D2.2: Report on Adaptation Methods

Project acronym: ADMORPH
Project full title: Towards Adaptively Morphing Embedded Systems
Grant agreement no.: 871259

Due Date:	Month 30
Delivery:	Month 30
Lead Partner:	UniLu
Editor:	Marcus Völp, UniLu
Dissemination Level:	Public (P)
Status:	final
Approved:	
Version:	1.0

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871259 (ADMORPH project).

DOCUMENT INFO – Revision History

Date and version number	Author	Comments
21/05/2022 ver. 1.0	Marcus Voelp	First draft

List of Contributors

Date and version number	Beneficiary	Comments
21/05/2022 ver. 1.0	Marcus Voelp	Initial report structure

GLOSSARY

BFT-SMR Byzantine Fault Tolerant Statemachine Replication

CPS(oS) Cyber Physical System (of Systems)

FIT Fault and Intrusion Tolerance

IPC Inter-process communication

QoS Quality of Service

SoS System of Systems

Contents

Executive summary **4**

1 Adaptation as Key Enabler of Individualistic and CPSoS-wide Resilience **5**

2 State-of-the-Art since D2.1b (month 22) **6**

3 Adaptation to Achieve Control-Aware Fault and Intrusion Tolerance **7**

4 Adaptation to Recover and Make Resilient Control Tasks **9**

5 Bounding Time of Adaptation and Reconfiguration **9**

6 Network Monitoring and Adapting Connectivity **10**
6.1 Diversity in NIDS for mitigating adversarial attacks 10
6.1.1 Approach overview 11
6.1.2 Approach specification 12
6.2 Summary and next steps 13

7 Interfacing with the Coordination Language Compiler Infrastructure **14**

8 Testing Runtime Systems and Adaptation Strategies **15**

9 Conclusions **16**

10 References **16**

Executive summary

Work package WP2 sets out to develop the adaptation building blocks necessary for maintaining or, in extreme situations, gracefully degrading the systems' quality of service guarantees. The focus is on methods, protocols, tools and techniques, to increase the resilience of controllers of Cyber-Physical-Systems of Systems (CPSoS), to optimize the mapping, partitioning and scheduling of system components, to automate the design transformation towards a reliable, resource and physical requirement aware system, and to analyze and limit system reconfiguration times.

This deliverable D2.2 reports on the adaptation methods and summarizes the achievements of all tasks of WP2, which at the moment this deliverable is written are all active. This deliverable builds upon and extends deliverable D2.1b.

Adaptation serves four main purposes:

1. to evade faults, including accidental ones and adversaries in their ongoing attacks;
2. to improve the resilience of systems after experiencing faults or during ongoing attacks, possibly by degrading the systems' quality of service, if necessary;
3. to return the system to a state that is considered safe and secure; and
4. to optimize the system whenever the perceived threat level drops (e.g., because the CPS entered less harsh environments or because adversaries lost interest).

We report on our progress in adapting individualistic CPS and CPSoS-wide resilience (Section 1), report on the SOTA since the last deliverable (in Section 2), look at adaptation to achieve control-aware fault and intrusion tolerance (in Section 3) and to recover control tasks and make them resilient (in Section 4). We report on our work to bound adaptation and reconfiguration times (in Section 5), including how we deal with situations where reconfiguration cannot be bounded. We report on challenges of integrating low-level control into the ADMORPH exchange format and the coordination language compiler infrastructure (in Section 7) as well as how to test control systems with a feedback loop for adaptation (in Section 8). Section 9 concludes.

