Continuous Consensus with Ambiguous Failures

Tal Mizrahi * and Yoram Moses **

Department of Electrical Engineering, Technion

Abstract. *Continuous consensus* (CC) is the problem of maintaining an identical and up-to-date core of information about the past at all correct processes in the system [1]. This is a primitive that supports simultaneous coordination among processes, and eliminates the need of issuing separate instances of consensus for different tasks. Recent work has presented new simple and efficient *optimum* protocols for continuous consensus in the crash and (sending) omissions failure models. For every pattern of failures, these protocols maintain at each and every time point a core that subsumes that maintained by any other continuous consensus protocol. This paper considers the continuous consensus problem in the face of harsher failures: general omissions and authenticated Byzantine failures. Computationally efficient optimum protocols for CC do not exist in these models if $P \neq NP$. A variety of CC protocols are presented. The first is a simple protocol that enters every interesting event into the core within $t+1$ rounds (where t is the bound on the number of failures), provided there are a majority of correct processes. The second is a protocol that achieves similar performance so long as $n > t$ (i.e., there is always guaranteed to be at least one correct process). The final protocol in the general omissions model makes use of active failure monitoring and failure detection to include events in the core much faster in many runs of interest. Its performance is established based on a nontrivial property of minimal vertex covers in undirected graphs. The results are adapted to the authenticated Byzantine failure model, in which it is assumed that faulty processes are malicious, but correct processes have unforgeable signatures. Finally, the problem of *uniform* CC is considered. It is shown that a straightforward version of uniform CC is not solvable in the setting under study. A weaker form of uniform CC is defined, and protocols achieving it are presented.

Keywords: Distributed computing, fault tolerance, consensus, continuous consensus.

1 Introduction

Fault-tolerant systems often require a means by which independent processes or processors can arrive at an exact mutual agreement of some kind. As a result, reaching consensus is one of the most fundamental problems in fault-tolerant distributed computing, dating back to the seminal work of Pease, Shostak, and Lamport [2]. When the independent processes need to reach compatible decisions *at the same time*, they often need to reach *simultaneous consensus* about particular aspects of the execution. While early

* deweastern@yahoo.com, Dept. of Electrical Engineering, Technion, Haifa, 32000 Israel.
** moses@ee.technion.ac.il, Dept. of Electrical Engineering, Technion, Haifa, 32000 Israel.

protocols in the synchronous model achieved consensus at the same round (e.g. [2]), it was noticed by Dolev et al. [3] that non-simultaneous solutions are sometimes advantageous. Later work showed that simultaneous consensus required *common knowledge* and this is a nontrivial requirement [4]. Nevertheless, the need for simultaneous decisions to be compatible is very natural in many cases: E.g., when different processes need to access distinct physical resources 'at the same time, or when one distributed algorithm ends and another one begins, and the two use similar message texts for different purposes.

We consider a synchronous message-passing system in which processes receive external inputs from the outside world at various times. Suppose that we are interested in maintaining a simultaneously consistent view regarding a set of events $\mathcal{E}$ in the system. The particular events that would be of interest is application-dependent, but it will typically record events such as inputs that processes receive at various times, values that certain variables obtain at given times, and faulty behavior in the form of failed or inconsistent message deliveries. A continuous consensus (CC) protocol maintains at all times $k \geq 0$ a *core* $M_i[k]$ of events of $\mathcal{E}$ at every site i. In every run of this protocol the following properties are required to hold, for all nonfaulty processes i and j.

Accuracy: All events in $M_i[k]$ occurred in the run.
Consistency: $M_i[k] = M_j[k]$ at all times k.
Completeness: If the occurrence of an event $e \in \mathcal{E}$ is known to process j at any point, then $e \in M_i[k]$ must hold at some time k.

Decisions performed by different correct processes in the same round can be chosen in a consistent manner if they are based on the core of a CC protocol. Indeed, once an event recording a particular value or vote enters the core, the processes automatically have simultaneous consensus regarding it. Finally, a CC protocol can replace the need for initiating separate instances of a consensus protocol. By monitoring different events in $\mathcal{E}$, the protocol can automatically ensure consensus on a variety of issues.

