Facing the Safety-Security Gap in RTES: the Challenge of Timeliness

Marcus Völp, David Kozhaya and Paulo Esteves-Verissimo
Critical and Extreme Security and Dependability Group (CritiX)
Interdisciplinary Center for Security, Reliability and Trust (SnT)
University of Luxembourg
L-2721 Luxembourg
Email: <name>.<surname>@uni.lu

***Abstract*—Safety-critical real-time systems, including real-time cyber-physical and industrial control systems, need not be solely correct but also timely. Untimely (stale) results may have severe consequences that could render the control system's behaviour hazardous to the physical world. To ensure predictability and timeliness, developers follow a rigorous process, which essentially ensures real-time properties a priori, in all but the most unlikely combinations of circumstances. However, we have seen the complexity of both real-time applications, and the environments they run on, increase. If this is matched with the also increasing sophistication of attacks mounted to RTES systems, the case for ensuring both safety and security through aprioristic predictability loses traction, and presents an opportunity, which we take in this paper, for discussing current practices of critical real-time system design. To this end, with a slant on low-level task scheduling, we first investigate the challenges and opportunities for anticipating successful attacks on real-time systems. Then, we propose ways for adapting traditional fault- and intrusion-tolerant mechanisms to tolerate such hazards. We found that tasks which typically execute as analyzed under accidental faults, may exhibit fundamentally different behavior when compromised by malicious attacks, even with interference enforcement in place.**

I. Introduction

In the past, real-time systems were closed and, despite telemetry, largely disconnected. They executed simple, controller-like tasks that read sensor inputs, feed them into a model of the controlled plant, and produce appropriate actuator signals for maintaining safe and energy-efficient operation. Simplicity brought predictability and hence safety in all but the most rare circumstances. Safety violations, including late control decisions, are only tolerated if it can be shown that either their likelihood of occurrence is sufficiently low to practically never occur over the lifetime of the fleet or if their consequences are marginal. In particular, safety assurance criteria demand replication only if the above properties could not be achieved with singular systems, but not as precaution against attacks.

Unfortunately, the coverage of the assumed level of safety behind the above classical, accidental fault-prevention driven development process, degrades when one assumes malicious (and thus intentional) faults. Firstly, because these faults are no longer stochastic, and in consequence, a hazard is much less a residual probability (of some defect being activated), and much more a likelihood, once a defect known, and accessible to an attacker. Secondly, because both these two latter conditions have "improved" over the later years: vulnerability diagnostic tools have improved, and highly-skilled adversaries target these systems, as parts of critical information infrastructures.

It can reasonably be argued that such attacks were infeasible in the past, e.g., due to a simplicity-enforced lack of exploitable vulnerabilities, and/or to limited connectivity of real-time systems. However, the same cannot be said about current critical application scenarios, autonomous or cooperative driving, for instance, being a blatant example. In fact, we have recently elaborated on the threat surface of the cooperative-driving ecosystem [1], revealing what we call the safety-security gap:

Vulnerabilities rarely triggered through combinations of natural events may well cause serious harm when exploited by adversarial teams.

Having said this, two important factors single out critical real-time applications from general IT ones. First, both the attack and the necessary defense are dictated by the environment dynamics, making the slow and imprecise human-in-the-loop approach to current IT security, infeasible, or ineffective at best. Second, Cyber-physical Systems (CPS) may cause severe impact upon failure, both to humans or to resources. Rather than discharging threats with "unlikelihood" arguments, we believe it is time to meet them with paradigms that can come to exhibit a power and an effectiveness commensurate to the adversarial power we begin to witness.

It seems intuitive that the decreased coverage of the level of safety, which we have discussed earlier, could be regained, if systems, albeit in the presence of defects and other faults that may now be explored by attacks — with considerable reachability and likelihood of activation — could still achieve a similar level of failure avoidance as in the past, through *automatic means*. Fault tolerance as a general predicate seems to have been performing up to the task, in the scope of accidental faults. Now, we would need both fault and intrusion tolerance.

Fortunately, the fault and intrusion tolerance body of knowledge (commonly called BFT, for 'Byzantine Fault Tolerance')

This work is partially supported by the Fonds National de la Recherche Luxembourg (FNR) through PEARL grant FNR/P14/8149128.

has had a dramatic development, and already gives us a few preliminary solutions and insights to mitigate these threats in an automatic way, at least if the system at hand is not real-time. Replication and voting mask the actions of a minority of compromised replicas behind a majority of healthy replicas, reaching consensus [2]. Reactive rejuvenation of known or suspected compromised replicas and occasional proactive rejuvenation counteract exhausting the set of healthy replicas and defy stealthy adversaries and detection flaws [3]. Rejuvenation is of particular importance if adversaries persist in their attack with the goal to eventually exceed the tolerance threshold of up to f compromised replicas. Last but not least, diversification ensures that adversaries cannot benefit from knowledge gained during previous attack runs [4].