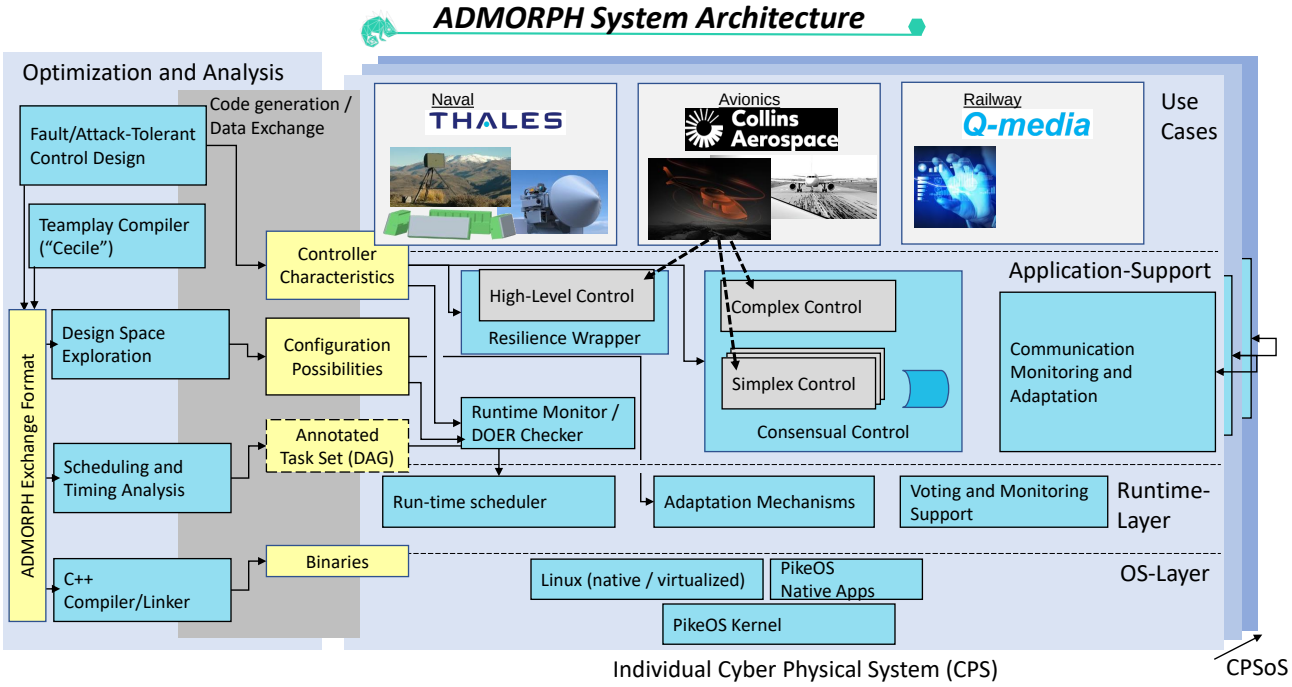

Figure 1: ADMORPH architecture: The figure shows the main components of the ADMORPH architecture and their interplay in relation to the three use cases. Shown is the tool support (left), the runtime components (right), and the interaction with other CPS of the CPSoS (back).

1 Adaptation as Key Enabler of Individualistic and CPSoS-wide Resilience

In ADMORPH and WP2 specifically, we take a holistic view on CPSoS and its components. We assume fault and threat models of incremental and possibly changing strength. This can be in terms of accidental faults or intentionally malicious faults, such as targeted attacks mounted by adversaries. Accidental faults are induced by the environment and change when the CPSoS changes where it is operating. Intentionally malicious faults change based on their intent, access to attack tools and skills of the team (see D2.1b for a more detailed discussion of fault and threat models).

Adaptation is the key to cope with faults, on the long run, provided the system can tolerate faults long enough for adaptation to become effective. In addition, adaptation will serve optimizing functional and non-functional properties. In this WP and Deliverable, we focus on adaptation in relation to faults and threats. As mentioned above, adaptation serves four purposes in this context:

1. to evade faults by relocating services to a different set of resources;

2. to improve resilience by including more resources in the tolerance of faults;

3. to rejuvenate the system by recovering faulty or compromised resources when possible; and
4. to match the systems' resilience to the perceived threat by allocating more or less resources to the defense.

We will report adaptation methods and their interplay by zooming into the runtime elements of the ADMORPH architecture, illustrated in Figure 1. ADMORPH supports both internal and external adaptation of tasks. Internal adaptation implies that the task or component knows by itself how to morph in order to tolerate faults and will trigger such measures autonomously. External adaptation are goverened by the ADMORPH toolchain and happen in a coordinated manner at predefined points in time. We use the former for time-critical configurations, such as restarting low-level control replicas to absorb the faults of compromised ones within a few control epochs. The latter governs more involved reconfiguration, such as transitioning to a new configuration at the end of a schedule's hyperperiod. In particular, internal adaptation is tasked to tolerate faults long enough so that external adaptation has the time to evade the cause and return the system to a secure state.

In the following, we will report on

- adaptation to achieve control-aware fault and intrusion tolerance (in Section 3),
- recovery and resilience of control tasks (in Section 4)
- bounding reconfiguration times (in Section 5) and in particular keeping them low enough so that timeliness can be maintained in spite of faults and subsequent reconfiguration to recover from them,
- network-level monitoring and adaptation (in Section 6),
- interfacing with the coordination language compiler infrastructure (in Section 7), and
- testing runtime systems and adaptation strategies (in Section 8).

We start by reporting on the SOTA advances since deliverable D2.1b.

2 State-of-the-Art since D2.1b (month 22)

In addition to the threats reported in D2.1b, Sargolzaei et al. [**?**] identified a new type of emerging threats: Time-Delay-Switch (TDS) faults. These faults are either injected by an adversary or the result of an accidentally faulty communication channel (e.g., sensor to actuator). In the former case, attacks will follow a planned strategy in order to fully compromise the system and cause damage. The latter will typically be of a stochastic nature and most often transient in nature. TDS manifests as unknown time-variable delays in a control signal, such as the feedback signal from the control regulator to the plant or the controller internal state update.

Several works have been dedicated to monitoring faults and attacks, including TDS attacks, by estimating the state in the control system [**?**].

Zhang et al. [?] developed a framework to lay out the worst-case behavior during design time such that TDS faults can be mitigated.