The continuous consensus problem was introduced in [1], where it was studied in the crash and sending omissions failure models. This generalized and simplified the earlier related work in [5]. The main results of [1] are simple and efficient optimum CC protocols for both crash and sending omissions failures, in the synchronous message-passing model and assuming an upper bound of $t < n-1$ on the total number of failures. The core provided by their protocol at time k given a particular behavior of the adversary is the union of *all* the cores provided by *all* correct CC protocols under the same adversarial conditions.

In this paper we extend the study of the continuous consensus problem to more problematic failure models: General omissions and Authenticated Byzantine failures. In the former, a faulty process may fail to send a subset of the messages prescribed by its protocol, *as well as* failing to receive a subset of the messages sent to it in a given round. In the authenticated Byzantine failure model, faulty processes are malicious, but correct processes have unforgeable signatures. This ability to sign messages implies that correct processes cannot be misquoted about messages that contain their signatures. As a result, the two failure models are actually quite similar. One property that they share is the fact that when a process i reports not having received a message that another process j was supposed to have sent it, this is proof that one of them is faulty. But this

proof is ambiguous regarding which one of them is the culprit. This is in contrast to the situation in the crash and sending omissions models. In those models, an unreceived message provides unambiguous proof that the intended sender is faulty. This distinction turns out to have significant implications on the efficiency of solutions to the continuous consensus problem. While the optimum protocols in [1] for the simpler failure models can be implemented using linear-time computations and $O(n)$-bit messages, we adapt a result of [5] to show:

Lemma 1. *If $P \neq NP$ then there exists no polynomial-time protocol implementing an optimum solution to continuous consensus in the general omission (resp. in the authenticated Byzantine) failure model.*

This result is not detrimental, since optimum solutions are rare in general. For eventual consensus, for example, it has been shown in [5] that no optimum protocol exists at all, even in the crash failure model. Moreover, searching for continuous consensus protocols that are not optimum is still quite subtle in the presence of failures. Finding relatively efficient ones (and proving their correctness) turns out to be a nontrivial task.

The further contributions of this paper are:

- We present a simple CC protocol that enters every interesting event into the core within $t+1$ rounds (where t is the bound on the number of failures), provided there are a majority of correct processes (i.e., $n > 2t$).
- We improve this protocol to one that achieves similar performance so long as $n > t$ (i.e., there is always guaranteed to be at least one correct process).
- Finally, we use fault monitoring and failure detection to obtain a protocol that in many runs will include events in the core within much fewer than $t+1$ rounds of their discovery by a correct process. This protocol (called VC-CC) is based on the construction of a conflict graph [5–7] and an analysis of failures in such a graph. Intuitively, this is a graph whose nodes are process names, and where an edge appears if there is an inconsistency between the two nodes implying that at least one of them must be faulty.The faulty processes must at all times form a vertex cover of the (edges of the) conflict graph. The correctness of the VC-CC protocol depends on a nontrivial graph-theoretical result that shows the following: If the size of the minimal vertex cover of a graph is b, then the size of the union of all minimal vertex covers of G is at most $2b$ (see Appendix C).
  To this end, we present two types of protocols. One type consists of computationally efficient protocols that have good behavior in the best case, and more theoretical protocols that make use of an NP oracle and produce good performance much more often.
- We then turn to consider the problem of obtaining uniform solutions to CC. In the simpler crash and sending omission failure models [1], the basic optimum protocol is enhanced to yield an optimum solution for uniform CC. The resulting protocol guarantees that all processes—both faulty and nonfaulty—have the same core at all times. Moreover, the core contains exactly what it would under the optimal (non-uniform) protocol. In the general omission and the authenticated Byzantine models, solutions to CC in which all processes have the same core are shown to be impossible. Essentially, if a faulty process might be blocked from receiving any

messages for arbitrarily long periods, then there is no way to ensure that it will maintain a growing core with useful and up-to-date information. We define weaker notion of uniformity that requires the cores of the faulty processes to always be a subset of those of the nonfaulty ones, and show that this version is attainable.

2 Continuous Consensus in the Generalized Omission Model

Using Authentication

Our analysis of the CC problem in the generalized omission model uses an authentication scheme. Pease, Shostak and Lamport [2] presented an algorithm that reaches agreement in a fault-prone system using authentication. Since the generalized omissions model is in fact a simplified private case of the authenticated Byzantine model, this analysis is valid in the generalized omissions model as well. Thus, our algorithms in this section are presented for the authenticated Byzantine model.