However, most of this research is concerned with asynchronous or partially synchronous systems, and further research is required on the extension of the paradigm to encompass real-time behavior. Namely, because the impact of the behavior of compromised components on system timeliness, is not well understood. We give a contribution in this position paper, at the specific level of task and component scheduling. We argue that intrusion tolerance mechanisms, despite their proven guarantees for facing attacks, lose their effectiveness if applied without a good understanding of the interaction between system tasks and components. To this end, we first revisit the traditional real-time system development process to highlight additional complications that arise when a subset of tasks may have been compromised.

In Section IV , this paper sketches our vision of a real-time BFT architecture featuring replication of "critical" components or sub-systems as the key to face faults and compromises through intrusions. More importantly, it shows in Section III, that traditional simple replication mechanisms may fall short of achieving their mission, in real-time systems. The reason lies in unanticipated interference: due to lower level system operations, tasks have access to different parts of common resources, which in case of malicious behaviour allows attackers to sabotage the whole resource. As such, replication, under contemporary interference analysis, may not yield the desired fault-/intrusion-tolerance. This paper concretely investigates such interferences, focusing on memory and cache interferences in multi-core systems, and highlights pitfalls of intrusion-agnostic analysis of task behavior deviation. We identify the challenges when facing the timeliness threats of compromised tasks, and sketch intrusion tolerance solutions for partially interfering-controllable resources. To our surprise, anticipating replication not only for fault but also for intrusion tolerance, though it complicates system analysis, it also bears opportunities to actually simplify the resulting scheduling problem, possibly even leading to more optimistic response times.

II. The Safety-Security Gap of the Cooperative-Vehicle Ecosystems

Lima et al. [1] identified threats to the cooperative-vehicle ecosystem, concluding that autonomous driving without cooperation is doomed to fail in the interim phase where both fully autonomous and human-controlled vehicles share the road.

For one, while driving, humans base their decisions on a variety of implicit protocols, interpreting driving styles, eye contacts, subtle movements and similar indicators as signs to evade an opposing car or to break aggressively (e.g., because the driver observed a scared look in a mother's face when chasing her child to prevent him from running onto the street). Cooperation can partially close this communication gap through explicit communication, until research incorporates these implicit protocols.

The second aspect, despite the need for cooperation, lies in the increased threat surface of autonomous and to a larger extent also cooperative vehicle ecosystems. Vehicles must defend against attacks on global V2I, I2I communication infrastructures [5], [6], against V2V attacks, but also against the classical in-vehicle communication networks such as CAN [7], [8] and Flexray [9]. Already today, diagnostics and infotainment access expose these safety-critical networks, with an often non-redundant gateway being the last line of defense against remote attacks.

Vulnerabilities in the software of this gateway, but also in other components such as the complex scenery detection tasks required for autonomous driving, put safe operation at risk. Similarly vulnerable, but more exposed are road-side units and cloud-based authentication mechanisms which are required in cooperative scenarios to distinguish authentic from fake events.

In addition to the above cyber attacks, autonomous cars (but likewise CPS and IoT systems) are also exposed to attacks against their plant and environment sensing capabilities [10], [11], a matter which although accidental already took their first life toll [12]. Sensor fusion and cross-validation amongst vehicles may be one solution to mitigate this threat. However, mitigation strategies of this kind heavily rely on reliable V2V and V2I communication, which is easily blocked through jamming in current substrates if cyber attacks are accompanied by physical attacks. In fact, Serageldin et al. [13] show that jamming becomes over-proportionally effective at higher DSRC bandwidths, leaving only low bandwidth solutions tolerant to such attacks.

Clearly, analyses fall short of correctly valuing safety threats if they anticipate only accidental faults and their likelihood, but not coordinated attacks to cause these faults. In particular, natural occurrences of combinations of independent faults are extremely rare and as such often overlooked or misinterpreted in terms of risk. However, adversaries in control of the system may easily trigger such combinations and thereby exploit the safety-security gap in security-agnostic analyses, a conclusion which is also shared by Hamad et al. [14] in their attack-tree based security analysis of automated obstacle avoidance.

III. Timeliness Threats

Clearly, the fundamental prerequisites for meeting the above challenges include (1) limiting the interference that compromised tasks can have on other tasks, and (2) developing mechanisms for enforcing these limits in a trustworthy manner.

Otherwise, any compromised task would be able to exceed the bounds to jeopardize the timeliness of its critical counterparts.

A. Memory Isolation and Cooperative-Scheduling

An intuitive consequence of the need to limit interference suggests isolation as a key factor for reducing the attack surfaces inside real-time systems. This implies reducing the ability of an adversary to compromise further components once it has successfully compromised one.

Clearly, in systems without sufficient memory protection, adversarial control may spread from one task to others until critical system components become compromised.

Common practice of embedded real-time systems today is to execute code directly from flash images, so one might argue against code-level compromises. However, examples like return-oriented programming [15] and similar techniques have demonstrated how programs can be compromised without altering their code. Also, over-the-air update capabilities demand for mechanisms to replace pre-installed code. Not to mention that higher-level tasks of autonomous driving, such as scenery detection and trajectory planning, exceed today's on-die flash capacity and require instruction caches or scratch-pad memory (i.e., modifiable storage) to keep up with their performance requirements.