3 Adaptation to Achieve Control-Aware Fault and Intrusion Tolerance

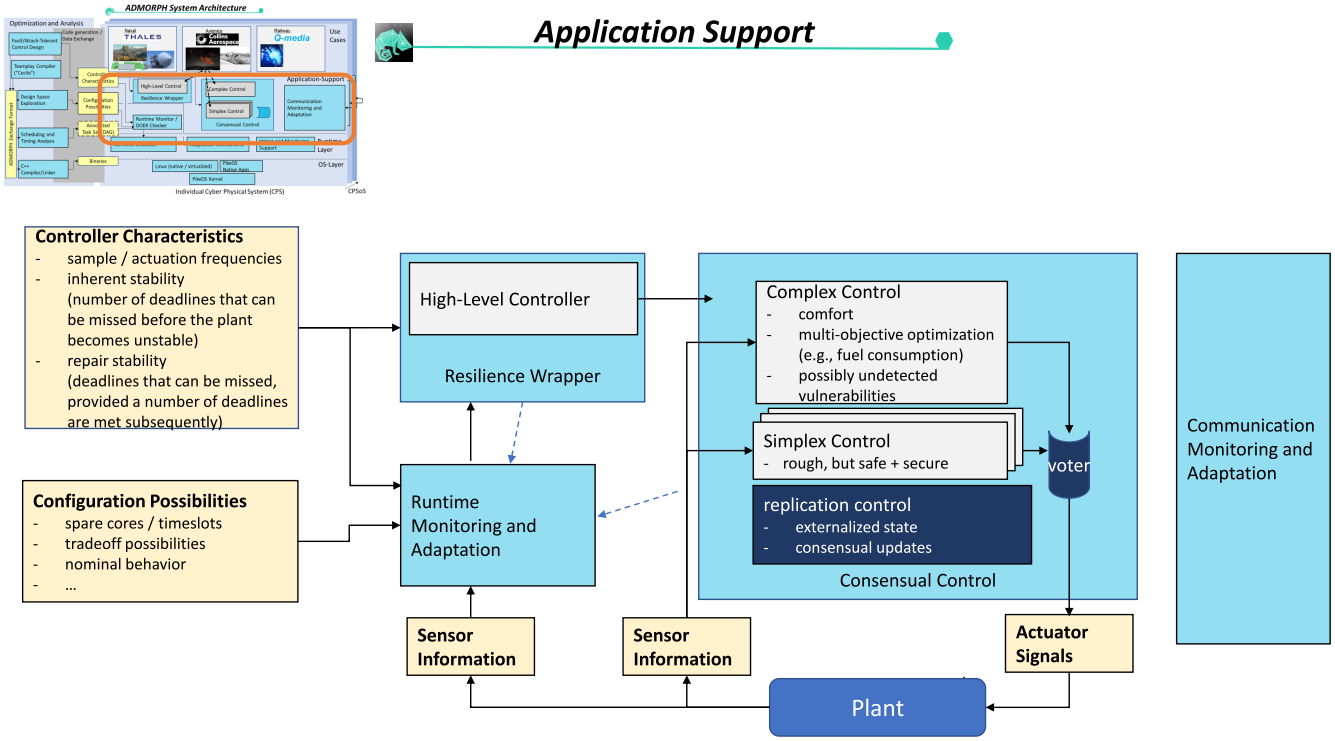

Figure 2: ADMORPH control architecture: The control architecture considers high-level controllers feeding commands into low-level controllers, which themselves are subdivided into a complex control task and simplex controllers, ready to take over in case complex fails. A replication control component coordinates the replication and hence the resilience of the simplex controller. It reacts to reconfiguration requests from the runtime monitoring and adaptation component.

Figure 2 shows the control architecture of ADMORPH. In many situations, including our use cases, high-level controllers steer the behavior of more low-level control loops that run at much higher frequency and have as additional task the stability of the plant. A common example are advanced cruise control systems, which aim at keeping the distance, lane and velocity, while a lower-level controller follows the path they dictate. Similar controls can be found in our taxiing use case, whereby the high-level controller might as well reside outside the controlled CPS, communicating steering commands through wireless connections. Low level controllers can themselves consist of multiple instances of complex and simplex control components, the latter possibly replicated.

We have identified the following adaptation possibilities in such systems:

1. *Functional adaptation*, by transitioning between multiple high-level controllers and the functionality they provide (e.g., manual vs. automated steering, but also internal vs. external control). Such adaptations typically happen in the form of configuration changes at pre-defined points in time and are coordinated by the coordination language, its compiler and tool chain. Functional adaptations at non-pre-defined points in time are event-triggered mode changes and must be considered ahead of time, including the transition period.

2. *Resource adaptations* of the current functionality (e.g., in response to changing loads or resource unavailabilities). Such adaptations are as well steered by the coordination language framework.

