D2.1: Identified Adaptation Possibilities and Methods

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

Due Date:	Month 12
Delivery:	Month 12
Lead Partner:	UniLu
Editor:	Marcus Völp, UniLu
Dissemination Level:	Public (P)
Status:	draft
Approved:	
Version:	0.1

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871259 (ADMORPH project).
This deliverable reflects only the author's view and the European Commission is not responsible for any use that may be made of the information it contains.

DOCUMENT INFO – Revision History

Date and version number	Author	Comments
12/12/2020 ver. 1.0	Marcus Voelp	First draft
21/12/2020 ver. 1.1	Marcus Voelp	Internal review

List of Contributors

Date and version number	Beneficiary	Comments
12/12/2020 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

Workpackage WP.2 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 CPS(oS), 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 is the first deliverable of workpackage WP.2. We report on preliminary work done in the context of Task 2.1: Control-aware fault and intrusion tolerance (FIT), on identified adaptation opportunities and methods.

Task 2.1 is the only task active in this period. Tasks 2.2 - 2.6 start in month M.13 and M.19 respectively. Task 2.1 suffered from severe hiring difficulties, which we describe in further detail in Sec. 3. We decided to not defer the submission of this delivery, but instead to compensate by providing an update on its subject matter in Deliverable D.2.2.

The goal of Task 2.1 is to develop FIT protocols and techniques to leverage application-domain knowledge (here control) in the design of advanced resilience mechanisms.

Adaptation serves four main purposes:

1. to evade adversaries in their ongoing attacks;
2. to improve the resilience of systems during ongoing attacks, possibly by degrading the systems' quality of service, if necessary;
3. to return the system to a state at least as secure as initially; and
4. to optimize the system whenever the perceived threat level drops.

We identified a tension between the classical world of real-time and embedded systems, focusing on predictability as first principle and hence known upper bounds for computation and communication latencies, and the body of knowledge on resilience mechanisms, in particular fault and intrusion tolerance. The latter operate in what researchers in that domain call asynchronous or partially synchronous systems and protocols, where such bounds are not given per se, in particular while the system is under attack.

Our goal is to reconcile both worlds and to research and develop adaptive resilience mechanisms (as far as Task 2.1 is concerned for controllers) that are able to provide timing guarantees, not from the predictability of all components, but from enough components not being affected by adversaries both in the value and time domain.

1 Towards Individualistic and CPSoS-wide Resilience

Concerning the specific contributions of this deliverable, it first provides a review of the state of the art (SOTA) on adaptive resilience, from which it is possible to conclude about open issues and opportunities that we will explore in ADMORPH. Then, the deliverable introduces some fundamental concepts on dependability and security, also describing baseline assumptions that pave the way for defining appropriate system and fault models. The deliverable is then devoted to describe our initial view on a generic FIT control architecture, including a description of its components, their role and possible interplay. Our view on how this architecture will support resilient adaptation, and what kinds of adaptation we anticipate at this stage of the project, are then described. The deliverable concludes with some considerations on the work done in this task so far, referring to aspects that conditioned the progress of the work as initially planned.

1.1 Background and Related Work on Adaptive Resilience

RTES typically operate under system assumptions that the dependability community [20] summarize under the *synchronous system model*. That is, RTES are typically assumed to operate in environments where communication and computation are sufficiently predictable to derive upper bounds (typically worst-case transmission or execution times). In the presence of highly skilled and well equipped adversaries opposing the system, the implied assumption, namely that communication and computation bounds also hold while the system experiences severe accidental faults or intentionally malicious attacks, cannot be justified with high confidence.

1.1.1 Partial Synchrony:

To tolerate situations where adversaries affect some of these properties, a large part of the dependability community switched to *partially synchronous system models* [10], where bounds of this kind only exist during frequently recurring "good" periods, while the system might exceed such bounds during those times when it is under attack. Denial of service attacks are one prominent example of a network-level attack that violates most if not all timing guarantees that one has derived for normal situations. Other examples of attacks, affecting cyber-physical systems more directly, include Aurora [8], Stuxnet [12] and the attack to the Ukrainian power grid [15].