Notice that it is purposeful to enforce strong isolation between tasks, even if one task produces a result, which is an essential input to the other. Strongly isolating these dependent tasks slows down adversaries, who are forced in that case to attack by either breaking the isolation or through the communication interfaces between tasks. Moreover, when we later introduce replication, dependent tasks in a chain may be replicated separately rather than the whole chain. This way, each dependent segment benefits from a majority of healthy replicas in the previous segment.

Now, for the same reasons that we have to disqualify real-time systems with lack of strong isolation, we must also disqualify cooperative scheduling and schedulers without time-slice enforcement. In these systems, tasks are expected to voluntarily relinquish control over allocated resources. However, compromised tasks, deviating from their analyzed behavior, may never relinquish such resources, thereby falsifying analyzed resource bounds.

Fortunately, memory isolation and enforced schedules are already state-of-the-art in many (though evidently, until recently, not all [16]) automotive systems. In particular the lower control levels run on physically isolated microcontrollers or on well isolating RTOSs. However, the same care must be exercised for the complex autonomous driving counterparts, in particular due to the imminent threat of legacy OS compromises.

B. Resource Bound Analysis

So far, car manufacturers refrained from using modern, superscalar, multicore processors with their innumerous latency hiding mechanisms. However, the performance demands of higher autonomy levels [18] and the expected data rates that need to be processed for cooperative driving may change the picture. Let us therefore investigate more closely what impact a deviation of a compromised task from its analyzed behavior can have in modern multicore architectures that are not specifically equipped with QoS mechanisms to ensure minimal guarantees for critical tasks (e.g., by employing AMBA 5's bandwidth-control mechanisms [19]).

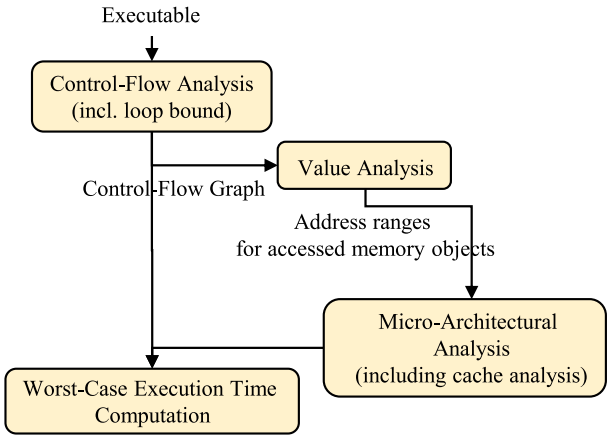

Fig. 1. Path and cache analysis for WCET estimation (adopted from [17] Fig.2).

Traditionally, resource bound analysis is concerned with finding the worst-case combination of task execution paths that lead to the worst-case overall execution time and hence the maximal interference tasks may have on each other. It is therefore tempting to subject task groups to such an analysis and run them with the obtained interference bounds.

When interference bounds can be controlled completely, following this approach preserves safety even under attack. However, not all resources, through which tasks may interfere with others, can be controlled to the required degree. For these resources, WCET analysis must not only compute the worst-case combination of execution paths of analyzed tasks, but also combinations where a subset of tasks execute in the worst possible pattern.

Compromised tasks may exhibit arbitrary execution patterns over the resources they can get hold of. That is, given a statically allocated set of resources R_i, a compromised task τ_i may access the resources in R_i in a pattern that maximizes interference on other tasks. In systems with dynamic resource allocation, as they will be required for more demanding applications, compromised tasks τ_i may request further resources, expanding R_i to the set of acquirable resources R_i^{max}, and then construct a worst-case interference pattern. In this situation, resource bound analysis has to anticipate arbitrary execution over the extended set R_i^{max}.

1) Caches: Let us exemplify the consequences of this observation with the example of resource bound analyses for processor data caches [17].

Caches are near core memories split into multiple pairs of tag and data RAM (called ways) and equipped with a logic to transparently resolve cache hitting and missing memory

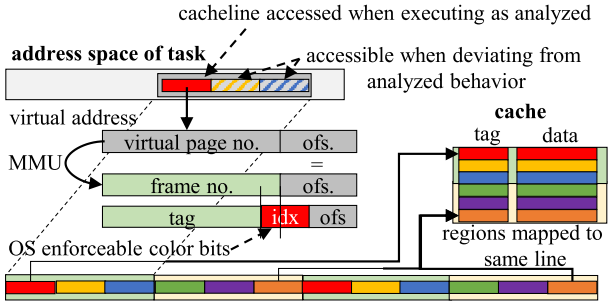

Fig. 2. OS controllable cache colors. Although only cachline-size regions of the same color collide in the cache, OS interference control is limited to those index bits that range into the frame number. Interference by compromized applications can therefore be much higher than when behaving as analyzed.

accesses. Extracting from the accessed address the lower most bits after the cacheline offset (i.e., the *index*), the cache logic determines the row in all ways and compares the tag RAM against the remaining higher order bits (the *tag*) to determine hits (exactly one match) or misses (no match). Since replacement is only among cachelines of the same row, it is possible to color memory locations by the rows in which they will be inserted in the cache. The bottom part of Figure 2 illustrates this coloring and the split of the physical address into tag, index and offset. Given cl large cachelines and an associativity (no. of ways) a, a cache of size $s = n \cdot a \cdot cl$ has n colors.

