SCRaM – State-Consistent Replication Management for Networked Control Systems

Ben W. Carabelli, Frank Dürr, Kurt Rothermel
Institute of Parallel and Distributed Systems, University of Stuttgart, Germany
{ben.carabelli | frank.duerr | kurt.rothermel}@ipvs.uni-stuttgart.de

***Abstract*—Networked control systems (NCS) consist of sensors and actuators that are connected to a controller through a packet-switched network in a feedback loop to control physical systems in diverse application areas such as industry, automotive, or power infrastructure. The control of critical real-time systems places strong requirements on the latency and reliability of both the communication network and the controller. In this paper, we consider the problem of increasing the reliability of an NCS subject to crash failures and message loss by replicating the controller component. Previous replication schemes for real-time systems have focused on ensuring that no conflicting values are sent to the actuators by different replicas. Since this property, which we call *output consistency*, only refers to the values within one time step, it is insufficient for reasoning about the formal conditions under which a group of replicated controllers behaves equivalent to a non-replicated controller. Therefore, we propose the stronger *state consistency* property, which ensures that the sequence of values produced by the replicated controller exhibits the same dynamical behaviour as a non-replicated controller. Moreover, we present SCRaM, a protocol for replicating generic periodically sampled controllers that satisfies both of these consistency requirements. To demonstrate the effectiveness of our approach, we evaluated it experimentally for the control of a cart-driven inverted pendulum.**

I. Introduction

Networked control systems (NCS) [1], [2] are an important class of cyber-physical systems (CPS) in manifold application areas such as the industrial Internet of Things (Industry 4.0), smart grids, telesurgery, or assisted and autonomous driving. They consist of sensors, actuators, and digital controllers that are connected over a packet-switched network, forming a feedback control loop to stabilize a physical process.

Considering their applications, it is obvious that many NCS are safety-critical. Component malfunctions might lead to severe monetary loss, environmental damage, or even human injury. Sensors, actuators, and controllers run on nodes that may fail, and communicate over faulty communication channels, which may lose messages or deliver them too late. Loss and delay of sensor outputs or actuator inputs can affect the performance and stability of the NCS.

Thus, reliability is one of the most important requirements of NCS. Clearly, the reliability of an NCS depends on all of its components, i.e., sensors, actuators, controller, and network. The fault-tolerance of a control loop with regard to sensor, actuator, and network failures can be addressed at the application layer, by employing a control design that is inherently robust to a certain degree of sensor and actuator message loss [1], [3]. Such design approaches typically yield some (probabilistic) bounds for the necessary availability of the involved components that ensures stability or required performance.

The availability of sensors can be increased through fusion of redundant sensors [4], [5]. Similarly, the availability of actuators can be increased by suitable redundant designs, e.g., [6]. In this paper, we focus on the availability of the *controller* function. For communication, we explicitly do *not* assume a perfectly reliable network since this is an unrealistic assumption [7].

In order to avoid a single point of failure, the controller has to be replicated over different nodes. In each control step, the controller replicas receive the sensor input and apply the control function to generate the controller output, which is then transmitted to the actuators. Obviously, an important requirement for a replication scheme is to ensure *output consistency* [8], meaning that all controller replicas send the same sequence of output messages. In this paper, we use a relaxed output consistency condition like in [9], where we allow replicas to occasionally produce no output. This is motivated by the fact that intermittent message loss in NCS—also from controller to actuator—is well-studied [10]–[12].

In general, output consistency could also be achieved by following the State Machine Replication (SMR) approach for implementing the controller function, where each control step corresponds to calling an operation that takes the sensor values as input and generates the controller output. In this case, a standard SMR protocol such as Viewstamped Replication [13] or RAFT [14] could be used to achieve consensus about the controller outputs. Unfortunately, as pointed out in [9], SMR is unsuitable for controller replication due to the time-constraints of NCS. Since each control step has to be finished within a certain time bound, the replicas only have limited time to achieve consensus on each output. If consensus cannot be achieved in time, no output can be sent. While a fault-tolerant controller can accept missing outputs in some control steps, the problem with SMR is that a replica cannot start processing a new control step before having completed the previous one. Therefore, a delay violation of consensus processing in one control step affects all the following steps.

In this paper, we investigate how the problems of SMR subject to real-time constraints can be avoided when consensus algorithms are used for controller replication. To this end, we first discuss the necessary consistency concepts for replicated control systems. While output consistency addresses the *spatial* correctness concerning the set of messages delivered to

Accepted for publication in *Proc. 11th IEEE/ACM International Conference on Cyber-Physical Systems (ICCPS)*, April 2020.

the actuators in one time step, it does not relate to any notion of *temporal* correctness regarding the *sequence* of controller outputs. However, any stateful control law specifies such temporal behaviour. Since the controller's dynamics is part of the closed-loop model, which is the basis for stability and performance analysis, it should be reproduced faithfully by the replicated controller. In other words, output consistency alone is not sufficient to ensure *replication transparency* for NCS.

Therefore, we argue that it should be complemented by an additional condition, which we call *state consistency*, that requires any new output to be based on the controller state that generated the most recent previous output. State consistency ensures that the replicated controller produces a sequence of outputs that appears to have been generated by a single controller that always executes uninterruptedly, and that any intermittent unavailability of the controller is indistinguishable from an omission of output messages. Consequently, the properties that have been proven for a non-replicated controller also hold for a replicated version of the same controller, with no need for modifying the controller design.[1] While SMR offers equivalent properties in the failure-free case, state consistency provides a well-defined correctness condition also for *incomplete* controller output sequences, which facilitates the temporal decoupling of consensus processing for consecutive control steps to avoid the problems of SMR.

We propose SCRaM, a State-Consistent Replication Management protocol for NCS controllers. For each iteration of the replicated control function, a single-value consensus algorithm based on a standard protocol [15], [16] is executed to ensure output consistency. However, SCRaM ensures that the replicated controller is not blocked unnecessarily waiting for consensus instances to terminate, and that the state of controller replicas producing new output is based on the state of replicas that have produced output previously, thus ensuring state consistency.

In summary, we make the following contributions:

- A formal definition of state consistency for replicated controllers, providing sufficient guarantees to make a replicated controller functionally indistinguishable from a non-replicated controller.
- SCRaM, a replication protocol for NCS ensuring state and output consistency with high availability.
- An experimental comparison of SCRaM with the Quarts protocol [9], which is—to the best of our knowledge—the most effective replication protocol for NCS to date, demonstrating the benefits of state consistency.

The rest of this paper is organized as follows. In Sec. II we discuss related work. In Sec. III we introduce a generic model for time-triggered control systems. In Sec. IV we extend this model for replicated controllers and derive our consistency conditions. In Sec. V we derive requirements for a replication protocol to satisfy these consistency conditions and present the SCRaM protocol. In Sec. VI we show our evaluation results. Sec. VII concludes the paper with a short summary and outlook.

[1]...assuming that the non-replicated controller is designed to tolerate output omissions to a degree that is equivalent to that achieved by the replicated controller for a given network.

II. Related Work

It is well known that in *synchronous systems* with bounded network and processing delay, consensus can be reached in bounded time, even for stronger failure models than we assume in this paper, such as Byzantine failures [17]. Therefore, some replicated systems rely on real-time communication networks such as Time-Triggered Ethernet [18] or Token Ring [8], adding sufficient redundancy in the network to practically avoid delay failures, which would otherwise violate the fundamental assumption of a synchronous system. However, real-time networks with deterministically bounded delay are mostly restricted to local or possibly metropolitan area. Therefore, we cannot assume a synchronous system to be available for CPS distributed over a larger geographic area, where one has to communicate over unreliable Internet connections.

Considering that in asynchronous systems one cannot guarantee both, agreement and termination within bounded delay, Saab et al. recently proposed the Quarts protocol [9]. It guarantees that in each time step, replicated controllers agree on a single output vector sent to actuators (output consistency) through plurality voting. The delay of attempting agreement is bounded such that non-agreement in one step cannot indefinitely block the next steps. The authors show that Quarts is superior to the classic asynchronous consensus protocol FastPaxos in terms of availability and latency. However, Quarts does not address the issue that the agreed actuator values of one step should also be consistent with the agreed values of another round. In contrast, in addition to output consistency, we enforce state consistency to ensure that the state of a replica producing the agreed output values of the current round is based on the state of the replica that has produced the previously agreed values. This makes the functional behavior of the replicated controller indistinguishable from a non-replicated controller. This fundamentally simplifies the design and analysis of the replicated controller based on a functionally equivalent non-replicated controller.