We assume that all messages sent in the system are authenticated by unforgeable signatures that enable processes to verify the source of the information they send and receive. Since there are no "liars" in the generalized model, a process may sign a message by simply adding its name to the message. In the authenticated Byzantine model, it is assumed that the signatures are unforgeable despite the fact that some of the processes may be "liars", and thus the signatures are also used to verify the reliability of the information. [1]

Adding signatures to the data sent in the system enables each process to keep track of the knowledge of other processes. When a process receives a piece of information, it can deduce about which processes have received and signed this piece of information by observing the signatures added to this information by other processes.

Notation

We now present some terminology and definitions referring to the usage of authentication in our system. Similar to the notation in [1], our logical language consists of propositions referring to a set of monitored events in our system. For simplicity, we identify the set of monitored events $\mathcal{E}$ with a subset $\Phi_{\mathcal{E}} \subseteq \Phi$, and restrict monitored events to depend only on the external inputs in the current run.

Define $\mathrm{SENT}_i(k)$ as the message that process i sends in round k to each of the other processes. We assume that each message is a sequence of atomic messages called *datagrams*. A datagram is either a proposition, $\varphi \in \Phi_{\mathcal{E}}$, or a *signed* proposition. A signed proposition consists of a proposition and a list of process signatures. Denote by Ψ as the set of all possible datagrams in the system. Clearly, $\Phi_{\mathcal{E}} \subset \Psi$. We also define the operation $sign_i : \Psi \to \Psi$ for a process i, so that if $\alpha \in \Psi$ then $sign_i(\alpha)$ is a datagram containing the information in α signed by process i. Notice that Ψ is defined inductively with primitive elements $\Phi_{\mathcal{E}}$, and is closed under the operations $sign_i(\cdot)$ for all $i \in \mathbb{P}$. In a given run, we denote by $\mathrm{RCVD}_i(k)$ the set of datagrams received by i in round k.

[1] In practice, by having a public-key infrastructure in our system, one may produce cryptographic signatures which are unforgeable with a very high degree of probability.

Let α be a datagram such that $\alpha = sign_{p_d}(\ldots sign_{p_2}(sign_{p_1}(\varphi)))$ with $d \geq 1$. If $p_1, ..., p_d$ are pairwise distinct processes, then α is a *d-signed* datagram, and φ is *d-authenticated* by α. We further define a function $\mathcal{F} : \Psi \rightarrow \Phi_{\mathcal{E}}$. Intuitively, $\mathcal{F}(\alpha)$ is the proposition embedded in the datagram α. Formally, for every datagram α, we have $\mathcal{F}(\alpha) = \varphi$ if there exists a sequence of processes (not necessarily distinct), $\langle p_1, ..., p_s \rangle$, such that $\alpha = sign_{p_s}(\ldots sign_{p_2}(sign_{p_1}(\varphi)))$.

A CC Protocol with Authentication

In this subsection we present two protocols that solve the continuous consensus problem in the generalized omissions model using Authentication.

In Figure 3 we present the first protocol, ACC (which stands for *Authenticated Continuous Consensus*). Each process i runs the protocol individually, and computes a *core* $M_i[k]$ in every round k. The core is guaranteed to be shared among the nonfaulty processes. Process i places a proposition φ in its core when it receives a $(t+1)$-*signed* datagram, α, such that $\mathcal{F}(\alpha) = \varphi$. Every process sends and receives messages according to a protocol we call SFIP, which is a full-information protocol in which signatures are used to authenticate every piece of information delivered in the system. In SFIP every process broadcasts its information adding its signature to it. More formally, an SFIP is a protocol with the following properties:

- An SFIP is, in particular, a FIP, i.e., in every round, every process sends a message encoding all of its information to all other processes.
- Every primitive proposition $p \in \Phi_{\mathcal{E}}$, sent by process i is signed by i.
- In every round k, each process i relays all datagrams received in the previous round, adding its own signature to every datagram.

```
ACC(i)
    M_i[k] ← ∅  for all k ≥ −1
    for k ≥ 0 in round k
        do
1           send and receive messages according to SFIP
2           for α ∈ RCVD_i(k)
3               if α is a (t+1)-signed datagram then
4                   M_i[k] ← M_i[k] ∪ F(α)
5           end for
6           M_i[k] ← M_i[k−1] ∪ M_i[k]
    end for
```

Fig. 1. The ACC protocol for process i.