Processors come with multiple levels of caches, some separately caching code and data, others both at the same time. Lower levels (e.g. L1) are typically private (i.e., exclusively used by a single core) while later levels (e.g. L2) act as victim caches to keep evicted cachelines from L1 or as possibly inclusive shared caches for all cores on the same die (e.g., L3). Inclusive means data cached in lower level caches is also cached in L3 and if evicted in the latter, must also leave the lower levels. Write-through caches update lower cache levels and RAM immediately, write-back defers these updates and, in case of L2 victim caches also the cacheline allocation, to the time of eviction.

Figure 1 shows the building blocks of a worst-case execution time (WCET) pipeline. Starting from the executable binary, the control flow graph and loop bounds are extracted, which are then fed into a further value analysis for determining address ranges for all accessed variables and other memory objects. With these ranges, a micro-architectural analysis is invoked, which includes a cache analysis. The cache analysis itself proceeds by abstractly tracking the locations that may and must hit in the cache along the replacement policy (e.g., sorted by age in case of least-recently-used (LRU) replacement) and by merging the abstract states at control-flow join points.

Figure 3 shows this merging for a 4-way set associative LRU cache after executing the following code from empty caches:

```
d = 0;
if (a > 0) { b = 1; c = 2; }
else { d = 1; e = 2; }
```

Assuming all variables are in cachelines of the same color, d and a must hit in the cache with age 4 and 3, respectively (maximum age to conservatively bound cache hits), while c, e may hit with age 1, d, b with age 2 and a with age 3 (minimum age to conservatively bound cache misses, potential write backs, and cross core evictions due to L3 cache inclusiveness).

2) Cache analysis under attack: From the above observation, it is tempting to extract the interference pattern of a task from the addresses it accesses and to compute from this cache related preemption delays (see e.g., [20]) and similar interference bounds. Assume for example two tasks τ_1 and τ_2 whereby the scheduler executes both tasks on the same core while allowing τ_2 to preempt each job of τ_1 once. If τ_2 accesses at most two cachelines of a particular color, the worst case interference that may happen to τ_1's memory accesses of this color are two evictions plus the write-back of two possibly dirty lines (see right part of Fig. 3).

The same scenario when executing τ_2 on a different core of the same die and with an inclusive last-level cache effectively reduces to two the ways available to τ_1 for this color. This is because τ_2 may repeatedly access the two cachelines to evict the memory cached by τ_1. Of course, further analysis of τ_2's access pattern allows exploiting more fine grain interleavings, e.g., allowing τ_1 a number of subsequent accesses in between any two of τ_2's accesses.

Unfortunately, the operating system (OS) can enforce cache colors only at the granularity of the smallest page size [21] when allocating page frames for an application and only at the cost of having to support paging, a feature necessarily required by more resource demanding applications, but, due to predictability concerns, rarely supported in real-time operating systems (RTOSs) [22]. This lack of control stems from paging, which allows the OS to define only those index bits that are part of the page number (i.e., above the page offset). Assume $ps = k \cdot cl$ holds for the size of the smallest page ps. Then, because k cachelines fit this page, the number of enforceable colors is reduced to n/k (e.g., yellow and green in Fig. 2).

Rephrasing the above statement slightly differently, a task τ_i analyzed to access memory in the sequence of colors S_i^c may exhibit arbitrary sequences S_i' of colors c when compromised, where c agrees with a color in S_i^c in the index bits above the page offset. In particular, compromised τ_i may access memory that it did not access when executing as analyzed.

Although not yet quantified in adversarial settings, the comprehensive benchmarks in [23]–[25] give a first indication on the impact that compromised tasks can have when executing outside analyzed behavior, in particular when interfering through shared implicit last-level caches. In addition, these works suggest hardware and software-level solutions to partially mitigate cross-core interference though shared caches. For example, Kim et al. [24] introduces hardware way- and set-partitioning mechanisms in MC^2 for last-level caches (LLCs), Kenna et al. [23] discuss page coloring for LLCs and Mancuso et al. [25] introduce colored lockdown.

