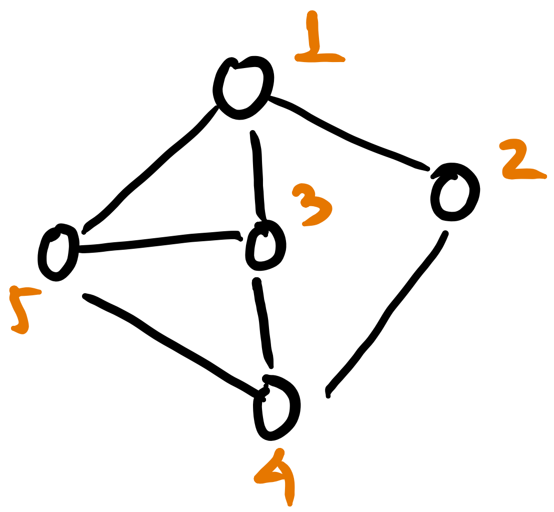
POIS-19

Protocols for multi party communication

- for $n > 3t$, we have perfectly secure schemes such as BGW88, GRR98 etc.
- for $n > 2t$, we also assume that we have a physical broadcast channel (unlike $n > 3t$ there are no protocols for reliable broadcasting), secrecy is shown statistically, with protocols such as RB89
- for $n > t$, we have computational proofs for secrecy (running against high scale computers to show that it takes extremely long to break the secrecy),with protocols such as GMN87

Dealing with incomplete networks

So far, all of the problems we have seen involve a complete communication graph, meaning that every node can communicate with every other node, however for most practical settings, this is not true. We can represent this communication through an undirected graph as follows



Here, nodes i and j can communicate, only if the edge (i, j) exists in the graph. Hence nodes 1 and 3 can communicate but nodes 2 and 3 cannot communicate with each other **directly** (they can obviously communicate via intermediate nodes).

In order to show that a particular protocol is secure, we need to show that the protocol can simulate a TTP (Trusted Third Party). What are the necessary and sufficient conditions to simulate a TTP on an incomplete communication graph

The conditions

We can show that the necessary and sufficient conditions for a multiparty communication protocol with n nodes (out of which t are byzantine corrupt) are :

- $n > 3t$

- The communication graph G is $(2t+1)$ - connected
  meaning that we can have a proper communication protocol if and only if these two conditions are satisfied.

Proof

Using Menger's Theorem, which says that a graph is k connected if and only if every pair of distinct nodes, is connected by k node disjoint paths (two paths are node disjoint if they do not share any vertex in common)

We first prove that if a graph is $2t+1$- connected, then there does exist a reliable communication protocol.

By Menger's Theorem, we can say that between any two non-faulty processes i and j, there exist $2t+1$ node disjoint paths. If we want to send a message from i to j, we can send the message along these $2t+1$ paths. Since only t paths are corrupted (have a corrupt node). Therefore, we can reliably reconstruct the message at node j by just taking the majority of the messages recieved.

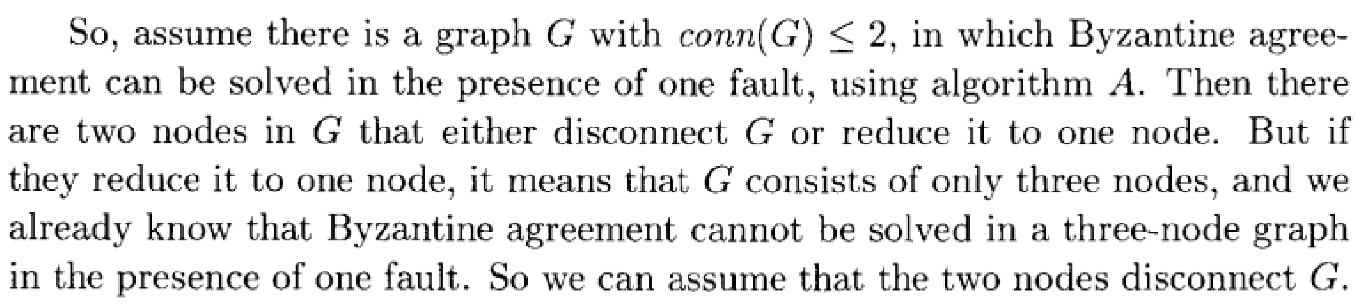
We now prove that if graph is $< 2t+1$ - connected, then there does not exist a reliable communication protocol (Assuming that $n > 3t$) [LynchBook 6.5]

```
not sure about kannan's proof, Lynchbook has a formal proof, so using that for
now
```

Main Proof

Note that any algorithm that solves the byzantine agreement problem, has three correctness conditions (meaning that it is a valid algorithm if and only if it satisfies these conditions)

- **Agreement** : All non-faulty nodes, must end up at the same value.
- **Validity** : If all non-faulty nodes started with the same initial value v, then v will also be the final value for these nodes
- **Termination** : All nodes must end up at a single value after finite time.



