Disposition 10: Asynchronous Byzantine Agreement

Mathias Ravn Tversted

January 14, 2020

Table of contents

Asynchronous Model

Asynchronous Broadcast

- Definition
- Bracha Broadcast

Asynchronous Byzantine Agreement

- Asynchronous Agreement
- Asynchronous Termination

Asynchronous Model

The asynchronous model

In the async model, we have the following properties: Messages can be delayed for arbitrarily long, and we have interested in the following properties:

- **Agreement**: All honest parties make the same decision
- **Validity**: Decision made must be sensible in some sensible
- **Termination**: If all parties run the protocol, eventually all honest parties will make a decision

Additionally, we say that the protocol does not guarantee liveness until all parties start running it, since people can fail arbitrarily long behind.

Asynchronous Broadcast

Defining Asynchronous Broadcast

We build it on ACast and we use a simple PKI *ToyPKI*.

Asynchronous Broadcast uses signatures. A broadcaster P_i will ask all n parties to sign message m it sends. All parties thus sign at most one m for each P_i. Then P_i waits for $n - t$ parties to send a signature. To broadcast, P_i sends m along with $n - t$ signatures. A receiver only outputs m if it has $n - t$ signatures. On output, pass it along to everyone else with a signature, who will then also output m.

Agreement - Dafuq

If two honest P_j, P_k output m_j, m_k, then we want $m_j = m_k$. If there are t corrupted (possibly including sender P_i) then we have the following argument: Each party sees $n - t$ distinct signatures. If all corrupt t parties sign both m_j, m_k (causing them to be output), this gives us at most $2t$ signatures. This gives us a total of $(n - t) + 2t = n + t$ distinct signatures on either m_j or m_k.

If $m_j \neq m_k$ then all signatures on both are distinct. So in this case we have $(n - t) + (n - t)$ distinct signatures. We know that $n - t = 2t + 1$ so
$(n - t) + (n - t) = (n - t) + (2t + 1) = n + t + 1$ which is greater than the $n + t$ we have just proved. So $m_j = m_k$.

Validity - Dafuq???

Honest parties output m from P_i if it sees at least $n - t$ signatures. But then it saw at least $(n - t) - t = t + 1$ signatures from honest parties. But then $t + 1 \geq 1$ honest parties signed m. And honest parties only sign the m that comes from P_i.

Termination

If P_i is honest, it asks all honest parties to sign m. At least $n - t$ honest grant signatures. P_i at some point receives $n - t$ signatures on m. It the nforwards them so all honest P_j receives m with $n - t$ signatures, and so they output m.

Async Broadcast from Authenticated Channels

Actually implementing it with an authenticated channel can be done with Bracha broadcast. It goes as follows: Assume P_1 is the broadcaster.

- P_1 gets input (P_1, bid, m) on $ACast_i$ to start. We say it got $(Broadcast, P_1, bid, m)$
- When party outputs P_n, bid, m on $ACast_j$ we say it has output $(Deliver, P_n, bid, m)$
- There are no rounds, only activation rules

Bracha Broadcast

Send: P_1: On input (BROADCAST, P_1, bid, m), send (SEND, P_1, bid, m) to all parties **Echo**: P_i: On message (SEND, P_1, bid, m) from P_1, send (ECHO, P_1, bid, m) **Ready 1**: P_i once message (ECHO, P_1, bid, m) has been received from $n - t$ parties, send (READY, P_1, bid, m) **Ready 2**: P_i once message (READY, P_1, bid, m) has been received from $t + 1$ parties, send (READY, P_1, bid, m) to all parties if not yet done **Deliver**: Once message (READY, P_1, bid, m), has been received from $n - t$ parties, output (DELIVER, P_1, bid, m) and terminate the protocol

About Bracha broadcast

Each of the preceding rules are activation rules. There are no rounds. The rules can be activated in any order (READY 2 before READY 1 for example).

We also wait for $n - t$ messages, and not n messages, because we cannot distinguish between late and never arriving messages. When we have enough information we act.

If you wait for $t + 1$ parties, you are also ensuring you hear from one correct party. We use this to learn that someone honest saw the mesage m.

We also need $n > 3t$ to ensure common correct party between two parties. (BEVIS SIDE 210?)