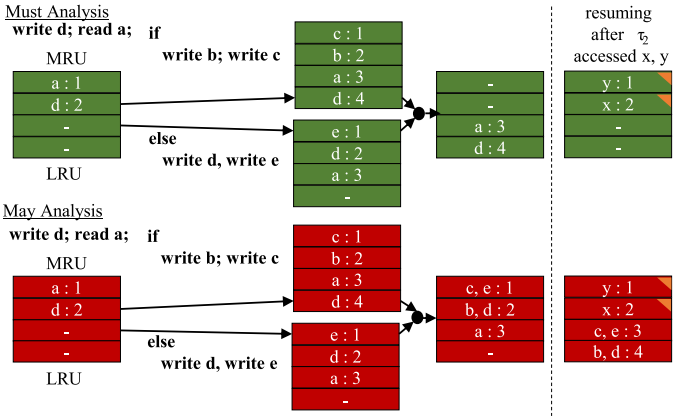

Fig. 3. Merging of abstract cache states at the join point after the conditional branch.

Krüger et al. [26] work on counteracting these threats through schedule randomization by probabilistically increasing the distance between attacking and attacked tasks.

3) Memory, Busses and Pipelines: Coloring not only applies to caches, but extends also down to memory banks for limiting DRAM refresh interferrence. However, like caches, color-based control over DRAM banks only overapproximates the banks that compromised tasks may access, which lets us expect a similar discrepancy between analyzed and compromised behavior.

Yun et al. [27] introduce a performance-counter based framework to enforce memory bus bandwidths. However, again, counters only have throttling capabilities, hence they may not change access patterns at finer granularity. In particular buses like the CPU / GPU interconnects, which allow bursts, may therefore exhibit large discrepancies between analyzed and compromised behavior.

Last but not least, most of the above arguments implicitly assumed a timing-anomaly free pipeline to enable the above analyses in the first place. Modern out-of-order, speculative and superscalar pipelines, as required for more demanding tasks such as scenery analysis, defy to a large extent predictability and can, through careful exploits, be turned into time consuming monsters.

IV. Towards an Intrusion Tolerant Architecture

Observing the possibly devastating effect compromised tasks can have on timeliness, leave alone correctness, we sketch in this section possible architectures to tolerate intrusions. Our focus is thereby on correctness matters, leaving timeliness opportunities from replication for the next section.

Figure 4 shows a birds eye view on our envisaged fault and intrusion tolerant real-time architecture. Exemplified are two complex tasks τ_1 and τ_2 controlling the plant, which may be the controlled physical system or another system of the same structure with lower-level control tasks. For example, for drones [28], a common architectural pattern is to couple the rotors and elevators with a flight stabilizing controller which

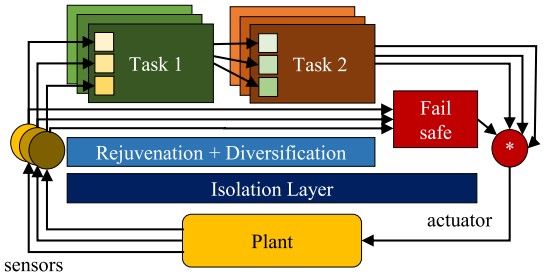

Fig. 4. Intrusion Tolerant Architecture for Complex Safety-Critical Real-Time Systems

in turn receives signals from a more powerful, decoupled system running more complex tasks such as flight planning and autonomous landing.

A. Isolation

In Section III-A, we have already seen the essential need for memory isolation to slow down adversaries. Candidates for this isolation layer are real-time microkernels [29]–[31] but but also hardware solutions are imaginable where replicas run on dedicated soft cores or on hard cores in ASICs. The remaining ingredients, which we discuss in the following, are voting, sensor fusion, fall-back to simplex actuator control and rejuvenation and diversification of replicas.

B. Replication and Voting

Tolerating complexity-induced vulnerabilities demands replicating τ_1 and τ_2 such that up to f of their replicas can be compromised. The remaining replicas should continue to reach consensus on the values that replicas read from the replicated sensors, on the inputs they receive from τ_1 and on the outputs τ_2 forwards to the plant actuators. Unfortunately, classical BFT consensus protocols, such as PBFT [2] or derived hybrid protocols [32], [33], operate on an asynchronous (i.e., time agnostic) system model. Applying them in a synchronous system setting is not trivial, despite bounded message transmission times and bounded execution times, as simply summing up the bounds through all protocol steps easily leads to intolerable worst case execution times.

For example, PBFT, MinBFT and CheapBFT achieve consensus by a leader proposing the next client request to vote on. However, in the presence of a faulty leader, this causes downtimes until the remaining replicas agree on a new leader. Leaderless protocols avoid this complication, however, they generally require more replicas. For example, PBFT requires $n = 3f + 1$ replicas to tolerate up to f faults. Hybrid protocols reduce this number to $2f+1$ respectively $f+1$ active plus f passive replicas. Leaderless protocols require $5f + 1$ replicas (or introduce further complexities [34] to maintain $n = 3f + 1$). An exception to this rule is BFT-TO by Correia et al. [35]. building on top of a trusted ordering wormhole, leaderless BFT-TO requires only $2f + 1$ replicas. If leader-based protocols are used in real-time systems, leader change must be bounded and anticipated in the schedule.

