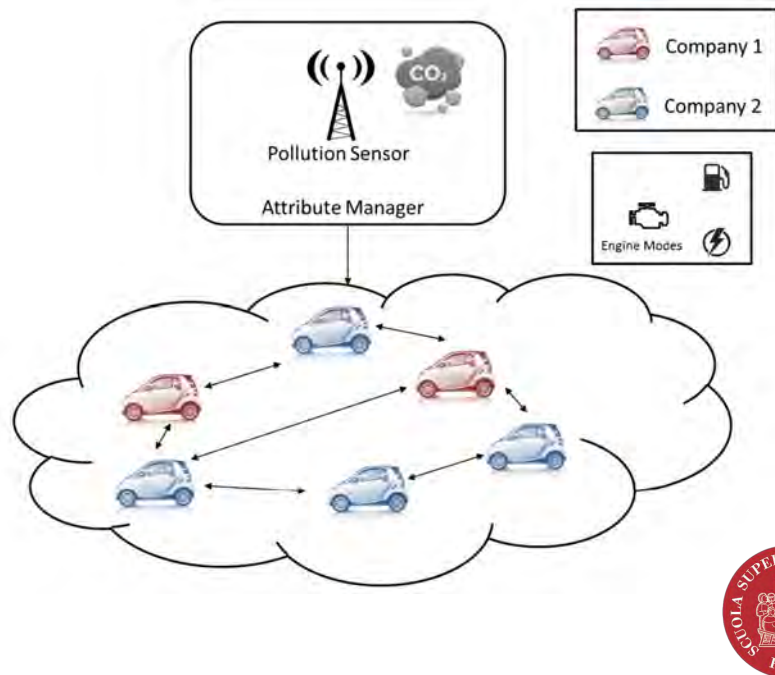
# Reference Scenario

- Fuel engine should be activated only if pollution level is below a threshold
- Each car has his own UCS instance
- Company 1 and 2 are competitors
- Fuel engine is considered more convenient for the company (less refilling time)

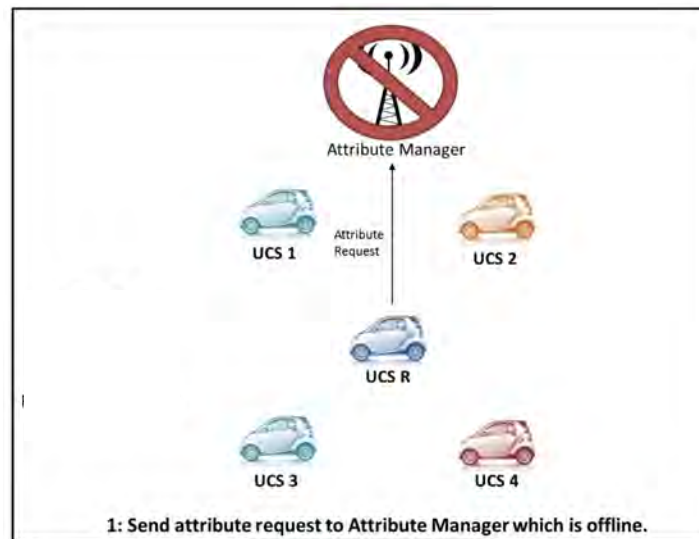


# Reference Scenario

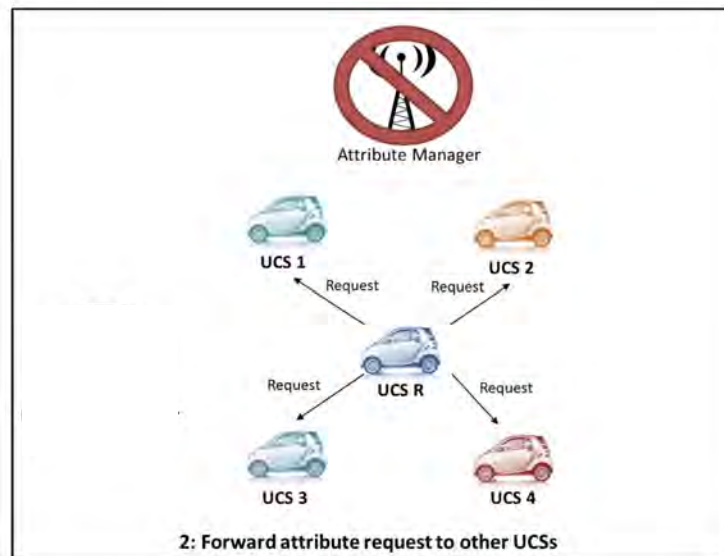
- Communication happens via mesh network among cars – V2V
- Car can read pollution sensor via Bluetooth if in range
- Car of Company 1 and Company 2 cache attribute values



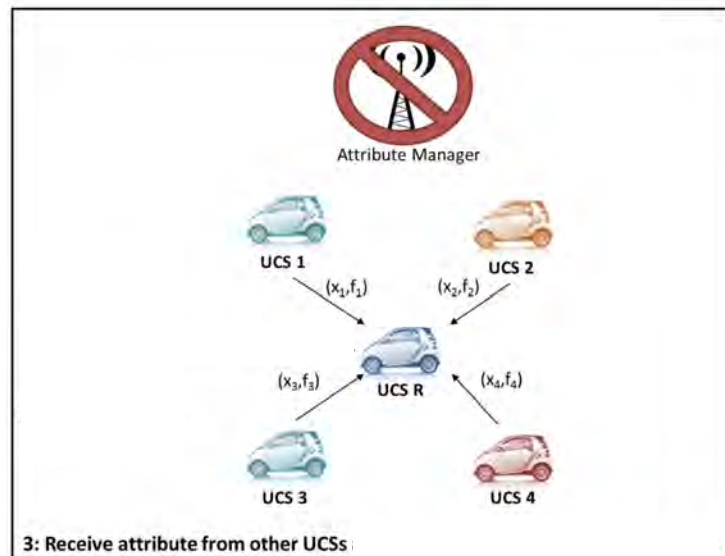
## Workflow [1]



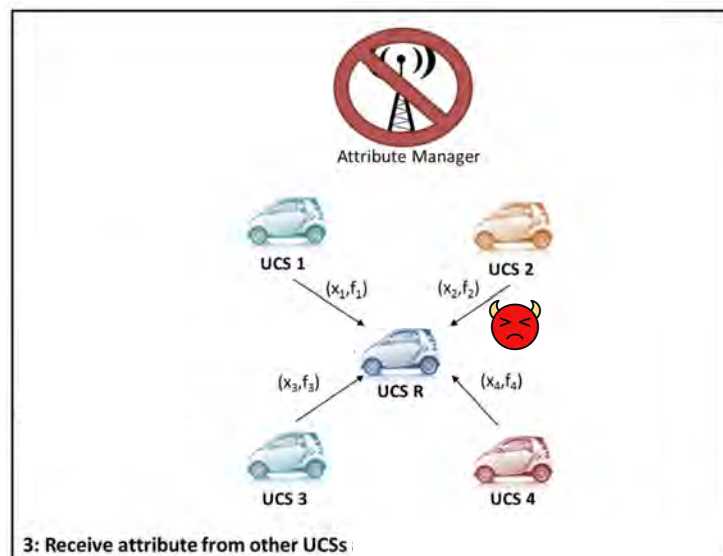
## Workflow [2]



## Workflow [3]



## Malicious Reports



# Malicious Reports

- Maliciously postponing the usage of fuel engine
- Causing a misbehavior because of a false report
  - Reputation tampering
- Getting unfair competitive advantage



# Solutions

- Majority voting
  - Using more than one report
- Root of trust
  - Not present in distributed systems
- Distributed trust mechanism
  - Does not need a root of trust
  - Pairs well with majority voting



# Dynamic Reputation

- **Belief**: The communicated reading from an agent has been verified as correct
- **Disbelief**: The communicated reading from an agent has been verified as non-correct
- **Uncertainty**: The communicated reading from an agent cannot be verified

$$r(t) = b(t) - d(t) - u(t)$$



## Reputation Update

---

**Algorithm 1** Updating reputation  $r_l$

---

```
 $r_l = b_l - d_l - u_l$   
 $b_l + d_l + u_l = 1$   
for all  $u \in U^j$  do  
  if  $u$  provides a correct value then  
     $b_l = b_l + \Delta_b$   
     $u_l = u_l - \frac{\Delta_b}{2}$   
     $d_l = d_l - \frac{\Delta_b}{2}$   
  else  
    if  $u$  provides a non fresh value then  
       $u_l = u_l + \Delta_u$   
       $b_l = b_l - \Delta_u$   
    end if  
  else  
    if  $u$  provides a malicious value then  
       $d_l = d_l + \Delta_d$   
       $b_l = b_l - \frac{\Delta_d}{2}$   
       $u_l = u_l - \frac{\Delta_d}{2}$   
    end if  
  end if  
end for
```

---



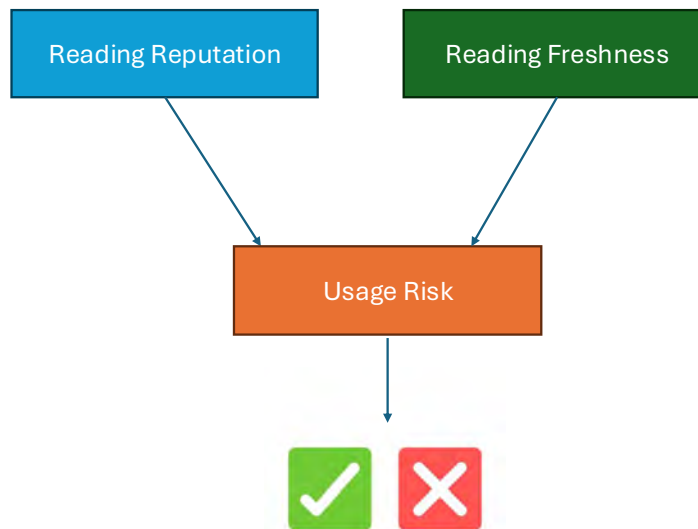


## Reputation value storage and retrieval

- Peer to Peer Distributed Hash Table infrastructure
- Each agent can provide value for a specific pollution reading based on location
- Verified with readings of other agents with similar location
  - Majority voting
- Reputation update
  - Agents cannot provide reading if reputation is below threshold



## Risk



## Validity

$$\mathcal{V}(v_k, t) = \frac{1}{n} \sum_{j=1}^n \delta_j(v_k, t) \times f_j(A, t) \times r_j(t)$$

$$\delta_j(v_k, t) = \begin{cases} 1 & \text{if } v_k = \pi_1(U_j(A, t)) \\ 0 & \text{Otherwise} \end{cases}$$



