Consensus

(And View Synchronisation)

Andrew Lewis-Pye, 28th June 2022

Joint work with Ittai Abraham:
The new result I'll talk about is a 'view synchronisation' method for 'optimistically responsive' blockchain protocols (like Hotstuff) which has $O(n)$ communication complexity per view in the worst case.

Joint work with Ittai Abraham:
The new result I'll talk about is a 'view synchronisation' method for 'optimistically responsive' blockchain protocols (like Hotstuff) which has $O(n)$ communication complexity per view in the worst case.

Combined with Hotstuff, this gives the first optimistically response blockchain protocol functioning in the partially synchronous setting which has:

- $O(n)$ complexity per confirmed block in the optimistic case;
- $O(n^2)$ complexity per confirmed block in the worst case.

A LITTLE PUZZLE FOR THOSE WHO KNOW CONSENSUS

Consider the synchronous setting, authenticated channels, Byzantine faults, PKI. Can you design a deterministic protocol to solve Byzantine Broadcast, in which each party speaks at most once?

'Speaking once' means that each party can send multiple messages, but they must all be sent at the same timeslot.

What is the problem a consensus protocol has to solve?

THE BYZANTINE AGREEMENT PROBLEM (INFORMAL)

Several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general. The generals can only communicate by messenger and must carry out a protocol to decide on a common plan of action, either 'retreat' or 'attack'. Initially, each general has their own private opinion as to the best plan of action. The difficulty is that some unknown subset of the generals may be dishonest traitors (and may deviate from the protocol). The protocol must satisfy:

THE BYZANTINE AGREEMENT PROBLEM (INFORMAL)

Several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general. The generals can only communicate by messenger and must carry out a protocol to decide on a common plan of action, either ‘retreat’ or ‘attack’. Initially, each general has their own private opinion as to the best plan of action. The difficulty is that some unknown subset of the generals may be dishonest traitors (and may deviate from the protocol). The protocol must satisfy:

(1) **Termination.** Each honest general must eventually reach a decision (retreat or attack).

THE BYZANTINE AGREEMENT PROBLEM (INFORMAL)

Several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general. The generals can only communicate by messenger and must carry out a protocol to decide on a common plan of action, either 'retreat' or 'attack'. Initially, each general has their own private opinion as to the best plan of action. The difficulty is that some unknown subset of the generals may be dishonest traitors (and may deviate from the protocol). The protocol must satisfy:

(1) **Termination.** Each honest general must eventually reach a decision (retreat or attack).

(2) **Agreement.** All honest generals must reach the same decision.

THE BYZANTINE AGREEMENT PROBLEM (INFORMAL)

Several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general. The generals can only communicate by messenger and must carry out a protocol to decide on a common plan of action, either 'retreat' or 'attack'. Initially, each general has their own private opinion as to the best plan of action. The difficulty is that some unknown subset of the generals may be dishonest traitors (and may deviate from the protocol). The protocol must satisfy:

(1) **Termination.** Each honest general must eventually reach a decision (retreat or attack).
(2) **Agreement.** All honest generals must reach the same decision.
(3) **Validity**. If all honest generals start with the same opinion, then that common opinion must be the same as their final decision.

THE BYZANTINE AGREEMENT PROBLEM (INFORMAL)

Several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general. The generals can only communicate by messenger and must carry out a protocol to decide on a common plan of action, either 'retreat' or 'attack'. Initially, each general has their own private opinion as to the best plan of action. The difficulty is that some unknown subset of the generals may be dishonest traitors (and may deviate from the protocol). The protocol must satisfy:

(1) **Termination.** Each honest general must eventually reach a decision (retreat or attack).
(2) **Agreement.** All honest generals must reach the same decision.
(3) **Validity**. If all honest generals start with the same opinion, then that common opinion must be the same as their final decision.

Without validity requirement, it would be trivial.

THE BYZANTINE AGREEMENT PROBLEM (INFORMAL)

There are n parties (generals), of which at most f are dishonest. Each starts with their own input. Must satisfy:

(1) **Termination.** Each honest party gives an output.
(2) **Agreement.** All honest parties give the same output.
(3) **Validity**. If all honest parties have the same input, this must be their output.

THE BYZANTINE AGREEMENT PROBLEM (INFORMAL)

There are n parties (generals), of which at most f are dishonest. Each starts with their own input. Must satisfy:

(1) **Termination.** Each honest party gives an output.
(2) **Agreement.** All honest parties give the same output.
(3) **Validity**. If all honest parties have the same input, this must be their output.

For clarity, suppose the honest generals know f (but not which are the dishonest generals).

THE BYZANTINE AGREEMENT PROBLEM (INFORMAL)

There are n parties (generals), of which at most f are dishonest. Each starts with their own input. Must satisfy:

(1) **Termination.** Each honest party gives an output.
(2) **Agreement.** All honest parties give the same output.
(3) **Validity**. If all honest parties have the same input, this must be their output.

If $f = 0$ then the problem is trivial (just do majority vote).

THE BYZANTINE AGREEMENT PROBLEM (INFORMAL)

There are n parties (generals), of which at most f are dishonest. Each starts with their own input. Must satisfy:

(1) **Termination.** Each honest party gives an output.
(2) **Agreement.** All honest parties give the same output.
(3) **Validity**. If all honest parties have the same input, this must be their output.

If $f = 0$ then the problem is trivial (just do majority vote).

So, it is clear that one of the basic questions we should be interested in is "what values of n and f can a protocol handle?".

THE BYZANTINE AGREEMENT PROBLEM (INFORMAL)

There are n parties (generals), of which at most f are dishonest. Each starts with their own input. Must satisfy:

(1) **Termination.** Each honest party gives an output.
(2) **Agreement.** All honest parties give the same output.
(3) **Validity**. If all honest parties have the same input, this must be their output.

Not possible if $f \geq n/2$.

THE BYZANTINE AGREEMENT PROBLEM (INFORMAL)

There are n parties (generals), of which at most f are dishonest. Each starts with their own input. Must satisfy:

(1) **Termination.** Each honest party gives an output.
(2) **Agreement.** All honest parties give the same output.
(3) **Validity**. If all honest parties have the same input, this must be their output.