III. Control System Model

An NCS consists of a *plant*, which denotes the physical system to be controlled, a set of *sensors* that take measurements of the plant, a set of *actuators* that apply actions to the plant, and a *controller* that calculates actuator values (denoted as its *output vector OV*) based on the sensor values (denoted as its *input vector IV*) and an internal controller state. This basic setup is shown in Figure 1. The controller communicates with the sensor and actuator components over a packet-switched network. Therefore, the inputs and outputs of the controller are discrete-time sequences of sensor and actuator values.

Accordingly, we assume that the plant is described by some dynamical system model that is sampled periodically at times t_k, which are determined by a sampling interval $T_s = t_{k+1} - t_k$. We denote the measurements from all sensors at sampling time t_k by y_k, and the actuator values by u_k.

The controller maintains its own internal state S^c_k. As described above, the controller periodically updates its state depending on the values received by the sensors, and generates a corresponding vector of actuator values. Because the sensor

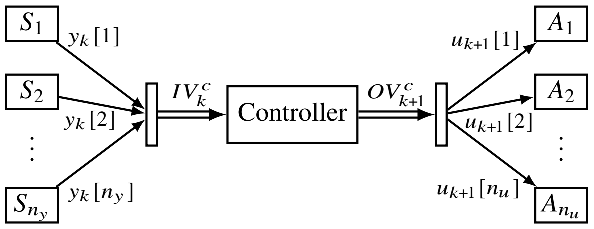

Figure 1. Setup showing sensors, controller, actuators and communication channels. The physically connected plant is omitted for clarity.

values are transmitted over a communication network, values may be lost or arrive too late, i.e., the input vector $IV_k^c \in \mathbb{R}_\perp^{n_y}$ received by the controller may contain only a subset of the sensor values $y_k \in \mathbb{R}^{n_y}$, such that $IV_k^c[i] = \perp$ if $y_k[i]$ was not received.[2] The input vector is used to update the controller's state $S_k^c \to S_{k+1}^c$, from which an output vector $OV_{k+1}^c \in \mathbb{R}^{n_u}$ is then generated and sent to the actuator nodes. The actuator vector $u_{k+1} \in \mathbb{R}_\perp^{n_u}$ may contain only a subset of the output vector components, again, due to delay and loss. More specifically, each actuator node is responsible (w.l.o.g.) for one component $u[i]$ of the actuator vector. Due to the real-time behaviour of the control system, the actuator node applies $u_{k+1}[i] = OV_{k+1}^c[i]$ if it receives the output vector by time t_{k+1}, otherwise the corresponding component $u_{k+1}[i] = \perp$ is undefined.[3]

The state and output of the controller evolve according to

$$S_{k+1}^c = \mathtt{Update}\big(S_k^c, IV_k^c\big), \tag{1}$$

$$OV_{k+1}^c = \mathtt{Output}\big(S_{k+1}^c\big). \tag{2}$$

We assume that both functions are deterministic. For convenience of notation, we use the following shorthand for the repeated `Update` function:

$$\begin{aligned} &\mathtt{Update}^{n+1}(S, IV_0, IV_1, \ldots, IV_n) \\ &\quad = \mathtt{Update}^{n}\big(\mathtt{Update}(S, IV_0), IV_1, \ldots, IV_n\big), \end{aligned} \tag{3}$$

where $\mathtt{Update}^0(S) = S$. In the following, unless noted otherwise, we consider *period* k to be synonymous with the time interval $(t_{k-1}, t_k]$, i.e., the time interval during which the controller updates its state to S_k.

Clearly, the behaviour of the NCS depends on the sequence of actuator values u_k delivered to the actuators. As mentioned above, an output vector sent by a controller may be lost due to faulty communication channels. Consequently, even in the case of a perfectly reliable controller, communication failures may leave actuators with undefined values. Since the possibility of missing actuator values is inevitable, we introduce the following definition.

[2]*Notation:* Indexing is denoted by bracket notation, so $x[i]$ indicates the ith component of a vector x. In most cases, subscripts denote sampling periods, i.e., time, and superscripts denote replicas, so x_k^r indicates the value of (a vector) x on (or associated with) replica r at time t_k. Undefined values are denoted by $\perp$. The set $\mathbb{R}_\perp = \mathbb{R} \cup \{\perp\}$ is used to denote possibly missing scalars. In pseudocode, an ampersand preceding arguments indicate call-by-reference semantics, i.e., `&var` denotes a reference to the variable `var`.

[3]Application-layer methods for dealing with missing values in NCS depend on the control design and are therefore beyond the scope of this paper. We refer, e.g., to [5], [10] for a discussion of missing sensor and actuator values.

Definition 1 (Faithful Controller): A controller is *faithful* if it maintains a state that evolves according to (1) but may omit output vectors sporadically, i.e.,

$$OV_k^c \in \big\{\mathtt{Output}\big(S_k^c\big), \perp\big\}. \tag{4}$$

The notion of a faithful controller facilitates the definition of state consistency in Sec. IV by allowing us to account also for a controller (replica) being unable to send an output (reach agreement) due to node failures or deadline misses. Most importantly, subject to faulty communication channels, the *values* of output vectors that are received by the actuators from a faithful controller are indistinguishable from those of a perfectly reliable controller. In short, faithful behaviour of the replicated controller ensures *replication transparency* [19].

It is important to note that key properties of control systems—such as stability, performance, and robustness w.r.t. disturbances—are sensitive to the end-to-end loss and latency characteristics of the entire control loop, which should always be taken into account, also when designing an NCS with replicated controller. For instance, if the the non-replicated controller is designed under the assumption of i.i.d. losses (including deadline violations) with a certain distribution, then the closed-loop system with the replicated controller can only be guaranteed to behave equivalently if it also exhibits i.i.d. losses with the same distribution. This is sometimes referred to as *performance transparency* [20]. However, these characteristics are not determined by the (replicated) controller alone but by all CPS components, i.e., sensors, actuators, and the network.

In this paper, we focus on the replicated controller, making as few assumptions as possible on the other components. From this perspective, we stress that faithful behaviour of the replicated controller is a *necessary* condition for the replication-agnostic NCS design, independent of the control design method being used. While modelling end-to-end losses is an orthogonal but equally important design prerequisite, it is beyond the scope of this paper. Its treatment for special cases (i.e., particular control system, component failure, and network models) should be considered in future research.

IV. Consistency Models

Now, we consider a group of N replicated controllers, which we denote by the set G. All controllers execute the same control functions, i.e., `Update` and `Output`. However, each controller replica $r \in G$ may experience different input vector sequences due to different (partial) arrival patterns of measurement components. Therefore, also the controller states S_k^r and outputs OV_k^r may differ in general among replicas. Moreover, replicas may be unavailable, e.g., due to crash failures, in which case they do not generate an output. For each period, we denote the subset of *influential* replicas that generate an output OV_k^r in period k as G_k^{out}, i.e.,

$$r \in G_k^{\text{out}} \iff OV_k^r \neq \perp. \tag{5}$$

Our goal is to design a replication scheme where the behaviour of a group of replicated controllers is equivalent to that of a faithful controller. In the following, we describe two consistency models which together provide this equivalence.

Definition 2 (Output Consistency): A controller group G is called *output consistent* if

$$\forall_{k>0}\forall_{r,s\in G_k^{\text{out}}}\quad OV_k^r = OV_k^s, \tag{6}$$

i.e., all influential replicas generate the same output.