Intuitively, once a process receives a sequence of $t+1$ signatures for φ, it is guaranteed that at least one nonfaulty process has received information about φ, and has forwarded it to all nonfaulty processes. These, in turn, are able to sign and forward the datagram. Thus, as stated in Lemma 2, all nonfaulty processes will add φ to their cores simultaneously.

Lemma 2. *Let R be a system with $n > 2t$. Then* ACC *solves the continuous consensus problem in R.*

In Figure 2 we present ACCD, which is a slight variant of ACC. In ACCD, for every datagram φ that i receives, it computes the round in which φ is expected to become a $(t+1)$-*authenticated* proposition. In a way, this is a bit similar to the concept of *horizon* presented in [1]: in each round of the CONCON protocol presented there every process i tries to compute a *horizon* based on the number of processes known to be faulty. The idea in ACCD also uses a "horizon," which is defined and computed differently. While in CONCON the core is uniquely determined by a *critical time* and a *critical set*, our approach in ACCD is different; The "horizon" is computed for *each* proposition individually, and thus the core cannot be represented by a particular critical time.

```
ACCD(i)
    M_i[k] ← ∅  for all k ≥ −1
    for k ≥ 0 in round k
        do
1           send and receive messages according to SFIP
2           for α ∈ RCVD_i(k)
3               if α is a d-signed datagram for some 1 ≤ d ≤ t+1 then
4                   M_i[k+(t+1)−d] ← M_i[k+(t+1)−d] ∪ F(α)
5           end for
6           M_i[k] ← M_i[k−1] ∪ M_i[k]
    end for
```

Fig. 2. The ACCD protocol for process i.

Lemma 3. *Let R be a system with $n > t$. Then* ACCD *solves the continuous consensus problem in R.*

Discussion of ACC and ACCD

Lower Bound on n Notice that while ACC requires $n > 2t$, ACCD requires just $n > t$. The reason we require $n > 2t$ for ACC is that a proposition is included in the core once it is signed by $t+1$ signatures. Thus, as the proof of Lemma 2 shows, once a proposition reaches a nonfaulty process, it is forwarded sequentially to other nonfaulty processes, and within at most $t+1$ rounds it is bound to become a $(t+1)$-*authenticated* proposition. Specifically, an event that occurs at a nonfaulty process i, must be forwarded through a sequence of t different nonfaulty processes in order for it to be included in the core. It must therefore by assumed that there are at least $t+1$ nonfaulty processes. ACCD, on the other hand, does not require $n > 2t$, since if a d-*signed* datagram α reaches a nonfaulty process at time m, it forwards the datagram to all nonfaulty processes in round $m+1$, enabling them to include $\mathcal{F}(\alpha)$ in the core at $m+(t+1)-d$.

More efficient implementations of SFIP We previously defined a full-information protocol with signatures, SFIP, as a protocol similar to FIP except that it authenticates all messages using signatures. While it has been shown in [1] that a FIP in our context may be implemented quite efficiently, a first glance at SFIP shows that the overhead added

by the authentication mechanism may by quite high. Since every datagram in SFIP is relayed by every process i, adding its own signature, it is easy to see that if a primitive proposition p is sent by some process i at time 0, then the number of datagrams regarding p in round k is $O(n^k)$. If every signature has length sl, and every datagram has $O(k)$ signatures, then a message regarding p requires a length of $O(sl \cdot k \cdot n^k)$. Since we have n such messages in round k, we obtain communication complexity of $O(sl \cdot k \cdot n \cdot n^k)$ of signatures regarding every proposition p in every round. It is possible, though, to derive a more efficient implementation of SFIP, by applying the following techniques:

(i) **Avoid multiple instances:** In order to avoid multiple datagrams for every proposition, we can have each process i, relay at most one datagram corresponding to every primitive proposition, p, in every round. In particular, for every such p, process i chooses a datagram corresponding to p with the maximal number of signatures. In addition, i completely ignores datagrams which already include i's signature, since they have already been sent by i. Avoiding multiple instances of the same proposition helps reduce the communication complexity of signatures regarding p to $O(n^2)$ in every round.