1.1.2 Partially Synchronous BFT-SMR Protocols:

Byzantine fault tolerant statemachine replication (BFT-SMR) protocols, built for such partially synchronous system models, either aim to replace time-based omission detection with time-free failure detection [14], or, like in the seminal protocol PBFT [5], use time merely as a mean to eventually react to faults. For example, PBFT implements a leader/follower scheme with the leader defining the order in which requests should be executed, while followers confirm this order or re-elect a new leader. Even though leader election is triggered by timeouts (e.g., if no consensus could be reached on a client's request), the protocol can be considered as *time-free* because the safety of the system does not depend on this leader change to happen. Lifeness is affected and regained during long enough "good" times.

We can therefore conclude this initial definition of terms, by saying that a system is *time-free* if its safety does not depend on some form of reliable time source and by introducing the *asynchronous system model* as one where communication and computation bounds cannot be known, even during "good" times.

1.1.3 Reconciling Synchrony:

Obviously, asynchronous and partially synchronous systems cannot give the time bounds real-time and embedded systems require, even in such harsh and unpredictable environments (e.g., during ongoing attacks). Our goal is therefore to carefully craft resilience mechanisms to rely as much as possible on *time-free* constructs, while guaranteeing timeliness as a property of, for example, enough replicas operating in a timely manner. We admit that this will not be possible during all times and explore adaptation, possibly in combination with architectural hybridization, to survive long enough and place replicas outside the adversaries' path of attack, where we hope to regain the timeliness we need, at least for long enough to fail operationally into a fail safe state.

1.1.4 Architectural Hybridization:

Architectural hybridization [19] includes trusted-trustworthy components that follow a distinguished system and fault model (e.g., they may operate or even communicate synchronously and fail by crash only) for the purpose of making the whole system will be more resilient by relying on functionality of these trusted subsystems. The argument that justifies this exceptional role, i.e., why trusted-trustworthy components can be made resilient by construction, stems from their simplicity, which is still a matter of ongoing research.

1.1.5 Adaptive Resilience

Barring a more elaborate review of the state-of-the-art in adaptive resilience, which we will prepare as requested in the technical review, we will focus here primarily on recent works about adaptive BFT-SMR protocols.

Most of the state-of-the-art on adaptive resilience mechanisms [4, 17, 6, 1], in particular in terms of Byzantine fault and intrusion tolerance techniques, suffer from the above described divergence of the dependability and real-time/embedded systems communitiess, which will require some reconsolidation effort to obtain time-free, partially synchronous resilience mechanisms (capable of withstanding time domain attacks) that exhibit predictability and timeliness as an emerging property.

In the first class of time-agnostic, but not real-time capable protocols, WHEAT [18] decreases client latency by selecting as leader the best connected replica. AWARE [3] improves over WHEAT by continuously monitoring connectivity and reconfiguring based on these observations. Adapt [2] builds on Abstract and Aliph [1] to perform AI-driven dynamic adjustments of the payload protocol.

Unlike the above protocols, which primarily focus on optimizing the system, we are currently investigating means to react to situations where the perceived threat level increases. An early result from this ongoing work is that group-membership based BFT-SMR protocols [4, 17, 1], because of the impossibility identified by Chandra et al. [7], necessarily require consensus to adapt to threats increasing adversarial strength and this consensus must be synchronous and hence provided as part of the trusted-component functionality. Otherwise, without time bounds on reaching consensus about the new configuration, it will be impossible to outpace adversaries, which are naturally not constrained by the partial synchrony we assume them to cause. Obviously, because the complexity of such a trusted wormhole would by far exceed the complexity of traditionally used trusted components, such a solution is not very desireable.

Our ongoing research therefore focuses on the one side on proactive means to prepare for situations where the system has to adapt to increasing threats and on the other side on more lightweight mechanisms that enable consensus at a much lower complexity, but limited to the individualistic system. In the following we describe how these and similar resilience mechanisms apply inside a single system.

1.2 An Abstract View on Control

Before we proceed, it will help to abstract from the concrete usage scenarios that drive our developments (see also Deliverable D.5.1) to then return in D.2.2 and see how the individual resilience mechanisms can interplay in an adaptive manner.

Figure 1 illustrates what we believe are essential building blocks for securing individual CPS, making them resilient to accidental faults and attacks of varying strengths. Individualistic resilience of each CPS will be essential for obtaining CPSoS-wide resilience because the impact each individual CPS has on the physical world can, if this CPS gets compromised, be subverted into a threat to other CPS or worse, the humans operating in their proximity. For example, by exploiting a vulnerability in the radio telemetry subsystem of their Jeep Cherokee [11], researchers Charlie Miller and Chris Valasek had full remote control over their vehicle and from there its only a matter of imagination and good or bad intend whether such a compromised vehicle is turned into a cyberkinetic weapon against the platoon of cooperatively driving cars.

