Continuous Consensus via Common Knowledge

Tal Mizrahi* Yoram Moses†

Continuous consensus is the problem of having each process i maintain at each time k an up-to-date core $M_i[k]$ of information about the past, so that the cores are guaranteed to be identical. A simple algorithm for continuous consensus in fault-prone systems called ConCon is presented, based on a knowledge-based analysis. Continuous consensus is shown to be closely related to common knowledge. Via this connection, the characterization of common knowledge in systems with crash and omission failures by Moses and Tuttle is used to prove that ConCon is optimal—it produces the largest possible core at any given time. Finally, we modify the ConCon algorithm to obtain a uniform solution, in which all processes (faulty and nonfaulty) obtain the same core information at any given time.

1. Introduction

Agents in a given setting, whether players in a game or processes executing a protocol in a distributed computer system, typically have asymmetric information. Part of this information—the facts that are common knowledge—is identical for all agents and, moreover, can in principle be identified by each agent. By acting on information that is common knowledge agents are guaranteed to be acting on consistent information available to all agents. This connection was perhaps first drawn by David Lewis in his work on conventions, which led to the original definition of common knowledge [1]. The role of common knowledge for consistent simultaneous actions has been firmly established in the literature [2, 3, 4, 5]. Dwork and Moses in [3] presented an optimal solution to simultaneous Byzantine agreement in the presence of crash failures. They proved that simultaneous agreement can be reached exactly when the value of at least one agent's initial vote becomes common knowledge. Moses and Tuttle in [4] extended this work to the more complex omission failure model, and presented optimal solutions for a broader class of simultaneous choice problems. Implicit in the latter work is the computation of a core of information that characterizes the common knowledge at any given point in time. This computation is based on a subtle fixed-point construction. The current paper extends both of these works by presenting a simple and intuitive algorithm for computing the information about the past that is common knowledge. Two varieties of the continuous consensus task are defined, which further generalize simultaneous choice problems. In both [3] and [4], as well as in the current paper, it is assumed that the processes operate in synchronous rounds of communication and there is an a priori bound on the overall number of potentially faulty processes. While some practical systems work in a synchronous setting very close to this one [7, 8], many do not. Nevertheless, it is often practical to maintain a core that is updated relative to a given time period on a daily, hourly, or per-minute basis.

Maintaining consistency of the information held by different nodes in an unreliable distributed system is a challenging problem. Moreover, providing an up-to-date consistent picture at different sites of the

*`dew@tx.technion.ac.il`, Department of Electrical Engineering, Technion, Haifa, 32000 Israel.

†`moses@ee.technion.ac.il`, Department of Electrical Engineering, Technion, Haifa, 32000 Israel. Work on this paper was performed in part during a sabbatical at the School of Computer Science and Engineering, The University of New South Wales, Sydney, NSW 2052, Australia, where it was partially supported by ARC Discovery Grant RM02036.

system can sometimes aleviate the need to explicitly activate voting or agreement protocols to handle individual transactions [6]. Weaker guarantees than simultaneous consistency are popular, where consistency is guaranteed over time: If one process can determine that an event has occurred, the others will eventually know this as well [9]. These weaker consistency conditions are essential in some systems since simultaneous coordination requires nontrivial common knowledge, and this is not attainable in truly asynchronous systems [2].

We now specify our problem more formally. With respect to a set E of *monitored events*, we would like each process i to hold a copy of a shared list of events of E. This list may include the events that took place and, depending on the application, possibly additional information such as the times and locations at which they occurred. We define a *continuous consensus* (*CC*) service to be a distributed protocol that at all times $k \geq 0$ provides each process i with a *core* $M_i[k]$ of annotated events of E. In every run of this protocol the following is required to hold for all nonfaulty processes i and j.

Accuracy: All events in $M_i[k]$ occurred.

Consistency: $M_i[k] = M_j[k]$ at all times k.

Completeness: If an event $e \in E$ takes place at a process j at any point, then at some later time k a record of e's occurrence will appear in $M_i[k]$.

The consistency property guarantees that the information in the local lists is in fact shared among the nonfaulty processes at any given time. Since an event can enter $M_i[k]$ for some nonfaulty process i, only if it also enters the lists $M_j[k]$ for all other nonfaulty processes j, it follows that a process i may know of the occurrence of a monitored event $e \in E$ long before i can add e to its list $M_i[k]$. In many cases it is, of course, desirable to have the shared list in a continuous consensus application be as up-to-date as possible. A variant of this problem, which we call *uniform continuous consensus* (*UCC*) is defined similarly, except that the processes i and j are arbitrary, and may be faulty in the run.

We shall use standard techniques from [2] to show that in any implementation of CC the information in the lists M_i is guaranteed to be common knowledge. Our solutions for CC and UCC, called ConCon and UniConCon, will be optimal in providing at any given time the largest and most informative core possible by any protocol. This is proven using the characterization of common knowledge in the crash and omission failure models given in [4]. Interestingly, the solution to UCC does not incur any degradation in the information contained in the core of shared information.

The structure of ConCon and UniConCon provides insight into how common knowledge comes about in synchronous systems with failures. The analysis in the crash failure model showed that *clean* rounds are central to common knowledge in that model [3]. In the crash failure model, a round of communication is clean if no new failure is discovered in the round. Following a clean round, all processes can have the same information about the past. Once it is common knowledge that a round was clean, the information available to nonfaulty processes before this round becomes common knowledge. In the omissions model, no natural notion analogous to a clean round was found. Indeed, a process failure can be discovered by a faulty process in one round, and then passed on to nonfaulty ones in a much later round. As a result, the round in which a failure is discovered is no longer a very useful measure. The ConCon protocol suggests a more subtle definition in the omission model that serves an analogous role. The failures we consider are benign, in the sense that faulty processes never send incorrect messages ("lies"). Nevertheless, information obtained from a process that is known to be faulty has a qualitatively different value compared to information received from a process that appears to be nonfaulty.

This paper is organized as follows. The next section provides definitions of the formal notions used in the paper. Section 3 presents the ConCon protocol for continuous consensus and proves its correctness. Section 4 relates continuous consensus to common knowledge. Results from [4] are used to prove that the

ConCon protocol yields a maximally up-to-date solution for continuous consensus. Section 5 presents the UniConCon protocol, which provides an optimal *uniform* solution to continuous consensus in which all processes (including the faulty ones) are guaranteed to have the same core at all times. Finally, a few concluding remarks are presented in Section 6. Proofs that do not appear in the body of the paper appear in the appendix.

2. Preliminaries

Our treatment of the continuous consensus problem will be driven by a knowledge-based analysis. A general approach to modelling knowledge in distributed systems was initiated in [2] and given a detailed foundation in a book on the subject [5] (most relevant to the current paper are Chapters 4 and 6). The lion's share of technical analysis in this paper will be performed with respect to a single protocol, which gives rise to a specific class of systems. For ease of exposition, our definitions will be tailored for this particular setting.

The Communication Network

We consider a synchronous network with $n \geq 2$ possibly unreliable processes, denoted by $\mathbb{P} = \{1, 2, \ldots, n\}$. Each pair of processes is connected by a two-way communication link. Processes correctly identify the sender of every message they receive. They share a discrete global clock that starts out at time 0 and advances by increments of one. Communication in the system proceeds in a sequence of *rounds*, with round $k+1$ taking place between time k and time $k+1$. Each process starts in some *initial state* at time 0. Then, in every following round, the process first sends a set of messages to other processes, and then receives messages sent to it by other processes during the same round. In addition, a process may also receive requests for service from clients external to the system (think, for example, of deposits and withdrawals at branches of a bank), or input from sensors with information about the world outside of the system (e.g., smoke detectors). Finally, the process may perform local computations based on the messages it has received. The history of an infinite execution of such a network will be called a *run*.

Nature's Role: Inputs and Failures

We think of a solution to the continuous consensus problem as a protocol operating (or playing) against an adversary called *nature*. Nature determines two central aspects of any given run: Inputs and failures.

Inputs. We have the set of monitored events E and for every process i a set of initial local states Σ_i. Nature determines the initial states of the processes and the external inputs they receive in every round. We represent the external inputs in an infinite execution as follows. Define a set $V = \mathbb{P} \times \mathbb{N}$ of *process-time nodes* (or *nodes*, for short). An *(external) input assignment* is a function ζ associating with every (initial) node $\langle i, 0 \rangle$ at time 0 an initial state from Σ_i and with each node $\langle i, k+1 \rangle$ an input from E. The structure of the elements of E will not concern our analysis.