(ii) **Early stopping:** Since in ACC we are not interested in datagrams with more than $t+1$ signatures, once process i receives a $(t+1)$-*signed* datagram α, it stops sending any datagrams containing $\mathcal{F}(\alpha)$. This reduces the communication complexity of sending p to $O(n \cdot t)$ in every round. As for ACCD, once process i receives a message containing a datagram α, it relays the datagram just *once*. In all future rounds, process i ignores all datagrams β with $\mathcal{F}(\alpha) = \mathcal{F}(\beta)$, which reduces the communication complexity of signatures about p to $O(n)$.

(iii) **Nominating relay processes:** In ACC it is possible to nominate $2t+1$ processes to sign the data they send, while the rest of the processes follow the standard FIP. This allows for a communication complexity of $O(t^2)$, while still enabling the protocol to work correctly. Similarly, in ACCD we can nominate $t+1$ processes, reducing the comm. complexity to $O(t)$.

Uniformity The uniform continuous consensus (UCC) problem was presented in [1]. UCC requires all processes, including the faulty ones, to have the same core at all times. It is an interesting observation that in the crash and omission models, the protocol ACC that we presented above solves UCC. Since in these models a faulty process is assumed to receive all messages sent to it, it is easy to verify that every faulty process receives a $(t+1)$-*signed* datagram α at the exact same time that the nonfaulty processes receive it, and can thus include α in its core. On the other hand, ACCD is not uniform, since its correctness depends on the process i running the protocol being nonfaulty; only a nonfaulty process may assume that a d-*signed* datagram is guaranteed to be relayed to all the nonfaulty processes, and become common knowledge in $t+1-d$ rounds.

In the generalized omissions model, however, we do not have uniformity for ACC, since a faulty process may fail to receive many or even all of the messages sent to it, keeping its core from growing as it should. Consequently, no protocol can guarantee both Completeness (which is defined for nonfaulty processes) and Uniform Consistency in the generalized omission model (see [1]). It is possible, however, to obtain a weaker variant of the Uniform Consistency property. Let j be a nonfaulty process, and let z be an arbitrary process, then:

Weak Uniform Consistency: $M_z[k] \subseteq M_j[k]$ at all times k.

We say that a protocol solves the *weak* UCC problem (WUCC for short), if it satisfies Accuracy, Consistency and Completeness, and in addition it also satisfies Accuracy for the cores of faulty processes and Weak Uniform Consistency, which is related to the cores of faulty and nonfaulty processes. It is easy to see that ACC solves the WUCC problem in the generalized omissions model.

3 CC in the Generalized Omission Model - Improved Protocols

In the previous section we presented ACCD and ACC. Both of these protocols solve the CC problem in the generalized model. However, in both of these protocols no event may be included in the core earlier than $t+1$ rounds after it occurs. In this section we discuss different solutions to the CC problem, which provide a richer core, and enable some propositions to join the core earlier than they would in either of the two protocols above.

Similar to the protocols in the previous section, our protocol uses message authentication. The idea is that if in round k it is confirmed that a subset $S(k) \subset \mathbb{P}$ contains at least s faulty processes, then we know that $\overline{S}(k)$, the complement of $S(k)$, contains at most $t-s$ faulty processes, allowing us to add $(t-s+1)$-*authenticated* propositions to the core.

The conflict Graph

In [5] Moses and Tuttle prove that the problem of testing for common knowledge in the generalized omission model is NP-hard by showing a Turing reduction from the Vertex Cover problem to the problem of testing for common knowledge. It follows by their analysis that information about the number and identities of faulty processes can be obtained by a Vertex Cover computation, as we shall now describe.