Our primary goal is therfore to make, through adaptation, each individual CPS strong enough to offer a minimal residual safety and security, and to ultimately recover, possibly with the help of healthy CPS in the individual's proximity.

What makes us confident that concentrating on the CPS will extend smoothly to CPSoS is that we can already identify self-similarities in the techniques and principles applied within the CPS. For example, much like a fernleaf repeats its own pattern, cascaded complex control (e.g., learning-based, complex components as they are required for autonomous driving, or in our case taxiing on the airfield) may well be tolerated to fail, or be protected by more lightweight mechanisms if they can rely on the abstraction of a resilient controller, capable of preventing both time- and value-domain faults from manifesting at the level of the actuator. The same general pattern recurs when coordinating the actions of a multitude of CPS in a CPSoS. If each CPS individually exhibits the notion of an in principle resilient system (possibly with degrading QoS in extreme situations),

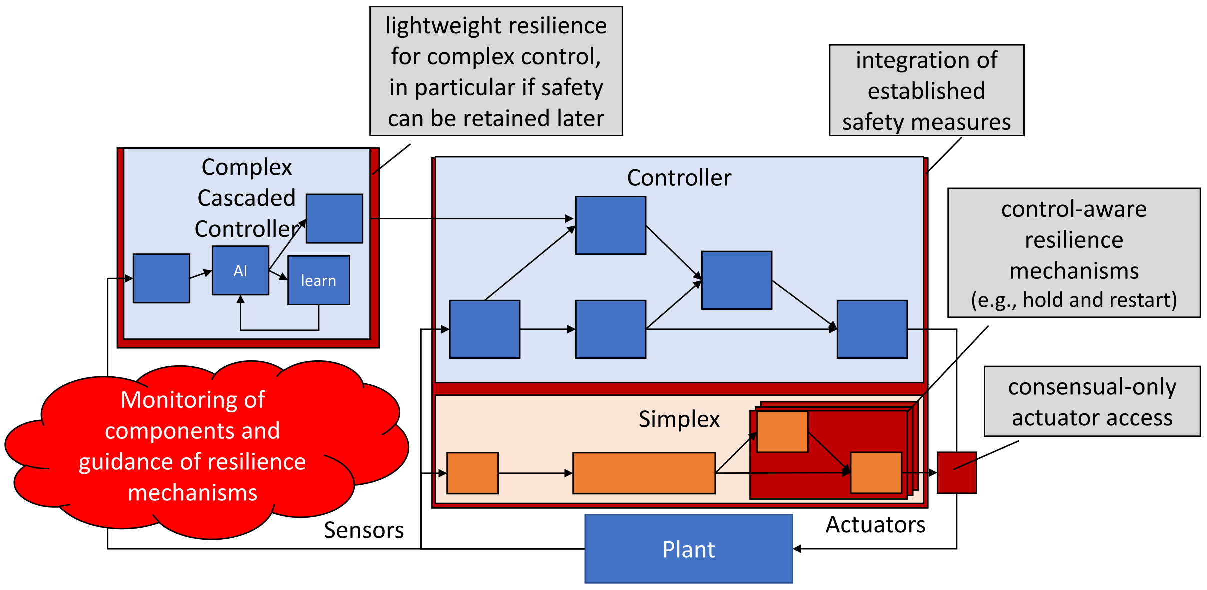
lightweight resilience for complex control, in particular if safety can be retained later
integration of established safety measures
Complex Cascaded Controller
AI
learn
Controller
control-aware resilience mechanisms (e.g., hold and restart)
consensual-only actuator access
Simplex
Monitoring of components and guidance of resilience mechanisms
Sensors
Plant
Actuators

Figure 1: Abstract view on a generic control architecture, its resilience measures and their interplay. Shown are three cascaded control subsystems with archetypical task graphs (blue and orange boxes) and resilience mechanisms (in red). Simplex control can take over and override the decision of the regular controller of the plant to enforce safety and security properties. The complex cascaded controller leverages this combined regular/simplex control subsystem for more complex decision making (e.g., AI driven and learning enabled).

CPSoS wide cascaded controls may leverage this internal resilience and the fact that individuals no longer fail in the most pessimistic manner, but perform a detectable, coordinated and fail operational reduction of QoS into fail safe states.