Failures. The second aspect of a run that is determined by nature is the identity of the faulty processes, and the details of their faulty behavior. These depend on the particular failure model being assumed. In this paper we consider two closely-related failure models, called *crash* and *omission*. For simplicity, a process will be considered faulty in a run if it displays faulty behavior at any point during the run. In the crash failure model, a faulty process *crashes* in some round $k \geq 1$. In this case, it behaves correctly in the first $k-1$ rounds and sends no messages from round $k+1$ on. During its crashing round k, the process may succeed in sending messages on an arbitrary subset of its channels. In the omission model, a faulty process may omit to send messages in any given round. It sends messages only according to its protocol (it cannot misrepresent or lie), and nature determines for every round what subset of its messages will successfully be delivered. We remark that even faulty processes receive all messages sent to them over non-blocked channels. If a message is not delivered, its sender is necessarily faulty. We formally represent the *failure pattern* in a given run via an edge-labelled graph (V,E,β), where V is the set of process-time nodes defined above, and $E = \{(\langle i,k\rangle,\langle j,k+1\rangle) : i \neq j, k \geq 0\}$. An edge $e = (\langle i,k\rangle,\langle j,k+1\rangle) \in E$ stands for the round $k+1$ communication in the channel from i to j. The labelling function $\beta : E \to \{\mathtt{Y},\mathtt{N}\}$ captures when such channels are blocked and when they operate correctly. Intuitively, $\beta(e) = \mathtt{N}$ means that e is blocked for communication, while $\beta(e) = \mathtt{Y}$ means that it is not blocked. In the latter case, a message on e, if sent, will be delivered.

Nature's combined contribution to a run r is captured by an *execution graph*. This is a labelled graph $\mathsf{G}^r = (V,E,\zeta,\beta)$ with labels ζ to the vertices giving the input assignment and labels β to the edges defining the failure pattern.[1] Notice that all execution graphs over n processes have the same edge and vertex sets (V,E)—a complete grid of $n \times \mathbb{N}$ nodes, with edges from each node $u \in V$ at level k to all nodes of level $k+1$ with a process name different from u's. Different execution graphs G differ only in the labelling functions ζ and β. Figure 1(a) contains an illustration of the nodes of an execution graph, with some of the edges describing round $k+1$. Observe that all edges from one time point to the next are in the graph—some crossed, depicting their being blocked by β, while the others are available for communication.

We now consider particular subgraphs of $\mathsf{G} = (V,E,\zeta,\beta)$ that will be useful later on. Define the β-*closure* of a node $u \in V$ with respect to G, denoted V_u, to be the smallest set containing u that satisfies for all $v,v' \in V$ both (i) if $\beta(v,v') = \mathtt{Y}$ and $v' \in V_u$ then $v \in V_u$ and (ii) for all $j \in \mathbb{P}$ and $k \geq 0$, if $\langle j,k+1\rangle \in V_u$ then $\langle j,k\rangle \in V_u$. Intuitively, the β-closure will contain all nodes about which u can receive information either directly or via a sequence of messages. We define the *maximal potential view* (or *view* for short) at node $u = \langle i,k\rangle$ in G, denoted by $\mathsf{V}_i(k)$, to be a subgraph generated by the β-closure of u. More formally, $\mathsf{V}_i(k) = (V_u,E_u,\zeta \restriction V_u,\beta \restriction E_u)$ where $E_u = E \restriction V_u$ is the restriction of E to the nodes of V_u, and similarly for the ζ and β functions. See Figure 1(a) for an illustration of a view $\mathsf{V}_i(k)$. Later on we will use the extension of a view to sets of processes $S \subseteq \mathbb{P}$, where $\mathsf{V}_S(k)$ is defined to be the union of the graphs $\mathsf{V}_j(k)$, over all $j \in S$.

Full-Information Protocols

A *full-information protocol* (FIP) is one in which processes have perfect recall and observe all incoming messages and external inputs that they receive. Moreover, in every round, every process sends a message encoding all of its information to all other processes. It is not hard to show that in any such protocol a process is able to reconstruct $\mathsf{V}_i(k)$ from its information at time k. Without loss of generality, we will assume for the sake of concreteness that the local state of a process is maintained in the form of a view $\mathsf{V}_i(k)$, and the message sent by i in round $k+1$ is $\mathsf{V}_i(k)$.

Since in a FIP a message is sent on every channel in every round, the execution graph describes all aspects of a run: What inputs are received by the processes, which processes are faulty, and, for every message, whether it is delivered or not. Moreover, the contents of delivered messages can also be derived from the graph G. From now on we shall identify a *run* r of a FIP with its execution graph G^r. It is a folk theorem, perhaps first shown by Coan [10], that any deterministic protocol can be simulated by a FIP.

[1]The run r appears in the superscript of G^r. Throughout the paper, we omit explicit reference to the run whenever it is clear from context.

Systems and Knowledge

Generally speaking, we identify a *system* with a set R of runs. For a general protocol, a run r is an infinite sequence of states, and there is a well defined *local state* $r_i(m)$ for every process i and time m. For the FIP we identify runs with execution graphs, while in general every execution graph will determine a run of a protocol P (cf. [4, 5]). The systems that we study in this paper are thus parameterized by a five-tuple $(n,t,fm,\Sigma,\mathcal{E})$, where $n \geq 2$ and $0 \leq t \leq n-2$ and $fm \in \{\text{crash}, \text{omission}\}$ is a failure model. $\Sigma = (\Sigma_1, \ldots, \Sigma_n)$ assigns a nonempty set of initial states Σ_i for every process, and $\mathcal{E}$ determines monitored inputs. The exact identity and internal structure of Σ and $\mathcal{E}$ is application-dependent, although to avoid degeneracies we assume that $|\mathcal{E}| > 1$ and $|\Sigma_i| \geq 1$ for every $i \in \mathbb{P}$. A FIP system $R = R(n,t,fm,\Sigma,\mathcal{E})$ is defined to be the set of all runs of the FIP with n processes, at most t of which fail according to the failure model fm, and where the input assignment assigns local states and external inputs from Σ and $\mathcal{E}$, respectively.

Our analysis makes use of the knowledge that processes achieve at different times in various runs. As is standard in the literature, formulas will be considered true or false at a *point*, which is a pair (r,m) consisting of a run $r \in R$ and a time $m \in \mathbb{N}$. Moreover, since notions of knowledge typically depend on truths at points in runs other than the current one, we shall define truth with respect to a system R. Let $\Phi = \{p,q,p',\ldots\}$ be a set of propositions. Intuitively, a proposition is a basic primitive fact. An example of a relevant proposition in the context of continuous consensus is "$\zeta(i,k) = e$", stating the occurrence of the event (or input) $e \in \mathcal{E}$ at a given process i at time k. We will also consider propositions that state what the contents of the core $M_i[m]$ are. Given a system R, each proposition $p \in \Phi$ is identified with a set $[\![p]\!]$ of points of r. A proposition $p \in \Phi$ holds at (r,m), which we denote by $(R,r,m) \models p$, if $(r,m) \in [\![p]\!]$.

We construct a logical language $\mathcal{L}$ by closing Φ under Boolean connectives $\wedge$ and $\neg$, and under modal knowledge operators K_i, D_S, C and C_N where $i \in \mathbb{P}$ and $S \subseteq \mathbb{P}$ is a set of processes. Here K_i stands for process i's knowledge, D_S corresponds to the *distributed knowledge* that is implicit in the set of processes S, C stand for common knowledge, and C_N stands for common knowledge among the nonfaulty processes. The semantics of the Boolean operators is standard, and we now review the definitions for K_i and D_S. Common knowledge will be considered in Section 4. The formal definitions (cf. [5]) of satisfaction for knowledge and distributed knowledge formulas are briefly stated as follows:

$(R,r,m) \models K_i\varphi$ iff $(R,r',m') \models \varphi$ for all points (r',m') with $r' \in R$ such that $r_i(m) = r'_i(m')$.