## Experimental Evaluation

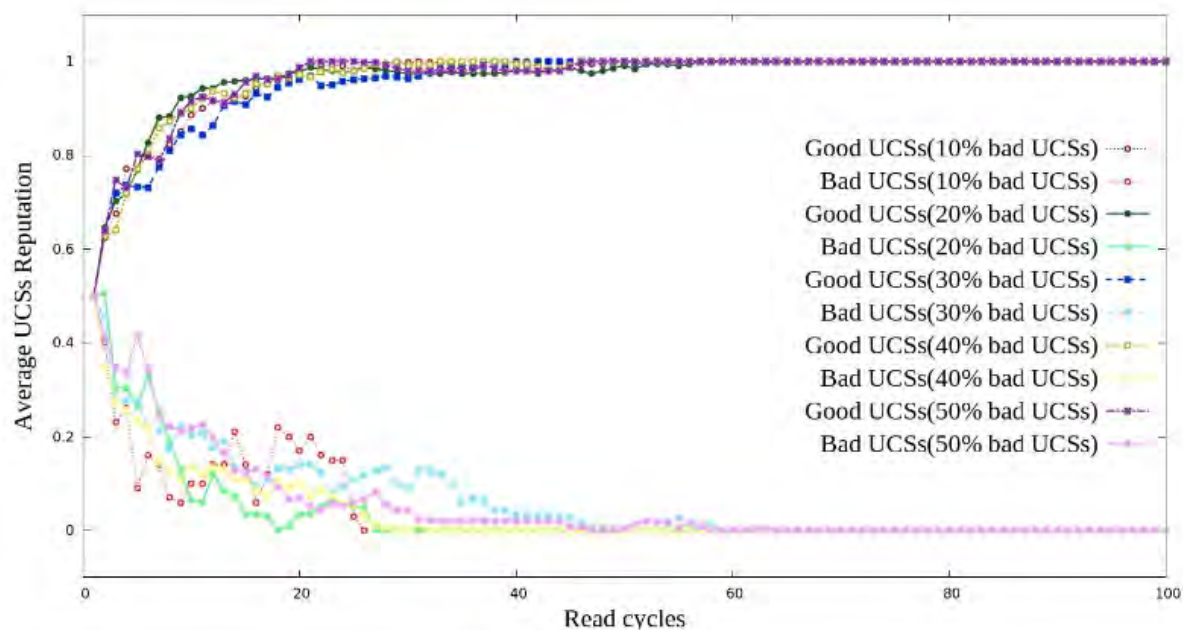
Simulation testbed  
with 100 agents.

Varying percentage  
of malicious users  
providing fake  
readings

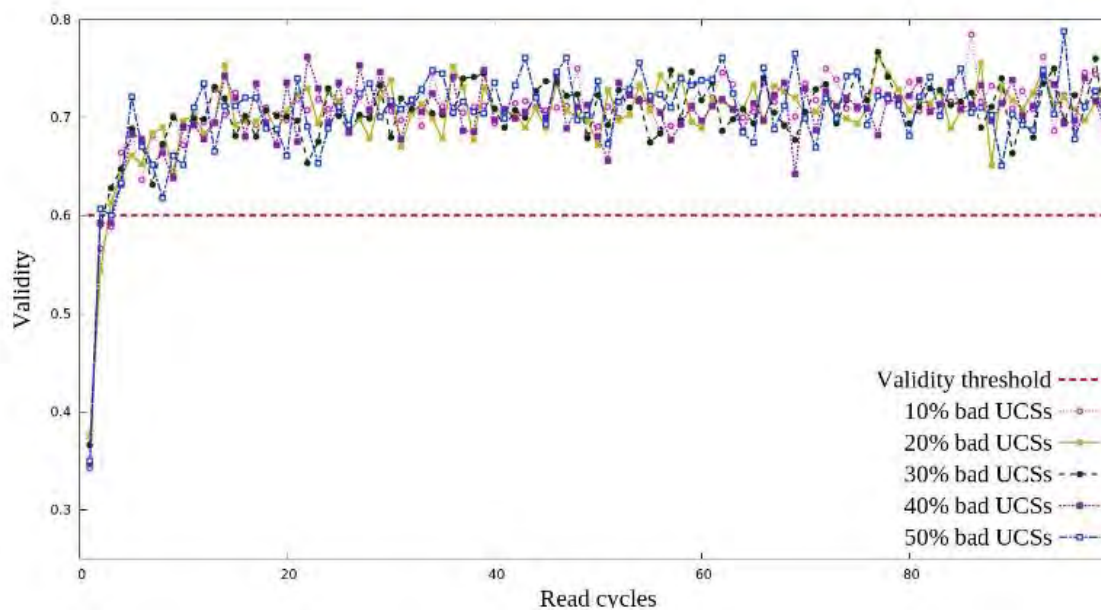
Hypothesis of  
colluding  
malicious users



# Experimental Evaluation

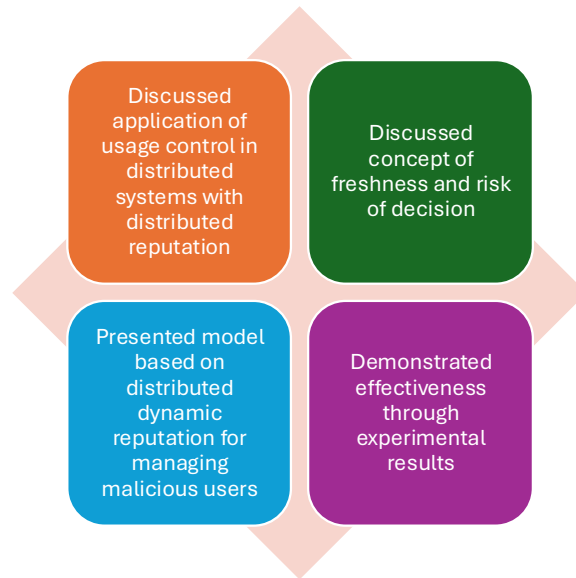


# Experimental Evaluation





## Conclusion



## Thank You for Your Attention



[andrea.saracino@santannapisa.it](mailto:andrea.saracino@santannapisa.it)



**Sant'Anna**  
Scuola Universitaria Superiore Pisa

# Acknowledgement

- This presentation is considered part of the dissemination activities for the Horizon Europe MEDiate project GA 101168465



**MEDIATE**



## Invited Talk

### Secure Time Synchronization in Packet-Switched Networks

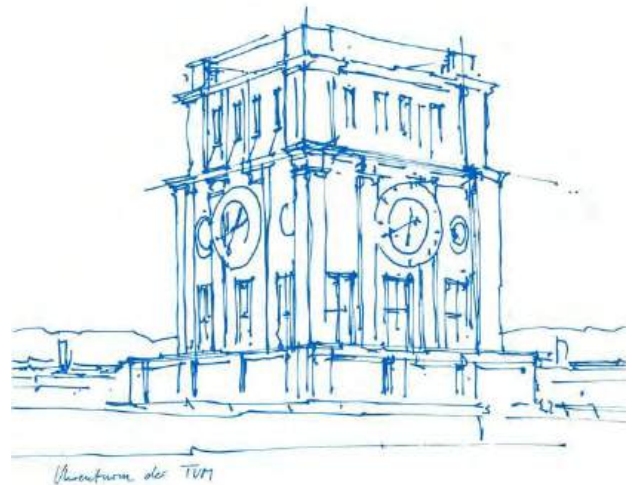
M. Sc. Andreas Finkenzeller  
*Technical University of Munich*

**Abstract:** Time synchronization in packet-switched networks has become essential for many modern applications and technologies, including Smart Grids, Time-Sensitive Networking (TSN), and 5G networks. Along with the required microsecond-level accuracy, reliability and security are of ever greater concern. While the Precision Time Protocol (PTP) can satisfy the accuracy requirement in trusted environments, it becomes unreliable in the presence of specific cyberattacks. The vulnerability arises from the protocol's susceptibility to time delay attacks, which deliberately introduce packet delays for critical PTP event messages. This enables attackers to manipulate targeted clocks without detection. Various presented attack strategies and a related reference implementation with additional evaluation on a hardware testbed demonstrate the attack feasibility. While several mitigation approaches exist, they are not optimal and cannot sufficiently mitigate sophisticated delay attacks. A newly presented method to detect and counteract delay attacks against PTP is based on cyclic path asymmetry measurements over redundant paths. In addition to the underlying theoretical concepts, the talk presents the PTPsec protocol as an extension of the latest IEEE 1588-2019 standard to secure the conventional PTP protocol. A complementary implementation and related evaluation results emphasize the effectiveness of the proposed solution. The talk concludes with a perspective on open research topics to further improve secure time synchronization over packet-switched networks.

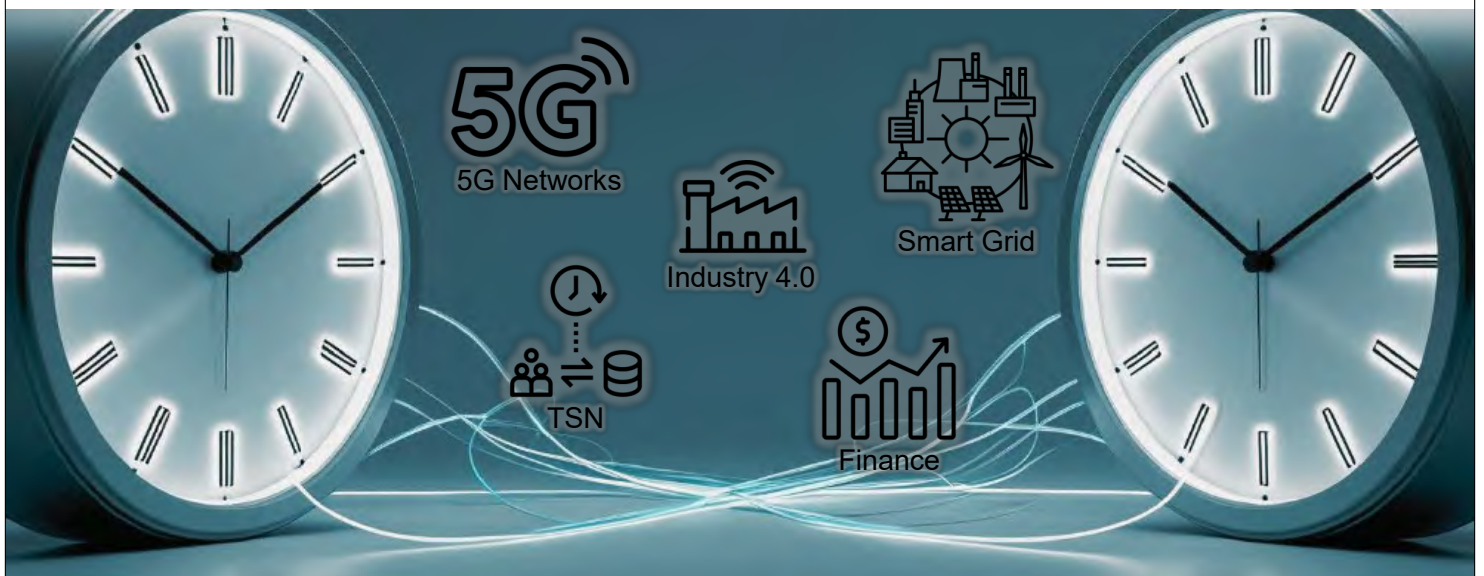
# Secure Time Synchronization in Packet-Switched Networks