So, assume there is a graph G with $conn(G) \leq 2$, in which Byzantine agreement can be solved in the presence of one fault, using algorithm A. Then there are two nodes in G that either disconnect G or reduce it to one node. But if they reduce it to one node, it means that G consists of only three nodes, and we already know that Byzantine agreement cannot be solved in a three-node graph in the presence of one fault. So we can assume that the two nodes disconnect G.

Therefore the graph looks something like this

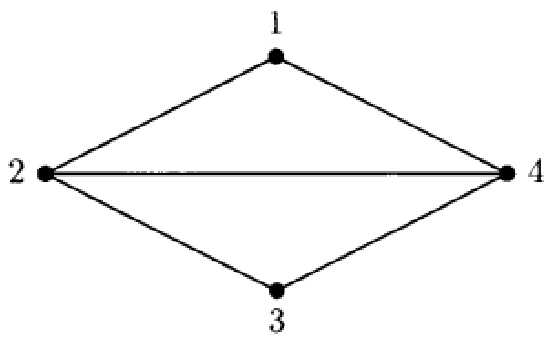


Figure 6.11: A graph G with $conn(G) = 2$.

Then the picture must be something like Figure 6.11, except that nodes 1 and 3 might be replaced by arbitrary connected subgraphs and there might be several edges between each of processes 2 and 4 and each of the two connected subgraphs. (The link between 2 and 4 could also be missing, but this would only make things harder.) Again for simplicity, we just consider the case where 1 and 3 are single nodes. We construct a system S by combining two copies of A. We start one copy of each process with input value 0 and the other with input value 1, as shown in Figure 6.12. As in the proof of Lemma 6.26, S with the given input assignment does exhibit some well-defined behavior. Again, we will obtain a contradiction by showing that no such behavior is possible.

The graph S looks as follows

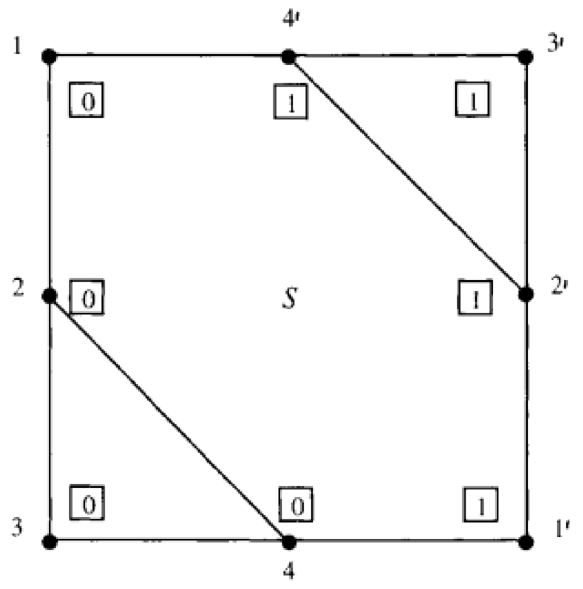


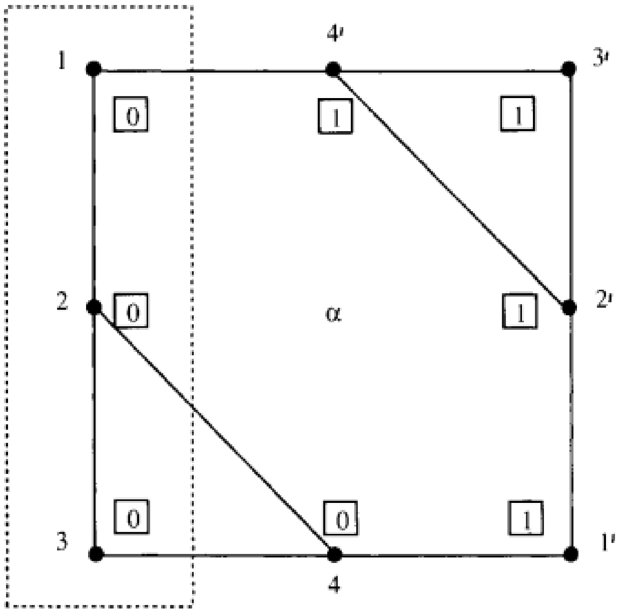
Figure 6.12: Combining two copies of A to get S.