Assume that at the end of every round k, process i constructs a graph, $\mathcal{G}_i(k) = (V, E_i(k))$. V consists of n vertices, each labelled by a unique process name. $E_i(k)$ contains an edge $\{p_j, p_s\}$ i knows at the end of round k that at least one message between p_j and p_s has not been delivered successfully. $\mathcal{G}_i(k)$ is called a *Conflict Graph*, since each of its edges stands for a conflict between its adjacent nodes: At most one of them is nonfaulty. It is thus easy to see that the nodes representing the processes which have displayed faulty behavior up to round k form a vertex cover of $\mathcal{G}_i(k)$. It follows that if $\mathcal{G}_i(k)$ has a *minimum* vertex cover of size s, then there must be at least s faulty processes in r. Our protocol, VC-CC is based on these properties of the conflict graph.

The analysis in this section uses the conflict graph to solve the CC problem. As we shall see, since the conflict graph represents information about potential failures in the system, this information may be used to include some facts in the core earlier than they would in either ACC or ACCD.

A Simple Protocol Using the Conflict Graph

We first present a very simple protocol, which is an extension of ACC. The protocol is called ACCI (short for Improved ACC). The idea is that instead of defining the core as

```
ACCI(i)
    M_i[k] ← ∅, B_M[k] ← ∅   for all k ≥ −1
    for k ≥ 0 in round k
        do
 1          repeat iteratively until B_M[k] stabilizes
 2              send and receive messages according to SFIP
 3              for α ∈ RCVD_i(k)
 4                  if α is (t − |B_M[k]| + 1)-signed by P \ B_M[k] then
 5                      M_i[k] ← M_i[k] ∪ F(α)
 6              end for
 7              M_i[k] ← M_i[k − 1] ∪ M_i[k]
 8              for all j ∈ P
 9                  B_M[k] ← j if M_i[k] implies that j is faulty
10          end repeat
    end for
```

Fig. 3. The ACCI protocol for process i in the generalized omission model.

all $t+1$-*signed* facts, we allow the information *in the core* to reduce this $t+1$ round margin. For example, if the fact that process z is faulty is included in the core, then the fact that there are at most $t-1$ faulty processes among $\mathbb{P} \setminus \{z\}$ is also in the core. Thus, receiving t signatures from processes in the set $\mathbb{P} \setminus \{z\}$ should be enough to include a fact in the core.

In ACCI, in every round, each process constructs a conflict graph according to the information in the core. We define $B_M[k]$ as the set of processes that are *confirmed* to be faulty according to information in $M_i[k]$. Specifically, once the conflict graph contains edges from a process z to at least $t+1$ different processes, z is confirmed to be faulty, and thus $z \in B_M[k]$. More generally, if a process z has at least $t - B_M[k] + 1$ conflicts with processes from $\mathbb{P} \setminus B_M[k]$, then z must be faulty, and thus we can include z in $B_M[k]$. Notice that updating $B_M[k]$ may require iterative repetition of the steps above, until $B_M[k]$ stabilizes. It is important to note that ACCI has polynomial running time, since it requires counting the number of edges connected to each node, which requires polynomial computations.

VC-CC

We now present VC-CC. In each round, every process computes a *conflict graph*, $\mathcal{G}_i(k)$. It is essential that the conflict graphs constructed by all processes are the same in every round, since the computation of the core is based on the information in the conflict graph, $\mathcal{G}_i(k)$. Thus we require that $\mathcal{G}_i(k)$ is constructed by i according to $M_i[k-1]$, which guarantees a consistent conflict graph for all nonfaulty processes. The subroutine AUTHENTICATEDVC(G, α) (used on line 6 in the protocol) performs a test on α according to G, and returns TRUE if α is to be included in the core, and FALSE otherwise.

After updating the core $M_i[k]$ at time k (lines 5 and 6), a new conflict graph, $\mathcal{G}_i(k)$ may now be constructed (line 8), based on recent information. Thus we iteratively up-

VC-CC(i)

0 $M_i[k] \leftarrow \emptyset$ for all $k \geq -1$
for $k \geq 1$ in round k **do**
1 send and receive messages according to SFIP
2 $M_i[k] \leftarrow M_i[k-1]$
3 $\ell \leftarrow 0$
4 compute $\mathcal{G}_i^{(\ell)}(k)$
repeat
5 **for** $\alpha \in \text{RCVD}_i(k)$
6 **if** $\mathcal{F}(\alpha) \notin M_i[k]$ **and** AUTHENTICATEDVC($\mathcal{G}_i^{(\ell)}(k), \alpha$) **then** $M_i[k] \leftarrow M_i[k] \cup \mathcal{F}(\alpha)$
7 $\ell \leftarrow \ell + 1$
8 compute $\mathcal{G}_i^{(\ell)}(k)$
9 **until** $\mathcal{G}_i^{(\ell)}(k) = \mathcal{G}_i^{(\ell-1)}(k)$
end for