M. Sc. Andreas Finkenzeller

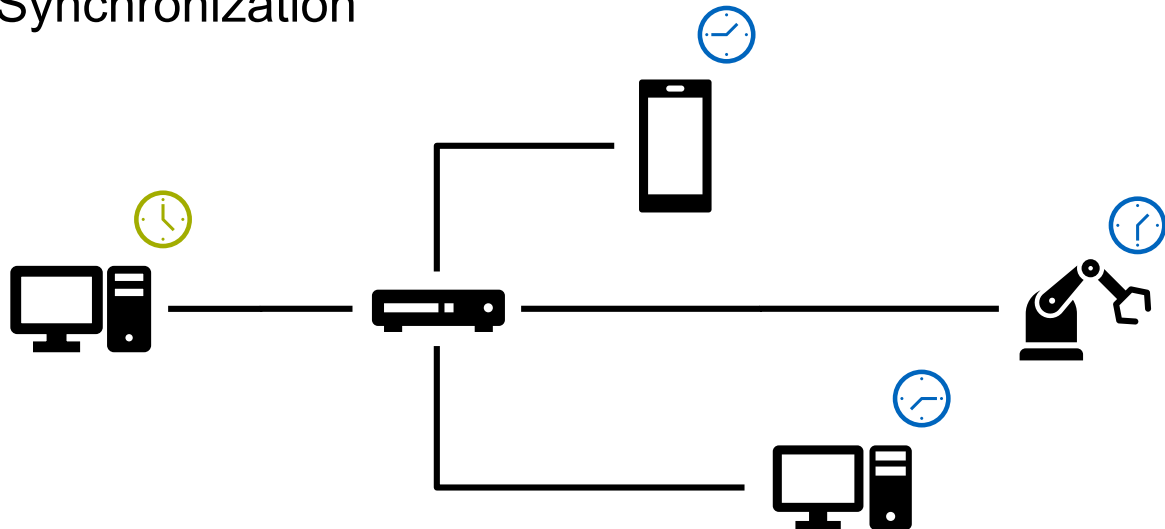
Brussels, 8<sup>th</sup> July 2025



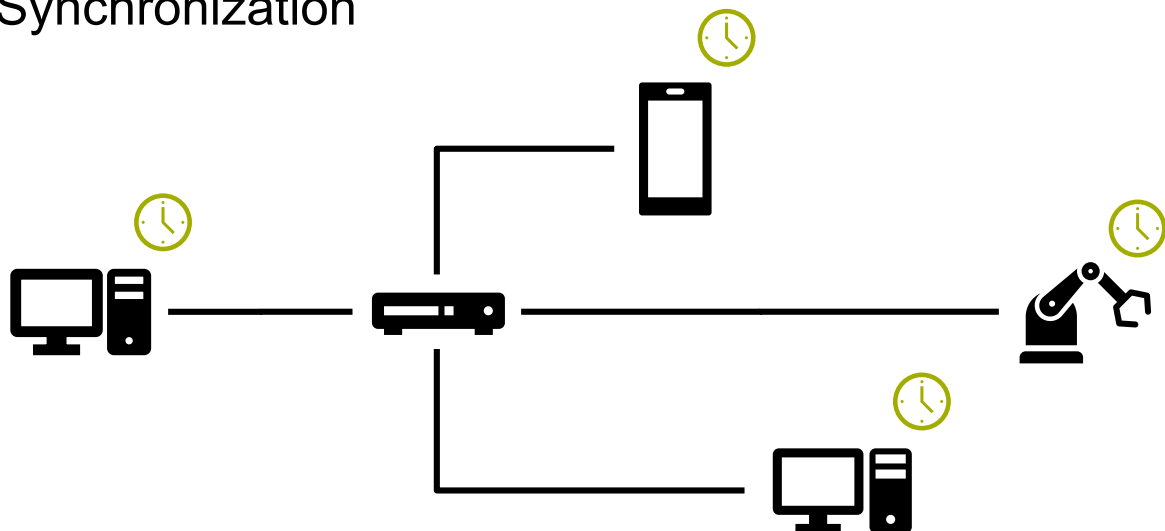
## Motivation



# Time Synchronization



# Time Synchronization



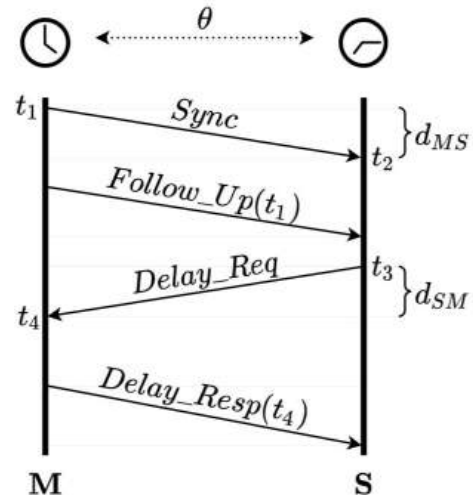
## Precision Time Protocol (IEEE 1588)

- Four timestamps:  $t_1, t_2, t_3, t_4$
- Assumption:  $d_{MS} = d_{SM}$
- Clock offset

$$\theta = \frac{(t_2 - t_1) - (t_4 - t_3)}{2}$$

- Path delay

$$d = \frac{(t_2 - t_1) + (t_4 - t_3)}{2}$$



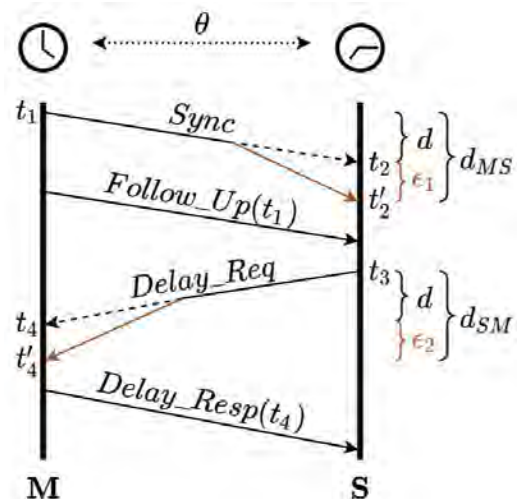
## Time Delay Attack

- Assumption:  $d_{MS} = d_{SM}$
- With  $\epsilon_1 \neq \epsilon_2$ , assumption breaks
- Clock offset

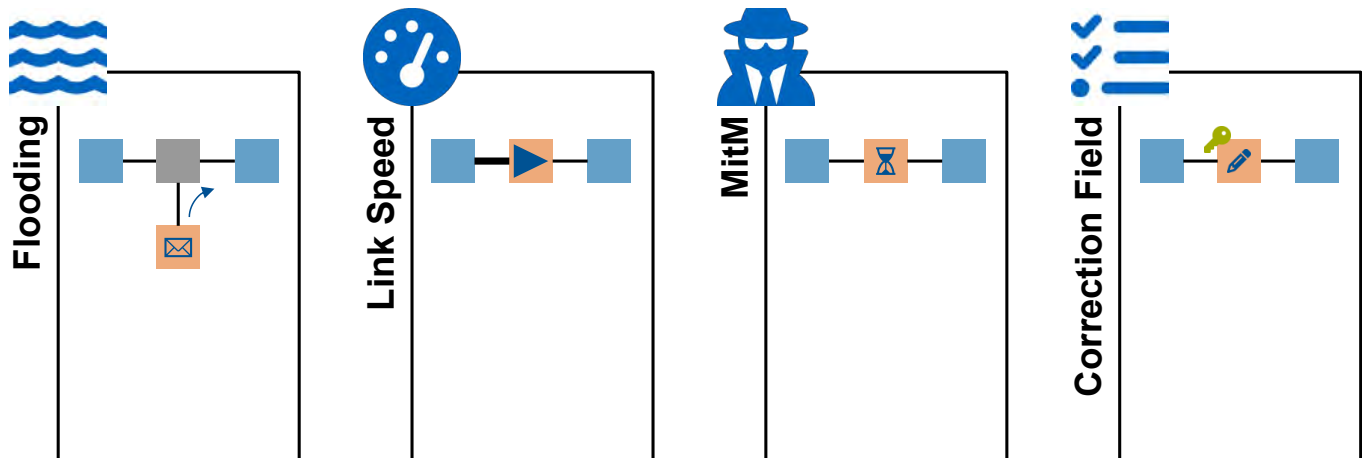
$$\theta = \frac{(t_2 - t_1) - (t_4 - t_3)}{2} - \frac{(\epsilon_1 - \epsilon_2)}{2}$$

- Path delay

$$d = \frac{(t_2 - t_1) + (t_4 - t_3)}{2} - \frac{(\epsilon_1 + \epsilon_2)}{2}$$



## Attack Strategies

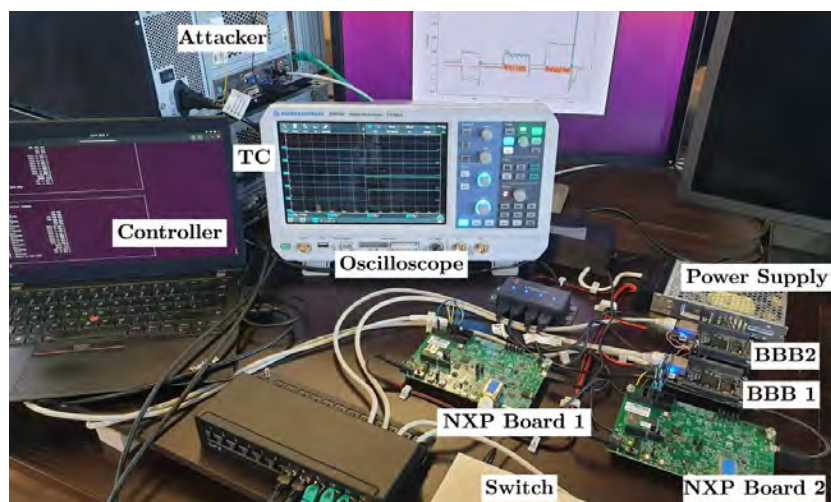


[1] "Feasible Time Delay Attacks Against the Precision Time Protocol", Finkenzeller et al., Globecom 2022

Andreas Finkenzeller | 08.07.2025 | Secure Time Synchronization in Packet-Switched Networks

7

## Attack Implementation



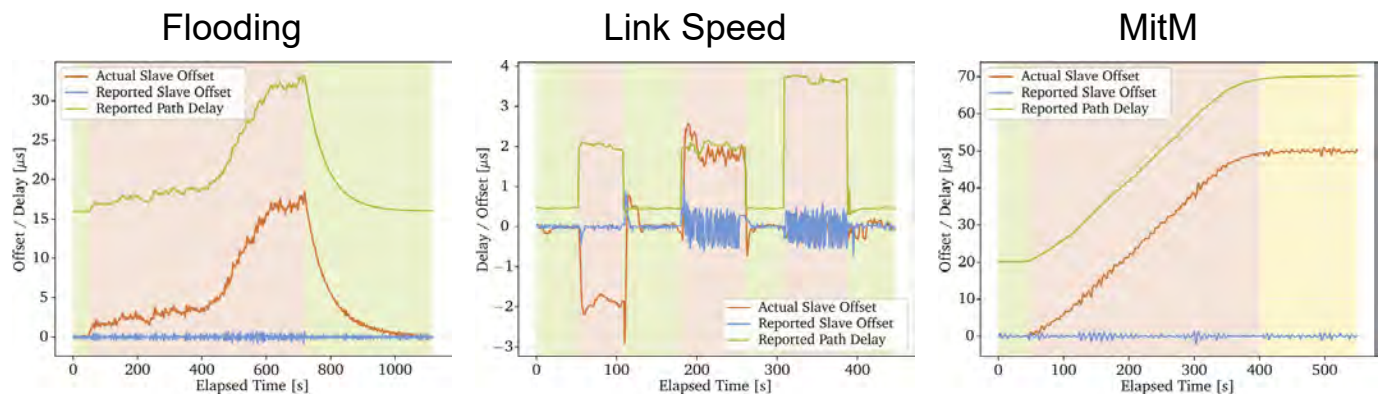
[1] "Feasible Time Delay Attacks Against the Precision Time Protocol", Finkenzeller et al., Globecom 2022