Output consistency guarantees that the group does not send "conflicting" commands to different actuators at any time (nor that any actuator receives several different commands). To illustrate the importance of output consistency in the context of NCS, let us consider tracking control for a milling machine as a simple example. Each controller estimates the current position and speed of the milling head and determines as its output a vector of forces $[F_x, F_y]$ along the x and y axes required to keep the tool moving along the desired trajectory. Assume that the target trajectory is a line at 45° and two controller replicas estimate the same position but slightly different speeds, leading to two different control outputs that both represent a force tangent to the trajectory, say $[0.9, 0.9]$ and $[1.1, 1.1]$, respectively. If the corresponding actuators should now receive outputs from *different* replicas, e.g., $F_x = 0.9$ and $F_y = 1.1$, the resulting force (at approximately 50.7°) is clearly not in alignment with the desired trajectory, at least in the current time step, leading to a reduced quality.

Note, however, that Definition 2 does not make any statements about the sequence OV_k itself *over time*. Since a faithful controller updates its state according to (1) in every period, any output vector is based on a state that can be obtained through a series of `Update` steps from the state on which the previous output vector was based, even if an arbitrary number of output vectors were omitted in the interim. We say that the corresponding state is *reachable* from the state from which the previous output was generated according to the following definition.[4]

Definition 3 (Reachability): The set of states S_{k+n} in period $k+n$ that are *reachable* from a state S_k in period k is $\mathcal{R}_n(S_k) = \{\texttt{Update}^n(S_k, IV_k, \ldots, IV_{k+n-1}) \mid IV_\tau \in \mathcal{L}(y_\tau)\}$.

Here, $\mathcal{L}(y)$ is the set of possible input vectors received for the measurement vector $y \in \mathbb{R}_\perp^{n_y}$ transmitted over the network, i.e., $IV \in \mathcal{L}(y) \iff \forall_i IV[i] = y[i] \vee IV[i] = \perp$. We are now ready to introduce the stronger concept of state consistency, which reflects the state interdependencies of a controller group that behaves faithfully.

Definition 4 (State Consistency): A controller group G is called *state consistent* if

$$\forall_{k\geq 0}\forall_{r,s\in G_k^{\text{out}}}\quad S_k^r = S_k^s \tag{7}$$

$$\forall_{k>0}\forall_{r\in G_k^{\text{out}}}\forall_{s\in G_{k-d(k)}^{\text{out}}}\quad S_k^r \in \mathcal{R}_{d(k)}\big(S_{k-d(k)}^s\big) \tag{8}$$

where $d(k) = \min\{\tau \mid \tau > 0,\ G_{k-\tau}^{\text{out}} \neq \emptyset\}$, i.e., all influential replicas share a controller state that is *reachable* from the controller state of the *most recent* influential replicas.

State consistency implies output consistency, since the state S_k^r determines the output OV_k^r, cf. (2). To illustrate the importance of state consistency, let us again consider the milling example from earlier. One important concern in machining (or other mechanical systems such as aircraft [21]) is the suppression of so-called chatter, which may impair quality or even lead to actuator damage, and controllers are often designed accordingly [22]. In our example, we therefore assume that each controller replica produces an output sequence where the rate of change in actuator force is limited. However, without state consistency the replicated controller could alternate in the worst case between outputs from both replicas such that the magnitude of the output oscillates, leading to an undesirable chattering motion of the actuator. In general, violation of state consistency may introduce (high-frequency) dynamics that amount to an *unmodelled* disturbance that was not considered during the controller design.

By contrast, the output sequence produced by a controller group that satisfies Definition 4 is equivalent to that of a faithful controller. Since a faithful controller can omit to send output vectors, we can show equivalence by considering output vectors in periods where $G_k^{\text{out}} \neq \emptyset$.

Lemma 1: Consider a controller group G with uniform initial controller state $\forall_{r\in G}\, S_0^r = S_0$ and a faithful controller c with initial controller state $S_0^c = S_0$. If G is *state consistent*, then there exists a sequence of input vectors $IV_k^c \in \mathcal{L}(y_k)$ for the faithful controller such that

$$\forall_{k>0}\forall_{r\in G_k^{\text{out}}}\quad OV_k^r = OV_k^c, \tag{9}$$

i.e., the outputs OV_k generated by the group G are identical to the corresponding outputs generated by a faithful controller.

The proof is described in Appendix A.

V. Replication Algorithm

Our goal is to design a replication protocol for a group G of replicated controllers whose behaviour is equivalent to that of a faithful controller. As we have shown, a sufficient condition for this is that the underlying replication protocol offers both output and state consistency. Before describing the SCRaM protocol in detail, we first present an outline of the algorithm and discuss the requirements it should satisfy.

A. Outline and Requirements

First and foremost, replicas must implement the execution model of the generic controller as described in Sec. III. Hence, the basic algorithm executed by each replica consists of a *controller loop* that repeats the following steps:

1) receive an input vector IV_k,
2) update the controller state,
3) calculate the output vector and send it to the actuators,
4) wait for the next sampling time t_{k+1} and go to step 1.

However, since output consistency is required, we need to ensure that all replicas agree on a unique output vector in step 3 of each period. For this reason, we also execute one instance of a *single-value consensus* [15], [23] algorithm in each iteration of the controller loop, which is the basic building block of our replication scheme.

[4]This definition differs from the usual reachability notion in control theory in that the input sequences are constrained.

Let us denote the consensus instance for period k (i.e., for output vector OV_k) as C_k. We consider C_k to be the problem of agreeing on the state S_k, from which the unique output vector OV_k can be calculated deterministically. Any algorithm for solving the single-value consensus problem must satisfy the following standard properties [15], [23]:

Agreement All correct replicas decide the same value for C_k.
Integrity If a correct replica decides the value S_k for C_k, S_k must have been proposed for C_k by some replica.
Termination Every correct replica eventually decides a value for C_k.

The *agreement* and *integrity* properties ensure output consistency, provided that replicas may only send an output vector after the corresponding consensus instance has been decided.

However, the *termination* property only requires that a decision is made eventually, and is therefore not sufficient for determining a unique output within one sampling period. Indeed, it is impossible to guarantee upper bounds on the time required to complete consensus in asynchronous distributed systems [23] or in synchronous systems subject to omission failures [24]. Because any component of an output vector OV_k produced from a state S_k is discarded if it does not arrive at the corresponding actuator by time t_k, a consensus instance that terminates any later will not provide any useful output. But since output vectors may be omitted by a faithful controller, consensus instances can be aborted at the end of their corresponding sampling period without jeopardizing output consistency.

Conversely, state consistency requires that the controller states from which consecutive outputs are generated satisfy the reachability condition expressed in (8). Therefore, whenever a consensus instance *does* terminate within its period, the decided state S_k becomes influential in the sense of Definition 4 and all future decided states must be reachable from S_k. We denote such an instance as successful.

Definition 5 (Successful Consensus Instance): Let the unique value decided by consensus instance C_k be denoted by $\mathtt{Decision}(C_k)$, and the time at which C_k yields its decision be defined as the *earliest* time at which any correct replica decides this value. A consensus instance C_k is *successful* if and only if it yields $\mathtt{Decision}(C_k)$ at some time $t \leq t_k$.

While a replica that decides a value S_k can infer that the consensus instance C_k was successful, the opposite is not true, i.e., a replica not deciding by time t_k cannot be sure whether C_k was successful, nor whether its controller state S_k is reachable from the most recently decided state. Since proposing a state that does *not* satisfy this reachability condition could threaten to violate state consistency, we must suitably restrict the set of controller states that may be proposed for subsequent consensus instances.

Based on these observations, we define the following requirements that the controller loop should satisfy with respect to the consensus instance executed in each period.

Requirement 1 (Output Constraint) The output vectors of each replica $r \in G$ must satisfy

$$OV_k^r \in \left\{ \mathtt{Output}(\mathtt{Decision}(C_k)),\ \perp \right\}, \tag{10}$$
$$OV_k^r \neq \perp \implies C_k \textit{ successful}. \tag{11}$$

Requirement 2 (Proposal Constraint) Any replica $r \in G$ may only propose values S_k^r for C_k that satisfy

$$S_k^r \in \mathcal{R}_{k-k'}\big(\mathtt{Decision}(C_{k'})\big), \tag{12}$$

where $k' = \max\{\tau \mid \tau < k, C_\tau \text{ successful}\}$. (Assume that there exists a successful consensus instance C_0 with $\mathtt{Decision}(C_0) \triangleq S_0$ for the initial state.)