3. *Threat-related adaptations*, such as increasing / decreasing the internal resilience of components in response to observed higher or lower perceived threat levels. Adaptation at this stage needs to be a combination of proactive planning of the resources used in case of imminent stronger threats and fast runtime adjustments of the components itself to ensure the system is in the desired state one the threat manifests [6].

4. *Adaptation of the runtime monitor*, in terms of strategies, depth of analysis and monitored components. Adaptation of the above kind typically entails also adapting the monitoring subsystem whose responsibility it is to observe components and identify abnormal behavior. Adapting also the observation and handling strategies allows monitoring to focus on the risks at hand.

The above strategies may lead to adaptations at runtime. These are internal to the control architecture to quickly respond to unforeseen situations and by leveraging excess resources and planned configurations that have already be anticipated during the design-space exploration. Such adaptations may include:

- take over by the simplex subsystem in case complex fails to provide correct information in time
- adjustment of the simplex replication policy by transitioning from a detection quorum, which establishes resilience over subsequent control epochs to immediate masking
- relocation of continuously failing controllers to spare resources
- adaptation of the frequency of rejuvenation

The above control-aware fault-and-intrusion tolerance techniques are enabled by the inherent stability of the plant.

We have implemented the above adaptation methods into the control infrastructure presented in D2.1b and evaluate them in our inverted pendulum demonstrator. Next steps include interfacing with the coordination language and compiler to limit the selected additional resources to those foreseen by the design-space exploration.

4 Adaptation to Recover and Make Resilient Control Tasks

Over time, control tasks fail or become compromised by adversaries aiming to take over the system and cause harm to the environment in which it operates. Recovery of such control tasks and the resources they use is essential to maintain healthy majorities and to continue tolerate failures. We have therefore extended both the state-capturing capability of the replicated controller and the its ability to restart replicas in a stateless manner with the means to also bring up additional replicas. These new replicas start from binaries that are drawn from a pool of pre-compiled and pre-analysed images that are sufficiently diverse to cancel adversarial knowledge how to attack replicas of this kind. Starting without state, the replication controller then injects the captured state and keeps these replicas up to date in the control tasks at hand before transitioning the responsibility to actually control the plant to them. This way, additional replicas can be created and later brought into the active voting group once they are up and running. In the next section, we detail why this two staged approach is required, i.e., why replicas must first be operational before we can consider their outputs in the majority decisions that control the cyber-physcial system and systems of the same.

5 Bounding Time of Adaptation and Reconfiguration

Normally, reconfiguration is limited to starting the new components and the tasks they implement it and transitioning responsibility to them. However, faults may also affect the resources that have been used to run these tasks and components. To not exhaust these resources, in particular in the presence of transient faults, resources must be rejuvenated and reconfigured themthelves. The latter typically implies rebooting the failing resource and testing it to see whether the fault persists.

Similarly, reconfiguration at software level creates new tasks, merges them into the existing schedules and sets up their communication to coordinate with the remainder of the system. Such reconfiguration is highly task and system dependent and may have a runtime that may by far exceed the deadline of the re-configured tasks, even when several of them can be missed.

Primary objective of Task T2.3 is to limit configuration times in order to guarantee bounded down times. We do so by integrating fault models into the scheduling analysis of redundant dataflow tasks. This way, we determine the WCET of such tasks in the presence of errors down to a specific probability.

Some aspects might however still lead to unbounded reconfiguration times, such as repeated re-boots of a resource after a crash or reactivation of the same software vulnerability. These situations typically indicate a systematic and persistent failure in the hardware resource and or the software component that is re-instantiated. For this reason, we will investigate software diversification and relocation possibly to computing resources of different characteristics and the implied consequences on the WCET of this software component. Our plan is to have pre-analyzed images pooled as well as a few resources on stand-by that can then be integrated by either slightly adapting the schedule

or by rescheduling an entire subsystem.

However, even though the above measures will bound the reconfiguration times of the system, we expect most of these bounds to be significantly larger than the worst case response times of tasks in the system. In particular, we consider the case where the system would become unsafe during reconfiguration if control is not maintained throughout the reconfiguration itself. This is where fault tolerance comes into play to buy the time that is necessary to bring up the new subsystems. A second challenge, related to the previous one, is the fact that although each component will be reconfigured in a bounded amount of time, component interdependencies may push the overall reconfiguration time beyond the limit what can be absorbed by our tolerance measures. Therefore, instead of creating reconfiguration cascades by accepting the downtime of a system while a depending system is reconfiguring (which may easily lead to transitive effects), we aim to decouple reconfiguration and inclusion into the operational group as much as possible.

In our control architecture, we have already implemented this decoupling of reconfiguration and inclusion by preparing our system to reconfigure in the following three steps: First, the new configuration starts up, by creating new replicas, by starting components implementing the new functionality or by starting new resources (kept as spares). Second, the new configuration connects with the existing setup, including by ensuring that the new configuration receives state updates and sensor inputs and is as well already monitored. Third, once this preparation phase concludes and all components report their readiness to be included, we transition control to them by atomically updating voters and associated components to consider the new subsystems instead of the previous configuration.