Andreas Finkenzeller | 08.07.2025 | Secure Time Synchronization in Packet-Switched Networks

8



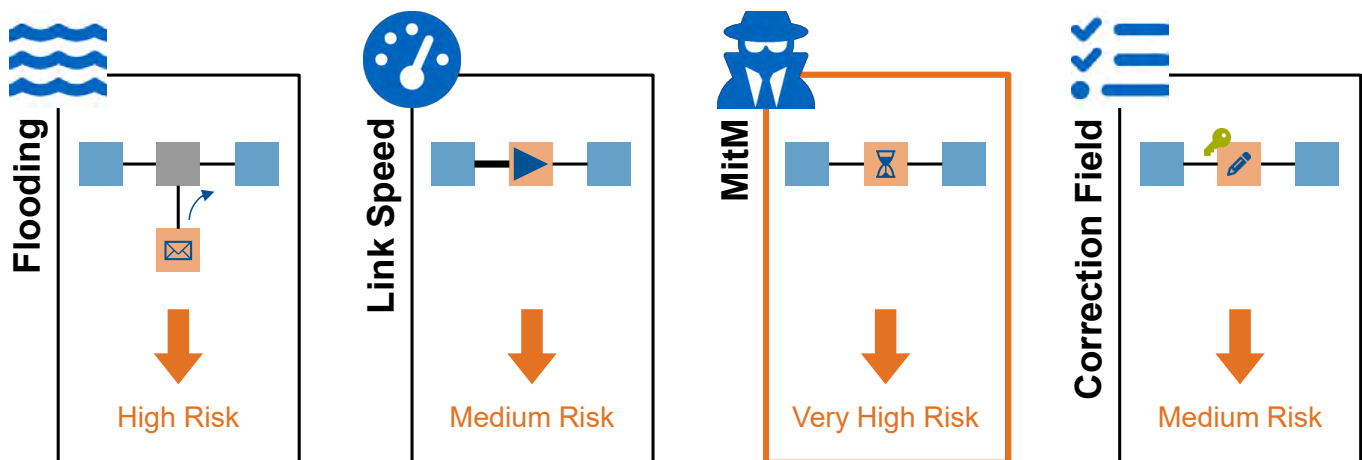
# Attack Evaluation



[1] "Feasible Time Delay Attacks Against the Precision Time Protocol", Finkenzeller et al., Globecom 2022  
 Andreas Finkenzeller | 08.07.2025 | Secure Time Synchronization in Packet-Switched Networks

9

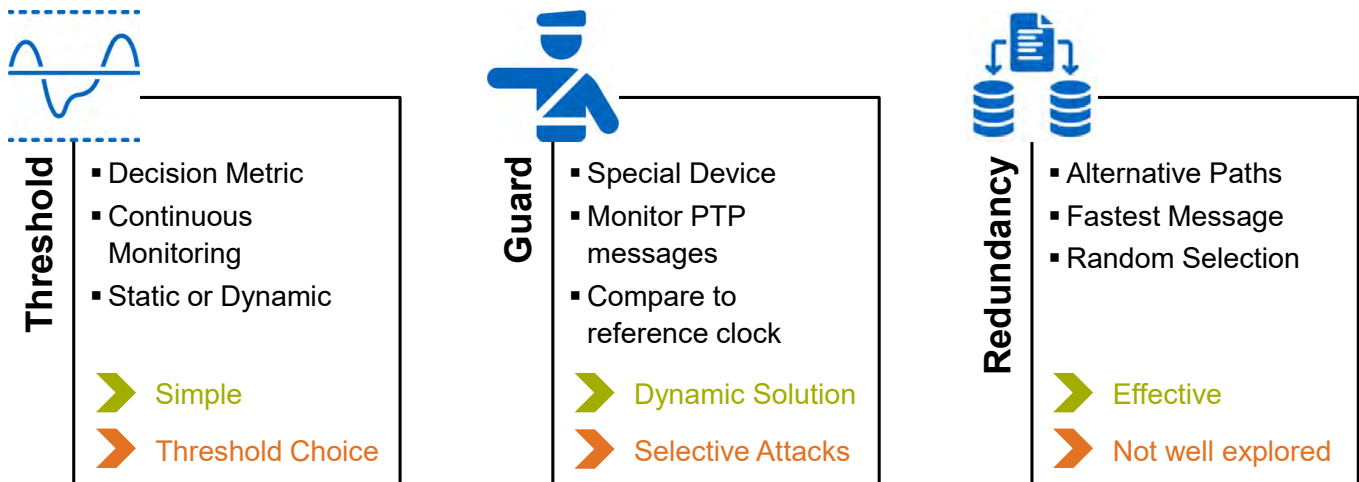
# Attack Strategies



[1] "Feasible Time Delay Attacks Against the Precision Time Protocol", Finkenzeller et al., Globecom 2022  
 Andreas Finkenzeller | 08.07.2025 | Secure Time Synchronization in Packet-Switched Networks

10

## Mitigation Techniques

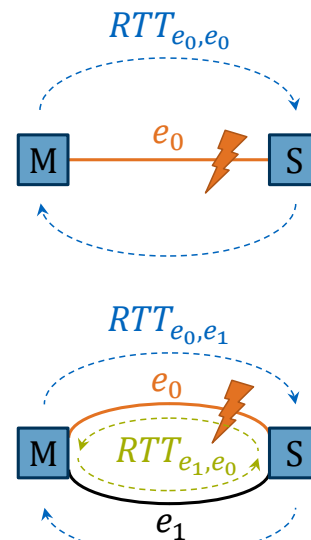


## Asymmetry Analysis

- One link is not sufficient
- $$\alpha_{e_0} = RTT_{e_0, e_1} - RTT_{e_1, e_0} = (e_0^+ + e_1^-) - (e_0^- + e_1^+)$$

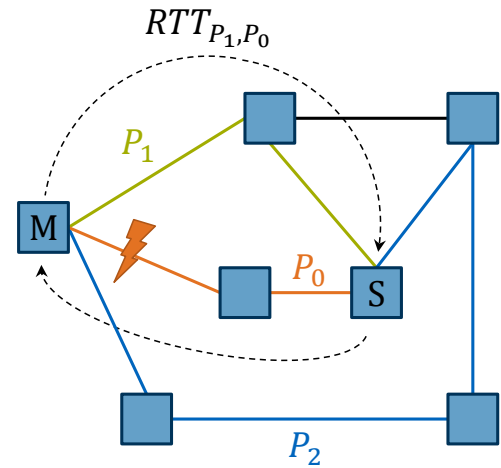
$$= (e_0^+ - e_0^-) \quad \text{if } e_1^- = e_1^+$$
- At least two links for unidirectional delay
- Redundant link must have symmetric link delay

➤ **At least one symmetric link required for analysis**



## Asymmetry Analysis

- $\alpha_{P_0}^{(1)} = RTT_{P_0, P_1} - RTT_{P_1, P_0} = (P_0^+ + P_1^-) - (P_0^- + P_1^+)$



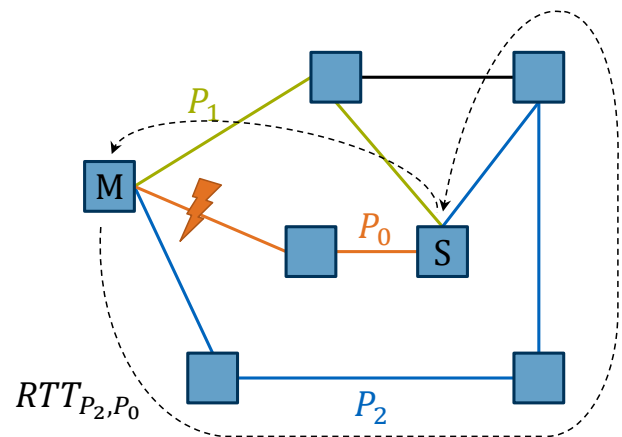
[2] "PTPsec: Securing the Precision Time Protocol Against Time Delay Attacks Using Cyclic Path Asymmetry Analysis", Finkenzeller et al., Infocom 2024  
 Andreas Finkenzeller | 08.07.2025 | Secure Time Synchronization in Packet-Switched Networks

13

## Asymmetry Analysis

- $\alpha_{P_0}^{(1)} = RTT_{P_0, P_1} - RTT_{P_1, P_0} = (P_0^+ + P_1^-) - (P_0^- + P_1^+)$
- $\alpha_{P_0}^{(2)} = RTT_{P_0, P_2} - RTT_{P_2, P_0} = (P_0^+ + P_2^-) - (P_0^- + P_2^+)$
- $\alpha_{P_0} = \alpha_{P_0}^{(1)} = \alpha_{P_0}^{(2)}$

➤ All symmetric paths lead to similar estimates for  $\alpha_{P_0}$



[2] "PTPsec: Securing the Precision Time Protocol Against Time Delay Attacks Using Cyclic Path Asymmetry Analysis", Finkenzeller et al., Infocom 2024  
 Andreas Finkenzeller | 08.07.2025 | Secure Time Synchronization in Packet-Switched Networks

14

# Attack Detection & Mitigation

- Attack Detection

$$delay\_attack \leftarrow \begin{cases} True, & \exists \alpha^{(i)} \neq 0, i \in [1, n] \\ False, & otherwise \end{cases}$$

- Attack Mitigation

$$\theta_{rect} = \theta_{rep} - \frac{\alpha_{P_0}}{2}$$

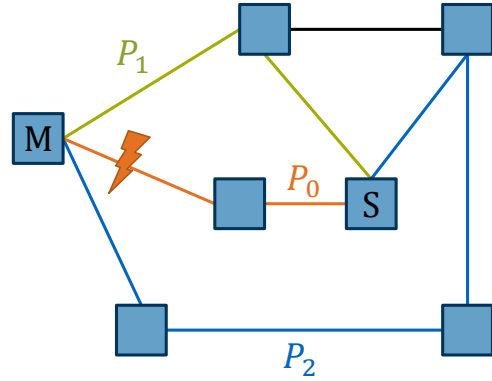
$\theta_{rep}$  reported clock offset

$\theta_{rect}$  rectified clock offset

[2] "PTPsec: Securing the Precision Time Protocol Against Time Delay Attacks Using Cyclic Path Asymmetry Analysis", Finkenzeller et al., Infocom 2024

Andreas Finkenzeller | 08.07.2025 | Secure Time Synchronization in Packet-Switched Networks