Not possible if $f \geq n/2$.

Attack

Retreat

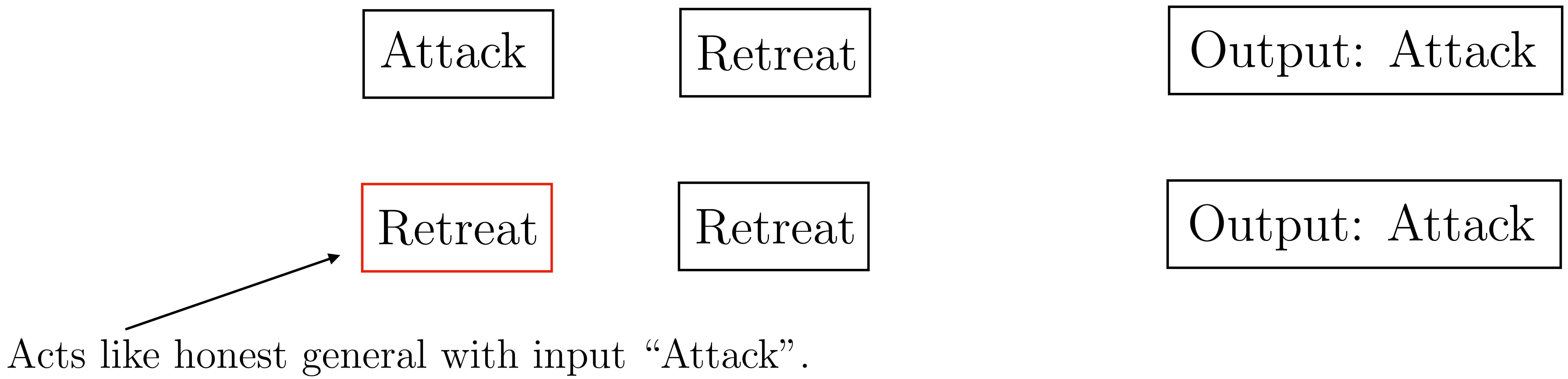
Output: Attack

THE BYZANTINE AGREEMENT PROBLEM (INFORMAL)

There are n parties (generals), of which at most f are dishonest. Each starts with their own input. Must satisfy:

(1) **Termination.** Each honest party gives an output.
(2) **Agreement.** All honest parties give the same output.
(3) **Validity**. If all honest parties have the same input, this must be their output.

Not possible if $f \geq n/2$.



THE BYZANTINE AGREEMENT PROBLEM (INFORMAL)

Is it trivial when $f < n/2$? Can we not just implement a majority vote argument?

THE BYZANTINE AGREEMENT PROBLEM (INFORMAL)

Is it trivial when $f < n/2$? Can we not just implement a majority vote argument?

The problem with this approach stems from the allowed form of communication between generals (which is intended to accurately reflect communication between processors in real world scenarios).

THE BYZANTINE AGREEMENT PROBLEM (INFORMAL)

Is it trivial when $f < n/2$? Can we not just implement a majority vote argument?

If the generals were standing in a circle and shouting out their votes – so that everybody can see who is shouting out a vote and any vote heard by a single honest general is immediately heard by all – then a simple majority vote approach would work. In the setting described above, however, communication occurs by messenger between one pair of generals at a time.

THE BYZANTINE AGREEMENT PROBLEM (INFORMAL)

Is it trivial when $f < n/2$? Can we not just implement a majority vote argument?

The problem now is that dishonest generals can tell different things to different generals. Suppose $n = 3$ and $f = 1$. One honest general initially wants to attack, while the other wants to retreat. If the dishonest general sends a 'retreat' message to the general who wants to retreat and an 'attack' message to the general who wants to attack, then the honest generals will see different majority votes.

THE BYZANTINE AGREEMENT PROBLEM (INFORMAL)

Is it trivial when $f < n/2$? Can we not just implement a majority vote argument?

In fact, we will see that (under a natural formalisation of the informal problem above) the Byzantine Agreement problem is not solvable when $f \geq n/3$ unless we endow the generals with certain extra abilities. So the problem is not as trivial as it might initially seem.

THE FORMAL FRAMEWORK

The setup:

- We formalise each general as a processor.

THE FORMAL FRAMEWORK

The setup:

- We formalise each general as a processor.
- The execution of the protocol is divided into discrete timeslots, beginning at time $t = 0$. At each time t, each processor receives a certain set of messages from other processors, and then carries out a finite set of instructions to decide what messages to send to other processors at that timeslot.