6 Network Monitoring and Adapting Connectivity

CPSoS must not only adapt the individual cyber-physical systems, but also the network of the same in case communication gets jammed or in case certain communication links fail. To that end, we develop machine learning techniques for monitoring the network, both for accidental faults and to detect intrusions. A particular research question is, thereby, how to deal with attacks to the network intrusion detection system itself, which we address by exploiting different forms of diversity to defeat attacks. These range from diverse model datasets and execution platforms to redundancy in the network layer itself.

We have already defined several strategies for exploiting the above diversity on which we elaborate in the following.

6.1 Diversity in NIDS for mitigating adversarial attacks

Cyber-Physical Systems (CPS) are often the target of cyber-attacks, sometimes perpetrated by highly motivated attackers [7]. ML-based NIDS can be used as a defense mechanism to mitigate some of these threats [1]. However, the NIDS itself can be targeted by adversarial machine learning (AML) attacks itself, which is a problem that is not sufficiently well covered in the literature. According to [2], learning-based cybersecurity systems should consider adaptive adversaries that

target the proposed system by searching for and exploiting ML weaknesses. We propose a distributed intrusion detection approach that considers diversity to face this issue. The approach allows the NIDS-based monitoring system be become more resilient to attacks by exploiting software and data diversity.

6.1.1 Approach overview

Our main objective is to mitigate traditional adversaries, which target the CPS being protected by the NIDS, and also mitigate the adversaries that target the NIDS itself, in particular its ML infrastructure. We propose an architecture that explores different forms of diversity (Figure 4). According to [5], diverse components have different vulnerabilities, and the greater distance/difference between them, then the lower chances of the same vulnerability occurring. Therefore, when using redundant components as means to tolerate attacks, diversity of the redundant components is essential to ensure that the attacker cannot simply replicate the attack to be successful. Diversity will force the attacker to find and explore different vulnerabilities in each component, thus ensuring that the system survives attacks and operates correctly for a longer amount of time.

The multiple forms of diversity that we consider and propose are the following:

- **Diversity of View**, which consists in exploiting different feature sets. If an attacker changes one of the views, it is possible to identify it through the other views.
- **Diversity of Model**, in which multiple and different models are used. The attacker will have to know the characteristics of different models, as needed to evade all them by applying different algorithms. Moreover, as reported in the literature, some ML algorithms are more robust than others against AML according to the vulnerability.
- **Diversity of Workers**, which consists in having the NIDS being replicated, which each replica being different from the others. To compromise the NIDS, the attacker has to compromise multiple workers. If an attacker compromises only one of the workers, it is possible to remove the compromised machine from the system without significant losses for the intrusion detector, working only with the remaining workers.

The approach overview is depicted in Figure 3. It is composed of a master node and n workers defined by the security manager. The master node is responsible for deploying the worker nodes and globally consolidating all local views from the workers. We note that this master function is an abstract function, which is not (and should not) be performed in a single component. In a real deployment, this function has to be implemented in a distributed and also replicated way, for instance by distributing it among the multiple worked nodes. Known techniques for Byzantine Fault Tolerance (BFT) can be used for this purpose, so at this point we do not explicitly focus on this aspect and we simply represents the function as central function.

The worker node is responsible for listening to network traffic at a given point. It has a diversity layer composed of different models trained with different views and algorithms. Also, the worker is responsible for sending its status frequently to the master node. The specification of each worker

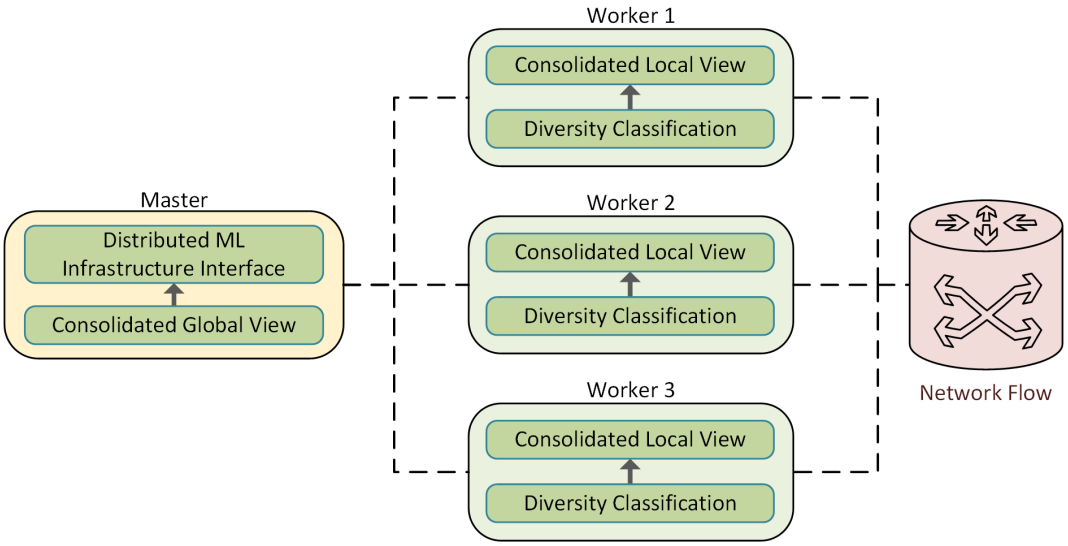

Figure 3: Proposal - Distributed and Resilient Network Intrusion Detection System Overview.