Clearly, Requirement 1 together with the agreement property ensures output consistency since it guarantees that no conflicting output vectors may be generated by different replicas for the same sampling period. Requirement 2 together with the integrity property ensures state consistency since it guarantees that only controller states that are reachable from previous influential controller states can be proposed and hence decided.

Finally, it is worth noting that replicas should seek to start a new consensus instance C_{k+1} as soon as new input from the sensors is available since this maximizes the likelihood of the new instance being successful and producing an output within the corresponding sampling period. This requires replicas to make a proposal at that point. Requirement 2 already ensures that any such proposal must not contradict the *possible* outcome of the previous consensus instance C_k (and, in fact, any earlier instance). Therefore, each replica can abort the previous consensus instance at the beginning of the new sampling period.

B. System Model

For describing the algorithm, we consider an asynchronous distributed system where replicas may suffer crash failures and the network may suffer omission failures. For simplicity, we assume that processing latency is negligible. Moreover, we refine our communication model by assuming that most messages are delivered with a network delay of at most δ, while $p < 1$ is the probability that a message is lost or experiences a delay greater than δ, which is denoted as *probabilistic synchronous* in [25].

Moreover, we assume that every replica is equipped with an unreliable failure detector [15], and let $\mathtt{Suspects}_r$ denote the set of replicas that r currently suspects to have failed. We assume that the failure detector satisfies strong completeness and eventual weak accuracy [16], which facilitates the use of an existing consensus algorithm as part of our protocol.[5]

Finally, we assume that clocks are synchronized among sensors, actuators, and controller replicas. Synchronization of sensors and actuators is an implicit assumption of the control system model, while sufficiently accurate synchronization of replicas is useful for obtaining tight bounds on the timeouts on listening for input vectors and sending output vectors.

[5]Such failure detectors can be implemented using timeout mechanisms [15].

C. Algorithm

We now present the SCRaM replication protocol by explaining the controller loop and the consensus instances executed by each replica in more detail. Algorithm 1 shows the algorithm executed by each replica. Lines 1–7 show the initialization of the necessary variables. As before, k denotes the current sampling period, S the replica's local controller state, and IV its current input vector. In addition, the "base" period k_b indicates the period of the most recent consensus instance C_{k_b} on whose (possible) decision the controller state S is based.[6] This value is updated whenever a replica modifies its state information through consensus and is used to satisfy Requirement 2 as will be shown in the following. The remaining variables, `mode`, v, and v_b, are related to the consensus algorithm and will be explained in the corresponding sections.

At this point we note that, although we defined a consensus instance C_k as a procedure for agreeing on a state S_k for simplicity, our algorithm is designed to agree on the tuple (S_{k-1}, IV_{k-1}). However, since the controller state S_k can be calculated deterministically as $\texttt{Update}(S_{k-1}, IV_{k-1})$, both descriptions are equivalent. We advocate the latter form since it leaves open the possibility of exchanging input vectors separately from controller states, either for the purpose of message reduction in the failure-free case when replicas already possess S_{k-1} from the previous period (assuming IV requires less memory than S), or for combining input vectors from multiple replicas in order to fill in missing components like in [9].

1) Controller Loop: The remainder of Algorithm 1 shows the controller loop executed by each replica. At the beginning of a sampling period, the input vector is received (l. 10), where the function `ReceiveAndConstructIV`(k) assembles the input vector IV_k from all received components of the measurement y_k. It returns after one network delay δ, setting components not received so far to $\perp$. The counter k is incremented (l. 11) after receiving the input vector in order to reflect the active period (i.e., to produce OV_k).

At this point, the replica starts a consensus instance. The quantities that are the subject of consensus, namely S, IV, and k_b, are collected into a data structure `est` (called the estimate[7]), which is passed as an argument to the consensus procedure.

In Line 15 the `Consensus` function is called for period k, and a reference to `est` is passed along with references to `mode`, v, and v_b, which are parameters of the consensus protocol described in Sec. V-C2. If `Consensus` returns within the current sampling period, the consensus instance was successful and `est` contains the agreed-upon values for S and IV. However, if the current sampling period expires before completion (l. 17), `Consensus` is aborted. (Of course, an abort does *not* imply that the corresponding consensus instance was unsuccessful, merely that no decision was received by the replica.)

[6]Note that C_{k_b} need not have been successful. Rather, the base period expresses that S is reachable from the most recent influential state $S_{k \le k_b}$ up to period k_b, while there may have been a more recent influential state $S_{k > k_b}$ from which S is not necessarily reachable.

[7]Terminology cf. [15]. An estimate may be proposed or decided by the consensus instance, and is therefore the replica's "best guess" of the consensus' outcome. We use a leader-based consensus protocol, cf. Sec. V-C2, where only one replica makes a proposal at a time. However, estimates of different replicas may be proposed when the leader changes.

Algorithm 1: SCRaM protocol executed on each replica $r \in G$

```
1  $k \leftarrow 0$;                                   // current period
2  $S \leftarrow S_0$;                                 // controller state
3  $IV \leftarrow [\perp, \ldots, \perp]$;                // input vector
4  $k_b \leftarrow 0$;                    // "base" period of estimate
5  mode ← normal;                                   // operation mode
6  $v \leftarrow 0$;                             // current view number
7  $v_b \leftarrow 0$;         // "base" view (most recent proposal)
   /* ============== MAIN LOOP ================    */
8  while true do
9      wait until $t \ge t_k$;           // await sampling period
10     $IV \leftarrow$ ReceiveAndConstructIV($k$);
11     $k \leftarrow k + 1$;              // advance period counter
12     decided ← false;
13     est ← $(S, IV, k_b)$;                    // prepare estimate
14     try                                // within current period
15         Consensus($k$, &est, &mode, &$v$, &$v_b$);        // $C_k$
16         decided ← true;
17     catch timeout at $t_k$                    // period expired
           // Consensus $C_k$ aborted
18     end
       /* Recover state from estimate and update:     */
19     $S \leftarrow$ Update(est.$S$, est.$IV$);
20     $IV \leftarrow [\perp, \ldots, \perp]$;
21     $k_b \leftarrow$ est.$k_b$;
22     if decided then
23         send $OV_k = \langle$Output(est.$S$), $k\rangle$ to actuators;
24     end
25 end
```

Note that, even if `Consensus` is aborted, `est` may have been modified. This is important for ensuring state consistency since aborted instances could have received proposals which were potentially decided. In order to ensure that a decided controller state is used as the basis for the controller state to be proposed (and possibly decided) in the following period, even under crash failures, it must be guaranteed that a majority of replicas are aware of that decided state. While it cannot be guaranteed that a successful decision is received by a majority of replicas within the current sampling period, a decision implies that a majority of replicas received and acknowledged the corresponding proposal, which is then contained in those replicas' estimate at the end of the sampling period.

Therefore, the new controller state S at the end of the sampling period is determined as the `Update` of the state and input vector contained in `est` (l. 19), IV is cleared (l. 20), and the base period k_b of the current state is set to that of the estimate (l. 21). Note that this update is performed regardless of the success of the consensus instance, so that S contains a controller state that is reachable from the decided state of any possibly successful consensus instance C_{k_b}. Finally, if `Consensus` was not aborted, then the replica calculates the output vector OV_k and sends it to the actuators (l. 23) before waiting for the next sampling period.