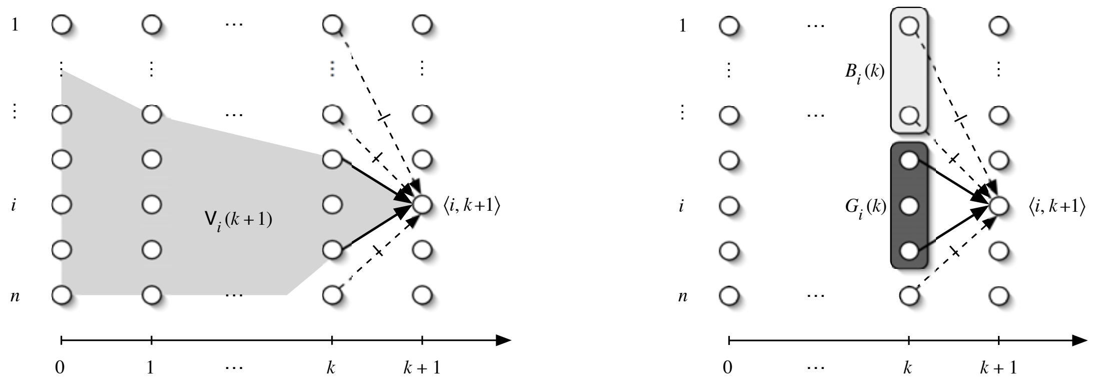
$(R,r,m) \models D_S\varphi$ iff $(R,r',m') \models \varphi$ for all points (r',m') with $r' \in R$ such that $r_j(m) = r'_j(m')$ holds for all $j \in S$.

A process knows φ by this definition if its local state (which captures the information it has access to) implies that φ holds. Distributed knowledge is defined similarly, but is based on the combined information available to the members of a set S of processes. In a full-information protocol, the distributed knowledge of S is equivalent to the knowledge of a process that would have as its local state at every point (r,m) of R the view $\mathsf{V}^r_S(m)$.

Knowledge in the FIP has a number of unique properties. For example, suppose that process i receives messages in round $k+1$ from the processes in the set S. Then by construction, $\mathsf{V}_S(k)$ is contained (as a subgraph) in i's view $\mathsf{V}_i(k+1)$ at the end of the round. This immediately implies that all facts about the past that are distributed knowledge to S at time k are known by i at time $k+1$. This observation plays a role in the solution to the continuous consensus problem described in the next section.

3. The ConCon Protocol

The existence of communication failures in a system may hinder processes to obtain a consistent view of the world. Information that one process receives in a given round might not reach another process in that round



(a) An execution graph and i's view $\mathsf{V}_i(k+1)$.

(b) Sets $G_i(k)$ & $B_i(k)$ are based on $\mathsf{V}_i(k+1)$.

Figure 1: An execution graph and its use in ConCon.

if the sender fails to deliver it to the second process. A vast literature shows that agreeing on even the value of a single bit is a nontrivial task in such a setting. The continuous consensus task aims to achieve more than this. It strives to make all interesting events part of a shared core of information, while continuously maintaining the consistency of this core across all nonfaulty processes. The early work on simultaneous agreement [3] allow a solution in which every monitored event triggers an agreement process that results in its inclusion into the shared core. The analysis of common knowledge carried out in [3, 4, 11] demonstrated that it is possible to compute the core without need for separate treatment of every input item. In fact, a core can be obtained using the subtle fixed-point construction of [4]. What we shall show is that ideas in the spirit of the simpler analysis of simultaneous agreement in crash failures can be simplified even more and extended to the more general problem of continuous consensus both in the crash and in the omission failure models.

The analysis of simultaneous agreement in the crash failure model in [3] showed that the problem reduces to computing when initial values become common knowledge. For the FIP with crash failures, this question has a particularly nice structure. Very roughly speaking, a *clean* round is one in which no process is discovered as faulty for the first time. In a precise sense, initial values become common knowledge once the existence of a clean round is common knowledge. A faulty process can keep at most one round from being clean by crashing. Hence, if process i that knows of f failures at time k, then it knows that a clean round will happen no later than time $h_i(k) = k+t+1-f$ rounds. We think of $h_i(k)$ as i's *horizon* at time k. Based on the properties of the crash-failure model it is possible to show that there will be a *critical* round c which is clean, and after which all nonfaulty processes can predict the same horizon. Moreover, once time $\ell = h_i(c)$ is reached, every nonfaulty process can detect that c was the critical round (for ℓ). This makes it common knowledge that round c was clean, allowing an efficient solution to simultaneous agreement. The analysis and proofs make strong use of the fact that for crash failures, if a process crashes in round k then all processes know of this in FIP by time $k+1$ at the latest. The omission model is significantly more complex, since information about failures can be kept for a while by faulty processes, and then delivered to nonfaulty processes much later on. Thus, in the omission model the information about failures behaves in a much more erratic fashion.

We shall now present a protocol that solves continuous consensus and operates in a fashion that resembles the intuition provided above. In every round $k+1$, each process will compute a value it considers its horizon for time k, which will be a time at which a particular part of its current view of the first k can be added to the shared core. (As we shall show later, this view will indeed become common knowledge.) The horizons predicted by different processes at the end of round $k+1$ need not be the same. When time ℓ is reached, however, the *latest* k whose horizon is ℓ will in fact be identical for all nonfaulty processes. This, in turn, will uniquely determine a shared core that can be used by the consensus task.

The crux of our construction depends on finding a replacement for the role played in the crash model for the number of failures known to i. This is suitably generalized by the following two definitions of sets of *good* and *bad* processes for i with respect to time k. The first set, which we denote by $G_i(k)$, consists of the processes that i does not know at time $k+1$ to be faulty:

$$G_i^r(k) \quad \triangleq \quad \{\ j\ : (R,r,k+1) \models \neg K_i(j \text{ is faulty})\ \} \tag{1}$$

As usual, we drop the superscript r from terms when it is clear from context. Notice that $G_i(k)$ is defined based on i's knowledge at the end of the following round $k+1$. In the crash failure model, $G_i(k)$ is the set of processes that i receives messages from in round $k+1$, while in the omissions model it is a possibly strict subset of these processes. The G stands for *good*. Based on $G_i(k)$, we define a second set, denoted by $B_i(k)$ (for *bad*), consisting of processes whose faultiness is distributed knowledge at time k among the members of the first set $G_i(k)$:

$$B_i^r(k) \quad \triangleq \quad \{\ j\ :\ (R,r,k) \models D_{G_i(k)}(j \text{ is faulty})\ \} \tag{2}$$

We denote by $b_i(k)$ the cardinality $|B_i(k)|$ in the current run. Since $B_i(k)$ consists of faulty processes, we will always have $b_i(k) \leq t$. Figure 1(b) illustrates the sets $B_i(k)$ and $G_i(k)$ in the FIP based on a particular execution graph.

The ConCon protocol, shown in Figure 2, is run by each process i individually in order to compute a shared view of the run at any given time ℓ. The exact same protocol can be used in either the crash or the omission protocol (although knowledge will be determined with respect to a different system R in either case). In every round $k+1$ of this protocol, every process performs two tasks: One is to update a current estimate (upper bound) for when the view $\mathsf{V}_{G_i(k)}(k)$ of $G_i(k)$ will be part of the shared core view. The other is to determine the shared core at time $k+1$, which is at the end of the round. This consists of reading the critical round c for time $k+1$ which is found in $Estimated_i[k+1]$. The shared core is then defined to consist of the input events in $\mathsf{V}_{G_i(c)}(c)$.

```
ConCon($i$)
    $Estimated[m] \leftarrow -1$ for all $m \geq 1$
    for $k \geq 0$ in round $k+1$
1       do send local state and receive messages according to FIP
2          $horizon \leftarrow k+1+t-b_i(k)$                    ▷ denoted $horizon_i(k)$
3          $Estimated[horizon] \leftarrow k$
4          $c \leftarrow Estimated[k+1]$                  ▷ critical round denoted $crit_i(k+1)$
5          $M_i[k+1] \leftarrow \begin{cases} \lambda & \text{if } c = -1 \\ \mathsf{V}_{G_i(c)}(c) \restriction E & \text{otherwise.} \end{cases}$
```

Figure 2: The ConCon protocol for process i.