Notice that this works because from the perspective of a particular node, they have the same set of connections (Note that nodes 4 and 4' are identical). Hence, for vertex 1, it is connected to 2 and 4 (4'), vertex 2 is connected to 3,4,1, similarly vertex 3 is connected to 2 and 4, and vertex 4 is connected to 2,3,1(1'). Same argument extends for the other vertices as well.

So suppose that the processes in S are started with the input values indicated in Figure 6.12, that is, the unprimed processes with 0 and the primed processes with 1; let α be the resulting execution of S.

We consider α from the point of view of processes 1, 2, and 3. To these processes, it appears as if they are running in system A, in an execution α_1 in which process 4 is faulty. See Figure 6.13. Then the correctness conditions for Byzantine agreement imply that eventually in α_1, processes 1, 2, and 3 must decide 0. Since α is indistinguishable from α_1 to processes 1, 2, and 3, all three must eventually decide 0 in α as well.

From viewpoint of 1,2,3 : 1 says that he recieved a 0-bit from 2 and 1-bit from 4(4'). On the other hand, 2 and 3 say that they recieved a 0-bit from 4. This would lead to the consensus that 4 is the faulty node since it is propagating different values to different nodes. Hence the non-faulty nodes in the subsystem are 1,2,3. Now because of validity condition on correctness, they should all terminate with a final value of 0 (they all had initial values of 0)



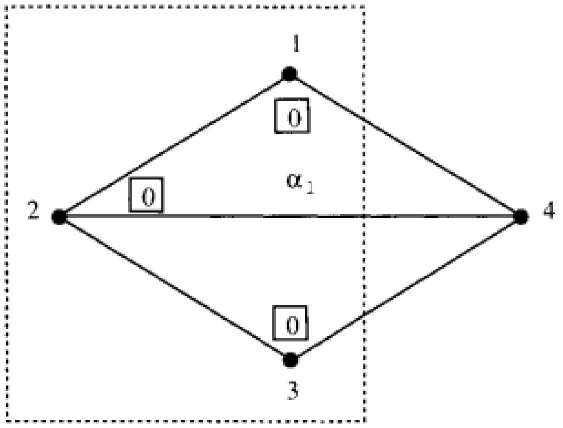
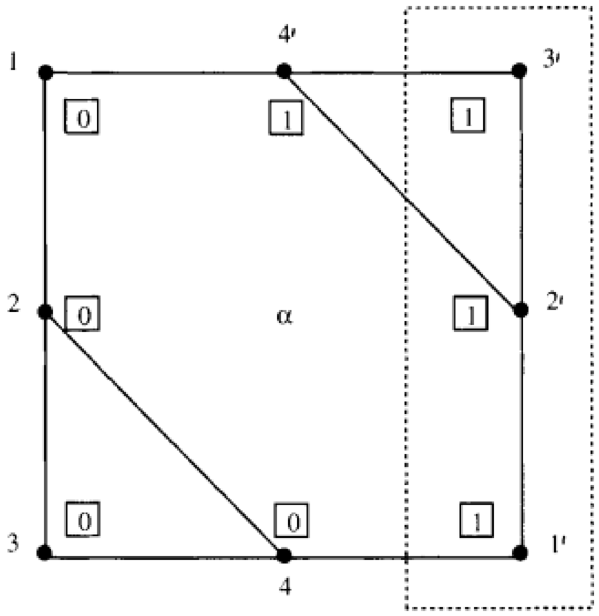


Figure 6.13: Executions α and α_1 are indistinguishable to processes 1, 2, and 3.

Next consider α from the point of view of processes $1'$, $2'$, and $3'$. To these three processes, it appears as if they are running in A, in an execution α_2 in which process 4 is faulty. See Figure 6.14. By the same argument, processes $1'$, $2'$, and $3'$ must eventually decide 1 in α.

Same argument goes for the right side as well, 1' says that he recieved a 0-bit from 4 . However 2' and 3' say that they recieved a 1-bit from 4(4') meaning that 4 is again the faulty node. By validity, 1',2' and 3' must end up with 1 as their final value.