Despite the late start of Task 2.1, we have already been able to identify a few significant building blocks towards this resilience, which we depict in Figure 1.

1.2.1 Cascaded Control and Safety Kernels

Underlying the separation into a "normal" controller and a cascaded higher-level controller is the observation that many properties of the plant (e.g., its stability or its ability to follow a certain directive, such as a trajectory). Once this split is identified and the cascaded controller separated and isolated, faults in the complex controller can propagate to the "normal" controller only through the directives / high-level control signals it emits or due the timing of the very same. If timing and value of this control signal can be validated and compensated (e.g., by injecting the high-level control signals of a fail operational maneuver) and if the "normal" controller would be resilient to attacks, then it would be possible from a safety perspective that the complex controller fails in an arbitrary way (of course degrading the quality of service of the system).

The role of the safety kernels is to ensure exactly that, namely with knowledge of the application at hand to ensure that emitted high-level control signals are correct relative to the sensor readings of the plant and relative to the internal state of the complex controller, which we will externalize to these safety kernels though additional "internal" sensors.

Recovery patterns for returning to full quality of service will focus on reestablishing the complex controller, possibly on a different subset of the system resources away from the adversaries' attack path.

1.2.2 "Normal" vs. Simplex Control

Unfortunately, not all of the desired plant properties can be decomposed into such cascaded properties, with the simpler ones already exhibiting most of the safety we desire. However, often the same safety-related properties can be achieved sometimes also by simpler, but in general undesirable low-level control decisions. If this is a case, we obtain additional resilience by supporting Complex/Simplex design patterns. Such patterns are characterized by a simplex controller being able to take over and overwrite the control decisions of a more complex (in our case "normal") controller, in particular in those situations where safety is at risk. A prominent example is a simplex autopilot, tasked to perform aggressive, fuel demanding maneuvers in case the complex autopilot fails to return the system into a safe state [13] (and similar for aerial vehicles [21]).

Again, the resilience pattern suitable for "normal" controllers of this kind is by recovery, because the simplex controller can take over in case the plant risks entering a state where safety is at risk, although again at the cost of degrading quality of service.

One of the research questions is to identify the control components that are part of the cascaded system and of the "normal" control system to then search for simpler, safe but not necessarily comfortable variants to form the simplex controller.

A second pertains to the resilience mechanisms of the simplex control system. As there is no further underlying controller to take over in case safety is at risk. Redundancy-based mechanisms suggest themselves at this level, which leads us to the next question.

1.2.3 Control-aware resilience mechanisms

The vast majority of resilience mechanisms are generic and therefore agnostic of the application they protect. This leaves unused much of the potential that an application-aware resilience mechanism could reveal. In case of control, we know from our partners at Uni Lund, Maggio et al. [16] that some properly controlled plans can already tolerate several omission faults if the previous control signal is held during omission and if the controller is researted. Leveraging this knowledge and the fact that the individual tasks, a controller is comprised of, rather exibit a graph structure than client-server invocation patterns, we can design control-aware resilience mechanisms to tap into this potential.

1.2.4 Consensual actuation

With all this in place, a majority of correct control signals should be available and just needs to be applied at the plant's actuators. However, most plants are resilience agnostic, in particular about the fact of changing responsibilities when we adapt the low level controller and relocate for example a replica to a differnt core. Therefore, to prevent a compromised control replica from interfering with the plant at exactly the right moment where no further overwrites by the other replicas will be possible, we ensure consensual-only access to actuators by means of architectural hybridization and a trusted-trustworthy component. Consensual-only access implies that only the majority decision among all active replicas will be forwarded to the actuator and otherwise the previous value will be held.

2 Adaptation Opportunities and Methods

In the following, we will report on several adaptation opportunities and methods, which we already identified and on the additional research we plan to conduct. As illustrated in Deliverable D1.1, adaptation will be the key to compensate accidental faults and to fend off adversaries mounting a targeted attack to the system. Adaptation will act along the following four lines:

1. to replace and relocate components along the attack pathway, creating a moving target defense to buy the time that is necessary to repair and recover compromised components and in turn strip adversaries from their foothold in the system;
2. to adjust the internal resilience of components, by replacing configurations with more resource demanding configurations that exhibit a better resilience to match the perceived threat level;
3. to adjust the functionality of the system to guarantee a degraded service in case not enough resources remain to sustain the full functionality of the system; and
4. to optimize the system whenever the perceived threat level drops.

