Protocols for Byzantine Agreement

Alessandro Panconesi
DSI - La Sapienza
via Salaria 113, piano III
00198 Roma, Italy

After the impossibility proof of Consensus for n synchronous procesess, t of which can be byzantine faults, $t \geq n/3$, we now come to showing the existence of protocols. We start with the simpler case of crash failures and switch later to analizing the case of byzantine faults.

1 Withstanding Crash Failures

The case of crash (or fail-stop) failures is quite easy to handle. We will consider the more difficult case of *dirty* faults. i.e. a process can fail during the transmission of a message. As a result the message could reach a proper subset of the set of processes. The protocol is shown in Figure 1 and its basic idea is as follows. Each process maintains a set S_p of pairs (q, x_q). Essentially S_p is the set of all input bits that are known to p. For $f+1$ rounds, every process broadcasts a pair only when it sees it for *the first time*, where by "broadcast" we mean that the message is sent to all. In other words, suppose that p receives the pair (q, x_q) at round k for the first time. Then, if $k+1 \leq f+1$, p will broadcast this pair at round $k+1$. This pair will never be broadcast by p again.

As soon as a new pair is seen the set S_p is updated accordingly. At the end (round $f+2$) the decision bit of process p is the smallest input bit contained in S_p. That is, if S_p contains a pair of the kind $(q, 0)$ the decision is 0, otherwise it is 1. Note that the decision rule is well defined because S_p contains at least (p, x_p).

As usual x_p and y_p denote, respectively, the input and the decision bit of process p. We now verify that the protocol satisfies Limited Bureaucracy, Non-Triviality, and Agreement.

round 1. Let x_p be the input bit of p. Send (p, x_p) to all. $S_p := \{(p, x_p)\}$

round r. $(2 \leq r \leq f+1)$ Let R_p^r be the set of pairs (q, x_q) received at the end of round $r-1$. If R_p^r contains a pair not in S_p then, send it to all. $S_p := S_p \cup R_p^r$.

round f+2. Let $S_p := S_p \cup R_p^{f+2}$, and let y_p be the minimum value x_q among all pairs contained in S_p.

Figure 1: Consensus protocol for processor p. The number of faults that the protocol withstands is denoted by f.

Limited Bureaucracy. The protocol terminates in $f+2$ rounds.

Non Triviality. Let X be the set of values of the input bit, and S be the set of values sent by processors during the execution of the protocol. Non Triviality follows from the observation that $S \subseteq X$.

Agreement. The proof is by case analysis. Initially, there are two cases two consider: (a) all correct processes decide 1. If this is the case, there is nothing to prove. Otherwise, (b) there exists a correct process p that decides 0. In this case we have to show that every other correct process q decides 0. We split this into two subcases. The first subcase is when x_p, p's input bit, is 0. In this case, p sends 0 to everybody at round 1. The bit reaches every other correct process before round 2. Hence every correct process sees a 0 value before the beginning of round $f+2$, and the conclusion follows.

The other case to consider is when $x_p = 1$. This being the case, p must have received 0 from another process at some round r, for p's decision is 0. Again, we have two sub-sub-cases. If $r < f+2$ then, p echoes the 0 value to all, and every correct process sees a 0 before the beginning of round $f+2$, and decides 0 accordingly. Otherwise, $r = f+2$ and p does not have the time to echo the value. How can we conclude the proof? Let a_1 be the process who sends the 0 value to p at round $f+1$. If a_1 is correct then, every correct process receives the 0 value by round $f+2$, and we are done. Suppose then, by contradiction, that a_1 is faulty. Now, from the definition of the protocol, a process echoes a value as soon as it sees this value for the first time. Therefore a_1 must have received the 0 value from another process a_2, who sent it at round f. If a_2 is correct we are done. Otherwise we iterate the above reasoning. In this fashion we exhibit a chain of $f+1$

faulty processes, each sending the 0 value:

$$a_1 \leftarrow a_2 \leftarrow \ldots \leftarrow a_{f+1}.$$

But this is impossible, since there are at most f faults.