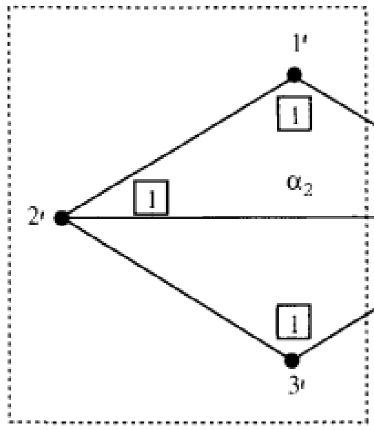
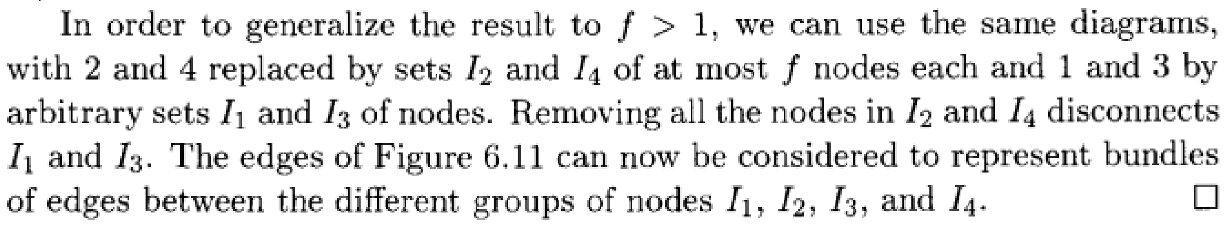


Figure 6.14: Executions α and α_2 are indistinguishable to processes $1'$, $2'$, and $3'$.

Finally, consider execution α from the point of view of processes 3, 4, and $1'$. To these processes, it appears as if they are running in A, in an execution α_3 in which process 2 is faulty. See Figure 6.15. By the correctness conditions for Byzantine agreement, these three processes must eventually decide in α_3, and their decisions must be the same. Then the same is true in α.

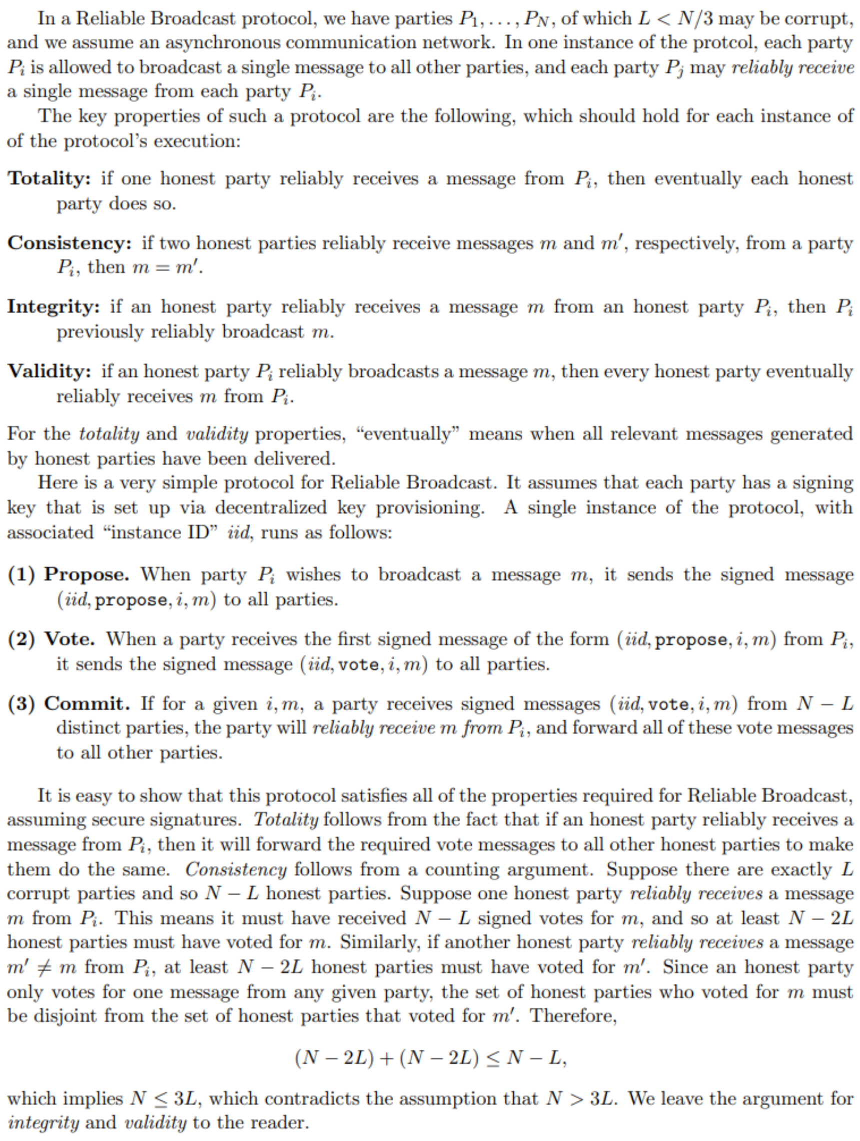
Similar argument extends for the top/bottom side as well. In this case vertex 2 would be the faulty vertex, and hence by agreement condition on correctness, verticews 3,4, and 1' must end up on the same value. However we previously showed that vertex 3 ends on 0-bit and vertex 1' ends on a 1-bit. Therefore there is no way they can have the same value. Hence we have a contradiction. Therefore there does not exist any protocol which can solve the byzantine agreement problem if the graph is $< (2t+1)$ - connected.