15



$$\alpha^{(1)} = RTT_{P_0, P_1} - RTT_{P_1, P_0}$$

$$\alpha^{(2)} = RTT_{P_0, P_2} - RTT_{P_2, P_0}$$

$$\alpha_{P_0} = \alpha^{(1)} = \alpha^{(2)}$$

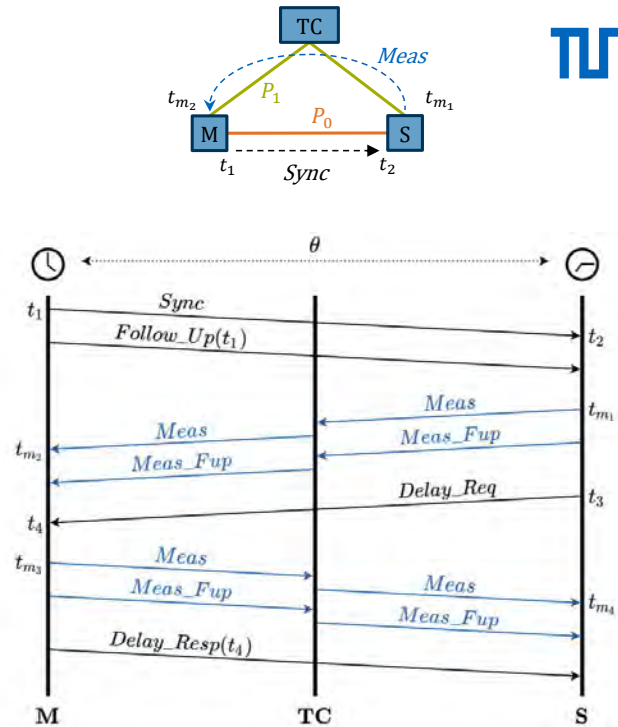
## PTPsec

- Two new packet types (*Meas*, *Meas\_Fup*)
- Measurements entangled with PTP

$$RTT_{P_0, P_i} = (t_{m_2} - t_1) - (t_{m_1} - t_2)$$

$$RTT_{P_i, P_0} = (t_{m_4} - t_3) - (t_{m_3} - t_4)$$

$$\alpha_{P_0} = RTT_{P_0, P_i} - RTT_{P_i, P_0}$$



[2] "PTPsec: Securing the Precision Time Protocol Against Time Delay Attacks Using Cyclic Path Asymmetry Analysis", Finkenzeller et al., Infocom 2024

Andreas Finkenzeller | 08.07.2025 | Secure Time Synchronization in Packet-Switched Networks

16

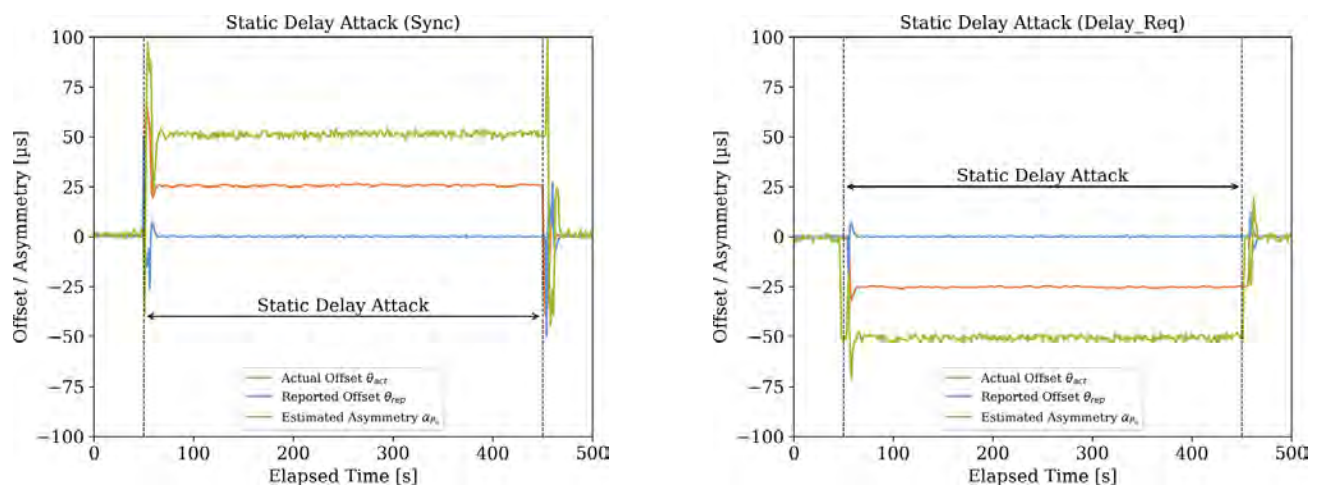
# PTPsec Implementation



[3] "PTPsec: Securing the Precision Time Protocol with SDN-enabled Cyclic Path Asymmetry Analysis", Finkenzeller et al., ACM TCPS 2025  
 Andreas Finkenzeller | 08.07.2025 | Secure Time Synchronization in Packet-Switched Networks

17

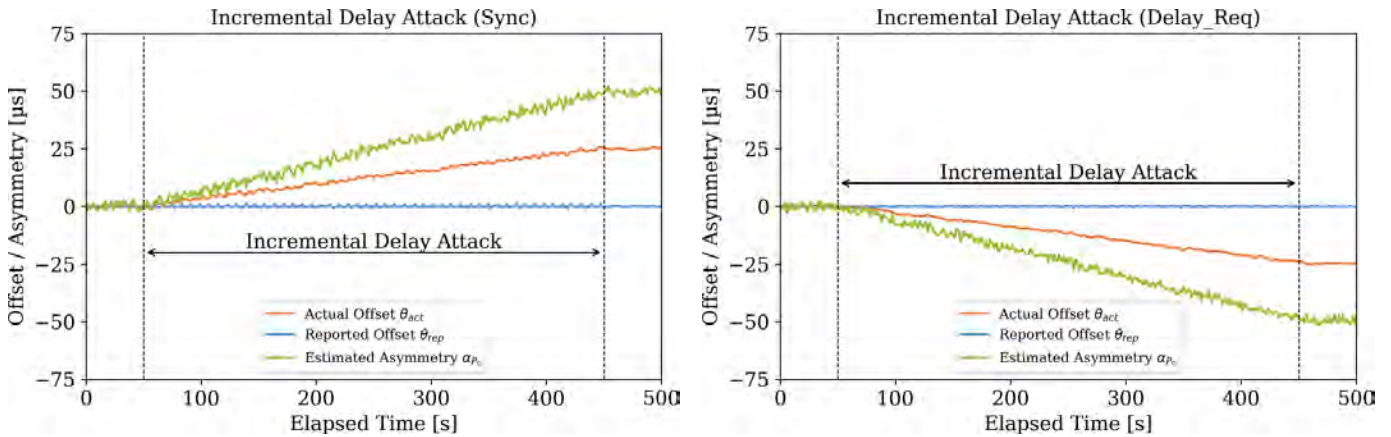
# PTPsec Evaluation



[3] "PTPsec: Securing the Precision Time Protocol with SDN-enabled Cyclic Path Asymmetry Analysis", Finkenzeller et al., ACM TCPS 2025  
 Andreas Finkenzeller | 08.07.2025 | Secure Time Synchronization in Packet-Switched Networks

18

## PTPsec Evaluation

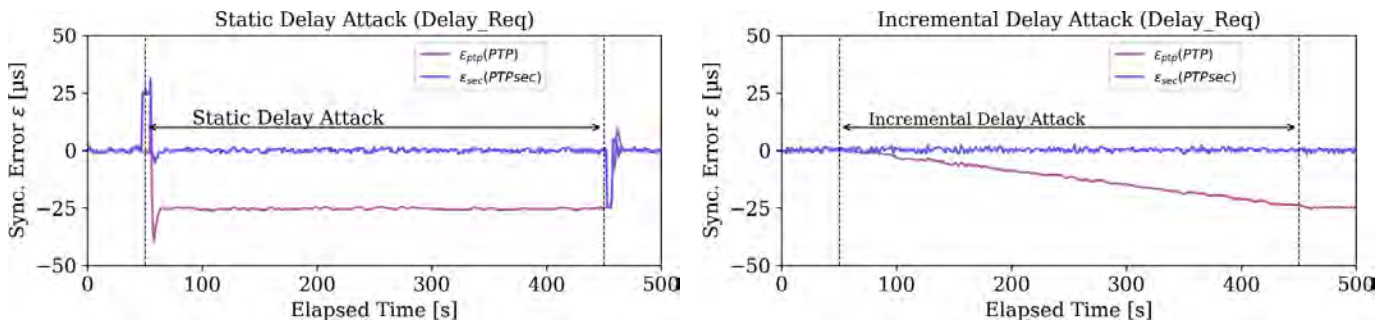


[3] "PTPsec: Securing the Precision Time Protocol with SDN-enabled Cyclic Path Asymmetry Analysis", Finkenzeller et al., ACM TCPS 2025

Andreas Finkenzeller | 08.07.2025 | Secure Time Synchronization in Packet-Switched Networks

19

## PTPsec Evaluation



[3] "PTPsec: Securing the Precision Time Protocol with SDN-enabled Cyclic Path Asymmetry Analysis", Finkenzeller et al., ACM TCPS 2025

Andreas Finkenzeller | 08.07.2025 | Secure Time Synchronization in Packet-Switched Networks

20

## What's next?



### Testing

- Large-scale Tests
- Complex Attacks



### Applications

- Understand Impact
- Sensor Fusion Security:



Session 2: Robotics  
and Control,  
Wed 1:30 – 3:30pm



### Wireless

- Transfer Attack/Defense
- Explore new possibilities

## Take Away





## References

- [1] Finkenzeller, A., Wakim, T., Hamad, M., & Steinhorst, S. (2022, December). Feasible time delay attacks against the precision time protocol. In *GLOBECOM 2022-2022 IEEE Global Communications Conference* (pp. 3375-3380). IEEE.
- [2] Finkenzeller, A., Butowski, O., Regnath, E., Hamad, M., & Steinhorst, S. (2024, May). PTPsec: Securing the Precision Time Protocol Against Time Delay Attacks Using Cyclic Path Asymmetry Analysis. In *IEEE INFOCOM 2024-IEEE Conference on Computer Communications* (pp. 461-470). IEEE.
- [3] Finkenzeller, A., Fucks, A., Regnath, E., Hamad, M., & Steinhorst, S. (2025). Securing the Precision Time Protocol with SDN-enabled Cyclic Path Asymmetry Analysis. *ACM Transactions on Cyber-Physical Systems*.
- Icons (sl. 7, 10, 20): [www.fontawesome.com](http://www.fontawesome.com), [www.flaticon.com](http://www.flaticon.com)
- Image (sl. 2, 21): AI-generated with [Canva](https://www.canva.com)





# Defending Event-Triggered Systems against Out-of-Envelope Environments

Marcus Völpl<sup>1</sup>, Mohammad Ibrahim Alkoudsi<sup>2</sup>, Azin Bayrami Asl<sup>1</sup>,  
Kristin Krüger<sup>2</sup> Júlio Mendonça<sup>1</sup>, Gerhard Fohler<sup>2</sup>

<sup>1</sup>Interdisciplinary Centre for Security Reliability and Trust (SnT), University of Luxembourg  
e-mails: marcus.voelp@uni.lu, azin.bayramiasl@uni.lu, julio.mendonca@uni.lu

<sup>2</sup>Rheinland-Pfälzische Technische Universität Kaiserslautern - Landau  
e-mails: alkoudsi@rptu.de, kristin.krueger@rptu.de, gerhard.fohler@rptu.de

**Abstract**—The design of real-time systems is based on assumptions about environmental conditions in which they will operate. We call this their safe operational envelope. Violation of these assumptions, i.e., out-of-envelope environments, can jeopardize timeliness and safety of real-time systems, e.g., by overwhelming them with interrupt storms. A long-lasting debate has been going on over which design paradigm, the time- or event-triggered, is more robust against such behavior.