THE FORMAL FRAMEWORK

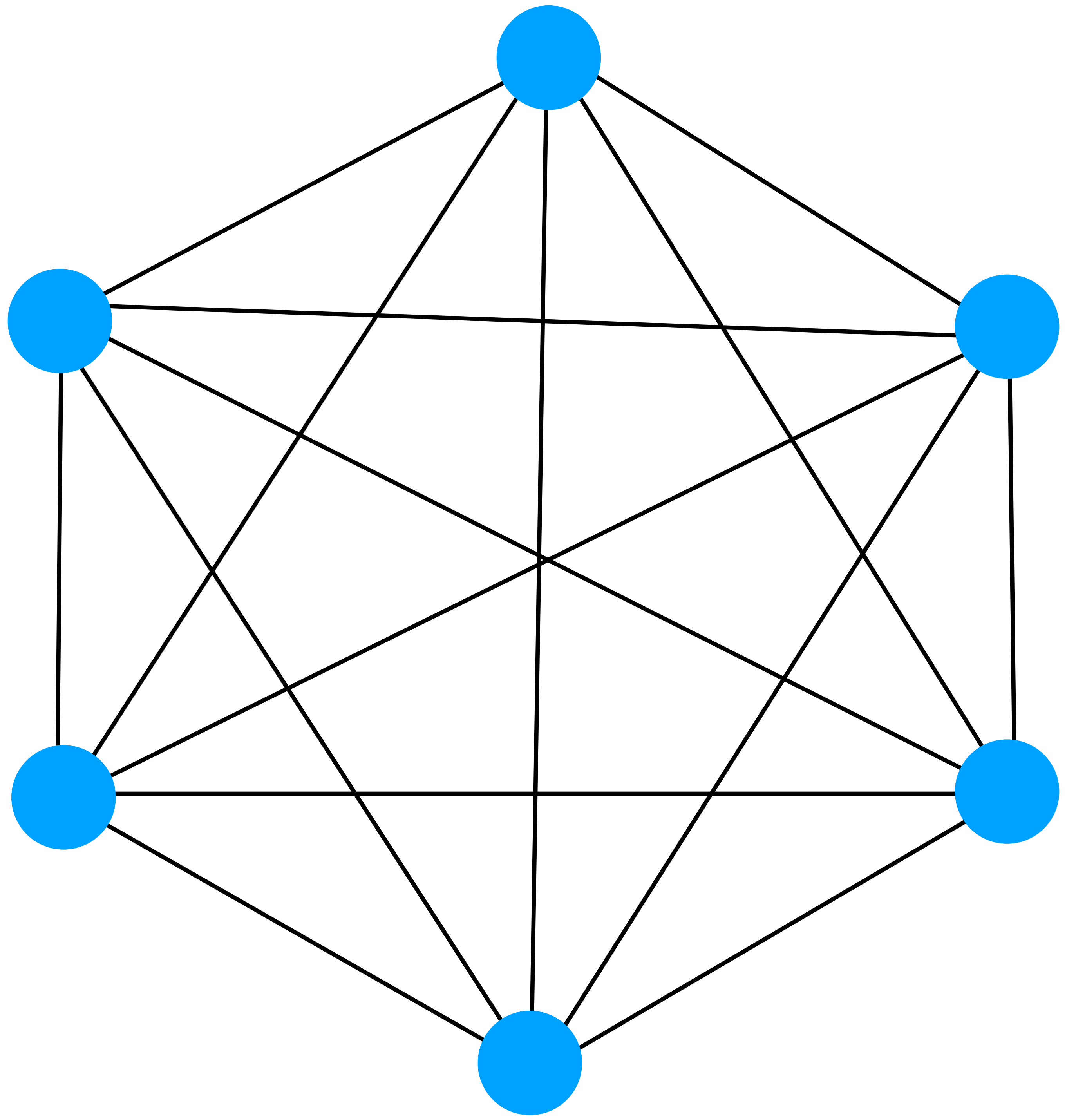
The setup:

- We formalise each general as a processor.
- The execution of the protocol is divided into discrete timeslots, beginning at time $t = 0$. At each time t, each processor receives a certain set of messages from other processors, and then carries out a finite set of instructions to decide what messages to send to other processors at that timeslot.
- There are n processors given names 0 to $n-1$. Each processor is told n as well as their own name i, i.e. this information is given as part of their input.

THE FORMAL FRAMEWORK

Authenticated channels. There exists a two-way authenticated communication channel $\{i, j\}$ for each pair of distinct processors i and j:

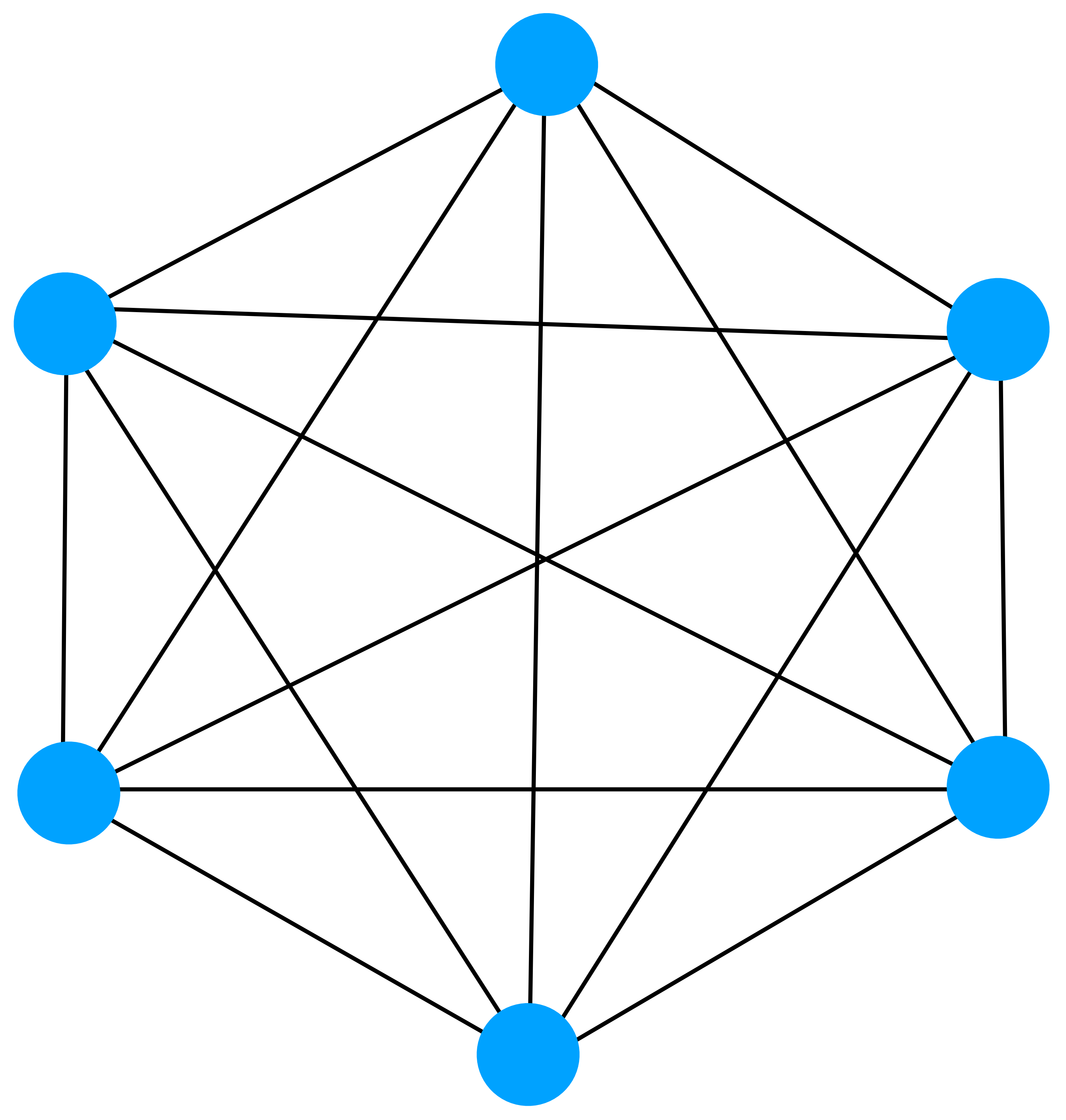
- Only i can send messages to j and only j can send messages to i on the channel $\{i, j\}$, and;