node is defined when the master node instantiates it. That approach works by monitoring only network traffic to detect suspicious activity.

It is important to emphasize that our point here is not just to create many systems replicas. While such a strategy could work for accidental failures, it won't be effective for cybersecurity attacks. Creating many system replicas without diversity will result in an environment running the same system's vulnerabilities. In that case, the adversary could put down the entire environment by exploiting a single vulnerability, common to all replicas.

6.1.2 Approach specification

Figure 4 provides a detailed view of the proposed architecture. The Network Assets component represents the network devices on the CPS. Those devices are connected to the network producing traffic. The role of the master is essentially deploying and managing the worker nodes. The worker nodes are deployed in a given network point, listening to the traffic. The network traffic is collected through a network packet analyzer, e.g., Wireshark or TCP-dump. The traffic is sent to the feature extractor, which transforms raw data into features that better represent the underlying network flow. The feature extractor extracts many different feature sets, called views. Views are sights from the same event, which look at the different features. Those built views feed many models. The models' layer is composed of n models, trained with many different combinations. The combinations range from different views to different classifiers. The model output or inferred values are sent to consolidate the local view. At this point, we are applying the majority vote. Where each model signs one vote, and the most voted class is picked as the winner class. Then, the worker reports its consolidated view to the master node, consolidating all the workers' views and raising alarms if anomalies are detected. Again, we note that this consolidation of the final decision

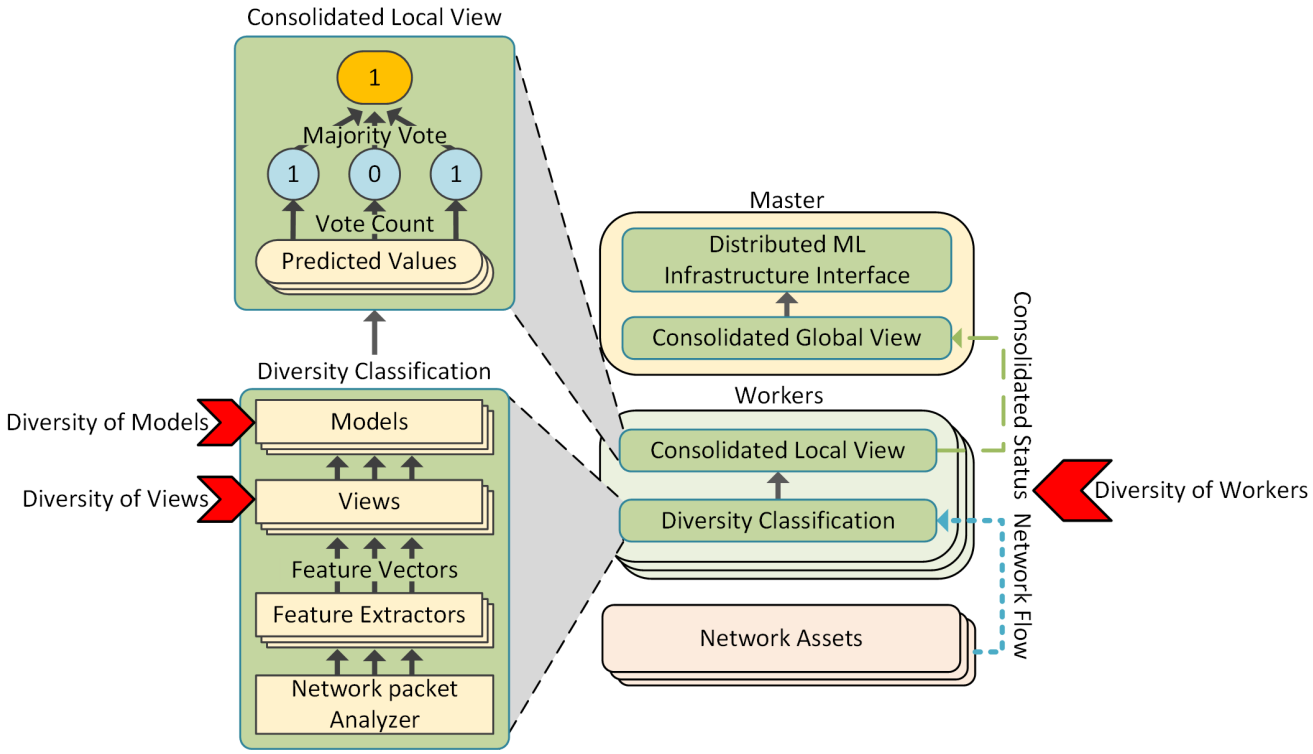

Figure 4: Proposal - Distributed Network Intrusion Detection System Detailed View.