Fortunately, most real-time tasks are triggered by the physical system triggering events such as alarms or timers. Reliable invocation of replicas may therefore avoid voting in different orders. However still, faulty replicas (including sensors) may lie about their values and in particular they can lie differently to different client replicas. These inconsistent faults (also called 'equivocation') are prototypical of the Byzantine fault phenomenon, and are avoided by either extended number of players and rounds of communication, or through cryptographic means such as signing messages for authenticity and unforgeability. Hybrid protocols make use of trustworthy sequencers (e.g., monotonic counters) which apply cryptographic means to ensure that only a single vote can be given for a single instance. However, naturally, cryptograpy means come at non-negligible costs, which are not tolerable in low latency real-time systems.

To avoid these costs, we instead propose to exploit the tight coupling between components and to introduce hybrid components that capture sensor and task values in a manner that prevents overwriting during the same instance. For sensors, capture/compare units suggest themselves as they also capture the timing of the sampled event, thereby preventing mixing of too time distant reads, provided of course the timing source is trustworthy. OS controlled FIFO buffers and synchronous IPC [30] achieve the same for intra task communication.

C. Sensor Fusion

Marzullo [36] shows that $3f+1$ interval-type sensor values are required to agree on an interval that contains the true value, Schmid and Schossmaier [37] refine this result to Lipschitz continuous intervals (i.e., small changes in the proposed intervals lead only to small changes in the agreed upon result), and Rushby proved correct both results in the theorem prover PVS [38]. The continuous nature of the controlled plant allows intra-task communication of largely diverse replicas to operate in a similar manner, provided tasks can project the intermediate results onto the control points of the other. The consequences of course are that voting is no longer on identical values but on semantically equivalent, yet possibly different control commands.

D. Simplex and Actuators

Even though replication and voting reduce dependability from a single instance to a majority of assumed healthy replicas operating in consensus, a residual risk of common mode failures remains, in particular if replicas exceed a certain size and complexity. Common mode failures are caused by systematic vulnerabilities in all replicas. They allow adversaries to exceed the tolerance threshold f, which BFT protocols require to maintain safe operation. It is therefore also crucial to explore simplistic fail-safes that reinstantiate safety in the rare situations when consensus-reaching values are wrong.

Bak et al. [39] and Veríssimo et al. [40] propose hybrid architectures wherein a simple controller monitors and returns the system to a stable state if more complex controllers fail to provide correct and timely results.

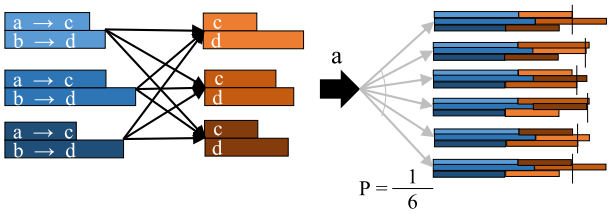

Fig. 5. Stochastic independent scheduling of dependent replicated tasks. Shown are two tasks (blue - left) and (orange - right) with clearly dependent execution times: an input of b leads to longer execution times in all replicas. However, they complete for input a in one of the patterns shown on the right with probability 1/6. In case all replicas respond correctly, the vertical bars indicate majority completion time.

E. Rejuvenation and Diversification

Diversification and rejuvenation are essential ingredients for maintaining a majority of healthy replicas and for reducing the threat of common mode failures. Diversification prevents adversaries from accumulating knowledge how to attack the system, as long as all replicas are rejuvenated periodically and faster than adversaries can compromise more than f replicas [3]. Rejuvenated replicas are down and must be compensated by additional replicas. The combination of the above requirements means diversification must be automatic (e.g., through obfuscating compilation [41]) and the real-time system needs access to a continuous stream of such diverse replicas. That is, they must either receive a continuous flow of updates from infrastructure components or must perform the compliation online in the same system. The flow/compilation task is thereby weakly real-time with an attacker speed determined periodicy, which means n new updates must be produced within this period or the fail safe must kick in (and possibly disrupt system continuity).

F. Actuators

Las but not least, when it comes to the actuation of the physical plant, agreement must be reached which control signal is applied to the plant. In the absence of consensus, this means actuators must fall back to the fail-safe while discarding the diverging replica decisions. Replication agnostic actuators must be driven by a trusted-trustworthy component condensing the multitude of proposed values.

V. Timeliness Challenges and Opportunities

The correctness challenges of the previous section also imply timeliness challenges. For example, all local downtimes and recovery mechanisms must be bounded and scheduled to guarantee (weak) timeliness of all tasks. In particular, WCET bound preserving or at least WCET bound providing compilation of diverse replicas is a challenging task that has yet to be solved. Many other questions remain open, as pointed by our reviewers: How to prove classical replication techniques to not cause interference themselves? How can sensor fusion assist in detecting interference in modern multicore architectures? How can that interference be incorporated in the system so that it can be tolerated? And how can rejuvenation and stochastic

scheduling be deployed safely without becoming a major source of interference? However, replication also brings some opportunities to simplify the WCET analysis and scheduling problem.