THE FORMAL FRAMEWORK

Authenticated channels. There exists a two-way authenticated communication channel $\{i, j\}$ for each pair of distinct processors i and j:

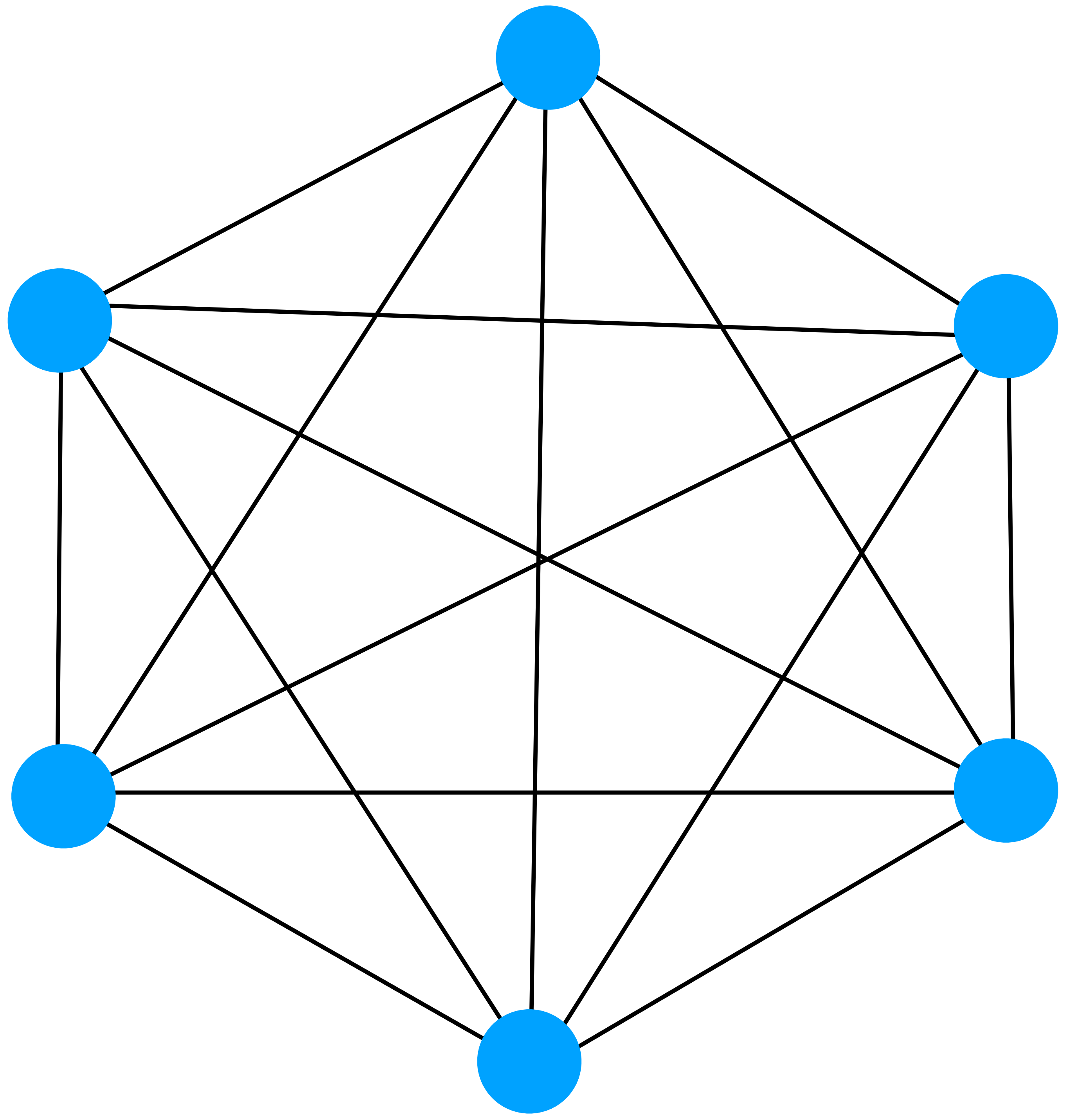
- Only i can send messages to j and only j can send messages to i on the channel $\{i, j\}$, and;
- When i receives messages it is aware of the channel by which the messages were sent, i.e. the instructions for i can depend not only on the messages received at any given timeslot but also which channels the messages arrived on.



THE FORMAL FRAMEWORK

Authenticated channels. There exists a two-way authenticated communication channel $\{i, j\}$ for each pair of distinct processors i and j:

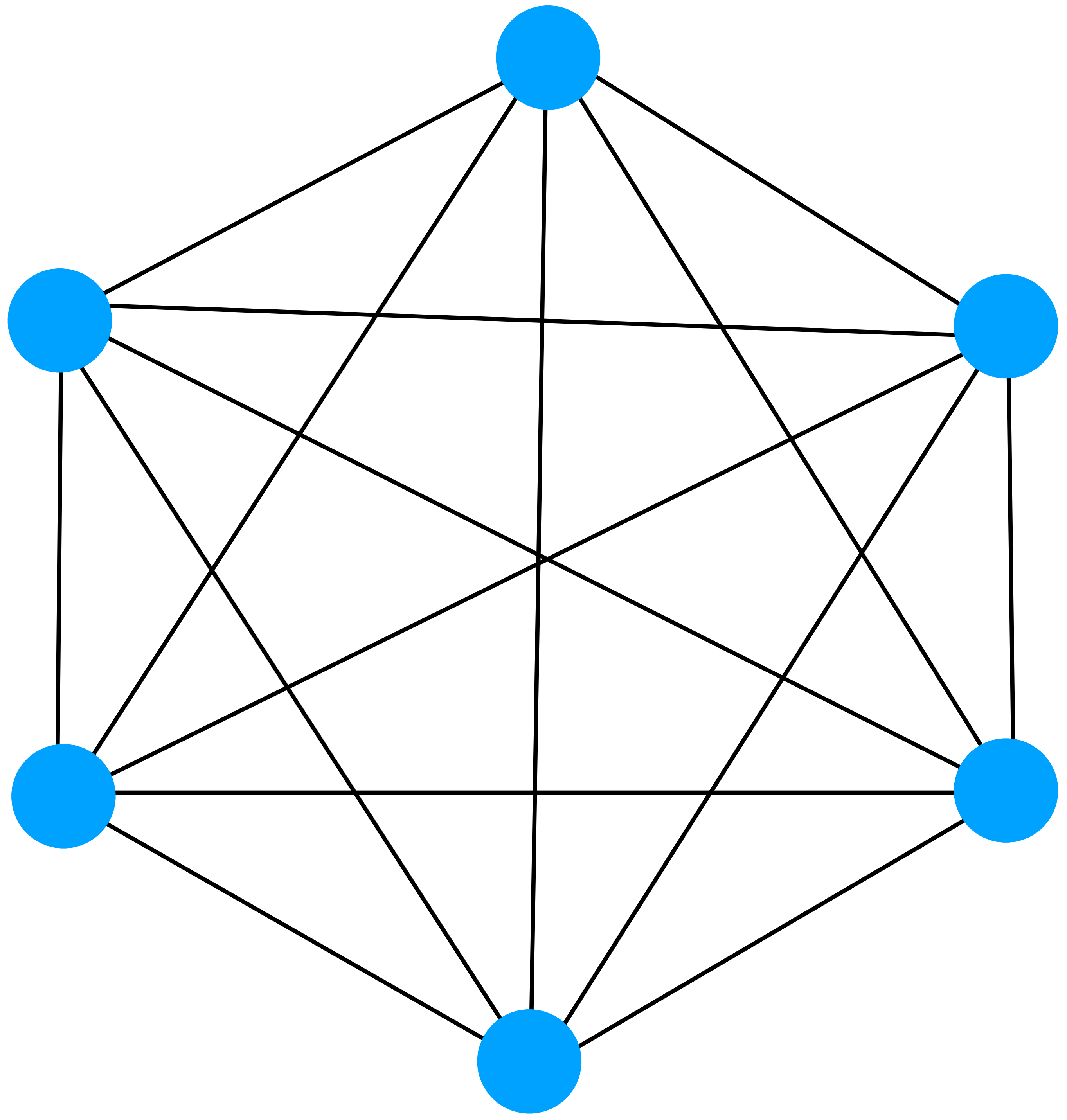
- Only i can send messages to j and only j can send messages to i on the channel $\{i, j\}$, and;
- When i receives messages it is aware of the channel by which the messages were sent, i.e. the instructions for i can depend not only on the messages received at any given timeslot but also which channels the messages arrived on.



At each timeslot, the instructions for processor i determine which messages it should send along each of its channels $\{i, j\}$.

THE FORMAL FRAMEWORK

Public Key Infrastructure (PKI). Sometimes we'll assume given a PKI, sometimes not. If given a PKI, this means each processor is provided with a $(\mathrm{sk}, \mathrm{pk})$ pair, and is told the public key of each of the other processors.



THE FORMAL FRAMEWORK

Message delay and the synchronous setting. To keep things simple, we start by considering what is known as the *synchronous* setting. This means that if i sends j a message at time t then j receives that message from i at time $t + 1$.

THE FORMAL FRAMEWORK

How can dishonest generals behave?

- Some of the processors may be *faulty*.

THE FORMAL FRAMEWORK

How can dishonest generals behave?

- Some of the processors may be *faulty*.
- Each processor is given an upper bound f for the number of faulty processors as part of its input.

THE FORMAL FRAMEWORK

How can dishonest generals behave?

- Some of the processors may be *faulty*.
- Each processor is given an upper bound f for the number of faulty processors as part of its input.
- Generally, we are most interested in analysing settings where the faulty processors can display arbitrary (and potentially malicious) behaviour. In this case, we say that the processors display *Byzantine* faults. Formally, this means that faulty processors can execute any arbitrary program.

THE FORMAL FRAMEWORK

How can dishonest generals behave?

- Some of the processors may be *faulty*.
- Each processor is given an upper bound f for the number of faulty processors as part of its input.
- Generally, we are most interested in analysing settings where the faulty processors can display arbitrary (and potentially malicious) behaviour. In this case, we say that the processors display *Byzantine* faults. Formally, this means that faulty processors can execute any arbitrary program.

Crash faults. Sometimes we will also be interested in a more benign form of faulty behaviour known as *crash faults*. In the crash fault setting, faulty processors must follow the protocol precisely until such a point as they crash, whereupon they execute no further instructions.

Byzantine Agreement (BA) and Byzantine Broadcast (BB)

Previously, we considered a binary version of the Byzantine Agreement Problem. Sometimes convenient to consider a more general form of the problem:

Byzantine Agreement (BA) and Byzantine Broadcast (BB)

Previously, we considered a binary version of the Byzantine Agreement Problem. Sometimes convenient to consider a more general form of the problem:

- We consider a set of n processors, of which at most f display Byzantine faults.

Byzantine Agreement (BA) and Byzantine Broadcast (BB)

Previously, we considered a binary version of the Byzantine Agreement Problem. Sometimes convenient to consider a more general form of the problem:

- We consider a set of n processors, of which at most f display Byzantine faults.
- For some set V, each processor is given an input in V (different processors potentially receiving different inputs). V is told to the processors and could be of any finite size ≥ 2.