In this work, we investigate the claim that time-triggered systems are immune against out-of-envelope behavior and how event-triggered systems can be constructed to defend against being overwhelmed by interrupt showers. We introduce importance (independently of priority and criticality) as a means to express which tasks should still be scheduled in case environmental design assumptions cease to hold, draw parallels to mixed-criticality scheduling, and demonstrate how event-triggered systems can defend against out-of-envelope behavior.

**Index Terms**—time-triggered systems, event-triggered systems, out-of-envelope behavior

## I. INTRODUCTION

Real-time computer systems are typically designed to function within an operational envelope that dictates all possible behaviors of the environment that encloses them. The specification of an operational envelope allows for design-time assurance that the real-time system will behave in a timely manner if it operates within this envelope [1].

A common argument in the decade-old debate between the time- and event-triggered paradigms' experts is the question how to cope with out-of-envelope behavior. On one side, Kopetz's tremendous achievement of predictable, fault-tolerant, composable, and modularly certifiable systems, enabled by the time-triggered architecture [2], and on the other side, there have been various attempts [3]–[5] towards achieving similar properties in event-triggered systems.

In the time-triggered architecture (TTA), all system components share a globally synchronized sparse-time base [2], [6]. All important events and activities, e.g., sampling the environment and exerting control over it, are internalized to periodically occurring time points and intervals on this time base. Intermediate control algorithms are also set to execute and communicate their results within predetermined intervals

of sparse time (time-slots), where each task or message gets a single or multiple exclusive time-slots on the respective resource. This temporal isolation rules out interference from other tasks or messages executing in different time-slots.

Therefore, as long as the environment matches the regime dictated by the specified sparse time-slots, and as long as the globally maintained time base remains immune to faults and attacks, we obtain the above properties and the elegance and beauty of the TTA. However, once the environment moves beyond this envelope, the situation becomes more complicated.

Event-triggered systems internalize events immediately when the sensors and devices at the periphery of the system observe them. Sensors, network interfaces, and other devices are connected to interrupt lines, which, when raised, signal the processor to interrupt its current control flow and activate a corresponding service routine. Such routines in turn handle the interruption, internalize the event (e.g., by associating it with a timestamp) and release tasks that are triggered by this event. The routines then invoke the scheduler to see whether the released tasks can be scheduled immediately or if their execution needs to be deferred (e.g. because a higher priority task is currently running or because resource locking protocols (such as stack-based ceiling [7]) demand deferred execution).

Immediate internalization makes event-triggered systems vulnerable to interrupt storms, triggered by faulty sensors and devices, and to unanticipated events, when environmental situations are in violation of the assumed operational envelope. Designers of event-triggered systems typically equate both situations as undesirable behavior and construct their systems to avoid both by ignoring events that occur in addition to those anticipated. This risks missing important deviations from the norm and taking appropriate actions to keep the system safe, even when normal operation can no longer be guaranteed.

In this work, we take a different approach and survey what it takes in event-triggered systems to defend against out-of-envelope behavior. Our goal is to design a system that will ultimately be able to respond in a timely and predictable manner to unforeseen situations while being equipped to carefully trade-off less important tasks, and to defend itself against error situations that would overwhelm the system.

We start in Section II, by giving concrete examples, to highlight what we mean by out-of-envelope behavior and

This research was funded by the Luxembourg National Research Fund (FNR) and by the German Research Council DFG through the FNR-Core-Inter grant ReSAC (C21/IS/15741419).

discuss how contemporary time- and event-triggered systems respond to such behavior. We then introduce in Section III a new feasibility criterion, inspired by mixed-criticality scheduling [8], [9], to ensure systems continue to respond to important events, even if they can no longer sustain all of their normal behavior. We will also discuss why one might want to introduce importance separate from criticality. Section IV illustrates how event-triggered systems can defend against being overwhelmed by out-of-envelope behavior.

Of course, we are not the first aiming to design defensive event-triggered systems and aspects will have to remain as open questions, which is why, in Section V and Section VI, we discuss limitations and open questions and we relate our work to the works of others, before concluding our work in Section VII.

## II. OUT-OF-ENVELOPE BEHAVIOR

Real-time computer systems interact with their environment by sensing environmental conditions together with the observable states of the machines they control. They receive alarms from specific sensors and influence the environment by actuating parts of these machines. Failure to do so in a correct and timely manner often has severe consequences and puts at risk the safety of the controlled machines, of the people in their proximity, or of the environment in which they operate.

When designing real-time systems, developers define an operational envelope, in which they characterize their assumptions about the environment and specifically what events they expect and how frequently and with which separation they expect them to occur [3]. For example, when controlling a nuclear plant, sensors reveal whether valves are open or closed. Other sensors read out the pressure and water level in the pressurizer and, more generally, dynamic processes are observed with at least double the frequency of the highest signal frequency that is to be expected.

In this work, we are primarily interested in external events, happening at the system's periphery, such as reactor pressure levels exceeding a certain threshold, which is observed by a sensor at an appropriate location. While pressure remains within certain bounds (i.e., within the envelope), the sensor-measured pressure values change only rarely and in a continuous manner. Beyond these bounds however, bubbles in the cooling water may cause fast alterations between low- and high-pressure situations at the same sensor, indicating a serious and unexpected situation, which in the case of Three-Mile Island has already led to a serious incident [10].

Time-triggered systems internalize these signals at pre-defined and, thanks to the globally synchronized sparse time base, globally known events. Let  $T_{sample}$  be the pre-defined sampling period of an observed real-time entity, e.g., a valve in the nuclear plant. Let  $t_{valve}$  and  $t'_{valve}$  denote the time at which the observed valve changes state to open and closed, respectively. If the duration  $t'_{valve} - t_{valve}$  is smaller than  $T_{sample}$ , then the time-triggered system will not be able to recognize and internalize the opening event of the valve  $t_{valve}$ . The operational envelope defines that developers do not expect

more frequent events and designing systems according to that envelope implies that such a behavior will not be considered by the developed system.

In event-triggered systems, in addition to the possibility of sampling events, changes are communicated almost immediately from the sensor to the computer system by means of raising interrupts at the latter. Interrupts are hardware signals (delivered over dedicated lines or as PCI and then memory bus messages) to one of the processors' interrupt controllers, which in turn causes the processor to preempt its current task and enter the operating system's *top-half* interrupt service routine. Once interrupt occurrence is recorded and, for level-triggered interrupts, the source is masked, the operating system (OS) decides whether it processes the interrupt immediately or returns to the preempted task for deferred handling in a so-called *bottom half*. The critique put forward is that this recording and masking in the top half together with the kernel entry, already interferes enough with the scheduled tasks to reduce predictability and hinder independent certification. In practice, however, as long as the environment behaves as depicted in the operational envelope, preemptions can be anticipated and accounted for in the tasks' worst-case execution times (WCETs), in particular if the handling of the remaining part of the interrupt (the bottom half) can be deferred and scheduled like regular jobs.

Out-of-envelope behavior happens when unexpected events occur. Event-triggered systems will then have more interrupts raised than anticipated or combinations of interrupts that were not considered. We have already seen that time-triggered systems are immune to such occasions, since they will not internalize out-of-envelope behavior, and will proceed as if none of the missed events have happened. This is where event-triggered systems have the potential for a more attenuated response, by continuing to respond also to unforeseen events and specifically to alarms. It will even be possible to plan for responses to unlikely event combinations, without reserving time for these responses in the regular schedule.

Unfortunately, by responding to out-of-envelope behavior, event-triggered systems also make themselves vulnerable to increased interference, in case multiple events happen close together, and specifically to interrupt storms. In fact, naive implementations, will simply crash event-triggered systems in such situations due to the kernel stack overflowing when pushing interrupted state in an uncontrolled manner. The execution of high-priority tasks may be delayed when executing the top-halves of a storm of interrupts pertaining to the execution of low-priority tasks, leading to deadline misses and risking safety. Of course there are several works that address this concern albeit within the assumed environmental envelope [11]–[16]. For instance, Parmer and West [16] schedule bottom halves in deferrable servers to limit the budget bottom halves may consume within a given amount of time. However, they dimension the budget to consider only events that have been anticipated in the system's operational envelope.

In this work, we also consider out-of-envelope behavior with the goal of equipping systems with the possibility to

respond to important events, specifically alarms, even if they happen more frequently than anticipated. At the same time, our goal is to defend systems against faulty sensors raising alarms continuously and, in turn, overwhelming the system.

### III. OUT-OF-ENVELOPE FEASIBILITY

Our goal is to allow event-triggered systems to remain responsive to out-of-envelope behavior, while defending against being overwhelmed. In normal situations, when the environment behaves as in the assumed envelope, all events should be internalized, and corresponding tasks released and scheduled, including the events' bottom-half handlers in case part of the immediate event response can be deferred. Once important events occur more frequently than anticipated, but still within certain bounds, the system should obtain the possibility to trade off the handling of less important events and their corresponding tasks and still inform operators about triggered alarms and still take the actions that are required to keep the system safe. However, when exceeding these bounds, sensors must be considered faulty and should be suppressed to these bounds, while indicating that more events occurred than could be internalized.

Since we specifically consider alarms, we assume a classical mix of sporadic and periodic tasks, as captured in the sporadic task model. Events (including alarms or the timers firing at specific points in time) release jobs  $\tau_{i,j}$  of tasks  $\tau_i$  in the task set  $\Gamma$ . Each task is characterized by the tuple  $(C_i, D_i, T_i, I_i)$ , where for simplicity we assume relative deadlines  $D_i$  are implicitly defined by the task's minimum inter-arrival time  $T_i$  (i.e.,  $D_i = T_i$ ). Tasks are feasible if all jobs receive  $C_i$  time between their release  $r_{i,j}$  and their absolute deadline  $r_{i,j} + D_i$ .

In order to characterize the importance of an event, and hence of the jobs it releases, we introduce an *importance* value  $I_i$ , which we use to create a total order of importance for each task. In mixed-criticality systems, importance can be mapped to criticalities (e.g., by assigning low-criticality tasks an importance up to a certain level  $l$  — tasks with importance  $I_i \in [0, l)$  are LO — and high-criticality tasks a higher value) and  $C_i$  and other task parameters may be criticality-level dependent (e.g.,  $C_i$  may be a vector  $C_i(l_i)$  where  $l_i$  is the criticality level of the task). In addition, importance may be mapped to priorities unless this interferes with priority assignment (e.g., tasks with a larger period  $T_i$  may be more important than those with a smaller period, while rate-monotonic scheduling would assign them lower priorities, which we would like to allow). However, there are also cases where importance should be handled independently of criticality and priority. Criticality is commonly used to grant additional resources to those tasks that are more critical to maintain the safety of the system. It does not distinguish internal and external failure modes as to why such a task would need additional resources. In our work, we are primarily concerned with internalizing and allowing the handling of external events, such as the frequent change of pressure levels when bubbles rise in the cooling water, to then take appropriate actions. In that sense, events arriving outside the anticipated

envelope may trigger a criticality change, but not necessarily vice versa. Therefore, in the above model, in normal situations, all events should be internalized and all tasks be scheduled, irrespective of their importance.