Asynchronous Byzantine Agreement

Defining Asynchronous Byzantine Agreement

In Asynchronous Byzantine Agreement. The parties try to agree on some decision. We tolerate $t = \frac{n}{3}$ Byzantine corrupted parties.

The main idea is to try to stabilise the network so that all honest parties have the same vote. You run a sequence of protocols to update votes. These have the properties that they stabilise the network and the parties end up in agreement.

Vote Updating Protocols

Definition 9.2: An async protocol is *vote updating protocol* (VUP). The protocol is as follows:

- Each party starts with $V_i \in \{0, 1\}$
- It should be *terminating*. All honest terminate with updated vote $W_i \in \{0, 1\}$
- Inputs are *stable on V* if all honest have $V_i = V$
- Protocol is *stability preversing* if when inputs are stable, then outputs are also stable
- VUP is a σ-stabiliser if for all configurations of input, that the outputs are stable with probability at least σ
- VUP is *detector* if it inputs to each party a truth vaue $D_i \in \{\top, \bot\}$. if input is stable, then $D_i = \top$ for all honest parties if output is stable

Basic VUP

We can implement a basic VUP using signatures. Each P_i sends signed V_i. Collect $n - t$ signatures and pick $J_i \in \{0, 1\}$ with $t + 1$ signatures. Send *justified vote* J_i with the signatures. Collect $n - t$ justified votes. If they are the same, then $W_i = j$ is a *hard vote*. Otherwise pick somehow a *soft vote*.

Stabilising Vote Updating Protocol

Each P_i runs: Send V_i to all parties with signature. Collect $n - t$ signatures. Pick $J_i \in \{0, 1\}$ with $t + 1$ distinct signatures. ACast the justified vote J_i to all parties with their signatures. Collect $n - t$ justified votes. If they are all the same, hard vote it. $W_i = J$. Otherwise uniformly randomly pick a soft vote W_i.

ACast is used to prove **Lemma 9.3 (SVUP)**:. It holds that if an honest party saw $t + 1$ justified votes for bit J, then no honest party gave or will give a hard vote for $W \neq J$.

SVUPPERE

Flere af disse kan evt tilføjes, men slides siger at man også skal nå at dække sektion 9.6

Asynchronous Termination

Definition 9.6 (Asynchronous Termination): When a party terminates, it terminates its own process, plus its processes in any sub-protocols. With one exception: It will forever run the process that implements delivery on the authenticated channels its using.

This could be delegated to to the OS or some other daemon. If a correct process terminates, it can affect liveness of sub-protocols. It cannot affect safety properties. Termination is equivalent to its messages being delayed arbitrarily. With *piggy bagging*, you can send these time capsule messages along with the next message in the main protocol. This way, messages can be delivered even if the sender terminates the protocol.

Vote Updating Protocol Byzantine Agreement

Input: From P_i, on input (VOTE, baid, v_i). Let $V_i^0 = v_i$. Let $r = 0$. Let $GaveOutput = \perp$. Start stabilise and then output

Stabilize: From P_i, keep running:

1. Run SVUP with input V_i^r to get W_i^r
2. Run DeVUP with input W_i^r to get V_i^{r+1} and D_i^r
3. If $D_i^r = \top$, then output (DECISION, baid, V_i^{r+1}). Let $GaveOutput = \top$ and send (TIMECAP, V_i^{r+1}) to all parties.
4. Let $r \leftarrow r + 1$ and repeat step 1

The TimeCapsule is delivered even if the sender terminates VUPABA.

Vote Updating Protocol Byzantine Agreement 2

Output: From P_i:

- **Term 1** On Receiving (TimeCAP, d) from $t + 1$ parties, send (TIMECAP, d) to all parties. If $\mathit{GaveOutput} = \bot$, then set $\mathit{GaveOutput} = \top$ and output (Decision, baid, d)
- **Term 2** On receiving (TIMECAP, d) from $2t + 1$ parties, terminate

ABA from VUP and Time Capsule

This protocol implements Asynchronous Byzantine Agreement. When a party P_i sees that $D_r^r = \top$, then it knows that the network is stable and other parties will realise it next round. It keeps running to ensure that others reach the next round. When enough reaches next round, ensuring that enough time capsules are around, it can terminate. The two tresholds in the shutdown mechanism is the same as in Bracha Broadcast.

ARGUE

NEEDS MORE OF IT RIGHT HERE POENTIALLY