In the protocol, each process performs the same set of actions in every round. Round $k+1$ starts at time k and ends at $k+1$. The first part of each round's computation consists of communicating according to the full-information protocol on line 1. In the next part, consisting of lines 2 and 3, process i computes an upper bound on the time at which a view of time k will be common knowledge. Finally, on lines 4 and 5 it records in $M_i[k+1]$ the view that is common knowledge at time $k+1$. Recall that the value of $b_i(k)$ is easily computable by i from its view (or local state) after the communication phase of round $k+1$. Process i can therefore perform the assignment of the value of $horizon_i(k)$ on line 2. We denote by $horizon_i(k)$ the value of $horizon$ that process i sets in round $k+1$ on line 2, and by $crit_i(k+1)$ the value of $crit$ that it sets on line 4.

By definition (see line 2), $horizon_i(m) = m+1+t-b_i(m)$. Since no failures are known initially, we have that $b_i(0) = 0$ and $horizon_i(0) = t+1$. The fact that the protocol communicates according to the full-information protocol, and that the number of failures is bounded from above by t, implies that $b_i(m) \leq b_i(m+1) \leq t$ for all m. This immediately implies for all m that both

(i) $horizon_i(m) \geq m+1$, and

(ii) $horizon_i(m+1) \leq horizon_i(m)+1$.

Notice from (ii) that the horizon cannot move forward at a fast pace. Observe, however, that it is possible for $horizon_i(m+1) < horizon_i(m)$ to hold if $b_i(m+1) > b_i(m)+1$, which can result from several failures being discovered by processes trusted by i in round $m+1$.

We now state very two useful properties of CONCON. The first says that the horizon is an upper bound on the time at which current round information is contained in the shared core. Indeed, given point (i) above this will imply that every round's information will become common knowledge within a fixed bound of roughly $t-f$ rounds, where f is the number of failures discovered. The second part proves that once the core is not empty, it grows by at least one round in every time step. Moreover, every round is assigned a critical round. This last fact is not immediate from the structure of CONCON.

Proposition 1 For all nonfaulty processes i and times m and ℓ:

(a) if $horizon_i(m) \leq \ell$ then $crit_i(\ell) \geq m$, and

(b) if $crit_i(\ell) \neq -1$ then $crit_i(\ell) < crit_i(\ell+1)$.

Proofs of Proposition 1 and of the upcoming Theorem 2 appear in the appendix. So far we have looked at the properties of the protocol as executed by a single nonfaulty process in isolation. The correctness of the algorithm depends on the relationship between executions of different processes in the same run. The following theorem shows the main correctness claim for CONCON, namely that all cores agree at all times.

Theorem 2 *The* CONCON *protocol solves the continuous consensus problem.*

More efficient implementations. The CONCON protocol sends messages according to FIP, which grow monotonically with time. We can, however, derive a more space- and communication-efficient implementation of CONCON. In order to compute $G_i(k)$ in round $k+1$, process i must know who is faulty according to $\mathsf{V}_i(k+1)$. Similarly, to compute $b_i(k)$ it needs to know the sets $G_j(k-1)$ for all $j \in G_i(k)$. All of the relevant information is accessible provided that in every round k, each process i sends $G_i(k-1)$ to all others. This can be a string of n bits. Process i would then know that z is faulty exactly if either z failed to deliver a message to i or some process j informed i that z is faulty. We can show that the sets $G_i(k)$ are the same under such a scheme as they are using FIP. In addition, it is needed that all information about monitored events from E be passed to everyone. If there are typically only few interesting events (e.g., fire alarms) then representing the relevant data regarding them may be succinct. Finally, we can see from CONCON that the protocol never requires historical data from more than $t+1$ rounds back. Indeed, as failures are discovered fewer rounds of history need to be kept track of. The result is that a solution to continuous consensus that is equivalent to ConCon exists that sends small messages and maintains bounded amount of state beyond the shared core $M_i[k]$.

4. Continuous Consensus and Common Knowledge

We have developed the ConCon protocol using intuitions obtained from the analysis of common knowledge in fault-prone systems. The continuous consensus application is in a precise sense very close to the problem of computing common knowledge. We now formalize this connection, and make use of it in order to prove an optimality result for ConCon.

The continuous consensus problem is specified in terms of the behavior of the nonfaulty processes, and does not require correct action from faulty ones. It was shown in [4] (see also [11, 5]) that the appropriate variant of common knowledge corresponding to such a situation is common knowledge among the *nonfaulty* processes, for which our language has the operator C_N. A semantic definition of satisfaction for C_N is given as follows.[2] We say that two points (r,m) and (r',m) are *N-neighbors*, and write $(r',m) \sim_N (r,m)$, if there is some process j that is nonfaulty in both r and r' for which $r'_j(m) = r_j(m)$. The point (r',m) is *N-reachable from* (r,m) in R, if there is a finite sequence of points $(r,m) = (r^0,m),(r^1,m),\ldots,(r^k,m) = (r',m)$ such that $(r^\ell,m) \sim_N (r^{\ell+1},m)$ holds for every $0 \le \ell < k$. Thus, N-reachability is the transitive closure of the $\sim_N$ relation. Moreover, it is an equivalence relation that defines a partition over the points of a system R. Common knowledge to the nonfaulty processes is then formalized by:

$$(R,r,m) \models C_N\varphi \quad \text{iff} \quad (R,r',m) \models \varphi \text{ for all points } (r',m) \text{ that are } N\text{-reachable from } (r,m) \text{ in } R\ .$$

This variant of *common knowledge among the nonfaulty processes* turns out to be the appropriate notion of knowledge to consider in applications such as continuous consensus, which are defined only in terms of the actions of nonfaulty processes. We say that a formula φ is *valid in* R, and write $R \models \varphi$, if $(R,r,m) \models \varphi$ for all points (r,m) with $r \in R$. We can now show a strong connection between common knowledge and continuous consensus:

Proposition 3 Let P be a protocol for continuous consensus and let R_P be the set of all runs of P with execution graphs in $R = R(n,t,fm,\Sigma,E)$. Fix a possible state X of the core $M_i[k]$ under P and let "Core $= X$" be a proposition that is true at a point (r,m) of R_P exactly if $M_i^r[m] = X$ for all nonfaulty processes i in r. Then

$$R_P \models \ (\text{Core} = X) \equiv C_N(\text{Core} = X)\ .$$

Proof: Assume that $M_i^r[m] = X$, where i is nonfaulty in the execution graph of r. Define the proposition q as "Core $= X$". It suffices to show that, for all $r,r' \in R$, if $(R,r,m) \models q$ and $(r,m) \sim_N (r',m)$ then $(R,r',m) \models q$, and the claim will follow by induction. Assume that $(R,r,m) \models q$ and $(r,m) \sim_N (r',m)$. Thus, $r_j(m) = r'_j(m)$ holds for some nonfaulty process j in both runs. Since $(R,r,m) \models q$ we have that Core $= X$ holds at (r,m) and since j is nonfaulty there it follows that, in particular, $M_j^r[m] = X$. From $r_j(m) = r'_j(m)$ we have that $M_j^{r'}[m] = X$ as well. Finally, since P solves continuous consensus, we have by Consistency that $M_i^{r'}[m] = X$ for all nonfaulty i in r', and hence $(R,r',m) \models q$. ■

Proposition 3 implies that the contents of the core in any protocol P for continuous consensus (e.g., ConCon) is common knowledge at any given instant. Hence, an event can be entered into the local copies of

[2]The fact that in our systems a process can always distinguish between points (r,m) and (r',m') with $m \neq m'$ simplifies the definitions here slightly.