For example, in a multicore system, the same non-replicated schedule can be reused when critical components are replicated. Rather than allocating replicas to fixed cores (e.g., replica r_i to core i) it may be more beneficial to randomize this allocation in order to exploit the stochastic independence of replicas of the same task, even if tasks are dependent. Consider the example in Figure 5 with two replicated tasks τ_1 and τ_2, which are I/O dependent as illustrated in Figure 4. Even though the output of τ_1 influences the execution performed in τ_2, different replicas have different execution times for producing this output (in the replicas τ_1^i of τ_1) and for consuming this output (in the replicas τ_2^j of τ_2). Randomizing the allocation of τ_2^j relative to τ_1^i leads to a distribution of combined execution times for this input/output pair of which, as it is common for replicated settings, only a correct majority of $2f + 1$ (respectively $f + 1$) replicas must reach agreement before the task's deadline D_i. The other replicas need only to complete by T_i (i.e., $T_i - D_i$ time units later for deadline constrained tasks) or not at all if the task itself is stateless.

VI. Acknowledgments

The authors like to thank the anonymous reviewers for their insightful comments and suggestions to improve the paper, raising in part additional challenges, which we took the freedom to include in the above description.

VII. Conclusions

Revisiting the threat surface of safety-critical real-time systems, we have identified several challenges but also opportunities, for applying fault and intrusion tolerance techniques, which not only mitigate accidental faults but also protect the system against targeted attacks, and do both in an automatic, unattended way. We analyzed the situation at the specific level of task and component scheduling, and studied problems arising when malicious fault and intrusion tolerance must take a timeliness and scheduling perspective into account. Major challenges and hence directions for future work include limited interference controls when tasks deviate from analyzed behavior, definition of safe fail stops to compensate common mode failures in complex replicated tasks, and analysis aware obfuscating compilation.

References

[1] A. Lima, F. Rocha, M. Völp, and P. Esteves-Verissimo, "Towards safe and secure autonomous and cooperative vehicle ecosystems," in *CPS-SPC*, ACM, Vienna, Austria, Oct. 2016.

[2] M. Castro and B. Liskov, "Practical byzantine fault tolerance," 1999, pp. 173–186.

[3] P. Sousa, A. N. Bessani, M. Correia, N. F. Neves, and P. Verissimo, "Highly available intrusion-tolerant services with proactive-reactive recovery," *IEEE Trans. on Parallel & Distributed Systems*, pp. 452–465, 2009.

[4] P. Larsen, A. Homescu, S. Brunthaler, and M. Franz, "Sok: Automated software diversity," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, ser. SP '14, Washington, DC, USA: IEEE Computer Society, 2014, pp. 276–291.

[5] J. R. Douceur, "The Sybil attack," in *Peer-to-peer Systems*, Springer, 2002, pp. 251–260.

[6] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Wormhole attacks in wireless networks," *IEEE Jour. on Selected Areas in Communications*, vol. 24, no. 2, pp. 370–380, 2006.

[7] K. Pazul, "Controller area network (CAN) basics," *Microchip Techn. Inc*, 1999.

[8] J. Staggs, "How to hack your mini cooper: Reverse engineering CAN messages on passenger automobiles," *Institute for Information Security*, 2013.

[9] F. Consortium *et al.*, "Flexray communications system-protocol specification," *Version*, vol. 2, no. 1, pp. 198–207, 2005.

[10] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[11] J. Petit, B. Stottelaar, M. Feiri, and F. Kargl, "Remote attacks on automated vehicles sensors: Experiments on camera and lidar," in *Black Hat Europe*, 2015.

[12] *Tesla's autopilot has had its first deadly crash*, https://www.wired.com/2016/06/teslas-autopilot-first-deadly-crash/, Accessed: 2016-07-05.

[13] A. Serageldin, H. Alturkostani, and A. Krings, "On the reliability of dsrc safety applications: A case of jamming," in *2013 International Conference on Connected Vehicles and Expo (ICCVE)*, Dec. 2013, pp. 501–506.

[14] M. Hamad, M. Nolte, and V. Prevelakis, "Towards comprehensive threat modeling for vehicles," in *1st Workshop on Security and Dependability of Critical Embedded Real-Time Systems*, M. Völp, P. Esteves-Verissimo, A. Casimiro, and R. Pellizzoni, Eds., co-located with the IEEE Real-Time Systems Symposium 2016, IEEE, Porto, Portugal, Dec. 2016, pp. 31–36.

[15] E. Buchanan, R. Roemer, H. Shacham, and S. Savage, "When good instructions go bad: Generalizing return-oriented programming to RISC," in *Proceedings of CCS 2008*, P. Syverson and S. Jha, Eds., ACM Press, Oct. 2008, pp. 27–38.

[16] C. of Oklahoma County, *Bookout vs toyota*, http://www.safetyresearch.net/Library/Bookout_v_Toyota_Barr_REDACTED.pdf, Accessed: 2016-07-22.