Byzantine Agreement (BA) and Byzantine Broadcast (BB)

Previously, we considered a binary version of the Byzantine Agreement Problem. Sometimes convenient to consider a more general form of the problem:

- We consider a set of n processors, of which at most f display Byzantine faults.
- For some set V, each processor is given an input in V (different processors potentially receiving different inputs). V is told to the processors and could be of any finite size ≥ 2.
- The protocol must satisfy the following conditions:
  - **Termination**. All non-faulty processors must give an output in V.
  - **Agreement**. All non-faulty processors must give the same output.
  - **Validity**. If all non-faulty processors have the same input v, then v must be their common output.

Byzantine Agreement (BA) and Byzantine Broadcast (BB)

The Byzantine Broadcast Problem. In the original papers in which Lamport, Shostak and Pease introduced the Byzantine Agreement problem, they actually focussed on a variant of the problem which is now known as *Byzantine Broadcast*:

Byzantine Agreement (BA) and Byzantine Broadcast (BB)

The Byzantine Broadcast Problem. In the original papers in which Lamport, Shostak and Pease introduced the Byzantine Agreement problem, they actually focussed on a variant of the problem which is now known as *Byzantine Broadcast*:

- We consider a set of n processors, of which at most f display Byzantine faults.

Byzantine Agreement (BA) and Byzantine Broadcast (BB)

The Byzantine Broadcast Problem. In the original papers in which Lamport, Shostak and Pease introduced the Byzantine Agreement problem, they actually focussed on a variant of the problem which is now known as *Byzantine Broadcast*:

- We consider a set of n processors, of which at most f display Byzantine faults.
- One processor is designated the 'broadcaster'. All processors are given the name of the broadcaster.

Byzantine Agreement (BA) and Byzantine Broadcast (BB)

The Byzantine Broadcast Problem. In the original papers in which Lamport, Shostak and Pease introduced the Byzantine Agreement problem, they actually focussed on a variant of the problem which is now known as *Byzantine Broadcast*:

- We consider a set of n processors, of which at most f display Byzantine faults.
- One processor is designated the ‘broadcaster’. All processors are given the name of the broadcaster.
- The broadcaster is given an input in some set V. The set V is told to all processors.

Byzantine Agreement (BA) and Byzantine Broadcast (BB)

The Byzantine Broadcast Problem. In the original papers in which Lamport, Shostak and Pease introduced the Byzantine Agreement problem, they actually focussed on a variant of the problem which is now known as *Byzantine Broadcast*:

- We consider a set of n processors, of which at most f display Byzantine faults.
- One processor is designated the 'broadcaster'. All processors are given the name of the broadcaster.
- The broadcaster is given an input in some set V. The set V is told to all processors.
- The protocol must satisfy the following conditions:
  - **Termination**. All non-faulty processors must give an output in V.
  - **Agreement**. All non-faulty processors must give the same output.
  - **Validity**. If the broadcaster is not faulty and has input v, then all non-faulty processors must output v.

Byzantine Agreement (BA) and Byzantine Broadcast (BB)

What is the relationship between the Byzantine Agreement (BA) problem and the Byzantine Broadcast (BB) problem? We saw BA cannot be solved when $f \geq n/2$. It is easy to see, though, that the same argument doesn't apply to BB – in fact, we'll see that, if a PKI is given and we work in the synchronous setting, then BB can actually be solved for any number of faulty processors.

Byzantine Agreement (BA) and Byzantine Broadcast (BB)

What is the relationship between the Byzantine Agreement (BA) problem and the Byzantine Broadcast (BB) problem? We saw BA cannot be solved when $f \geq n/2$. It is easy to see, though, that the same argument doesn't apply to BB – in fact, we'll see that, if a PKI is given and we work in the synchronous setting, then BB can actually be solved for any number of faulty processors.

So, there are certainly scenarios in which BB can be solved although BA cannot be.

Byzantine Agreement (BA) and Byzantine Broadcast (BB)

On the other hand, if we work in the synchronous setting and if $f < n/2$ then the two problems reduce to each other quite easily:

- If we can solve BB, then to solve BA we have all processors broadcast their inputs using the protocol for BB (meaning that we carry out n simultaneous executions of BB). Once a value is decided corresponding to each processor, processors then decide by majority vote, breaking ties in some previously arranged but arbitrary fashion.

Byzantine Agreement (BA) and Byzantine Broadcast (BB)

On the other hand, if we work in the synchronous setting and if $f < n/2$ then the two problems reduce to each other quite easily:

- If we can solve BB, then to solve BA we have all processors broadcast their inputs using the protocol for BB (meaning that we carry out n simultaneous executions of BB). Once a value is decided corresponding to each processor, processors then decide by majority vote, breaking ties in some previously arranged but arbitrary fashion.
- If we can solve BA, then to solve BB we have the broadcaster send their input to all other processors at time 0. Each processor then takes the value received at time 1 as their input value, choosing some arbitrary value in V if no value is received from the broadcaster. We then have the processors carry out the protocol for BA on those input values.

BA and BB: Synchronous setting with PKI

We'll prove the following:

Theorem. *Consider the synchronous setting with PKI given. There exists a protocol that solves the Byzantine Broadcast problem for any number of faulty processors.*

BA and BB: Synchronous setting with PKI

We'll prove the following:

Theorem. *Consider the synchronous setting with PKI given. There exists a protocol that solves the Byzantine Broadcast problem for any number of faulty processors.*

This also deals with BA. By the reductions discussed before, the theorem also suffices to show that we can solve BA when working in the synchronous setting with PKI iff $f < n/2$.

BA and BB: Synchronous setting with PKI

Why isn't it trivial?

- An obvious way to try solving BB when given a PKI would be to have the broadcaster send out signed values of their input to each of the other processors.

BA and BB: Synchronous setting with PKI

Why isn't it trivial?

- An obvious way to try solving BB when given a PKI would be to have the broadcaster send out signed values of their input to each of the other processors.
- The processors could then repeatedly share all of the signed values they have seen produced by the broadcaster.

BA and BB: Synchronous setting with PKI

Why isn't it trivial?

- An obvious way to try solving BB when given a PKI would be to have the broadcaster send out signed values of their input to each of the other processors.
- The processors could then repeatedly share all of the signed values they have seen produced by the broadcaster.
- If they only ever see a single value produced, then they output that value. If they ever see two different values produced, then they realise the broadcaster is faulty, so they give some 'default' value as output.