It is important to realize that the first three happen while the system has detected the presence of an imminent threat, whereas the latter applies in those situations where the system has gained confidence that it is no longer exposed to the high risks it has prepared for. In consequence, any adaptation of the former three kinds has to outpace the adversary in adapting faster to improve system resilience before the adversary can exploit the current vulnerable state from which the system tries to evade.

2.1 Adapting the System along Attack Pathways

Components are characterized by their state and executable and will interact with other components through well defined communication interfaces (e.g., the inter-process communication mechanism (IPC) provided by the PikeOS microkernel). Relocating a component from a faulty core and redirecting IPC connections to or from this components evades faulty resources and presents adversaries a fresh instance, in particular if a component is not replaced by an instance the adversary was already able to analyze, but instead by instances the adversaries has not seen, yet. Depending on whether or not the state has been compromised, it may be either transferred from the previous component or re-instantiated from scratch or from a checkpoint. Adaptation of a component generally entails adaptation of other elements of the system, most notably the schedule, which defines how applications are mapped and multiplexed to resources.

Dynamically changing the attack pathways in the manner described above evades accidentally faulty resources because the new instance will be mapped to a different subset of resources and it requires adversaries to redo the work they have already spent to prepare the attack of the old instance.

Let us exemplify this for an attacked component with a network connection to the outside. Such a component typically interacts with a network stack and network interface card (NIC) driver, which implement the communication protocol (e.g., TCP/IP) and network hardware interaction, respectively. Naturally, being exposed to the environment, the NIC driver and network stack are typical candidates for a first compromise of the system. However, aside from ongoing transmissions and open connections, the internal state of the network stack and NIC driver is largely disconnected from the state of the receiving component. It is therefore possible to periodically rejuvenate the stack into a state frozen after initialization and to repeat NIC initialization to remove adversaries that have entered this most external layer of the system. Rejuvenating and replacing the networked component, requires adversaries to repeat the work they have already performed for compromising this network layer before they can identify and exploit vulnerabilities in the receiving component. The latter is of course provided we can present the adversary with an instance he was not able to analyze.

2.2 Adapting Internal Resilience

Many components or conglomerate of components already exhibit some form of internal resilience to accidental and malicious faults. For example, triple modular redundant systems, or more generally replicated systems implementing Byzantine agreement operate n replicas out of which f replicas may become compromised before the adversary gains control over the conglomerate of these replicas. n and f are typically related (e.g., $n \geq 3f + 1$ for homogeneous, partially synchronous consensus [5]). Thus, increasing n also increases f and the internal resilience of this conglomerate.

By homogeneous consensus protocol, we refer to a protocol running in a homogeneous system. In such a system, all components follow the same fault model. In contrast architectural hybridization [9, 19] allows identifying trusted-trustworthy components that follow a distinguished fault model (e.g., they fail only by crashing, while the remaining system may exhibit arbitrary, possibly Byzantine faults).

In the most general case, changing the membership of which components belong to a group of replicas boils down to a consensus problem, requiring agreement for addition and removal. It therefore depends on the system model whether reaching such an agreement faster than the adversary compromises more than the initial f replicas is still possible. We are currently exploring possibilities to proactively prepare for such an adjustment to later be able to react without first having to reach consensus how this reaction should look like. In particular, we investigate how the inherent resilience of a plant to missed, wrong or held actuation helps performing these adaptations more efficiently.

For example, the state-of-the-art body of knowledge requires $2k$ additional replicas to operate safely and securely through attacks while up to k replicas are repaired and returned to a state at least as secure as initially. The ability to tolerate some deadline misses in the case of continuous disagreement among smaller quorums allows us to delay the activation of these replicas or to avoid them in the first place.

2.3 Adapting Functionality

Deliverable D5.1 already identifies how the ADMORPH use-case scenarios can adapt their functional and non-functional properties to adjust to those situations where not enough resources remain to sustain full system functionality. We draw functionally reduced components from an initial pool of deployed alternatives, but consider also means to supply such a pool dynamically.

2.4 Adapting to Lowering Threat Levels

As already mentioned, the fundamental difference between the above three and this fourth adaptation opportunity is the time until which the adaptation must have happened in relation to the time the adversary needs to break into the system. When optimizing, the system is already confident about the absence of faults and attacks at the system's current threat level. Therefore, there is no bound until which the adaptation must succeed other than the desire to quickly return to a more efficient modus operandi and to do so in steps that will not jeopardize the timing guarantees the system has to provide. In particular, optimization can always be aborted to react to increasing threat levels.