2) Consensus: For each instance C_k, we use a modified version of the consensus algorithm by Dolev et al. [16], which is an adaptation of the algorithm by Chandra and Toueg

Algorithm 2: Consensus algorithm executed by $r \in G$ for period k. Modifications w.r.t. [15], [16] are shown in blue.

```
1  Function Consensus($k$, &est, &mode, &$v$, &$v_b$)
2      while true do
3          if mode = viewchange then
4              $v \leftarrow v + 1$;
5              NextView($k$, &est, &mode, &$v$, &$v_b$);
6          end
7          $c \leftarrow$ coord($v$);          // $c$ = current coordinator
8          if $r = c$ then
9              est.$k_b \leftarrow k$;          // set base period of est
10             send ⟨Propose, $k$, $v$, est⟩ to $G$;
11         end
12         wait until received Propose from $c$ or $c \in$ Suspects$_r$;
13         if received ⟨Propose, $k$, $v$, est′⟩ then
14             est ← est′;
15             $v_b \leftarrow v$;
16             send ⟨ACK, $k$, $v$⟩ to $c$;
17         else if received message with $v' > v$ then
18             $v \leftarrow v'$;
19             NextView($k$, &est, &mode, &$v$, &$v_b$);
20             continue;
21         end
22         if $r = c$ then
23             wait until received $\lceil \frac{N+1}{2} \rceil$ ⟨ACK, $k$, $v$⟩;
24             send ⟨Decide, $k$, $v$, est⟩ to $G$;
25         end
26         wait until received Decide from $c$ or $c \in$ Suspects$_r$;
27         if received ⟨Decide, $k$, $v$, est′⟩ then
28             est ← est′;
29             $v_b \leftarrow v$;
30             return ;                              // decided
31         else if received message with $v' > v$ then
32             $v \leftarrow v'$;
33         else                // coordinator failure suspected
34             $v \leftarrow v + 1$;
35         end
36         NextView($k$, &est, &mode, &$v$, &$v_b$);
37     end
38 end
```

[15] (Sec. 6.2) for considering omission failures. It is shown in Algorithm 2.[8] We strive to keep the presentation close to the original algorithms, with our core modifications—for distinguishing consensus instances of different periods and for achieving state consistency—highlighted in blue. However, we split the algorithm into two parts described separately in the following, in order to better explain the (failure-free) normal operation of the algorithm as opposed to the handling of node failures. Instances of the modified consensus algorithm are aborted after the corresponding sampling period has ended, as described in the previous section.

The protocol uses a dedicated *coordinator* replica that is responsible for deciding on an estimate. In such leader-based protocols, we can distinguish between the *normal* operation mode when there is a single persistent coordinator without failures, and a special *view change* mode where a new coordinator has to take over (e.g., due to a crash failure of the previous coordinator) before normal operation can resume. We will first describe normal operation before discussing the mechanism for changing the coordinator.

a) Normal Operation Mode: For the moment, we ignore lines 3–6, as they do not pertain to normal operation. At the beginning of the consensus instance C_k, the coordinator c sets the base period est.k_b of its estimate to the period k of the current consensus instance (l. 9) since this estimate is to be proposed and may end up being decided. (This value is used to maintain state consistency when electing a new coordinator, as described later.) It then multicasts a Propose message containing its estimate (l. 10). Upon receiving a proposal for the current period, all replicas (including c) accept it by logging the contained values and acknowledge this to c (ll. 14–16). After receiving acknowledgments from a majority of replicas, the coordinator confirms the decision by multicasting a Decide message (l. 24). Upon receiving a decision for the current period, all replicas (including c) accept it by logging the contained values (in case the corresponding Propose had not yet been received) and return to the controller loop, indicating successful termination (ll. 28–30).

In normal operation, state consistency is satisfied by design since the decided state S_k is always reachable from the coordinator's previous state S_{k-1}. Incidentally, this is either the previous *decided* state if the coordinator decided in the previous period, or it is reachable from the last decided state since the coordinator always performs the state update in Algorithm 1, ll. 19–20 otherwise. Output consistency is satisfied since outputs are only generated upon deciding, and we assume that actuators apply output vectors labelled for period k only if received within the time interval $(t_{k-1}, t_k]$.

b) View Change Mode: Normal operation is interrupted if at least one replica suspects the coordinator to have failed. Coordinator failures (real or suspected) are handled by electing a different replica to be the coordinator and starting over in normal operation mode. As in [15], [16], our algorithm uses a rotating coordinator approach, such that coordinators are chosen deterministically. To this end, a monotonically increasing view[9] number v is incremented upon suspected coordinator failure and used to agree upon the next coordinator coord(v) $= (v \bmod N) + 1$. This "view change" is performed in the procedure NextView shown in Algorithm 3, which is invoked either if no Decide message was received (Alg. 2, ll. 34, 36) or if a message with a higher view number was received (Alg. 2, ll. 18–20, 32, 36). Note that messages with outdated view number are always discarded.

In normal operation, the persistent coordinator decides a single state for each period that is reachable from previously decided states. In contrast, when switching to a different coordinator, special care has to be taken that

[8] *Notation:* Received messages are matched to a certain format, indicated in the pseudocode. Fields marked prime are placeholders for arbitrary values. For instance, the predicate *received* ⟨Propose, k, v, est′⟩ specifies that a Propose message with the specified values for k and v is expected, accepting an arbitrary value est′ for field est. Non-matching messages are discarded or treated specially where indicated. In particular, messages with the wrong period k are discarded.

[9] While the term *round* is used in [15], [16], we use the term *view* introduced in [13] because it is less likely to be confused with a *period*. The same concept is also referred to as *term* in [14].

Algorithm 3: View change executed on $r \in G$. Modifications w.r.t. [15], [16] are shown in blue.

```
1  Procedure NextView($k$, &est, &mode, &$v$, &$v_b$)
2      mode ← viewchange;
3      send ⟨Estimate, $k$, $v$, $v_b$, est⟩ to coord($v$);
4      if $r$ = coord($v$) then
5          wait until received
               {⟨Estimate, $k$, $v$, $v'_b$, est′⟩} from $G \setminus$ Suspects$_r$;
6          if received $\lceil \frac{N+1}{2} \rceil$ Estimate messages then
7              select message with max($v'_b$, est′.$k_b$);
8              est ← est′;
9              $v_b \leftarrow v'_b$;
10         else if received message with $v' > v$ then
11             $v \leftarrow v'$;
12             NextView($k$, &est, &mode, &$v$, &$v_b$);
13         else
14             $v \leftarrow v + 1$;
15             NextView($k$, &est, &mode, &$v$, &$v_b$);
16         end
17     end
18     mode ← normal;
19     return;
20 end
```

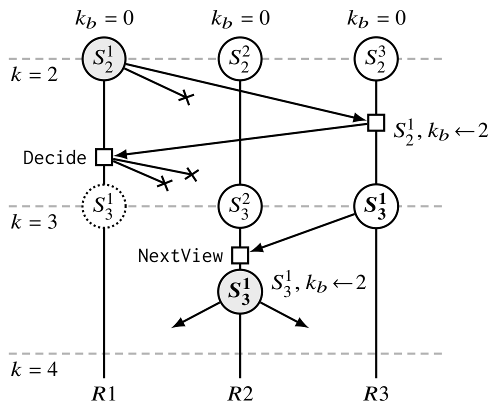

Figure 2. Example for state-consistent viewchange with three replicas. Coordinators are indicated by a shaded state node. Period $k = 2$ starts with different states and equal base period $k_b = 0$. $R1$ (coordinator) proposes S_2^1, which is acknowledged by $R3$ and then decided. $R3$ adopts the proposed state (and base period) for its next update, without generating an output. The Decide messages from $R1$ are lost. In the next period, $R1$ becomes unavailable and $R2$ is elected. The new coordinator's state is *not* reachable from the state S_2^1 decided in $k = 2$. However, it receives an estimate with S_3^1 (which *is* reachable from S_2^1). Because this estimate has a higher base period ($k_b = 2$) than its own ($k_b = 0$), $R2$ adopts the state and proposes it for period $k = 3$.

(a) no two coordinators decide different states S_k for the same sampling period k (agreement), and
(b) no coordinator proposes a state S_k that is not reachable from all states $S_{k'}$ decided in earlier periods $k' \leq k$.

The agreement concern (case *a*) is only relevant for view changes within the same consensus instance. It is already addressed by the standard consensus algorithm and described in [15], [16]. The interested reader can find a brief description of how that algorithm avoids conflicting decisions within a consensus instance in Appendix C.