(which is simpler) can be (and should be) also done in a distributed way, to avoid introducing a single point of failure/attack in the system.

6.2 Summary and next steps

This deliverable explored the stat-of-the-art of dependable network monitoring that supports systems adaptation. We proposed a distributed network intrusion detection system that employs distinct diversity as its main characteristic to mitigate adversarial attacks.

NIDS is a well-known cybersecurity mechanism to detect cyber threats that give support for system adaptation. The proposed architecture employs NIDS ML-based, which is recommended according to the CPS environment. Its environment is dynamic and usually composed of dozens or hundreds of network devices producing traffic. Usually, the CPS domain produces a significant amount of data, and there is a need to analyze current traffic in time. The NIDS ML-based can analyze network data and identify new attacks as time passes without needing a specialist to create security rules. The intrusion detector ML-based learns rules from the data.

The diversity was not sufficiently explored in the NIDS literature, but it is well-known in cybersecurity in general. As previously cited, OS and classifier diversity contributed to vulnerability reduction. When we talk about AML, there is a challenge that it is not possible to deploy all known techniques to mitigate AML attacks. Usually, those techniques decrease the model's accuracy and

increase computational complexity.

The proposed approach exploits the diversity concept. Instead of consuming the computational resources to create systems replicas (which does not work for cybersecurity), the alternative is to deploy multiple software variants with the same functionalities but different designs. Also, using many views from the same event, looking at different perspectives. The assumption is that the diversity technique will help mitigate adversaries indirectly by deploying systems diversity and data diversity, which, according to literature, contribute to vulnerability reduction.

The preliminary studies resulted in a survey paper about AML in NIDS, which describes attacks and defense techniques used in the literature. One of the contributions of our survey is about the feasible techniques of AML in the NIDS context. Usually, the techniques used in AML to NIDS come from other areas that have no or weak constraints in the feature space, such as object detection or spam, and the network is a constraint domain. The adversary can not arbitrarily change the network packet field. Having explored the state of the art, our next steps are: 1) building relevant datasets. Since the literature on NIDS lacks datasets and datasets for the CPS domain, are even more scarce. 2) Building models exploiting the proposed architecture, employing various ways diversities. Also, building models for baseline solutions, exploiting the state of the art approaches used to mitigate adversaries. 3) Evaluating the proposed architecture by comparing the results to the baseline models through traditional ML and AML metrics and improving models.

7 Interfacing with the Coordination Language Compiler Infrastructure

Interfacing with the coordination language compiler infrastructure is based on the ADMORPH Exchange Format (AXF), a text-based graph representation with additional attributes. For task sets that can be represented as directed acyclic graph, AXF will soon allow interfacing with the AROMA compiler and runtime system and it supports running TeamPlay coordination codes on AROMA supported hardware, such as the Kalray MPPA.

However, we have also already identified several challenges to integrate implicitly synchronizing tasks and tasks with different inherent periods. Communication in the AXF needs to be expressed explicitly. Moreover, since the whole task set is characterized as graph, periodic tasks would need to be unrolled and all instances up to the hyperperiod be represented. In particular, reconfiguration of tasks would only be possible at the end of this hyperperiod respectively at certain reconfiguration boundaries that cut across the DAG.

To address the above challenges, we are are currently reserving additional resources for low-level control tasks to adapt internally for the purpose of tolerating faults until the time for a more elaborate adaptation of the system comes within reach. We are currently investigating how to better integrate internal and external reconfiguration and the interplay between the two.