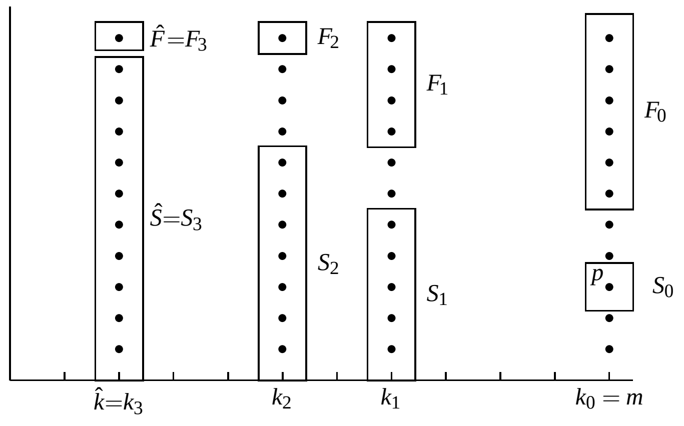


Figure 3: An instance of Moses and Tuttle's fixed-point construction by p at (r,m).

the core only once its occurrence has become common knowledge. We shall now argue that the ConCon protocol places events in the cores $M_i[m]$ as early as possible. More formally, we are setting out to prove the following

Theorem 4 *Let R be a* FIP *system, let $r \in R$ and assume that i is nonfaulty in r. Let P be a correct protocol for continuous consensus. Finally, denote i's core at (r,m) under* ConCon *by $M_i^{\mathsf{C}}[m]$ and its core under P by $M_i^{\mathsf{P}}[m]$. Then $M_i^{\mathsf{P}}[m] \subseteq M_i^{\mathsf{C}}[m]$ holds for all times m.*

Theorem 4 shows that ConCon is optimal in terms of recording events as early as possible in the core. We will prove the theorem by showing that the core $M_i^{\mathsf{C}}[m]$ produced by ConCon is precisely the view of the run that is common knowledge at (r,m). In [4] Moses and Tuttle completely characterized the connected components of the N-reachability relation systems for FIP in crash and omission models, thereby characterizing common knowledge as well. To set up the necessary background for the proof, we now briefly review their fixed-point construction and related characterization of common knowledge. The construction is performed individually by every process p based on its view $\mathsf{V}_p(m)$ at a given point (r,m). It defines a sequence of pairs (k_i,S_i) consisting of a time and set of processes, for $i \geq 0$. In the construction, F_i denotes the set $\{\ j\ :(R,r,k_i) \models D_{S_i}(j \text{ is faulty})\ \}$ of processes known at time k_i to be faulty by processes in S_i. The construction proceeds inductively as follows.

Base: Set $k_0 = m$ and $S_0 = \{p\}$.

Step: Set $k_{i+1} = m-(t+1-|F_i|)$ and $S_{i+1} = \mathbb{P} \setminus F_i$.

For nonfaulty processes p it can be shown that the F_i's are a descending chain. As a result, the S_i's are an ascending chain and the k_i's form a descending sequence. Since $|F_i| \leq t$, for some index h we must have that $F_h = F_{h-1}$. When this happens for the first time, the construction reaches a fixed-point because $S_{h+1} = S_h$ and $k_{h+1} = k_h$. We denote the first such values k_h and S_h at which a fixed-point is reached by $\hat{k} = \hat{k}(r,m)$ and $\hat{S} = \hat{S}(r,m)$, respectively. Finally, the outcome of the construction at (r,m) is given by

Output: is the view $\mathsf{V}^r_{\hat{S}}(\hat{k})$.

Figure 3 illustrates an example computation of the fixed-point construction.

The fixed-point construction is shown to characterize N-reachability relation (and hence also the common knowledge) in the crash and omission models:

Proposition 5 (Moses and Tuttle) Let r and r' be runs of a FIP system R. Let the view $\mathsf{V} = \mathsf{V}_{\hat{S}}(\hat{k})$ at (r,m) be the output of the fixed-point construction performed by a nonfaulty process at (r,m), while $\mathsf{V}' = \mathsf{V}_{S'}(k')$ is the output for some nonfaulty process at (r',m). Then (r',m) is N-reachable from (r,m) iff $\mathsf{V}' = \mathsf{V}$.

Based on this characterization, we can now present the proof of our optimality claim:

Proof of Theorem 4: Fix $r \in R$ and $m \geq 0$, and let $\mathsf{V} = \mathsf{V}^r_{\hat{S}}(\hat{k})$ be the view produced by the construction at (r,m). By proposition 5 this view is clearly common knowledge. We claim that only input events of E that appear in this view are common knowledge. This follows from the fact that the values of $\hat{k}$ and $\hat{S}$ produced by the construction depend only on the pattern of failures. Since $|E| > 1$, the inputs outside of V can be altered without changing the failure pattern. Hence, there is an N-reachable point instantiating any possible input assignment outside of V, and hence no input event outside V is common knowledge. By Proposition 3 it follows that $M^{\mathsf{P}}_i \subseteq \mathsf{V} \upharpoonright E$ holds for all continuous consensus protocols P, including ConCon. We will complete the argument by showing that $\mathsf{V}_{G(c)}(c) = \mathsf{V}$ for $c = crit^r_p(m)$, which implies that $M^{\mathsf{C}}_i[m] = \mathsf{V} \upharpoonright E$. We do this by considering p's execution in ConCon during the same run r. It suffices to show that $horizon_p(k_h) \leq m$ because by Proposition 1(a) we then have that $crit^r_p(m) \geq k_h$ and hence $\mathsf{V}_{G(c)}(c) \supseteq \mathsf{V}$ and we are done.

Assume that $\hat{k} = k_h$ and $\hat{S} = S_h$ are the limit values at the fixed-point of p's construction at (r,m). Notice that since $t \leq n-2$, necessarily $h > 0$. Observe that the sets F_i in the fixed-point construction only ever contain faulty processes. Since $p \in S_0$ and p is nonfaulty, it follows that $p \in S_i$ for every $i \leq \hat{h}$. In particular, $p \in S_{h-1}$. Since by definition, every proces known to be faulty by members of S_{h-1} at time $k_{h-1} > k_h$ is included in F_{h-1}, and since by construction $S_h = \mathbb{P} \setminus F_{h-1}$, it follows that $S_h \subseteq G_p(k_{h-1}-1)$, where G_p is the set of *good* processes according to p in ConCon. From $k_h < k_{h-1}$ we have that $k_h \leq k_{h-1} - 1$. The perfect recall property of the FIP implies that the sets $G_p(k)$ are monotonically decreasing in the weak sense. Thus, $S_h \subseteq G_p(k_{h-1}-1) \subseteq G_p(k_h)$. It follows that $B_p(k_h) \supseteq F_h$ and hence clearly also $b_p(k_h) \geq |F_h|$. Since $k_h = \hat{k}$ is the fixed, we have that $k_h = m - (t+1-|F_h|)$ and hence $m = k_h + t + 1 - |F_h|$. By definition of ConCon, $horizon_p(k_h) = k_h + t + 1 - b_p(k_h)$. Since $b_p(k_h) \geq |F_h|$ we obtain that $horizon_p(k_h) \leq m$ and we are done. ■

5. Uniform Continuous Consensus

The continuous consensus problem specifies constraints only on the cores of nonfaulty processes, guaranteeing nothing about faulty ones. Observe, however, that failures in our models are not considered malicious—there is no lying and faulty behavior is closely related to crashing or communication malfunction. It is thus natural to consider a stronger version of the problem, which could be called *uniform continuous consensus* (*UCC*), in which we require the Accuracy, Consistency, and Completeness properties of continuous consensus to hold for *all* processes i and j, whether faulty or nonfaulty. One can expect a solution to UCC to be similar in style and quality to ConCon, because the failures we consider only affect the ability of a process to send messages, while even faulty processes receive all incoming messages. In this section we consider how the ConCon protocol can be modified to obtain UniConCon, an optimal protocol for UCC.

As shown in [2, 5], simultaneously consistent behavior by all participants is closely related to (standard) common knowledge, that is, the traditional notion equivalent to an infinite conjunction of "everybody knows". Using the operator C for common knowledge among all processes, we briefly review the semantic definition of $C\varphi$. We say that the point (r',m) is *reachable* from (r,m) in R if there is a finite sequence of

```
UNICONCON(z)
    Estimated[m] ← −1 for all m ≥ 1
    for k ≥ 0 in round k+1
1       do send local state and receive messages according to FIP
           if $b_z(k) = t$
2             then $Estimated[k+1] \leftarrow k$
3             else $chosen[k] \leftarrow j$ for an arbitrary $j \in G_z(k)$
                  if $b_j(k-1) = t-1$
4                    then $Estimated[k+1] \leftarrow k-1$
5                    else $horizon \leftarrow (k-2)+1+t-b_j(k-2)$
6                         $Estimated[horizon] \leftarrow k-2$
7          $c \leftarrow Estimated[k+1]$                    ▷ critical round denoted $crit_z(k+1)$
8          $M_z[k+1] \leftarrow \begin{cases} \lambda & \text{if } c=-1 \\ \bigvee_{G_z(k)}(k) \restriction E & \text{if } c=k \\ \bigvee_{G_{chosen[k]}(c)}(c) \restriction E & \text{otherwise} \end{cases}$      ▷ $t$ failures discovered
```

Figure 4: Process z's computation in UNICONCON.