In contrast to agreement, the second concern (case *b*) is also relevant when proceeding from one *period* to the next, and therefore requires special treatment. While the agreement property is maintained by the consensus algorithm when coordinators change within the same period (i.e., consensus instance), we must also prevent newly elected coordinators from proposing any state that is not reachable from the decided states of previous periods, which is crucial for maintaining state consistency. Since this introduces a dependency between subsequent consensus instances, we modified the view change mechanism to ensure that the new coordinator always selects an estimate containing a state that is valid w.r.t. state consistency. For this reason, each estimate contains the base period est.k_b of the corresponding state, which is equal to the last received Propose or Decide message (cf. Alg. 2, ll. 15, 29). Because the only way to modify a replica's state (S or est.S) without touching est.k_b or v_b is through an Update in the main loop, and because k is monotonically increasing on each replica, we know that any controller's state estimate est.S must be reachable from a state S' that was contained in an estimate proposed or decided in period est.k_b and view v_b. Conversely, if any coordinator decided a state S' in period k' and view v', then a majority of replicas must have received the corresponding Propose message and therefore have est.$k_b \geq k'$ and $v_b \geq v'$. The majority of estimates collected by the new coordinator in the NextView procedure necessarily contains at least one "witness" to the most recent proposal. Now, the new coordinator can identify the most recent *possibly successful* consensus instance $C_{k'}$ by choosing the largest est.$k_b = k'$ *among* the received estimates with the most recent view[10] v_b (l. 7). If $C_{k'}$ was successful, the corresponding state est.S is necessarily reachable from $S_{k'}$ = Decision($C_{k'}$). Finally, note that coordinators can only propose values as long as they are in normal operation mode. If a new coordinator was still waiting for estimates when the previous consensus instance was aborted (i.e., mode = viewchange at the beginning of the new period), it performs another view change (cf. Alg. 2, ll. 3–6). Figure 2 shows a small example of how view change maintains state consistency.

Note that NextView is aborted whenever the calling consensus instance is aborted, and that receiving a message with a higher view number within NextView triggers another viewchange (ll. 11, 12). Note also that viewchange is part of the single consensus loop in the original consensus algorithm, e.g., comprising Phase 1 and Phase 2 (except for the last line where the proposal is sent) in Fig. 6 of [15]. Our presentation of the algorithm is equivalent since every round of consensus with a different coordinator must be preceded by a completed view change, except for the initial round where the coordinator proposes its own estimate.

D. Correctness

Now we asses the presented algorithm with respect to the requirements laid down in Sec. V-A. The correctness of individual consensus instances using Algorithms 2–3 with respect to *agreement*, *integrity*, and *termination* follows from

[10]Choosing an est with largest v_b is necessary for agreement [15].

the correctness of the original consensus algorithm which is proved in [15], [16]. We note that we make only three essential modifications to the algorithm, none of which affect the properties of individual consensus instances. First, we add a period counter k to all relevant messages for distinguishing between different instances, which ensures that messages not belonging to the current consensus instance are rejected but does not alter the message pattern within the isolated instance. Second, we set the base period field $\texttt{est}.k_b$ of proposed estimates to k, which would be equivalent to setting the same part of all estimates to a fixed value in an isolated consensus instance. Third, we refine the selection of estimates during view change in Algorithm 3, l. 7 by adding a secondary criterion based on $\texttt{est}.k_b$. However, the chosen estimate is still from the set of messages with maximum v_b, and in the original algorithm an arbitrary estimate from this set would be selected.

The termination property may be violated because consensus instances are aborted in Algorithm 1 when the next period begins. But as shown in Sec. V-A, we do not require the termination property to hold for individual consensus instances. (Indeed, consensus termination within one period cannot be guaranteed [23], [24], while eventual termination of each instance is not sufficient to argue about the availability of the replicated controller.)

Requirement 1 is satisfied due to the agreement property ensuring (10) and Algorithm 1, ll. 22–24, ensuring (11).

Lemma 2: The protocol defined by Algorithms 1–3 satisfies Requirement 2.

The proof is described in Appendix B.

E. Discussion of the Algorithm

While we presented the SCRaM protocol in a way that is focused on reasoning about its correctness, it admits several optimizations for reducing the message overhead: For instance, the coordinator can omit the estimate from `Decide` messages to all replicas that acknowledged the corresponding `Propose`. Also, if the coordinator already decided for the previous period, it can omit the controller state from its estimate in `Propose` messages for the current period to replicas that acknowledged its `Propose` for the previous period. During view change, the `Estimate` messages can be reduced to contain only k, v, v_b, and k_b. The new coordinator can then request missing state information from only one replica.

While crash recovery was neither part of our system model nor considered in our description of the algorithm, replicas may also recover from crash failures. If a majority of replicas are always available, recovering replicas can listen for a `Propose` or `Decide` message and adopt the state information from the estimates contained therein, or they can receive `Estimate` messages from a majority of participants as part of normal leader election. After obtaining a valid state in that fashion, replicas can resume the algorithm as usual. However, if fewer than a majority of replicas may be available, they are required to log their state information in stable storage. Upon recovery, the controller state can be brought up to date (only in terms of the sampling period) by repeatedly

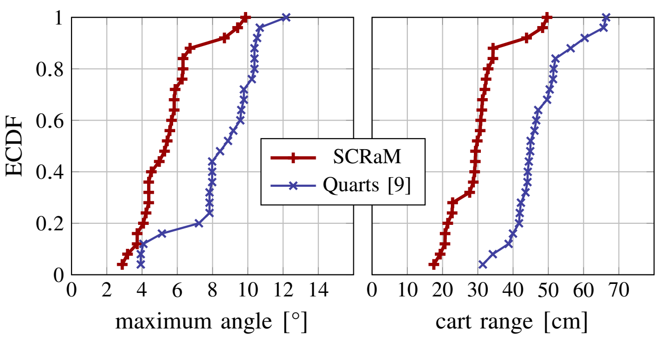

Figure 3. Empirical CDF of the maximum absolute pendulum angle $\max_k |\theta_k|$ (left) and of the cart position range $\max_k x_k - \min_k x_k$ (right) over all experiments for both replication protocols.

applying $\texttt{Update}(S, [\perp, \perp, \ldots, \perp])$, with values for k_b and v_b remaining as prior to crash failure. Of course, the protocol can only make progress while a majority of replicas are available.

VI. Evaluation

In this section, we present an evaluation of SCRaM using an experimental CPS setup. The Quarts protocol as described in [9] is considered for comparison. We implemented both protocols in the Julia language [26], using the `SimJulia` package to build an event-based simulation of the sensor and actuator components and message exchanges. Message loss follows a Bernoulli process with probability $p = 10^{-3}$, while the delay for each message is drawn from a uniform distribution on $(0\,\text{ms}, 0.5\,\text{ms}]$. Process failures are simulated as a Gilbert-Elliot process with (stationary) crash probability $\theta_c = 0.1$ and mean time to repair (MTTR) $R = 3\,\text{s}$.

The physical part of the experiment is an inverted pendulum (angle from upright denoted by θ) balancing on a cart (position denoted by x). The cart is driven by a stepper motor using a belt assembly. A microcontroller drives the stepper, reads θ using an incremental rotary encoder, and communicates with the PC running the event-based simulation of the replicated controller, which is periodically suspended until receiving a new measurement, to keep in sync with real time.

We used a standard LQG design for the $N = 3$ controller replicas, i.e., `Update` is a Kalman filter iteration, while `Output` applies an LQR gain to the filter's state estimate. The sampling period is $T_s = 50\,\text{ms}$. Closer details of the experimental setup are provided in Appendix D.