To capture out-of-envelope behavior, we introduce as additional parameters for the task-releasing events the number  $n_i$  of events and a time-window length  $W_i$  in the sense that the system should still respond to up to  $n_i$  such events within any sliding window of length  $W_i$  in case the environment leaves the assumed envelope and issues task- $\tau_i$ -releasing events more frequently than once every  $T_i$ . The fraction  $n_i/W_i$  gives us a rate and upper bound within which we still consider event-internalizing sensors as correct and the environment out of the anticipated envelope. Beyond this bound, that is, when the  $n_i^{th}$  event occurs within  $W_i$ , we raise as additional alarm, indicating that the event-triggering sensor may be faulty. It is then up to the system to decide how to respond.

Once a task  $\tau_i$  releasing event occurs more frequently than once within  $T_i$ , feasibility changes and requires handling up to  $n_i$  events within  $W_i$  provided more important events and the tasks  $\tau_i$  they release can still be scheduled. Like mixed-criticality feasibility, this requirement makes no claims about less important tasks, but of course we would like to maintain as many tasks as possible, ordered by importance, and ideally all of them. We therefore add as additional constraint that:

*Definition 1 (Out-of-Envelope Feasibility):* At any time, system responses for up to  $n_i$  task- $\tau_i$ -releasing events in every sliding window of length  $W_i$  must be considered and if the response is to release all instances of  $\tau_i$ , then those instances must be guaranteed to receive  $C_i$  time between their release and deadline before any less important task  $\tau_j$  (i.e.,  $I_j < I_i$ ) is scheduled.

Note that the formulation so far also allows only responding in exceptional situations, by considering that the normal occurrence of a task has already occurred and setting this task's period to infinity.

From here, three questions need to be answered, namely

- How should the system respond to more than one releasing event per  $T_i$ ?
- How to schedule released tasks so as to guarantee out-of-envelope feasibility? and
- How to enforce that for all tasks at most  $n_i$  events are handled within  $W_i$ , respecting the importance  $I_i$  and that the system recognizes important alarms to make the appropriate adjustments?

In this preliminary work, we shall focus on the third question, but let us summarize some early conclusions about the first two questions as well.

Of course, we cannot avoid internalizing all  $n_i$  events, which creates  $n_i$ -fold top-half load within the sliding window of  $W_i$ . However, our formulation of out-of-envelope feasibility does not require releasing all  $n_i$  tasks, although for some situations this may be desirable. Alternatively, one could

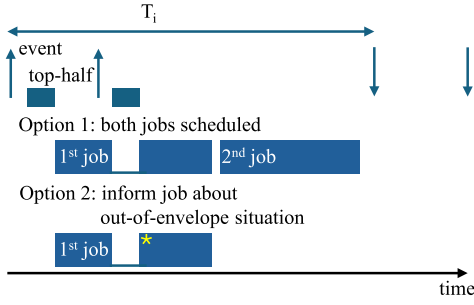


Fig. 1. Internalization and scheduling options for a task, which deviates from the assumed envelope. Both events need to be internalized and generate top-half overheads. Depending on the kind of task, out of envelope behavior can be addressed by releasing all jobs for this event (option 1) or by informing an already running job (option 2), e.g., by invoking an exception handler in this task.

release some of these tasks, informing them about the out-of-envelope behavior, which then may result in the tasks triggering a criticality change to properly respond to out-of-envelope situations. Figure 1 illustrates these two situations.

Clearly, out-of-envelope feasibility shows similarities to mixed-criticality feasibility, and we expect some of the results to carry over immediately. For example, drawing inspiration from criticality-monotonic scheduling [17], fixed task priority assignments according to the importance of tasks will guarantee that more important tasks are scheduled before less important tasks will be considered, even in out-of-envelope situations, provided of course that under this priority assignment the task set remains schedulable in normal situations.

It is also relatively easy to construct a counterexample, showing that such “importance-monotonic scheduling”, when releasing all tasks instances upon their releasing event, cannot be optimal. Consider the two tasks shown in Figure 2. The first is characterized as  $\tau_l = (2, 3, I_l)$  and the second as  $\tau_h = (2, 6, I_h)$ , with importance  $I_l < I_h$ . For  $\tau_h$ , we consider an out-of-envelope behavior of  $n_h = 2$  in every sliding window of length  $W_h = T_h = 6$ . Clearly, releasing the task twice is an exceptional situation, which however, we would still like to handle (e.g., to respond to an unanticipated important alarm sent in short succession). As shown in the figure, importance-monotonic priority assignment will lead to a deadline miss, even in normal situations (left). However, if we would raise the priority of  $\tau_l$ ’s first job above the priority of  $\tau_h$ , we obtain a schedulable task set in normal situations (middle), but also in the exceptional situation, where  $\tau_h$  is released twice (right). Remember, in the latter case we can sacrifice  $\tau_l$ ’s second job, since it is less important than  $\tau_h$ . The system remains responsive to the unanticipated occurrence of the second alarm, but cannot retain the full service it had in normal situations. Also, an alarm is raised indicating that the sensor triggering this alarm operates at its bound and should be interpreted with care.

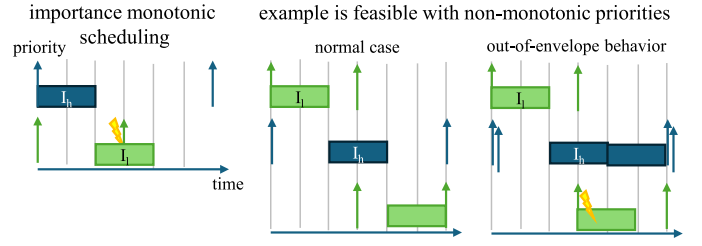


Fig. 2. Example showing that importance-monotonic scheduling is not optimal. The low-importance task  $\tau_l$  misses its deadline, even in normal situations (left), whereas both jobs of it can meet their deadline if the first one is higher prioritized than the high-importance task  $\tau_h$  (middle). Even if  $\tau_h$  is released outside the anticipated envelope (up to  $n_h = 2$  events within  $W_h = T_h$  in this example),  $\tau_h$  meets all deadlines, at the cost of the less important second job of  $\tau_l$ .

#### IV. DEFENDING AGAINST OVERWHELMING OUT-OF-ENVELOPE BEHAVIOR

Ensuring that event-triggered systems will not get overwhelmed by out-of-envelope behavior requires limiting the internalization of task-releasing events to at most  $n_i$  within any sliding window of length  $W_i$ , while providing the operating system with the signal it needs to adjust the task schedule. In particular, if a task is released more than once every  $T_i$ , this already constitutes out-of-envelope behavior and may result in sacrificing the internalization of less important events and possibly the tasks they release. This is to ensure more important events, specifically alarms, continue to be processed.

##### A. Vectored Interrupt Controllers

In this section, we illustrate how vectored interrupt controllers, such as the Advanced Programmable Interrupt Controller (APIC) variants of x86 processors or ARM’s Nested Vectored Interrupt Controller (NVIC) can be configured to provide these indicators while protecting from interrupt storms.

Vectored interrupt controllers (VICs) multiplex the processor’s interrupt mechanism by providing multiple external interrupt lines. In addition, they often allow interrupt lines to be assigned different priorities. In this case, the interrupt controller features an interrupt-priority level such that lower-priority interrupts can be masked simply by raising this priority level above the priority of the line.

In this work, we assume VICs to provide such interrupt priorities and an interrupt priority level (both ARM’s NVIC and various x86’s APIC variants do). We further assume that interrupts can be masked individually and, for simplicity, will not consider sharing interrupt lines among multiple interrupt sources. Moreover, we shall assume that the timer interrupt can be configured to be independent of the above constraints. This is possible, for example, by giving the timer the highest interrupt priority. Lastly, we shall assume devices or external interrupt controllers (e.g., IO-APIC) expose a counter per interrupt line that is incremented for each event occurring. Capture compare units already provide various such counters, albeit not for this purpose.

In the following, we shall continue to talk about priorities with the implication that these are the interrupt priorities considered in the VIC. We do not require interrupt priorities to correlate to task priorities and will divert from the recommendation to set the interrupt priority level to the priority of the currently running task. Instead, we shall leverage this mechanism to control importance and prevent too many undesired interruptions.

### B. Detecting Out-of-Envelope Behavior

If a task were to be released more than  $n_i$  times within a sliding window of size  $W_i$ , the releasing sensor could be considered faulty and no further events would need to be handled and, for that matter, be internalized.

To capture out-of-envelope behavior precisely, we would have to allow all events to happen and be internalized so as to record them in ring buffers of size  $n_i$ , measure their time of occurrence and compute when the event leaves the task's sliding window of length  $W_i$ .

### C. Defending Against Being Overwhelmed

We mask interrupts to defend against being overwhelmed, which implies not internalizing such events.

In case the buffer is full, the OS masks the interrupt to prevent all subsequent occurrences of this event from being internalized, raises an alarm and sets a timer to the time of the earliest event in the buffer plus  $W_i$ . Being masked, subsequent events will not be internalized but are still recorded in the devices' counters. Once the timer fires, the OS unmask the interrupt line of the event to consider further occurrences.

At this point in time, the OS compares the counter at the device against the value it read when masking the event to identify whether the sensor is faulty (i.e., more than  $n_i$  events occurred in that time). It is then up to the OS whether it will consider this sensor permanently damaged or resume using this sensor once the alarm rate drops below  $n_i/W_i$ .

Unfortunately, for sensors that still operate within the bound of  $n_i$  occurrences within  $W_i$ , capturing out-of-envelope behavior precisely comes at significant costs in terms of top-half overheads and hence the interference that top-half handling causes on task execution. In the worst case, this top-half handling, even if we capture the timestamp of the respective next event through a capture unit, amounts to  $\sum_{\tau_i \in \Gamma} n_i \Delta_{TH}$ , where  $\Delta_{TH}$  denotes the time needed to enter the kernel, record the timestamp of the event in the ringbuffer, take the decision to and mask it, and returning from the kernel, since all  $n_i$  occurrences of every task-releasing event may occur at once.