points of R $(r,m) = (r^0,m),(r^1,m),\ldots,(r^k,m) = (r',m)$ such that for every $0 \le \ell < k$ there is some $j = j_\ell$ for which $r^\ell_j(m) = r^{\ell+1}_j(m)$. Then

$$(R,r,m) \models C\varphi \quad \text{iff} \quad (R,r',m') \models \varphi \text{ for all points } (r',m') \text{ that are reachable from } (r,m) \text{ in } R\ .$$

Common knowledge and UCC are related in the same way as C_N and continuous consensus are, and a completely analogous result to Proposition 3 holds. An interesting result by Neiger and Tuttle in [11] shows that in our settings the two notions of common knowledge coincide:

Theorem 6 (Neiger and Tuttle) $R \models (C\varphi \equiv C_N\varphi)$ *holds for every formula* φ *and* FIP *system R.*

Intuitively, this can be explained by the fact that the failures considered in these models only affect the ability of a process to send messages, while even faulty processes receive all incoming messages. Consequently, one can expect a solution to UCC to be similar in style and quality to CONCON. A natural question in light of Theorem 6 is whether CONCON itself solves the UCC problem. Unfortunately, it does not. The problem in using CONCON by a faulty process z is that z might know of failures at time k that it does not pass on to nonfaulty processes. If $z \in G_z(k)$, then these failures will be counted in $b_z(k)$ and will therefore play a role in determining $horizon_z(k)$. If z is silent from round $k+1$ on, for example, then these failures might never figure in the calculations of other processes. An inconsistency between z's core and those of the nonfaulty processes will arise as a result.

The intuition behind the UNICONCON protocol is based on the following. Whether or not a process z is faulty, it is still guaranteed in our models to receive all messages that are sent to it. In addition, for any pair of processes j and z, if $j \in G_z(k)$ then no nonfaulty process has discovered that j is faulty by time k. The information available to j at time $k-1$ is thus transmitted to the (truly) nonfaulty processes in round k, and they are guaranteed to relay it to z in round $k+1$. Roughly speaking, in order to obtain an algorithm for UCC we have a process z choose an arbitrary member j of $G_z(k)$ in round $k+1$ (which we denote by *chosen*$[k]$) and simulate what j would have done two rounds earlier in ConCon.[3] This works, except if the nonfaulty processes detect all t of the faulty processes, and so the horizon becomes one round ahead of the current time (i.e., $horizon_z(k) = k+1$), then such simulation would be too late to be timely. This is easily detectable even by a faulty process, however, and is therefore treated as a separate case. The algorithm is given in Figure 4. The underlying lemmas used in the proof of correctness of UniConCon are given in Section B. Using them, the following theorem summarizes the correctness claim. We use superscripts u and c to denote cores computed in the in UniConCon and ConCon protocols, respectively.

Theorem 7 *Fix a run r in a* FIP *system R. Then*

1. $M^u_y[m] = M^u_z[m]$ *holds for all* $m \geq 1$ *and every pair* y, z *of processes, and*
2. $M^u_i[m] = M^c_i[m]$ *holds for all* $m \geq 1$ *and every nonfaulty process* i *in* r.

An immediate corollary of Theorems 6 and 7(2) is that UniConCon is an optimal solution of UCC.

6. Conclusions

This paper considered the continuous consensus problem, which generalizes simultaneously consistent action in fault-prone systems. Two variants were considered and solved, one in which consistency is required only among nonfaulty processes and the other being the uniform variant ensuring consistency among arbitrary pairs of processes. A striking aspect of the solutions is their simplicity: At every round, each process updates a single value based on a straightforward computation. Moreover, while the solutions are stated in the context of the full-information protocol, they can be implemented in a more efficient manner. The problems are closely related to that of computing what is common knowledge at any given point. The optimality of the solutions is shown using the characterization of common knowledge given in [4] and the equivalence shown in [11].

This paper bridges a gap between the analysis of common knowledge with crash failures in [3] and that for omission failures in [4]. In the case of crash failures, clean rounds are rounds in which no new failures are discovered. As shown in [3], clean rounds play an important role in the analysis of common knowledge there. The situation in the omission model seemed less transparent. The fixed-point construction of [4] did not suggest a similar notion. Indeed, as we discussed in Section 3, there can be an arbitrary delay between the time a failure is first discovered in this model and the point at which this information may affect the set of facts that are common knowledge. The connection between fault discovery and common knowledge seemed much less clear. The ConCon and UniConCon protocols presented in this paper, do, however, suggest a natural generalization of clean rounds to the omission model. We can define round $k+1$ to be *clean* in this case if no nonfaulty process i discovers that $b_i(k) > b_i(k-1)$. Thus, it is not the discovery of a failure in a round that makes it dirty; rather, a round is dirty if, for some process z, it is the first round in which z's failure is reported by a process that is trusted by some nonfaulty process. It can be shown that if $crit_i(m) \neq -1$ then the identity of $crit_i(m)$ and the fact that it is a clean round, become common knowledge at time m.

Finally, our algorithms shed an interesting light on the distinction between evidence supplied by processes that are known to be faulty and ones that are not. Recall that failures in the models we considered are benign. No process ever deviates from the protocol by sending incorrect messages. Thus, every piece of information obtained from a process can be trusted. Nevertheless, the computation of the horizon by process i in round $k+1$, and thus ultimately the times at which common knowledge is obtained, depends only the set $B_i(k)$. Thus, despite the fact that information from faulty processes is correct, this central

[3]It would be ok to have a process choose itself for as long as it does not know of its own faultiness.

computation considers only failures reported by potentially nonfaulty processes. This distinction seems an essential aspect of the evolution of common knowledge over time in these models.

Acknowledgement We wish to thank Mark Tuttle for discussions on the topic of this paper, and Kai Engelhardt for useful comments that improved its presentation.

References

[1] D. Lewis, *Convention: A Philosophical Study*. Cambridge, Mass.: Harvard University Press, 1969.

[2] J. Y. Halpern and Y. Moses, “Knowledge and common knowledge in a distributed environment,” *Journal of the ACM*, vol. 37, no. 3, pp. 549–587, 1990.

[3] C. Dwork and Y. Moses, “Knowledge and common knowledge in a Byzantine environment: crash failures,” *Information and Computation*, vol. 88, no. 2, pp. 156–186, 1990.

[4] Y. Moses and M. R. Tuttle, “Programming simultaneous actions using common knowledge,” *Algorithmica*, vol. 3, pp. 121–169, 1988.

[5] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi, *Reasoning about Knowledge*. Cambridge, Mass.: MIT Press, 1995.

[6] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[7] G. Berry and G. Gonthier, “The esterel synchronous programming language: Design, semantics, implementation,” *Science of Computer Programming*, no. 19, pp. 87–152, 1992.

[8] N. Halbwachs, *Synchronous Programming of Reactive Systems*. Kluwer Academic Publisher, 1993.

[9] L. Lamport, “The part-time parliament,” vol. 16, no. 2, pp. 133–169, 1998.

[10] B. Coan, “A communication-efficient canonical form for fault-tolerant distributed protocols,” in *Proc. 5th ACM Symp. on Principles of Distributed Computing*, pp. 63–72, 1986.

[11] G. Neiger and M. R. Tuttle, “Common knowledge and consistent simultaneous coordination,” in *Proceedings of the 4th international workshop on Distributed algorithms*, pp. 334–352, Springer-Verlag New York, Inc., 1991.

A. Proofs for ConCon

This appendix consists of the proofs of correctness of the ConCon algorithm. We start with the following technical lemma based on observations (i) and (ii) that appear after the ConCon protocol. This lemma serves to simplify later proofs.

Lemma 8 Let $k < k'$, and assume that $horizon_i(k) \leq \ell < horizon_i(k')$. Then some time $\hat{k}$ where $k \leq \hat{k} < k'$ satisfies both (a) $horizon_i(\hat{k}) = \ell$ and (b) $horizon_i(\hat{k}+1) = \ell + 1$.