Fig. 4. The VC-CC protocol for process i in the generalized omission model.

date $M_i[k]$ according to $\mathcal{G}_i^{(\ell)}(k)$, and construct $\mathcal{G}_i^{(\ell)}(k)$ according to the recent computation of $M_i[k]$, until a fixed-point of $\mathcal{G}_i^{(\ell)}(k)$ is reached (line 9).

The subroutine AUTHENTICATEDVC(G, α) performs a test on α according to G, the conflict graph. It returns TRUE if the conflict graph G and the signatures in α verify that $\mathcal{F}(\alpha)$ should be included in the core.

Lemma 4. VC-CC *solves the Continuous Consensus problem for* $n > 3t$.

We will now briefly discuss the intuition behind the proof of Lemma 4. The challenging part of the proof is to show that VC-CC satisfies Consistency. There are three locations where the subroutine AUTHENTICATEDVC(G, α), returns TRUE, indicating that α should be included in the core. We consider each of these cases separately:

(i) **α is a *d-signed* datagram by a set of processes D, and it is guaranteed that at least one of the processes in D is nonfaulty** (lines 2 to 5 in the AUTHENTICATEDVC subroutine). For every α which is *d-signed* by a set of processes, D, we

AUTHENTICATEDVC(G, α)

1 compute C and c according to G
2 **if** α is *d-signed* by a set of processes D such that $d \leq t$ **then**
3 $G' \leftarrow G \restriction \mathbb{P} \setminus D$
4 $b \leftarrow$ size of min-VC of G'
5 **if** $d \geq t - b + 1$ **then return** TRUE
6 $h \leftarrow t - c$
7 **if** α is signed by a sequence of processes, at least $h+1$ of which are from $P \setminus C$ **then**
8 **return** TRUE
9 **if** α is a $t+1$-*signed* datagram **then**
10 **return** TRUE
11 **return** FALSE

Fig. 5. The AUTHENTICATEDVC procedure used in VC-CC.

compute the min-VC of the set $\mathbb{P} \setminus D$. If the size of the min-VC is b, then the set D contains at most $t-b$ faulty processes, and thus any if $d > t-b$, then at least one nonfaulty process exists in D. In this case α is inserted into the core at $k+t+1-d$, i.e., to $M_i[k+t+1-d]$. Proving consistency for this case is the delicate part of the proof. It depends on the following graph-theoretical property: [2]

Lemma 5. *If the size of the minimum vertex cover of an undirected graph is b, then the union of all its vertex covers has size* $2b$ *at most.*

(ii) α **is** $(t-c+1)$***-signed*** **by processes from** $\mathbb{P} \setminus C$ (lines 6 to 8 in the protocol). AUTHENTICATEDVC computes all vertex covers of G with at most t nodes. Let $\nu_1, \ldots, \nu_s$ be these VC's. Define $C \triangleq \bigcap_{j=1}^{m} \nu_j$, and $c \triangleq |C|$. The set C is the conjunction of all vertex covers of size t, and thus includes all nodes which are *guaranteed* to represent a faulty process. It follows that the set $\mathbb{P} \setminus C$ contains at most $t-c$ faulty processes, and thus $t-c+1$ signatures are enough to authenticate a datagram and return TRUE.

(iii) α **is** $t+1$***-signed*** (lines 9 and 10).
We note that proving the consistency property in cases (ii) and (iii) above, is very similar to proving the consistency of ACC.

4 Byzantine Continuous Consensus

So far we discussed the generalized omission model. However, since we are using an authentication scheme, a natural extension to our analysis is to consider the authenticated Byzantine model. Our assumption in this model is that although faulty processes may be "liars," they can only lie about their own local states and inputs, and cannot alter any relayed information.