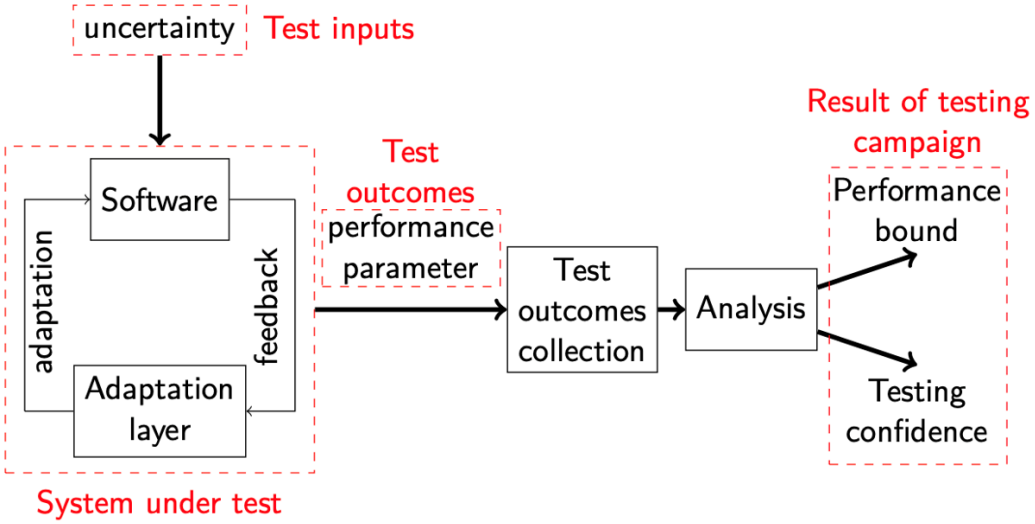

Figure 5: Testing framework

8 Testing Runtime Systems and Adaptation Strategies

Being able to test runtime systems and adaptation strategies, in particular in extreme situations, is a crucial for safety critical systems. Tests are commonly performed in a testing framework, such as the one depicted in Figure 5, which we use in ADMORPH. The system under test, which in our case contains both the main hardware and software components and the adaptation layer which feeds back into the software, adapts it and changes its behavior, is provided test inputs and generates in return the test outcomes at a certain measurable performance. The latter are fed into an analysis to determine performance bounds and test confidence.

In Mandrioli et al. [3, 4], we have investigated the specific setup where software inside the system under test has a feedback loop that allows it to adapt to various situations. In particular, we have investigated how to test such system. One example of such a feedback loop concerns the number of replicas for a controller to adapt to different threat levels and fault rates. Another example adjusts video processing on the fly.

Drawing from statistics, one typically finds three methods to test a system. Monte Carlo investigates the average behavior of a system by observing how the system reacts to examples randomly drawn from an input distribution. However, while this accurately represents the system behavior in the average case, we must also look at extreme situations. Extreme value theory argues about such extreme behavior based on the observed maxima to determine the likelihood of missing the extreme in the observed case.

Our approach is based on scenario theory, which tests the confidence β of having missed a scenario above the probabilistic bound ϵ in a number n of tests.

We are currently investigating the level at which tests are best performed (e.g., hardware-in-the-loop vs. software-in-the-loop) to achieve a good testing coverage. Moreover, we investigate how

the testing of control software can be guided by the control design process in a general manner. In other words, we identify properties that the behaviour of the controlled system (composed of physical plant and controller) should expose, based on the control design. We are developing a testing methodology that would allow us to certify that the closed-loop behaviour is coherent with the controller specification.

9 Conclusions

In this deliverable, we have presented the recent advances made in Task T2.1 – T2.6 of WP2 and our findings in terms of adaptation methods and how to embed them into the ADMORPH architecture. Although adaptation is governed by offline-computed strategies, possibly originating from design-space exploration, the actual adaptation performed at runtime requires care to secure fast response times of the individual CPSoS building blocks. In particular tasks with stringent timing requirements may need to be equipped with internal resilience mechanisms to absorb faults of an accidental and malicous nature. We have identified and already partially implemented several strategies for adapting to different situations, for bounding reconfiguration times and, in particular, for decoupling reconfiguration from the operational behavior of components, specifically controllers. The latter secures low response times by already bringing the new configuration into an operational state before responsibility is transferred.

Next steps include investigating how to more closely integrate internal adaptation and the coordination language and how to secure the monitoring and adaptation layer from faults.

10 References

[1] Zeeshan Ahmad, Adnan Shahid Khan, Cheah Wai Shiang, Johari Abdullah, and Farhan Ahmad. Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Transactions on Emerging Telecommunications Technologies*, 32(1):e4150, 2021.

[2] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. Dos and donts of machine learning in computer security. In *Proc. of the USENIX Security Symposium*, 2022.

[3] Martina Maggio Claudio Mandrioli. Testing self-adaptive software with probabilistic guarantees on performance metrics. In *ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020)*, November 2020.

[4] Martina Maggio Claudio Mandrioli. Testing self-adaptive software with probabilistic guarantees on performance metrics - extended and comparative results. *IEEE Transactions on Software Engineering (TSE 2021)*, January 2021.

[5] Miguel Garcia, Alysson Bessani, Ilir Gashi, Nuno Neves, and Rafael Obelheiro. Analysis of operating system diversity for intrusion tolerance. *Software: Practice and Experience*, 44(6):735–770, 2014.

[6] Douglas Simoes Silva, Rafal Graczyk, Jeremie Decouchant, Marcus Voelp, and Paulo Esteves-Verissimo. Threat adaptive byzantine fault tolerant state-machine replication. In *2021 40th International Symposium on Reliable Distributed Systems (SRDS)*, pages 78–87, 2021.

[7] Jean-Paul A Yaacoub, Ola Salman, Hassan N Noura, Nesrine Kaaniche, Ali Chehab, and Mohamad Malli. Cyber-physical systems security: Limitations, issues and future trends. *Microprocessors and microsystems*, 77:103201, 2020.