BA and BB: Synchronous setting with PKI

Why isn't it trivial?

- An obvious way to try solving BB when given a PKI would be to have the broadcaster send out signed values of their input to each of the other processors.
- The processors could then repeatedly share all of the signed values they have seen produced by the broadcaster.
- If they only ever see a single value produced, then they output that value. If they ever see two different values produced, then they realise the broadcaster is faulty, so they give some 'default' value as output.
- If the broadcaster is non-faulty then they will only produce a single signed value and all non-faulty processors will output that. If the broadcaster produces two different signed values and shows them to non-faulty processors then (the idea is) everyone will eventually see those values and give the default output.

BA and BB: Synchronous setting with PKI

Why isn't it trivial?

- An obvious way to try solving BB when given a PKI would be to have the broadcaster send out signed values of their input to each of the other processors.
- The processors could then repeatedly share all of the signed values they have seen produced by the broadcaster.
- If they only ever see a single value produced, then they output that value. If they ever see two different values produced, then they realise the broadcaster is faulty, so they give some 'default' value as output.

The problem. When should processors stop sharing values and terminate? If they share until time t, then the adversary can choose to show one signed value to all non-faulty processors until time t, and then show some subset of the non-faulty processors a second signed value at time t (when it is too late to share anymore), causing the 'agreement' requirement of BB to be violated.

BA and BB: Synchronous setting with PKI

The trick. What we need is a clever mechanism to ensure that if any non-faulty processor ‘recognises’ a certain signed value produced by the broadcaster, then all non-faulty processors will also ‘recognise’ that value.

BA and BB: Synchronous setting with PKI

The trick. What we need is a clever mechanism to ensure that if any non-faulty processor 'recognises' a certain signed value produced by the broadcaster, then all non-faulty processors will also 'recognise' that value.

That way, either they all recognise a single value and give that as output, or they all recognise multiple values and so give the default output.

BA and BB: Synchronous setting with PKI

The mechanism described by Dolev and Strong is quite elegant:

- At time 0 the broadcaster sends signed versions of their input to each processor.

BA and BB: Synchronous setting with PKI

The mechanism described by Dolev and Strong is quite elegant:

- At time 0 the broadcaster sends signed versions of their input to each processor.
- At time 1, the processors look to see whether they have received a signed value from the broadcaster, and if so then they 'recognise' that value. Now though, rather than just passing on that signed value, they attach their own signature to the message so that now it has been signed twice – first by the broadcaster and then secondly by them. Then they send this new version of the message to all processors.

BA and BB: Synchronous setting with PKI

The mechanism described by Dolev and Strong is quite elegant:

- At time 0 the broadcaster sends signed versions of their input to each processor.
- At time 1, the processors look to see whether they have received a signed value from the broadcaster, and if so then they 'recognise' that value. Now though, rather than just passing on that signed value, they attach their own signature to the message so that now it has been signed twice – first by the broadcaster and then secondly by them. Then they send this new version of the message to all processors.
- Then we stipulate that if a processor is to 'recognise' a new value at any time t, the message must have been signed by t distinct processors. If they recognise a new value at time t, then they add their signature to the list and send that message (now with $t+1$ distinct signatures) to all other processors.

BA and BB: Synchronous setting with PKI

The mechanism described by Dolev and Strong is quite elegant:

- At time 0 the broadcaster sends signed versions of their input to each processor.
- Then we stipulate that if a processor is to 'recognise' a new value at any time t, the message must have been signed by t distinct processors. If they recognise a new value at time t, then they add their signature to the list and send that message (now with $t+1$ distinct signatures) to all other processors.

BA and BB: Synchronous setting with PKI

The mechanism described by Dolev and Strong is quite elegant:

- At time 0 the broadcaster sends signed versions of their input to each processor.
- Then we stipulate that if a processor is to 'recognise' a new value at any time t, the message must have been signed by t distinct processors. If they recognise a new value at time t, then they add their signature to the list and send that message (now with $t+1$ distinct signatures) to all other processors.
- At time $f+1$ we give the processors a last chance to recognise new values (but not to share again) before either outputting the single value they have recognised or else a default value.

BA and BB: Synchronous setting with PKI

Why does this approach work? We have to show that if any non-faulty processor recognises a certain value $v \in V$, then all non-faulty processors will also recognise that value. There are two cases to consider:

BA and BB: Synchronous setting with PKI

Why does this approach work? We have to show that if any non-faulty processor recognises a certain value $v \in V$, then all non-faulty processors will also recognise that value. There are two cases to consider:

- **Case 1**. Suppose that some non-faulty i first recognises v at a time $t < f + 1$. In this case, i receives a message relaying the value v at time t which has t distinct signatures attached. Processor i then adds their signature to form a message with $t + 1$ distinct signatures and sends this message to all processors. This means all non-faulty processors will recognise v by time $t + 1$ $(\leq f + 1)$.

BA and BB: Synchronous setting with PKI

Why does this approach work? We have to show that if any non-faulty processor recognises a certain value $v \in V$, then all non-faulty processors will also recognise that value. There are two cases to consider:

- **Case 1**. Suppose that some non-faulty i first recognises v at a time $t < f + 1$. In this case, i receives a message relaying the value v at time t which has t distinct signatures attached. Processor i then adds their signature to form a message with $t + 1$ distinct signatures and sends this message to all processors. This means all non-faulty processors will recognise v by time $t + 1$ ($\leq f + 1$).
- **Case 2**. Suppose next that some non-faulty i first recognises v at time $f + 1$. In this case, i receives a message relaying the value v at timeslot $f + 1$ which has $f + 1$ distinct signatures attached. At least one of those signatures must be from a non-faulty processor j (since there are at most f faulty processors), meaning that Case 1 applies w.r.t. j.

BACK TO THAT PUZZLE...

Consider the synchronous setting, byzantine failures, authenticated channels, PKI. The question is, ‘can we describe a deterministic protocol solving BB and in which each party only speaks once?’

BACK TO THAT PUZZLE...