Obviously, this is not very feasible. We therefore overapproximate subsequent events as if they occur with the first, in case they do not affect the current scheduling decision. That is, we ignore all but the next occurrence of each event until when the bottom half is processed, by masking the specific interrupt in the top half and unmasking it only after the bottom half finishes executing. Still, multiple events may occur and the device indicates their number, but we assign all of them the timestamp of the event that masked them. This reduces the

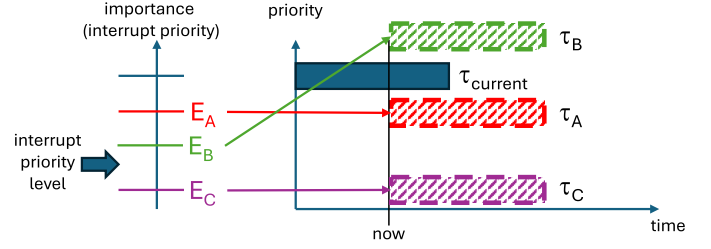


Fig. 3. Example showing how the interrupt priority level may prevent internalizing some of the events that have no consequence for scheduling. In the figure, three tasks ( $\tau_A, \tau_B, \tau_C$ ) and their events ( $E_A, E_B, E_C$ ) are shown that may preempt the currently running task  $\tau_{current}$ . We set the interrupt priority level to below the importance of the least important task ( $\tau_B$ ) that may still preempt  $\tau_{current}$ . This way,  $\tau_C$ , whose event importance  $I_C$  is lower than or equal to the interrupt priority level gets masked. However, the event  $E_A$  of task  $\tau_A$  remains enabled, since  $\tau_A$  is more important than  $\tau_B$  (although less important than  $\tau_{current}$ ).

sliding window during which we do not allow further events to be internalized and creates more pressure in the system<sup>1</sup>, but ensures we do not miss events that we should have handled.

In the same way, we mask events that have no immediate consequence on the scheduling decision. That is, we raise the interrupt priority level to prevent internalizing the events of tasks that would not preempt the current task. These are lower-important tasks whose next job the scheduler will prioritize higher than the priority of the current running task. Again, when lowering this level, we internalize these events with the timestamp when we masked them and evaluate the device event counts to identify whether sensor values were off. Notice that even though this timestamp may be before the actual release time of the task, the task will not be executed before the point in time when its releasing event is internalized. Notice also that the above mechanism is not perfect, because there may be lower prioritized and less important tasks than the currently running task, which are still more important than the task with the minimum importance whose next job will be higher prioritized. Figure 3 illustrates this point. It would of course be possible to mask these tasks manually, albeit at significant costs, in particular if the interrupt controller does not allow installing and changing interrupt masks for multiple interrupt lines at once.

## V. LIMITATIONS AND OPEN QUESTIONS

Of course, we are not yet at the end of our journey and several limitations and criticisms remain valid, which raise open questions and require further research.

For example, one could argue that our differentiation of the still to be considered out-of-envelope behavior and the bounds after which we consider sensors to wrongly produce events is artificial in the sense that we anticipate as out-of-envelope behavior what should actually be anticipated. We admit this contradiction and lack of a better definition of what out-of-envelope behavior actually is, but hope the intuition will be

<sup>1</sup>Considering events to happen earlier means the system will, at the brink of the sensor being faulty, sooner unmask interrupts, since the sliding window ends earlier.

clear. Out-of-envelope behavior is what we did not anticipate for the normal behavior of the system but which we still might be afraid to see and therefore plan for.

Focusing on enforcement, we could of course only sketch some of the scheduling problems that occur when the system should remain feasible in out-of-envelope situations. We therefore leave as an open question, the investigation of scheduling algorithms that can guarantee schedulability of the most important tasks when inter-separation constraints are violated.

Our approach so far completely gives up on inter-separation constraints once the environment moves out of the anticipated envelope. This is mainly to capture unanticipated alarms. However, if arguments about the physical processes demonstrate that it will be infeasible that events occur more frequently than once every  $T_i^{out}$ , then integrating such bounds in the task model would greatly improve the schedulable utilization of the system, even if the out-of-envelope period  $T_i^{out}$  is small compared to the period  $T_i$ .

Likewise, our approach so far assumes importance to be a total order among tasks. It might be interesting to investigate more well-defined structures that can capture different sets of tasks that should be sacrificed depending on which event violates the assumptions made in the system's operational envelope.

Being a report on preliminary work, we obviously did not quantify the overhead of our approach on existing vectored interrupt controllers, nor did we evaluate the design of VICs that are specifically designed to capture out-of-envelope behavior. For example, the constraints which prevented us from precisely tracking out-of-envelope events in Section IV vanish if ringbuffers containing timestamps are maintained in the VIC. Like capture units, such VICs could record the timestamp of occurrence of interrupt-triggering events (provided the  $n_i$ -element ringbuffer is not yet full) while triggering the processor's interrupt service routine only when the processor is ready to receive such an interrupt and when it would affect scheduling decisions (communicated by the masking scheme we discussed above).

## VI. RELATED WORK

In this work, we have already discussed similarities but also differences to mixed-criticality scheduling [8], whose state of the art is captured in the review by Davis and Burns [9]. Importance allows us to define trading off tasks independent of those that originate from high-criticality tasks exceeding their low-criticality expectations and we hope its use to respond to clearly unanticipated situations avoids some of the misconceptions of mixed-criticality systems [18].

We already mentioned several works to investigate event-triggered systems and what properties they may retain despite giving up on internalizing events only according to the globally synchronized sparse time base. For example, Scheler and Schroeder-Preikschat [4] ask whether the difference between event- and time-triggered is just a matter of configuration.

We believe this can be answered only after extending our observations to the network level.

We likewise mentioned that others have also proposed solutions for defending against interrupt storms (with the purpose of remaining within the operational envelope, unlike what we propose — to slightly step out of it and focus on the most important tasks). For example, Parmer and West [16] describe a means for predictable interrupt management with the help of deferrable servers, which, among others, has enabled core-local reasoning and predictable cross-core communication [19] in the M3 microkernel [20]. Scheler et al. [11] discuss hardware supported interrupt handling for event-triggered real-time operating systems. Kim et al. [12] explore interrupt handling and enforcement in real-time system virtualization. Ley vadel Foyo et al. [13] propose to integrate interrupt and task handling. Brinkschulte et al. [15] suggest in a work-in-progress presentation the notion of interrupt service threads and Elliott and Anderson [14] propose robust interrupt handling for multiprocessor systems, by drawing inspiration from GPUs.

## VII. CONCLUSIONS

In this work, we raise the question whether real-time systems can still remain responsive in case unanticipated and unforeseen combinations of events move them beyond the operational envelope for which they have been designed. We distinguish normal situations from situations where such out-of-envelope behavior occurs and from situations where faulty sensors overwhelm the system by generating interrupt storms or event combinations that the system can no longer sensibly handle. We propose a vectored interrupt controller-based event internalization and handling scheme that is capable of defending against the latter, while supporting the two former. Our scheme can be implemented using existing interrupt controllers (e.g., on ARM and x86), but would at the same time greatly benefit from a dedicated capture unit that records the timestamps of up to  $n_i$  interrupt-triggering events in per interrupt-line ringbuffers.

Directions for our future work include evaluating such hard- and software implementations, further exploring the similarities and differences between importance-based scheduling (once the system leaves its operational envelope) and mixed-criticality scheduling, and investigating whether out-of-envelope behavior can also be tolerated at the network level.

## VIII. ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their helpful comments.

## REFERENCES

- [1] D. Powell, G. Bonn, D. Seaton, P. Verissimo, and F. Waeselynck, "The delta-4 approach to dependability in open distributed computing systems," in *[1988] The Eighteenth International Symposium on Fault-Tolerant Computing, Digest of Papers*, 1988, pp. 246–251.
- [2] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.
- [3] D. P. Borgers and W. P. M. H. Heemels, "Event-separation properties of event-triggered control systems," *IEEE Transactions on Automatic Control*, vol. 59, no. 10, pp. 2644–2656, 2014.

- [4] F. Scheler and W. Schroeder-Preikschat, "Time-triggered vs. event-triggered: A matter of configuration?" in *ITG FA 6.2 Workshop on Model-Based Testing, GI/ITG Workshop on Non-Functional Properties of Embedded Systems, 13th GI/ITG Conference Measuring, Modelling, and Evaluation of Computer and Communications*, 2006.
- [5] M. J. Khojasteh, P. Tallapragada, J. Cortés, and M. Franceschetti, "Time-triggering versus event-triggering control over communication channels," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, 2017, pp. 5432–5437.
- [6] H. Kopetz, "Sparse time versus dense time in distributed real-time systems," in *[1992] Proceedings of the 12th International Conference on Distributed Computing Systems*, 1992, pp. 460–467.
- [7] T. Baker, "A stack-based resource allocation policy for realtime processes," in *[1990] Proceedings 11th Real-Time Systems Symposium*, 1990, pp. 191–200.
- [8] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, 2007, pp. 239–243.
- [9] A. Burns and R. Davis, "Mixed criticality systems-a review," *Department of Computer Science, University of York, Tech. Rep.*, pp. 1–69, 2013, current version: <https://www-users.york.ac.uk/~ab38/review.pdf>.
- [10] C. Perrow, "Normal accident at three mile island," *Society*, vol. 18, 1981.
- [11] F. Scheler, W. Hofer, B. Oechslein, R. Pfister, W. Schröder-Preikschat, and D. Lohmann, "Parallel, hardware-supported interrupt handling in an event-triggered real-time operating system," in *Proceedings of the 2009 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, ser. CASES '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 167–174. [Online]. Available: <https://doi.org/10.1145/1629395.1629419>
- [12] H. Kim, S. Wang, and R. Rajkumar, "Responsive and enforced interrupt handling for real-time system virtualization," in *2015 IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications*, 2015, pp. 90–99.
- [13] L. E. Leyva-del Foyo, P. Mejia-Alvarez, and D. de Niz, "Integrated task and interrupt management for real-time systems," *ACM Trans. Embed. Comput. Syst.*, vol. 11, no. 2, Jul. 2012. [Online]. Available: <https://doi.org/10.1145/2220336.2220344>
- [14] G. A. Elliott and J. H. Anderson, "Robust real-time multiprocessor interrupt handling motivated by gpus," in *2012 24th Euromicro Conference on Real-Time Systems*, 2012, pp. 267–276.
- [15] U. Brinkschulte, C. Krakowski, J. Kreuzinger, and T. Ungerer, "Interrupt service threads - a new approach to handle multiple hard real-time events on a multithreaded microcontroller," *RTSS WIP Sessions*, 08 2000.
- [16] G. Parmer and R. West, "Predictable interrupt management and scheduling in the composite component-based system," in *2008 Real-Time Systems Symposium*, 2008, pp. 232–243.
- [17] T. Fleming and A. Burns, "Extending mixed criticality scheduling," in *Workshop on Mixed Criticality Systems (WMC)*, vol. 11, 2013, pp. 7–12.
- [18] R. Ernst and M. Di Natale, "Mixed criticality systems—a history of misconceptions?" *IEEE Design and Test*, vol. 33, no. 5, pp. 65–74, 2016.
- [19] N. Asmussen, S. Haas, A. Lackorzynski, and M. Roitzsch, "Core-local reasoning and predictable cross-core communication with m3," in *2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2024, pp. 199–211.
- [20] N. Asmussen, M. Völz, B. Nöthen, H. Härtig, and G. Fettweis, "M3: A hardware/operating-system co-design to tame heterogeneous manycores," *SIGARCH Comput. Archit. News*, vol. 44, no. 2, p. 189–203, Mar. 2016. [Online]. Available: <https://doi.org/10.1145/2980024.2872371>