Proof: Assume that $k < k'$ and $horizon_i(k) \le \ell < horizon_i(k')$. By observation (ii) the function $horizon_i$ can advance only in steps of one. Hence, $horizon_i(k'') = \ell$ for some intermediate time $k \le k'' < k'$, establishing part (a) of the claim. Let $\hat{k} = \max\{k'' : k'' < k' \text{ and } horizon_i(k'') = \ell\}$. We claim that $horizon_i(\hat{k}+1) > horizon_i(\hat{k})$. Assume otherwise, by way of contradiction. If $horizon_i(\hat{k}+1) \le horizon_i(\hat{k})$ then $horizon_i(\hat{k}+1) \le \ell < horizon_i(k')$. From $\hat{k} < k'$ we have that $\hat{k}+1 \le k'$ and since in addition we have that $horizon_i(\hat{k}+1) \ne horizon_i(k')$ it follows that $\hat{k}+1 < k'$. We can now apply part (a) to $\hat{k}+1 < k'$ we obtain that $horizon_i(h) = \ell$ for some h satisfying $\hat{k}+1 \le h < \ell$, contradicting the maximality of $\hat{k}$. Since $horizon_i(\hat{k}+1) > horizon_i(\hat{k})$, we have by property (ii) that $horizon_i(\hat{k}+1) = horizon_i(\hat{k})+1$ and we are done. ■

Proof of Proposition 1: For part (a), assume that $horizon_i(m) \le \ell$. Since $horizon_i(\ell) > \ell$ we have by Lemma 8 (a) that $horizon_i(\hat{k}) = \ell$ for some $\hat{k}$ that satisfies $m \le \hat{k} < \ell$. In particular, $Estimated_i[\ell] = \hat{k} \ge m$ after line 3 is executed in round $\hat{k}+1$. The fact that $\hat{k} < \ell$ implies that $\hat{k}+1 \le \ell$. Moreover, since the value of $Estimated_i[\ell]$ is monotonic nondecreasing in time, it follows that $Estimated_i[\ell] \ge \hat{k}$ holds when line 4 is reached in round ℓ. It now follows by line 4 that $crit_i(\ell) \ge \hat{k} \ge m$ and we are done with (a). For part (b), assume that $crit_i(\ell) = m \ne -1$. Then from line 2 we have that $horizon_i(m) = \ell$. Since $horizon_i(\ell) > \ell$ we have by Lemma 8(b) that there exists $\hat{k}$ satisfying $m \le \hat{k} < \ell$ such that $horizon_i(\hat{k}+1) = \ell+1$. Applying part (a) we obtain that $crit_i(\ell+1) \ge \hat{k}+1$. Since $m \le \hat{k}$ we have that $\hat{k}+1 > m$ and thus $crit_i(\ell+1) > m = crit_i(\ell)$, which completes the proof. ■

Proof of Theorem 2: Since $M_i[\ell]$ is a view of the run, all events in $M_i[\ell]$ have occurred, and thus the Accuracy property holds. Completeness requires every event $e \in E$ occurring at a nonfaulty j to eventually appear in $M_i[\ell]$. Suppose that e occurs no later than time k. Notice that necessarily $j \in G_i(k)$ for every nonfaulty i, since j is nonfaulty. By definition, $horizon_i(k) \le k+t+1$, and Proposition 1(a) implies that $crit_i(k+t+1) \ge k$. It follows that the occurrence of e will appear in $M_i[k+t+1]$ and we have Completeness.

Finally, for Consistency, we need to show that $M_i[\ell] = M_j[\ell]$ holds for all times $\ell \ge 0$ and nonfaulty processes i and j. The variable $crit_i(\ell)$ is assigned the value of $Estimated_i[\ell]$ in round ℓ by line 4. Lines 0 and 3 guarantee that $Estimated_i[m] \ge -1$ and $Estimated_j[m] \ge -1$ holds for all indices m at all times. It follows that $crit_i(\ell) \ge -1$. We distinguish two cases. First consider the case in which $crit_i(\ell) = crit_j(\ell) = -1$. In this case we have by line 5 and $crit_i(\ell) = crit_j(\ell) = -1$ that $M_i[\ell] = M_j[\ell] = \lambda$ as desired. Second, suppose without loss of generality that $m = crit_i(\ell) \ne -1$. We claim that $G_j(m) \supseteq G_i(m)$ and $crit_j(\ell) \ge crit_i(\ell)$. If $G_j(m) \not\supseteq G_i(m)$, then j knows at time $m+1$ of some $z \in G_i$ that is faulty. Since j is nonfaulty, $j \in G_i(m+1)$, and hence $z \in B_i(m+1)$ so that $b_i(m+1) > b_i(m)$ and $horizon_i(m+1) \le horizon_i(m) = \ell$. Proposition 1(a) implies $crit_i(\ell) \ge m+1$, contradicting the assumption that $crit_i(\ell) = m$. From the fact that $G_j(m) \supseteq G_i(m)$ it follows that $\mathsf{V}_{G_i(m)}(m)$ is contained in $\mathsf{V}_{G_j(m)}(m)$ and hence $B_j(m) \supseteq B_i(m)$. This implies that $b_j(m) \ge b_i(m)$ and thus $horizon_j(m) \ge horizon_i(m)$. Again by Proposition 1(a) we can conclude that $crit_j(\ell) \ge m = crit_i(\ell)$ and the claim is established. Moreover, since $crit_i(\ell) > -1$, it follows that $crit_j(\ell) \ne -1$. Applying the above claim with respect to j instead of i, we obtain also that $G_j(m) \subseteq G_i(m)$ and $crit_j(\ell) \le crit_i(\ell)$. We thus have that $crit_i(\ell) = crit_j(\ell)$ and that $G_i(m) = G_j(m)$. Finally, from the fact that $G_i(m) = G_j(m)$ we have by line 5 that $M_i[\ell] = \mathsf{V}_{G_i(m)}(m) = \mathsf{V}_{G_j(m)}(m) = M_j[\ell]$ and we are done. ■

B. Correctness of UniConCon

In this section we provide the technical lemmas used in the proof of Theorem 7.

Lemma 9 Fix a run of the FIP, and let z be an arbitrary process in this run. Then (i) $\mathsf{V}_z(k+1)$ contains $\mathsf{V}_{G_z(k)}(k)$ for every $k \ge 0$, and (ii) $\mathsf{V}_z(k+1)$ contains $\mathsf{V}_{G_j(m)}(m)$ for every $m \le k-1$ and $j = chosen[m]$.

Proof: By definition, the view of a process grows monotonically with time, so whatever is contained in such a view at a given time will be contained in it at all later times. According to the FIP, every process is required to send a message to all processes in every round. Moreover, the crash and omission failure models are such that if a message from y to z does not arrive then y must be faulty. Hence, in particular, y will no longer be in G_z following such an event. It follows that $\mathsf{V}_z(k+1)$ contains $\mathsf{V}_y(k)$ for every $y \in G_z(k)$. This immediately implies part (i). For part (ii), let $m \leq k-1$ and assume that $j = chosen[m]$. By part (i) we have that $\mathsf{V}_j(m+1)$ contains $\mathsf{V}_{G_j(m)}(m)$. Since $j \in G_z(k+1)$ we have that $\mathsf{V}_z(k+1)$ contains $\mathsf{V}_j(k)$. Finally, since $m \leq k-1$ we have that $m+1 \leq k$ and by monotonicity of $\mathsf{V}_j(\cdot)$ we have that $\mathsf{V}_j(k)$ contains $\mathsf{V}_j(m+1)$ which, by part (i), contains $\mathsf{V}_{G_j(m)}(m)$. Part (ii) now follows. ■

Lemma 10 Fix a run of FIP and let y, z be arbitrary processes. If $b_z(k) = t$ then $G_y(k) = G_z(k)$ and $b_y(k) = t$.

Proof: Assume that $b_z(k) = t$. This means that the processes in $G_z(k)$ have distributed knowledge that all faulty processes are already recorded in $B_z(k)$, and everyone in $G_z(k)$ is correct. Being correct, they all succeed in sending messages to y. Finally, since their joint view implies that they are (precisely) the set of nonfaulty processes, it follows that $G_y(k) = G_z(k)$. Since $b_z(k)$ and $b_y(k)$ are a function of $G_y(k)$ and $G_z(k)$, we also have that $b_z(k) = b_y(k)$. ■

Lemma 11 Fix a run of FIP and let y, z be arbitrary processes. Moreover, let $k+1 \geq 2$ and let $j \in G_z(k)$ and $h \in G_y(k)$. If $b_z(k) \neq t$ and $b_j(k-1) = t-1$ then $G_j(k-1) = G_h(k-1)$.