[17] M. Lv, N. Guan, J. Reinecke, R. Wilhelm, and W. Yi, "A survey on static cache analysis for real-time systems," *Leibnitz Transactions on Embedded Systems*, vol. 3, no. 1, pp. 1–48, Jun. 2016.

[18] C. Hayes, *Driving along at full speed for autonomous vehicles*, Jan. 2016.

[19] ARM, *AMBA5 ahb protocol specification*, Oct. 2015.

[20] Z. Zhang and X. Koutsoukos, "Cache-related preemption delay analysis for multi-level inclusive caches," in *2016 International Conference on Embedded Software (EMSOFT)*, Oct. 2016, pp. 1–10.

[21] J. Liedtke, H. Hartig, and M. Hohmuth, "Os-controlled cache predictability for real-time systems," in *Proceedings Third IEEE Real-Time Technology and Applications Symposium*, Jun. 1997, pp. 213–224.

[22] *Free RTOS*.

[23] R. Mancuso, R. Dudko, E. Betti, M. Cesati, M. Caccamo, and R. Pellizzoni, "Real-time cache management framework for multi-core architectures," in *19th IEEE International Conference on Real-Time and Embedded Technology and Applications Symposium (RTAS 2013)*, Philadelphia, PA, USA.

[24] B. Ward, J. Herman, C. Kenna, and J. Anderson, "Making shared caches more predictable on multicore platforms," in *25th Euromicro Conference on Real-Time Systems*, Jul. 2013, pp. 157–167.

[25] N. Kim, B. Ward, M. Chisholm, C.-Y. Fu, J. Anderson, and F. Smith, "Attacking the one-out-of-m multicore problem by combining hardware management with mixed-criticality provisioning," *Real-Time Systems, special issue of outstanding papers from the 22nd IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2016)*, vol. 53, no. 5, pp. 709–759, Sep. 2017.

[26] K. Krüger, M. Völp, and G. Fohler, "Improving security for time-triggered real-time systems against timing inference based attacks by schedule obfuscation," in *Euromicro Conference on Real-Time Systems (ECRTS) - Work-in-progress Session*, 2017.

[27] H. Yun, R. Mancuso, Z.-P. Wu, and R. Pellizzoni, "PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms," in *Intl. Conference on Real-Time and Embedded Technology and Applications Symposium (RTAS)*, IEEE, 2014.

[28] P. Vivekanandan, G. Garcia, H. Yun, and S. Keshmiri, "A simplex architecture for intelligent and safe unmanned aerial vehicles," in *International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, IEEE, 2016.

[29] D. Hildebrand, "An architectural overview of qnx," in *Proceedings of the Workshop on Micro-kernels and Other Kernel Architectures*, 1992, pp. 113–126.

[30] J. Liedtke, "On micro-kernel construction," in *Proceedings of the 15th ACM Symposium on Operating System Principles*, 1995, pp. 237–250.

[31] (). Fiasco, [Online]. Available: https://os.inf.tu-dresden.de/fiasco/.

[32] G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung, and P. Veríssimo, "Efficient byzantine fault-tolerance," *IEEE Trans. Computers*, vol. 62, no. 1, pp. 16–30, 2013.

[33] T. Distler, C. Cachin, and R. Kapitza, "Resource-efficient byzantine fault tolerance," *IEEE Trans. Computers*, vol. 65, no. 9, pp. 2807–2819, 2016.

[34] F. Borran and A. Schiper, "A leader-free byzantine consensus algorithm," in *Int. Conf. on Distributed Computing and Networking (ICDCN)*, 2010.

[35] M. Correia, N. F. Neves, and P. Verissimo, "Bft-to: Intrusion tolerance with less replicas," *The Computer Journal*, vol. 56, no. 6, pp. 693–715, Jun. 2013.

[36] K. Marzullo, "Tolerating failures of continuous-valued sensors," *ACM Transactions on Computer Systems*, vol. 8, no. 4, pp. 284–304, Nov. 1990.

[37] U. Schmid and K. Schossmaier, "How to reconcile fault-tolerant interval intersection with the lipschitz condition," *Dirstibuted Computing*, vol. 14, no. 2, pp. 101–111, May 2001.

[38] J. Rushby, "Formal verification of marzullo's sensor fusion interval," SRI International, Tech. Rep., Jan. 2002.

[39] S. Bak, D. Chivukula, O. Adekunle, M. Sun, M. Caccamo, and L. Sha, "The system-level simplex architecture for improved real-time embedded system safety," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, IEEE, 2009, pp. 99–107.

[40] P. Verissimo and A. Casimiro, "The timely computing base model and architecture," *IEEE Transactions on Computers*, vol. 51, no. 8, pp. 916–930, Aug. 2002.

[41] R. Pucella and F. B. Schneider, "Independence from obfuscation: A semantic framework for diversity.," in *19th IEEE Work. on Computer Security Foundations*, 2006, pp. 230–241.