**Introduction**

Our project for this semester was to implement a distributed agreement algorithm. As assigned, we were to implement Lamport’s Paxos algorithm [1] incorporating Garg’s weighted Byzantine algorithm [2]. The first step in this implementation was to implement Paxos, then Byzantizing Paxos in a manner similar to what Lamport describes in [3], and finally incorporating Garg’s weights to the Byzantized Paxos. The result is a Byzantized Paxos algorithm that can tolerate a weight of w < 1/3 of failed processes.

**Paxos**

Paxos as described by Lamport is a distributed, non-Byzantine fault tolerant consensus algorithm. It accounts for benign failures, but not malicious (Byzantine) faults. The model assumed is therefore an asynchronous, non-Byzantine model.

You begin with a set of processes which can propose values, and single value amongst the proposed must be chosen. If no value is proposed, no value is chosen, and if a value is chosen, all processes can learn of the selection. This leads to the below safety requirements per Lamport:

1. Only a value that has been proposed may be chosen
2. Only a single value is chosen
3. A process never learns that a value has been chosen unless it actually has been.

Lamport does not specify liveness requirements other than that some value proposed is eventually chosen and that if a value is chosen, a process can eventually learn of the selection.

We will not delve into the derivation of the algorithm, and will be operating off of a modified version of Paxos which Lamport refers to as PCon in [3]. As in vanilla Paxos, there are three classes of agents, *proposers*, *acceptors*, and *learners*. The actions they perform should be self-explanatory. Paxos performs numbered ballots, each of which is orchestrated by a leader (the proposer). If N is the number of acceptors (which can include the proposer itself), where N > f, a quorum us any N – f acceptors. A simple way to require that any two quorums have a non-empty intersection (required for safety), we require that N > 2f. If a quorum of acceptors vote for a value, then that value is considered chosen.

In the language of the algorithm, a proposer can make a proposal ballot number *b*(proposal number in [1]) and value *v*. The proposal number is used to determine which proposal an acceptor will accept, and the value is the value which will be accepted. There are two-phases to the commit, *prepare* and *accept*.

The below properties follow the requirements of the algorithm as per PCon. Those familiar with the vanilla Paxos algorithm will notice several small differences.

P1. An acceptor can vote for a value *v* in ballot *b* only if *v* is safe at *b*

P2. Different acceptors cannot vote for different values in the same ballot.

P3a. If no acceptor in the quorum has voted in a ballot numbered less than *b*, then all values are safe at *b*

P3b. If some acceptor in the quorum has voted, let *c* be the highest-numbered ballot less than *b* in which such a vote was cast. The value voted for in ballot *c* is safe at *b*. (By P2, there is only one such value.)

The vanilla Paxos algorithm is as below. The modifications to create PCon as mentioned above are in red.

**Phase 1a** The ballot-*b* leader sends a 1*a* message to the acceptors.

**Phase 1b** An acceptor respond’s to the leader’s ballot-*b* 1*a* message with a 1*b* message containing the number of the highest-numbered ballot in which it has voted and the value it voted for in that ballot, or saying that it has cast no votes.

**Phase 1c** Using the 1*b* messages from a quorum of acceptors, the leader chooses a set of values that are safe at *b* and sends a 1*c* message for each of those values.

**Phase 2a** Using the 1*b* messages sent by a quorum of acceptors, the leader chooses a value *v* that is safe at *b* and sends a 2*a* message containing *v* to the acceptors.

**Phase 2a** The leader sends a 2a message for some value for which it has sent a 1c message.

**Phase 2b** Upon receipt of the leader’s ballot-*b* 2*a* message, an acceptor votes for *v* in ballot *b* by sending a 2*b* message.

**Byzantizing Paxos**

Byzantizing Paxos requires several modifications. The first is that a simple majority quorum is no longer satisfactory. Instead, we require what Lamport refers to as a Byzquorum. Say that you have a set of *N* processes which would normally require a quorum *q* in order to choose a value. However, you are operating under the assumption that you now have at most *f* Byzantine acceptors which may act in a malicious manner. To guarantee that you still have a valid quorum, you must now have a Byzquorum of *q* + *f* byzacceptors. This satisfies P3a above, but P3b is an issue. This is because there is no method of determining if a single message is from a real or Byzantine acceptor. We get around this by assuming that N > 3f, which means that any two quorums have at least f + 1 acceptors in common. This leads to

P3a’. If there is no ballot numbered less than *b* in which f + 1 acceptros have voted, then all values are safe at *b.*

P3b’. If there is some ballot *c* in which acceptors have voted and there is no higher-numbered ballot less than *b* in which *f* + 1 acceptors have voted, then the value *v* voted for in *c* is safe at *b*

This leads to a requirement of more than 4f acceptors, which Lamport gets around with the modifications to Paxos resulting in PCon.

**Incorporating Weighted Byzantine Agreement**

**Results**

**Conclusion**