In order to generalize the result to $f > 1$, we can use the same diagrams, with 2 and 4 replaced by sets I_2 and I_4 of at most f nodes each and 1 and 3 by arbitrary sets I_1 and I_3 of nodes. Removing all the nodes in I_2 and I_4 disconnects I_1 and I_3. The edges of Figure 6.11 can now be considered to represent bundles of edges between the different groups of nodes I_1, I_2, I_3, and I_4. $\square$

Reliable Broadcasting

```
Refer to A Graduate Course in Applied Cryptography Page 1083
```



In a Reliable Broadcast protocol, we have parties $P_1, \ldots, P_N$, of which $L < N/3$ may be corrupt, and we assume an asynchronous communication network. In one instance of the protcol, each party P_i is allowed to broadcast a single message to all other parties, and each party P_j may *reliably receive* a single message from each party P_i.

The key properties of such a protocol are the following, which should hold for each instance of of the protocol's execution:

Totality: if one honest party reliably receives a message from P_i, then eventually each honest party does so.

Consistency: if two honest parties reliably receive messages m and m', respectively, from a party P_i, then $m = m'$.

Integrity: if an honest party reliably receives a message m from an honest party P_i, then P_i previously reliably broadcast m.

Validity: if an honest party P_i reliably broadcasts a message m, then every honest party eventually reliably receives m from P_i.

For the *totality* and *validity* properties, "eventually" means when all relevant messages generated by honest parties have been delivered.

Here is a very simple protocol for Reliable Broadcast. It assumes that each party has a signing key that is set up via decentralized key provisioning. A single instance of the protocol, with associated "instance ID" *iid*, runs as follows:

(1) Propose. When party P_i wishes to broadcast a message m, it sends the signed message $(iid, \texttt{propose}, i, m)$ to all parties.

(2) Vote. When a party receives the first signed message of the form $(iid, \texttt{propose}, i, m)$ from P_i, it sends the signed message $(iid, \texttt{vote}, i, m)$ to all parties.

(3) Commit. If for a given i, m, a party receives signed messages $(iid, \texttt{vote}, i, m)$ from $N - L$ distinct parties, the party will *reliably receive m from* P_i, and forward all of these vote messages to all other parties.

It is easy to show that this protocol satisfies all of the properties required for Reliable Broadcast, assuming secure signatures. *Totality* follows from the fact that if an honest party reliably receives a message from P_i, then it will forward the required vote messages to all other honest parties to make them do the same. *Consistency* follows from a counting argument. Suppose there are exactly L corrupt parties and so $N - L$ honest parties. Suppose one honest party *reliably receives* a message m from P_i. This means it must have received $N - L$ signed votes for m, and so at least $N - 2L$ honest parties must have voted for m. Similarly, if another honest party *reliably receives* a message $m' \neq m$ from P_i, at least $N - 2L$ honest parties must have voted for m'. Since an honest party only votes for one message from any given party, the set of honest parties who voted for m must be disjoint from the set of honest parties that voted for m'. Therefore,

$$(N - 2L) + (N - 2L) \leq N - L,$$

which implies $N \leq 3L$, which contradicts the assumption that $N > 3L$. We leave the argument for *integrity* and *validity* to the reader.