*Package Utiles*

*P2PSocket*Implements the underlying communication model between the servers.  
Every P2PSocket holds 5 port for communication, and execute an infinite loop in which it accept requests from the servers and stores them in an inner buffer.

When the server tried to read from the socket it reads the massages from within the buffer.

*Messenger*

A middleware between the server and the socket.

Every Messenger holds 2 *P2PSockets* one for the acceptor massages of the server and the other to the leader massages (As every server is acceptor including the leader).

The Messenger pass to the server only the massages that relate to the current Paxos instance and therefore simplify the implementation of the Paxos algorithm.

*Peer*

Represent a peer of a servers.

The class contains an address, and 10 ports for the P2PSockets of the server.

*JsonSerializer*

A simple serializer which accept an instance of an arbitrary class ad generate a String Json from it.

It also able to deserialize a given object back to an instance of it’s class.

*Config*

A simple static class which holds all the configuration data in a public static variables.

That way the configuration is accessible from every point of the project without any more efforts.

*SystemUtiles*

Implements a basic functions which initiate the logging system and the Config class.

*ZooKeeperClient*

Implements the very primitive ZooKeeper API function that are needed to our system.

*LeaderFailureDetector*

Implements a Leader Failure Detector Ω using the ZooKeeper service.

Ω is implements in the following way:  
every process creates an ephemeral sequential node which contains it’s ID, and set a watcher for changes. At the starting of a server it register to Ω and elect the server with the minimal sequential number as a leader.

If the leader fell then the watcher is fired and elect the next leader among the remaining nodes.

Note: the LeaderFailureDetector is implements under the \ELECTION directory in the ZooKeeper service.

*MembershipDetector*

In order to know how participate the protocol without pre-configuration to each server.

The method is quite the same as the LeaderFailureDetector:

Using ZooKeeper, every server creates an ephemeral sequential with the following data inside it:

*[serverID, serverAddress, leaderPorts[5], acceptorePorts[5]]* where *[leader\acceptor]Ports* is an array of the 5 ports of the corresponding P2PSocket.

Note: the MembershipDetector is implemented under the \MEMBERS ZooKeeper directory.

*Package DataTypes*

*Block*

Represent a block in the blockchain.

List of transaction.

Every block contains:

* The list of transaction helps us to perform a batching on the transactions in order to decide on more than one transaction each time the algorithm runs.
* The previous block hash – in our implementation due to the non-byzantine assumption and in order to simplify the system we use it as a block number.

*Transaction*

*Address*

*Package Paxos*

*Package Paxos.PaxosMsgs*

The package contains the data type which are represent the Paxos massages.

*PaxosMsg*

A base class for all the types of paxos massages.

It contains the following fields:

* type – the massage type.
* serverID – the Id of the sender.
* serverAddr – the address of the sender.
* round – the paxos instance number.

*PrepareMsg*

Contain the additional fields:

* r – the r of paxos.

*PromiseMsg*

Contains the additional fields:

* acc\_r – the r that accepted from the leader.
* ack – ACK\NACK answer.
* lastGoodRound – the last\_good\_round of paxos.
* block – the proposal of the server.

*AcceptMsg*

Contains the additional fields:

* r – the r of paxos.
* block – the proposal the leader choose.

*AcceptedMsg*

Contains the additional fields:

* r – the r of paxos
* ack – ACK\NACK answer

*CommitMsg*

Contains the additional fields:

* block – the value to decide

*PaxosDecisions*

A class which represents a snapshot of decision of the current paxos instance. It required for our improvement of the broadcast operation of the commit phase.

*Paxos*

Contains the main logic of our paxos implementation.

The class expose a *propose* function which accept a block and runs our improved paxos algorithm.

While running the function a process mainly perform the following operations:

* Every 100 ms it check if it is a leader, if so it start the leader phase.
* Else it asks from the messenger service to deliver an acceptors massages and for each type of massage invoke the corresponding phase.
* If a massage from a previous round of paxos has accepted it answer to it with the appropriate data which is taken from the corresponding PaxosDecision.
* If the process is the leader it runs the leader phase with our improvement which take a sub group of non-contradict propose and unite them to a sub chain.