2 Withstanding Byzantine Faults

We now come to showing a polynomial-time protocol for Consensus withstanding $t < n/3$ byzantine faults. The idea is to adapt the protocol for crash failures to the byzantine scenario. This will be accomplished by implementing a *consistent broadcast*, a mechanism for passing messages between processes with certain gurantees.

2.1 Consistent Broadcast

Consistent broadcast has two operations associated with it: $\mathsf{BCast}(m)$ and $\mathsf{Accept}(m)$ that we want to implement by means of the communication primitives *send* and *receive*. Recall that the underlying communication mechanism provides point-to-point connections which are secure and authenticated. That is, each process can send a message directly to any other process and the receiver can be sure of the identity of the sender and of the content of the message. Messages sent with the consistent broadcast facility are made of three parts: $m = (c, p, r)$, where c is the message content, p the sending process, and r the round in which the message is sent. Consistent broadcast has the following three properties:

correctness If $m = (c, p, r)$ is BCast by $p \in L$ at round r then all $q \in L$ Accept m by round $r+2$. Here, L, as before, denotes the set of correct processes.

unforgeability If $m = (c, p, r)$ is not BCast by $p \in L$, then no $q \in L$ ever accepts it.

relay If $q \in L$ accepts m at round r, then every $p \in L$ has accepted it by round $r + 1$.

The protocol that is used to achieve the above is described next. As before, there are at most $f < n/3$ faulty processes (implying that there are at least $2f + 1$ correct, or loyal). Recall that the communication channels satisfy the following properties. The underlying network is a complete graph.

A message that is sent is received by the receiver in one round, and the identity of the sender is known.

In the following, sending a message to "all" includes sending it to oneself. The following describes the implementation of $\mathsf{BCast}(m)$ and $\mathsf{Accept}(m)$ in terms of the primitive send and receive operations. Here a process p wants to $\mathsf{BCast}(m)$ at round r.

round r The process p sends $[\text{INIT}, (c, p, r)]$ to all processes.

round $r+1$ If q receives $[\text{INIT}, (c, p, r)]$ from process p, it sends $[\text{ECHO}, (c, p, r)]$ to all processes.

round $t \geq r+2$ If process q receives $[\text{ECHO}, (c, p, r)]$ from at least $f+1$ distinct processes, and it has not yet sent an echo, then it sends $[\text{ECHO}, (c, p, r)]$.

round $t \geq r+2$ If process q receives $[\text{ECHO}, (c, p, r)]$ from at least $2f+1$ processes it performs $\mathsf{Accept}(c, p, r)$.

The protocol has the correctness property since when $p \in L$ sends a message at round r, it is echoed by $2f+1$ different $q \in L$ at round $r+1$, and thus accepted by every $q \in L$ at round $r+2$.

The unforgeability property follows because to accept a message at least $2f+1$ echoes are required. An $[\text{ECHO}, (c, p, r)]$ is triggered by two events only. The first way to trigger an echo is for a process to receive the message $[\text{INIT}, (c, p, r)]$. Since the underlying mechanism is point-to-point and the identity of the sender known, if an INIT message is not sent it cannot be forged. By the rules of the protocol then the only possibility is for a process to receive $f+1$ echoes. Of these at most f are traitors. Therefore there must be a correct process q_1 that has sent the echo. Again, q_1 must have received $f+1$ echoes. Of these one must come from a new correct process q_2, because the protocol specifies that echoes be sent only once by every process. In this fashion we construct a chain of correct processes $q_1 q_2 \ldots q_\ell$, where ℓ is the tital number of correct processes. The last process, q_ℓ, again must have sent an echo because it has received $f+1$ echoes, one of which from a correct process. But there are no more corret processes, a contradiction.

The verification of the relay property is left as an exercise.

2.2 Implementation of Byzantine Agreement

The protocol for consistent broadcast can be used to implement Byzantine agreement in $2f+3$ rounds. Time is divided into $f+1$ *phases*, denoted by s

in the range 1 to $f+1$ for the rest of this section. Each phase consists of 2 *rounds*, and one extra round is needed at the end of the protocol to decide. There is no communication in the last round.