Proof: Observe that if $x \in B_x(k-1)$ then $\mathsf{V}_x(k)$ contains proof that x is faulty. Any process z that receives a message from x in round $k+1$ obtains proof that x is faulty. Moreover, if z does not receive a message from x in round $k+1$ it, too, knows that x is faulty. Hence, if $x \in B_x(k-1)$ then necessarily $x \notin G_z(k)$ for all processes z. Let $j \in G_z(k)$ and $h \in G_y(k)$ and assume that $b_j(k-1) = t-1$ while $b_z(k) \neq t$. Assume by way of contradiction that $G_j(k-1) \neq G_h(k-1)$. We consider two cases.

- Assume that there is a process $x \in G_j(k-1) \setminus G_h(k-1)$. Thus, x is a faulty process not in $B_j(k-1)$. Since $b_j(k-1) = t-1$ it follows that the set of faulty processes is precisely $B_j(k-1) \cup \{x\}$. Consider following possibilities regarding the correctness of j and h:
  - j is nonfaulty: in this case $j \in G_y(k)$ and hence $B_y(k) \supseteq B_j(k-1)$ and by assumption $h \in G_y(k)$, which implies that $x \in B_y(k)$ as well. It follows that $b_y(k) = t$. By Lemma 10 this implies $b_z(k) = t$, contradicting the assumption.
  - j is faulty: By the above observation, $j \in G_z(k)$ implies that $j \in G_j(k-1)$ (and thus $j \notin B_j(k-1)$). The fact that $b_j(k-1) = t-1$ implies that there is at most one faulty process not in $B_j(k-1)$ thus necessarily $j = x$ for the assumed $x \in G_j(k-1) \setminus G_h(k-1)$. It follows that h knows at time k that j is faulty. If h is nonfaulty, then $h \in G_z(k)$ and we obtain that $B_z(k) \supseteq B_j(k-1) \cup \{x\}$ and thus $b_z(k) = t$ again contradicting the assumption. The remaining possibility is that both j and h are faulty. Clearly $j \neq h$ since $G_j(k-1) \neq G_h(k-1)$. Since $B_j(k-1)$ contains all faulty processes except for j, it follows that $h \in B_j(k-1)$. Moreover, all members of $G_j(k-1)$ except for j must be nonfaulty. Let i be a nonfaulty process. We claim that $\mathsf{V}_i(k)$ must contain $\mathsf{V}_x(k-1)$ for every $x \in G_j(k-1)$. Clearly $j \in G_i(k-1)$ since otherwise i's round $k+1$ message to z, which is guaranteed to dbe delivered because i is nonfaulty, would prove that j is faulty, contradicting the fact that $j \in G_z(k)$. Moreover, i must receive messages from all other members of $G_z(k-1)$, since they are all nonfaulty. It follows $\mathsf{V}_i(k)$ contains $\mathsf{V}_{G_j(k-1)}(k-1)$. From $h \in B_j(k-1)$ it thus follows that i knows at time k, based on $\mathsf{V}_i(k)$, that h is faulty. But then process y would know this after receiving i's message in round $k+1$, contradicting the assumption that $h \in G_y(k)$.

- Assume that $G_j(k-1) \subseteq G_h(k-1)$ and there is a process $m \in G_h(k-1) \setminus G_j(k-1)$. Since $G_j(k-1) \subseteq G_h(k-1)$ we have that $B_j(k-1) \subseteq B_h(k-1)$ so that $b_h(k-1) \geq t-1$. If $b_h(k-1) = t$ then since $h \in G_y(k)$ we have that $b_y(k) = t$ which is a contradiction as before. Otherwise the claim follows by the previous case with the roles of x and y reversed, since we now have $h \in G_y(k)$, $b_h(k-1) = t-1$ and $x \in G_h(k-1) \setminus G_j(k-1)$. ■

Lemma 12 Fix a run of FIP, let y,z be arbitrary processes, and let $j \in G_z(m)$ and $h \in G_y(m)$. Finally, assume that $b_j(m-2) < t-1$. If $G_j(m-2) \neq G_h(m-2)$ then $b_i(m-1) > b_j(m-2)$ will hold for some nonfaulty process i.

Proof: Fix a run, let $j \in G_z(m)$ and $h \in G_y(m)$, and fix a nonfaulty process i. Observe that from $j \in G_z(m)$ we obtain that $j \in G_i(m-1)$ and, similarly, $h \in G_y(m)$ implies that $h \in G_i(m-1)$. Assume that $b_j(m-2) < t-1$. Moreover, assume that $G_j(m-2) \neq G_h(m-2)$. We consider two cases. First assume that there is a process $x \in G_j(m-2) \setminus G_h(m-2)$. Thus, h knows that x is faulty at time $m-1$. Since $j,h \in G_i(m-1)$, we have that $B_i(m-1) \supseteq B_j(m-2) \cup \{x\}$ and thus $b_i(m-1) > b_j(m-2)$. For the second case, assume that $G_j(m-2) \subseteq G_h(m-2)$ and there is a process $x \in G_h(m-2) \setminus G_j(m-2)$. Since $G_j(m-2) \subseteq G_h(m-2)$ we have that $B_j(m-2) \subseteq B_h(m-2)$, and hence $b_h(m-2) \geq b_j(m-2)$. As in the first case, since $j,h \in G_i(m-1)$, we have that $B_i(m-1) \supseteq B_h(m-2) \cup \{x\}$. It follows that $b_i(m-1) \geq b_h(m-2)+1 \geq b_j(m-2)+1 > b_j(m-2)$ and we are done. ■

We can now finally prove the correctness of UNICONCON:

Proof of Theorem 7: For part 1 we need to show that $M_y^u[m] = M_z^u[m]$ holds for all $y,z \in \mathbb{P}$ and $m \geq 1$. Observe that Lemma 9 establishes that the views used in the assignments on Line 8 of UNICONCON are avaliable to the process running the protocol at that stage. Moreover, recall that it is an invariant of UNICONCON that of $Estimated_z[\ell] = k$ only if it was assigned this value in round $k+1$. Hence, $crit_z(m) \leq m-1$ is guaranteed for all m. Let $m \geq 1$. We consider four cases:

- Assume that $crit_z(m) = m-1$. By Line 7, $Estimated_z[m] = m-1$ at the end of round m. Since $Estimated_z[m]$ was assigned in round m, this could only happen on Line 2 of UNICONCON. Thus, $b_z(m-1) = t$ and by Lemma 10 we have that $b_y(m-1) = t$ and $G_y(m-1) = G_z(m-1)$. Line 2 of UNICONCON when executed by y in round m will thus set $Estimated_y[m] = m-1$, and by Line 8 (middle case) $M_z[m] = \mathsf{V}_{G_z(m-1)}(m-1)$ while $M_y[m] = \mathsf{V}_{G_y(m-1)}(m-1)$. Since $G_y(m-1) = G_z(m-1)$ these are the same, and the claim now follows.

- Assume that $m > 1$ and $crit_z(m) = m-2$. The argument in this case is completely analogous to the first case. It is obtained as before by substituting (i) $b_z(m-1) = t$ and $b_y(m-1) = t$ by $b_z(m-1) = t-1$ and $b_y(m-1) = t-1$, (ii) Line 4 for Line 2 and (iii) Lemma 11 for Lemma 10.

- Assume that $m > 2$ and $-1 < crit_z(m) < t-1$. By induction on $(t-1) - crit_z(m)$ we can use Lemma 12 to show that $crit_z(m) = crit_y(m)$. Denote this value by c. Then for $j = chosen_z[c]$ and $h = chosen_y[c]$, Lemma 12 also shows that $G_j(c) \neq G_h(c)$. It follows from Line 8 (bottom case) that that $M_z[m]$ and $M_y[m]$ are assigned to equal values and the claim holds.

- Assume that $crit_z(m) = -1$. By the previous case we have that $crit_y(m) = -1$ as well, so by the top case of Line 8 we have that $M_z[m] = M_y[m] = \lambda$ and we are done. can be used to shw

Part 2 of Theorem 7 follows from the fact that if process i is nonfaulty in r then $i \in G_i(m)$ for all $m \geq 0$. Thus, setting $chosen_i[k] \leftarrow i$ whenever $i \in G_i(k)$ is compatible with UNICONCON, and would yield by part 1 that $M_i^u[m] = M_z^u[m]$ for all processes z in r. It is easy to check, however, that when $chosen_i[k] = i$ for nonfaulty i, then $M_i^u[m]$ is set to the same view as $M_i^c[m]$ at all times m, and hence $M_i^u[m] = M_i^c[m]$ is guaranteed. ■