Our experiment consisted of balancing the pendulum for 180 s, starting upright with the cart at the origin. We repeated the experiment 25 times for each protocol, always with the same realization (seed value) of the crash failure model. The average availability—i.e., the proportion of periods with $OV \neq \perp$—was 99.2 % for all experiments. Fig. 3 shows the cumulative distribution of the maximum absolute pole angle and the position range covered by the cart pre experiment. Even though both replication protocols execute the same controller function and achieve the same availability, the NCS replicated using SCRaM keeps the angle 2.97° (or 35 %) smaller and the cart range 17.15 cm (or 36 %) smaller on average than with

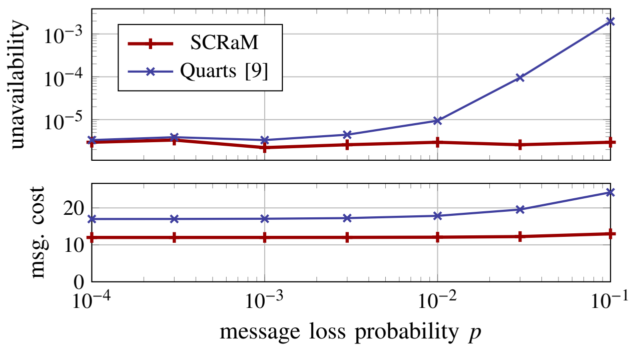

Figure 4. Unavailability (top) and messaging cost (bottom) of both replication protocols for $p \in [10^{-4}, 10^{-1}]$, $\theta_c = 10^{-3}$

Quarts. This demonstrates that providing state consistency with SCRaM can offer a clear performance advantage.

Furthermore, Fig. 4 shows a comparison of SCRaM and Quarts for varying message loss probability with respect to unavailability and messaging cost (in average protocol messages sent per period) using 5 h simulations (i.e., 9×10^5 periods at $T_s = 20\,\text{ms}$). The unavailability of both protocols is comparable at about 3×10^{-6} for low message loss rates. However, the unavailability of Quarts increases at high message loss rates, while that of SCRaM remains low. Moreover, the messaging cost of SCRaM is consistently lower that that of Quarts by 30–45 %. This demonstrates that SCRaM is efficient and more robust to message loss, which implies that state consistency comes at no additional cost in general.

VII. Summary and Outlook

In this paper, we defined two consistency concepts for controller replication in networked control systems (NCS), namely *output* and *state consistency*, which together ensure that a replicated controller can be used as a functionally indistinguishable drop-in replacement for a non-replicated controller. With SCRaM, we presented a corresponding efficient replication protocol.

While SCRaM can be used very generically for periodically sampled controllers and is agnostic to the underlying control design methodology, we plan to incorporate semantic knowledge for specific controller designs into our replication algorithm in future work. In particular, control-specific performance metrics could be used to inform the choice of states to propose by modifying the view change algorithm, or state consistency could even be replaced with a condition related to optimal control performance.

Acknowledgments

We thank Steffen Linsenmayer for his valuable feedback. This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 285825138.

References

[1] J. P. Hespanha, P. Naghshtabrizi, and Y. Xu, “A survey of recent results in networked control systems,” *Proc. IEEE*, vol. 95, no. 1, pp. 138–162, Jan. 2007.

[2] X.-M. Zhang, Q.-L. Han, and X. Yu, “Survey on recent advances in networked control systems,” *IEEE Trans. Industrial Informatics*, vol. 12, no. 5, pp. 1740–1752, Oct. 2016.

[3] L. Schenato, B. Sinopoli, M. Franceschetti, K. Poolla, and S. S. Sastry, “Foundations of control and estimation over lossy networks,” *Proc. IEEE*, vol. 95, no. 1, pp. 163–187, Jan. 2007.

[4] F. Koushanfar, M. Potkonjak, and A. Sangiovanni-Vincentelli, “Fault tolerance techniques for wireless ad hoc sensor networks,” in *IEEE Int. Conf. Sensors*, vol. 2, Jun. 2002, pp. 1491–1496.

[5] X. Liu and A. Goldsmith, “Kalman filtering with partial observation losses,” in *43rd IEEE Conf. Decision and Control (CDC)*, Dec. 2004, pp. 4180–4186.

[6] J. W. Bennett, G. J. Atkinson, B. C. Mecrow, and D. J. Atkinson, “Fault-tolerant design considerations and control strategies for aerospace drives,” *IEEE Trans. Industrial Electronics*, vol. 59, no. 5, pp. 2049–2058, May 2012.

[7] P. Bailis and K. Kingsbury, “The network is reliable,” *Communications of the ACM*, vol. 57, no. 9, pp. 48–55, Sep. 2014.

[8] M. Chérèque, D. Powell, P. Reynier, J.-L. Richier, and J. Voiron, “Active replication in Delta-4,” in *22nd Int. Symp. Fault-Tolerant Computing*, Jul. 1992, pp. 28–37.

[9] W. Saab, M. Mohiuddin, S. Bliudze, and J.-Y. Le Boudec, “Quarts: Quick agreement for real-time control systems,” in *22nd IEEE Conf. Emerging Technologies & Factory Automation*, Sep. 2017, pp. 1–8.

[10] L. Schenato, “To zero or to hold control inputs with lossy links?” *IEEE Trans. Automatic Control*, vol. 54, no. 5, pp. 1093–1099, 2009.

[11] W.-A. Zhang and L. Yu, “Stabilization of sampled-data control systems with control inputs missing,” *IEEE Trans. Automatic Control*, vol. 55, no. 2, pp. 447–452, Feb. 2010.

[12] D. E. Quevedo and D. Nešić, “Robust stability of packetized predictive control of nonlinear systems with disturbances and markovian packet losses,” *Automatica*, vol. 48, no. 8, pp. 1803–1811, 2012.

[13] B. M. Oki and B. H. Liskov, “Viewstamped replication: A new primary copy method to support highly-available distributed systems,” in *7th ACM Symp. Principles of Distributed Computing*, 1988, pp. 8–17.

[14] D. Ongaro and J. Ousterhout, “In search of an understandable consensus algorithm,” in *USENIX Annual Technical Conf.*, Philadelphia, PA, Jun. 2014, pp. 305–319.

[15] T. D. Chandra and S. Toueg, “Unreliable failure detectors for reliable distributed systems,” *J. ACM*, vol. 43, no. 2, pp. 225–267, Mar. 1996.

[16] D. Dolev, R. Friedman, I. Keidar, and D. Malkhi, “Failure detectors in omission failure environments,” Cornell University, Ithaca, NY, USA, Tech. Rep. 1813/7263, Sep. 1996.

[17] L. Lamport, R. Shostak, and M. Pease, “The byzantine generals problem,” *ACM Trans. Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, 1982.

[18] A. Ballesteros, J. Proenza, M. Barranco, and L. Almeida, “Reconfiguration strategies for critical adaptive distributed embedded systems,” in *48th IEEE/IFIP Int. Conf. Dependable Systems and Networks Workshops*, Jun. 2018, pp. 57–58.

[19] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems: Concepts and Design*, 3rd ed. Addison-Wesley, 2001.

[20] J. Crowcroft, *Open Distributed Systems*. Artech House, 1996.

[21] A. Levant, A. Pridor, R. Gitizadeh, I. Yaesh, and J. Z. Ben-Asher, “Aircraft pitch control via second-order sliding technique,” *J. Guidance, Control, and Dynamics*, vol. 23, no. 4, pp. 586–594, 2000.

[22] J. L. Dohner, J. P. Lauffer, T. D. Hinnerichs, N. Shankar, M. Regelbrugge, C.-M. Kwan, R. Xu, B. Winterbauer, and K. Bridger, “Mitigation of chatter instabilities in milling by active structural control,” *J. Sound and Vibration*, vol. 269, no. 1, pp. 197–211, 2004.

[23] M. J. Fischer, N. A. Lynch, and M. S. Paterson, “Impossibility of distributed consensus with one faulty process,” *J. ACM*, vol. 32, no. 2, pp. 374–382, Apr. 1985.

[24] U. Schmid, B. Weiss, and I. Keidar, “Impossibility results and lower bounds for consensus under link failures,” *SIAM J. Computing*, vol. 38, no. 5, pp. 1912–1951, 2009.

[25] D. Dzung, R. Guerraoui, D. Kozhaya, and Y.-A. Pignolet, “Never say never – probabilistic and temporal failure detectors,” in *IEEE Int. Parallel and Distributed Processing Symp.*, May 2016, pp. 679–688.

[26] J. Bezanson, J. Chen, B. Chung, S. Karpinski, V. B. Shah, J. Vitek, and L. Zoubritzky, “Julia: Dynamism and performance reconciled by design,” *Proc. ACM Program. Lang.*, vol. 2, no. OOPSLA, Oct. 2018.