When analyzing the CC problem in the context of the authenticated Byzantine model, we find that satisfying the Accuracy property may prove a bit problematic: a faulty process, z, may falsely claim that an event e occurred, without any process ever realizing that z is faulty. In this case including e in the core would spoil its Accuracy. We solve this problem by modifying the sets of primitive propositions and monitored events, Φ and $\mathcal{E}(\mathsf{V})$, defined above. Let Φ^B be a set of primitive propositions of the form "process i claims that p" for $i \in \mathbb{P}$ and $p \in \Phi$. Similar to our definition in Section 2, we restrict our analysis to a set $\mathcal{E}^B$ of monitored events that is identified with $\Phi^B_{\mathcal{E}} \subseteq \Phi^B$, such that monitored events depend only on what processes *claim* about their inputs. When using authentication, propositions from $\Phi^B_{\mathcal{E}}$ cannot be forged, since when a process receives a proposition $p \in \Phi^B_{\mathcal{E}}$, it may verify its signature. Thus, in this context the Accuracy property does not present a problem in the authenticated Byzantine model.

By restricting the core to the set $\mathcal{E}^B$ of monitored events, it is easy to verify that both the protocols shown in Section 2, as well as the ones shown in Section 3 provide a solution to the CC problem. The usage of authentication eliminates the possibility of

[2] As with all claims in this paper, the proof of Lemma 5 is deferred to the full paper.

forging *relayed* messages, while a message *produced* by a faulty process j regarding a proposition q is not considered a lie in our context, since it is indeed true that "j claims that q". Notice, however, that the core may still contain both "j claims that q" and "j claims that $\neg q$", provided that j is faulty. Such inconsistencies may be settled by protocols for Byzantine agreement that are beyond the scope of this work.

5 Conclusion

In this paper we discussed the continuous consensus problem in the generalized omission model. We presented two very simple protocols, ACC and ACCD, that solve the CC problem in this model by using an authentication. We showed that these protocols may be implemented quite efficiently. Whereas ACC requires $n > 2t$, we have shown that ACCD applies for $n > t$. In addition, we have shown that ACCD is early stopping, and that ACC, on the other hand, satisfies the *weak uniformity* property.

While ACC and ACCD both solve the CC problem in the crash and omission models, none of them is optimal. In both protocols it takes at least $t + 1$ rounds from the time that an event takes place until it is included in the core. As shown in [1], in the crash and in the sending omission models, some events may enter the core much sooner than this, by using a protocol called CONCON.

It was shown that by maintaining a *conflict graph* of the system, processes obtain information about failures in the run, which allows them to add facts to the core sooner than $t + 1$ rounds after their occurrence. We showed a very simple protocol called ACCI, which uses the conflict graph to identify a set of confirmed faulty processes, thus enabling processes to include in the core datagrams with less than $t + 1$ signatures.

Finally, we presented VC-CC, which, in addition to the techniques used by the previously described protocols, uses a subtle vertex-cover computation to obtain further information about failures in the system. While this solution produces a richer core, it is computationally problematic, since it requires computing the minimal vertex-cover, which is an NP-hard computation.

References

1. T. Mizrahi and Y. Moses, "Continuous consensus via common knowledge," *Distributed Computing*, to appear 2008. An early version appears in the *Proceedings of TARK X*, pp. 236–252, 2005.
2. M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *Journal of the ACM*, vol. 27, no. 2, pp. 228–234, 1980.
3. D. Dolev, R. Reischuk, and H. R. Strong, "Eventual is earlier than immediate," in *Proc. 23rd IEEE Symp. on Foundations of Computer Science*, pp. 196–203, 1982.
4. 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.
5. Y. Moses and M. R. Tuttle, "Programming simultaneous actions using common knowledge," *Algorithmica*, vol. 3, pp. 121–169, 1988.
6. P. Berman and J. A. Garay, "Cloture votes: $n/4$-resilient distributed consensus in $t + 1$ rounds.," *Mathematical Systems Theory*, vol. 26, no. 1, pp. 3–19, 1993.
7. J. A. Garay and Y. Moses, "Fully polynomial byzantine agreement for $n > 3t$ processors in $t + 1$ rounds," *SICOMP: SIAM Journal on Computing*, vol. 27, pp. 247–290, 1998.