The protocol mimics the one described earlier in the context of crash failures. The idea is that processes whose input bit is 1 would like to attack, while processes whose input bit is zero would not. Let us call the processes that want to attack *bushonauts*.

BCasts are executed at odd numbered rounds only, while Accepts can be performed at any round. Processes with input bit 1 are bushonauts and start by broadcasting their intention to attack. Process that initially are not bushonauts join in if they receive an ATTACK! message from a sufficient number of bushonauts. The number of bushonauts needed at round $r := 2s-1$ to trigger a decision to join and become a bushonaut is $f+s-1$. This is called the *threshold* of round r. The threshold is $f+1$ at round 3, $f+2$ at round 3, $f+3$ at round 5, and so on. In the final round there is no communication. Processes decide to attack if they have received opinions to attack from at least $2f+1$ bushonauts, otherwise they decide not to attack.

Note that a process BCasts at most once and it does so either at the very beginning, if its input bit is 1, or in the middle of the protocol as soon as the number of attack messages is above or equal to the threshold.

As usual, the inputs to the processes are denoted as x_p, and the outputs as y_p. Below is a description of the protocol, which runs concurrently with the protocol for consistent broadcast described before. Messages are of the form $m :=$ (ATTACK!, *sender*, *timestamp*) where the timestamp is the round during which the message is sent.

Protocol Precognitive Attack

round 1 If $x_p = 1$ then p executes BCast(ATTACK!,p,1).

round $r := 2s-1, (s > 1)$ ***(odd rounds)*** Let $M_{p,r}$ be the number of different processes from which p has accepted messages at round r or earlier. If $M_{p,r} \geq f+s-1$, and process p has not done it already, then p executes BCast(ATTACK!,p,r).

round $r := 2f+3$ ***(final round)*** Let $M_{p,r}$ be defined as above. Process p decides ATTACK! (i.e., $y_p = 1$) if $M_{p,r} \geq 2f+1$; otherwise, it decides not to attack (i.e. $y_p = 0$).

Notice that, as remarked, every correct process broadcasts at most once. We now prove the three properties we require: non-triviality, agreement, and limited bureaucracy.

First observe, that if at any round $2s-1$, for $1 \leq s \leq f+1$, all correct processes have broadcast, all correct processes will eventually decide 1. This follows from the correctness property of consistent broadcast which ensures that a message sent by a correct process is accepted within two additional rounds.

The non-triviality follows since if $x_p = 1$ for all $p \in L$, all correct processes start by broadcasting, and the observation above applies. If $x_p = 0$ for all $p \in L$, none of the correct processes will ever start broadcasting because (a) at round 1 no correct process broadcasts and (b) the threshold for broadcasting at subsequent phases is $f+s-1 \geq f+1$. Therefore no correct process will receive the $2f+1$ messages needed to decide on 1.

For the agreement property, we prove that for $p \in L$ we have that if p decides 1, then all correct processes decide 1. If $d_p = 1$, process p has accepted messages from at least $2f+1$ processes by round $2f+3$. Let I denote the set of correct processes from which p has accepted messages; we have $|I| \geq f+1$.

If all processes in I had input 1, every correct process accepts $f+1$ messages at round 3 and then broadcasts, so again, the observation above applies and we are done.

Otherwise, there is a process q for which $x_q = 0$. Thus, process q broadcasts at round $2s-1$ for some $2 \leq s \leq f+1$, which means that at round $2s-1$, process q has accepted messages from at least $f+s-1$ processes other than itself. These messages are accepted by all other correct processes at round $2s$ (because of the relay property of consistent broadcast), and the message from q itself reaches all other correct processes by round $2s+1$ (because of correctness of consistent broadcast). It follows that at round $2s+1$ (phase $s+1$), each correct process has accepted messages from at least $f+s$ processes, and thus broadcasts since the new threshold for phase $s+1$ is reached (if $s < f+1$), or decides 1 (if $s = f+1$).

Finally, the bureaucracy is limited to $2f+3$ rounds.