Appendix A

Proof of Lemma 1: We use inductive reasoning to show that there exists a sequence $(IV_k^c)_{k \geq 0}$ of input vectors for which (9) is satisfied. Since the output of c obeys (4), we can assume $G^{\text{out}}_k = \emptyset \implies OV^c_k = \bot$ without loss of generality. Let the periods when G produces an output be denoted by k_i, where $k_{i+1} = \min\{\tau \mid \tau > k_i,\ G^{\text{out}}_\tau \neq \emptyset\}$, $k_0 \triangleq 0$.

Basis: Since S_0 is fixed, the initial output of replicas $r \in G^{\text{out}}_0$ is $OV^r_0 = OV^c_0 = \texttt{Output}(S_0)$. Hence, (9) is satisfied for $k \leq k_0 = 0$ with an empty input vector sequence $\sigma_0 = \varepsilon$.

Inductive step: Assume that (9) is satisfied for $k \leq k_i$ with some input sequence $\sigma_i = \left(IV^c_k\right)^{k_i - 1}_{k=0}$ and that all replicas $s \in G^{\text{out}}_{k_i}$ hold the same state $S^s_{k_i} \equiv S^c_{k_i} \triangleq S_{k_i}$ as c.

Now consider a replica $r \in G^{\text{out}}_{k_{i+1}}$ and note that $k_{i+1} = k_i + d(k_{i+1})$. Because G is state consistent, $S^r_{k_{i+1}}$ must be reachable from S_{k_i}, cf. (8). Therefore, there exists a sequence of input vectors $\sigma' = \left(IV'_k\right)^{k_{i+1}-1}_{k=k_i}$ such that

$$S^c_{k_{i+1}} = S^r_{k_{i+1}} = \texttt{Update}^{d(k_{i+1})}\left(S_{k_i}, \sigma'\right)$$

and, consequently, $OV^c_{k_{i+1}} = OV^r_{k_{i+1}}$. Because of (7), this is true with identical values for all replicas $r \in G^{\text{out}}_{k_{i+1}}$. Therefore, (9) is satisfied for $k \leq k_{i+1}$ with the input sequence $\sigma_{i+1} = \sigma_i \circ \sigma'$. ■

Appendix B

Proof of Lemma 2: Assume that C_k is successful and the controller state $S_k = \texttt{Update}(S_{k-1}, IV_{k-1})$ is decided in view v. Therefore, a majority of replicas must have base period $k_b = k$, base view $v_b \geq v$, and controller state $S = \texttt{Decision}(C_k)$ at time t_k (i.e., Alg. 1 after l. 21).

Now assume any replica r proposes an estimate with (S'_k, IV'_k) in view $v' \geq v$ for C_{k+1}. If $v' = v$, then $r = c$ and $S'_k = S_k$. If $v' > v$, then r must have chosen an estimate with S_k or (S_{k-1}, IV_{k-1}) in Alg. 3, which was acknowledged by a majority of replicas for C_k. Therefore, only an estimate with state S_k (and arbitrary IV) can be proposed for C_{k+1}, which is equivalent to the proposal of a state $S'_{k+1} \in \mathcal{R}(S_k)$ in terms of Property 2 and satisfies (12). By extension of the same argument, this is true for all $C_{k'}$, $k' > k$. ■

Appendix C

While the mechanism for guaranteeing agreement in Alg. 2–3 is treated in [15], [16], we briefly explain it here for the sake of completeness. Each replica only accepts messages with its own current view number v, and stores the view number of the last received `Propose` or `Decide` message as v_b (cf. Alg. 2, ll. 15, 29). Therefore, if any coordinator decided a value in view v', then a majority of replicas must have received the corresponding `Propose` message and therefore have $v_b \geq v'$. For a successful viewchange, the new coordinator must collect estimates from a majority of replicas (ll. 3,6), which necessarily contains at least one "witness" to the most recent proposal received by a majority. The new coordinator adopts this estimate by choosing a message with the largest v_b (ll. 7–8). Therefore, if a value was decided in the previous view, it is impossible for a different value to be decided.

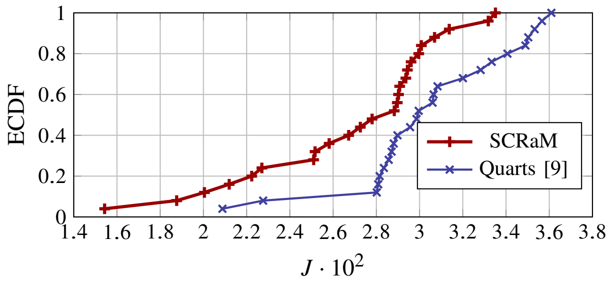

Figure 5. Empirical CDF of LQR cost J for both replication protocols.

Appendix D

The cart-driven inverted pendulum used in our experiments is 0.6 m long and runs on a track with a usable range of 1.2 m. We denote the state of the system by $\xi = [x, \dot{x}, \theta, \dot{\theta}]^\top$, where x is the distance of the cart from the origin (track center) in meters and θ is the angle of the pole from upright in radians. Sensor measurements are $y = [x, \theta]^\top$ and the actuation input is the cart acceleration $u = \ddot{x}$. For the design of the LQR controller, we use the continuous-time cost metric

$$J = \lim_{T\to\infty} \frac{1}{T} \int_0^T \xi^\top \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 \\ 0 & 0 & 1.5 & 0 \\ 0 & 0 & 0 & 0.01 \end{bmatrix} \xi + 0.02 \cdot u^2 \ \mathrm{d}t.$$

Linearization and discretization at $T_s = 50$ ms of the nonlinear pendulum equation yields the discrete-time LTI system

$$\xi_{k+1} = \begin{bmatrix} 1 & 0.05 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1.018 & 0.05 \\ 0 & 0 & 0.705 & 1.018 \end{bmatrix} \xi_k + \begin{bmatrix} 0.00125 \\ 0.05 \\ 0.00179 \\ 0.07185 \end{bmatrix} u_k + w_k$$

$$y_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \xi_k + v_k$$

$$J_T = \frac{1}{T} \sum_{k=0}^{T} \xi_k^\top Q \xi_k + 2\xi_k^\top H u_k + R u_k^2, \text{ where}$$

$$Q = \begin{bmatrix} 5.0 & 0.125 & 0 & 0 \\ 0.125 & 0.054 & 0 & 0 \\ 0 & 0 & 7.596 & 0.207 \\ 0 & 0 & 0.207 & 0.057 \end{bmatrix} \cdot 10^{-2},\ H = \begin{bmatrix} 2.083 \\ 1.328 \\ 5.359 \\ 1.975 \end{bmatrix} \cdot 10^{-5},\ R = 10^{-3}.$$

The LQR optimal input for this system is

$$u_k = \begin{bmatrix} 5.295 & 5.967 & -42.519 & -11.239 \end{bmatrix} \cdot \xi_k.$$

We use a Kalman filter variant presented in [5] to handle IVs with missing components. As noise covariance matrices for w_k and v_k we take

$$W = \begin{bmatrix} 0.0504 & 0.0125 & 0 & 0 \\ 0.0125 & 0.5 & 0 & 0 \\ 0 & 0 & 5.143 & 4 \\ 0 & 0 & 4.301 & 102 \end{bmatrix} \cdot 10^{-3},\ V = \begin{bmatrix} 0.15 & 0 \\ 0 & 2.5 \end{bmatrix} \cdot 10^{-3}.$$

Fig. 5 shows the cumulative distribution of the LQR cost J for all experiments, which is 13.5 % higher on average for Quarts compared to SCRaM.

The Gilbert-Elliot model for crash failures consists of a two-state Markov chain. In the good state (G), a replica may be sporadically unavailable with probability p. In the bad state (B), a replica remains unavailable (crashed) until it recovers to the good state. The transition probability for G⟶B (crash) is $\frac{T_s \theta_c}{R(1-\theta_c)} \approx 0.19$ %, while that for B⟶G (recover) is $\frac{T_s}{R} \approx 1.7$ %. Because the number of concurrently crashed replicas cannot be bounded in this model, we use a recovery operation based on the assumption of stable storage as outlined in Sec. V-E.