3 Covid-19

Workpackage WP.2 is affected by Covid-19 related hiring difficulties at the two partners, which were supposed to start in this WP from month 1: FC.ID and UNILU. We have taken measures to compensate for this delay by covering those parts of the work at PI level that cannot be delayed without also delaying other partners. To this end, we have identified and defined the main two building blocks — *safety kernels* to monitor the system and guide adaptation decisions, and *trusted voters* to establish a root of trust for the majority of resilience agnostic devices — and are in the process of implementing them as part of the runtime support system (WP.4). These two mechanisms will already support adapting the replication degree of controllers and will be subsequently extended towards more advanced resilience patterns.

4 Conclusions

In this deliverable we gave account of the progress achieved in WP.2 and how the active partners compensated Covid-related hiring difficulties. We identified the importance of individualistic resilience in CPSoS and provided an abstract view of a control architecture we believe is capable of achieving this resilience in an adaptive manner. Moreover, we identified adaptation opportunities and methods.

5 References

[1] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The next 700 bft protocols. *ACM Trans. Comput. Syst.*, 32(4), 2015.

[2] Jean-Paul Bahsoun, Rachid Guerraoui, and Ali Shoker. Making BFT protocols really adaptive. In *2015 IEEE International Parallel and Distributed Processing Symposium*, pages 904–913. IEEE, 2015.

[3] Christian Berger, Hans P Reiser, João Sousa, and Alysson Bessani. Resilient wide-area byzantine consensus using adaptive weighted replication. 2019.

[4] Carlos Carvalho, Daniel Porto, Luís Rodrigues, Manuel Bravo, and Alysson Bessani. Dynamic adaptation of byzantine consensus protocols. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, pages 411–418, 2018.

[5] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, page 173–186, USA, 1999. USENIX Association.

[6] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, November 2002.

[7] Tushar Deepak Chandra, Vassos Hadzilacos, Sam Toueg, and Bernadette Charron-Bost. On the impossibility of group membership. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '96, page 322–330, New York, NY, USA, 1996. Association for Computing Machinery.

[8] CNN. Xmouse click could plunge city into darkness, experts say", April 2018.

[9] Miguel Correia, Nuno Ferreira Neves, and Paulo Verissimo. How to tolerate half less one Byzantine nodes in practical distributed systems. In *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, 2004.*, pages 174–183. IEEE, 2004.

[10] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, April 1988.

[11] Andy Greenberg. Hackers remotely kill a jeep on the highway—with me in it, July 2015.

[12] Gregg Keizer. Is stuxnet the 'best' malware ever?, Sept. 2010.

[13] Seong Kyung Kwon, Ji Hwan Seo, J. Lee, and K. Kim. An approach for reliable end-to-end autonomous driving based on the simplex architecture. *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 1851–1856, 2018.

[14] Gérard Le Lann and Ulrich Schmid. How to implement a time-free perfect failure detector in partially synchronous systems. Technical Report Research Report 28/2005, Technische Universität Wien, 2005.

[15] Robert M. Lee, Michael J. Assante, and Tim Conway. Analysis of the cyber attack on the ukrainian power grid, March 2016.

[16] Martina Maggio, Arne Hamann, Eckart Mayer-John, and Dirk Ziegenbein. Control-System Stability Under Consecutive Deadline Misses Constraints. In Marcus Völp, editor, *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*, volume 165 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:24, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[17] Michael K Reiter. A secure group membership protocol. *IEEE Transactions on Software Engineering*, 22(1):31–42, 1996.

[18] Joao Sousa and Alysson Bessani. Separating the WHEAT from the chaff: An empirical design for geo-replicated state machines. In *2015 IEEE 34th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, sep 2015.

[19] Paulo E. Veríssimo. Travelling through wormholes: A new look at distributed systems models. *SIGACT News*, 37(1):66–81, March 2006.

[20] Paulo Veríssimo and Luís Rodrigues. *Distributed Systems for System Architects*. Springer, 2001.

[21] Prasanth Vivekanandan, Gonzalo Garcia, Heechul Yun, and Shawn Keshmiri. A simplex architecture for intelligent and safe unmanned aerial vehicles. In *RTCSA*, 2016.