Consider the synchronous setting, byzantine failures, authenticated channels, PKI. The question is, 'can we describe a deterministic protocol solving BB and in which each party only speaks once?'

(1) If you consider the binary version of BB, it's easy. Just run DS, but only with respect to one of the two possible values (0 say). Then, either every honest processor 'recognises' 0, or none do. If any honest processor recognises 0, it outputs 0, otherwise it outputs 1.

BACK TO THAT PUZZLE...

Consider the synchronous setting, byzantine failures, authenticated channels, PKI. The question is, 'can we describe a deterministic protocol solving BB and in which each party only speaks once?'

(1) If you consider the binary version of BB, it's easy. Just run DS, but only with respect to one of the two possible values (0 say). Then, either every honest processor 'recognises' 0, or none do. If any honest processor recognises 0, it outputs 0, otherwise it outputs 1.

(2) Then extending it to the general case is also easy, if one doesn't care about the number of stages. Run DS for the first value of V first. If you 'recognise' the first value at the end of that, stop there and output.

    If not, then that means no honest processor (other than maybe the broadcaster) has spoken yet. So, run DS for the second value, and so on.

State Machine Replication

Very roughly, SMR is the problem that blockchain protocols are designed to solve: Clients send in a sequence of transactions of their choosing and the processors implementing the SMR protocol have to agree on an order in which to implement those transactions.

State Machine Replication

Even though Bitcoin exists, there is considerable interest in permissioned SMR protocols:

- Could be interested in implementation in the permissioned setting.

State Machine Replication

Even though Bitcoin exists, there is considerable interest in permissioned SMR protocols:

- Could be interested in implementation in the permissioned setting.
- Permissioned protocols can be implemented as permissionless protocols (PoS) of a sort, and may be more efficient.

State Machine Replication

Even though Bitcoin exists, there is considerable interest in permissioned SMR protocols:

- Could be interested in implementation in the permissioned setting.
- Permissioned protocols can be implemented as permissionless protocols (PoS) of a sort, and may be more efficient.
- Can function in the partially synchronous setting (where message delivery is less reliable).

State Machine Replication

A metric of interest is the *communication* compexity: How many bits have to be exchanged per new block of transactions:

State Machine Replication

A metric of interest is the *communication* compexity: How many bits have to be exchanged per new block of transactions:

- Typically, the instructions are divided into 'views' (rounds), with a different leader suggesting a block of transactions for agreement in each view.

State Machine Replication

A metric of interest is the *communication* compexity: How many bits have to be exchanged per new block of transactions:

- Typically, the instructions are divided into 'views' (rounds), with a different leader suggesting a block of transactions for agreement in each view.
- $O(n)$ communication complexity inside each view was known, but;
- Best known complexity per view change was $O(n^2)$.

State Machine Replication

A metric of interest is the *communication* compexity: How many bits have to be exchanged per new block of transactions:

- Typically, the instructions are divided into 'views' (rounds), with a different leader suggesting a block of transactions for agreement in each view.
- $O(n)$ communication complexity inside each view was known, but;
- Best known complexity per view change was $O(n^2)$.
- New method gives $O(n)$ view changes.

Joint work with Ittai Abraham

The underlying protocol. We suppose view synchronisation is required for some underlying protocol (such as Hotstuff) with the following properties:

- Instructions are divided into views. Each view v has a designated *leader*, denoted $\texttt{lead}(v)$.
- If the honest processors spend long enough in a view with an honest leader when network conditions are good, the view will complete successfully, and produce a 'certificate' (QC) signed by $n - f$ processors.

The underlying protocol. We suppose view synchronisation is required for some underlying protocol (such as Hotstuff) with the following properties:

- Instructions are divided into views. Each view v has a designated *leader*, denoted $\texttt{lead}(v)$.
- If the honest processors spend long enough in a view with an honest leader when network conditions are good, the view will complete successfully, and produce a 'certificate' (QC) signed by $n - f$ processors.

The view synchronisation task:

- We have to ensure that all non-faulty processors eventually spend long enough in the same view that it completes successully.
- We also want a protocol which is *optimistically responsive*: i.e. can go as fast as the network can handle.

Clock-times. To synchronise processors while maintaining optimistic responsiveness, we have a predetermined 'clock-time' corresponding to each view: The clock-time corresponding to view v is $t_v := \Gamma v$. At certain points in the execution, a processor may instantaneously forward their clock to some clock-time t_v and enter view v.

Clock-times. To synchronise processors while maintaining optimistic responsiveness, we have a predetermined 'clock-time' corresponding to each view: The clock-time corresponding to view v is $t_v := \Gamma v$. At certain points in the execution, a processor may instantaneously forward their clock to some clock-time t_v and enter view v.

The safety condition. We want to ensure that when processors forward their clocks a certain *safety condition* is maintained: For some fixed Γ, and for each honest processor, at least f other honest processors have their clocks at most Γ behind.

Clock-times. To synchronise processors while maintaining optimistic responsiveness, we have a predetermined 'clock-time' corresponding to each view: The clock-time corresponding to view v is $t_v := \Gamma v$. At certain points in the execution, a processor may instantaneously forward their clock to some clock-time t_v and enter view v.

The safety condition. We want to ensure that when processors forward their clocks a certain *safety condition* is maintained: For some fixed Γ, and for each honest processor, at least f other honest processors have their clocks at most Γ behind.

This will allow us to achieve view synchronisation. When an honest processor enters a new view v, they send a message to the leader telling them. Once the leader receives $f+1$ of these, it combines these into a 'start view v' message, telling other processors to start the view. The condition above means this will happen in sufficient time.

HOW DO WE MAINTAIN THE SAFETY CONDITION?

We only forward a clock to t in two cases:

- We see attestations from $n - f$ processors that they are at most Γ behind t (we see a QC for the previous view).

HOW DO WE MAINTAIN THE SAFETY CONDITION?

We only forward a clock to t in two cases:

- We see attestations from $n - f$ processors that they are at most Γ behind t (we see a QC for the previous view).
- We see attestations from $f + 1$ processors that their clock is $\geq t$ (we see a message saying we should start view v).

Inductively, it's easy to see that the safety condition will never be violated.

Thanks for listening!