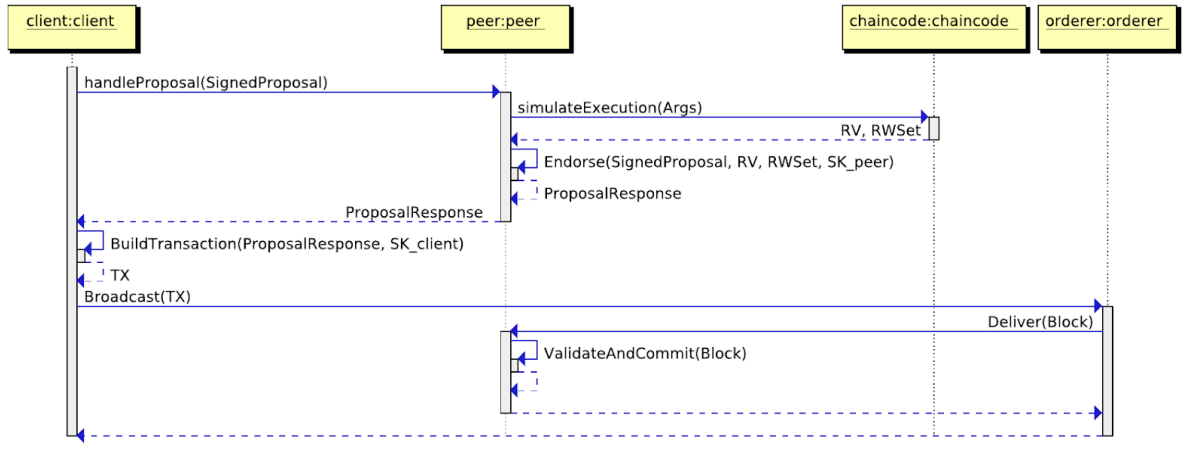
An informal description of the endorsement protocol of Hyperledger Fabric

Alessandro Sorniotti

This document describes the protocol used in fabric to modify state. The protocol operates as shown in the figure below.



The flow involves the following steps:

1. The client sends a `SignedProposal` to the peer
2. The peer invokes the appropriate chaincode supplying the arguments provided by the client and obtains a `RWSet` (i.e. a proposed update of the world state) and a return value
3. The peer endorses the execution and its results, creates a `ProposalResponse` and sends it back to the client
4. The client creates a transaction and sends it for ordering
5. The ordering server orders the transaction
6. The peer receives an ordered block, validates it and commits it

Step 1

Initially the client creates a `SignedProposal` message, which is defined as follows

```
SignedProposal {
      Proposal Proposal {
            ChannelHeader ChannelHeader {
                  ChannelId string
                  ChaincodeName string
                  ChaincodeVersion string
                  TxId string
            }

            SignatureHeader SignatureHeader {
                  Creator []byte
                  Nonce []byte
            }

            Args [][]byte
      }

      Signature []byte
}
```

Notable fields include:

- `ChannelId` Is the name of the fabric channel that is the target of this transaction; a channel is a way of creating more than one logical instance of a ledger/world state and chaincode namespace within a single physical instance of fabric.
- `ChaincodeName` and `ChaincodeVersion` are the name and version of the chaincode that is being invoked by this proposal.
- `TxId` Is the transaction identifier, computed as the hash of `SignatureHeader`.
- `Creator` is the serialized identity of the client.
- `Nonce` is an array of random bytes.
- `Args` is the set of arguments for this chaincode invocation.
- `Signature` is a signature of the client over the `Proposal`.

Step 2

In this step, the peer contacts the chaincode docker container and requests it to execute based on the supplied arguments (`Args`). The chaincode executes and may perform `GetState` calls to read the current world state. Any call to modify world state (`PutState` or `DeleteState`) does not affect world state; it is instead added to the `RWSet` that keeps track of the changes to the world state that this proposal would require if committed. The chaincode returns to the peer the `RWSet` and a return value.

Step 3

The peer uses the proposal, its signing key and the results of the chaincode execution to generate a `ProposalResponse` message, which is defined as follows

```
ProposalResponse {
      ProposalResponsePayload ProposalResponsePayload {
            ProposalHash []byte
            ChaincodeName string
            ChaincodeVersion string
            RWSet []byte
            RV []byte
      }

      Endorsement Endorsement {
            Endorser []byte
            Signature []byte
      }
}
```

Notable fields include:

- `ProposalHash` is the hash of the `Proposal` message.
- `Endorser` is the serialized identity of the peer.
- `Signature` is a signature over `ProposalResponsePayload` and `Endorser`.

Step 4

The client uses the ProposalResponse and its signing key to create a `Transaction` message, which is defined as follows

```
Transaction {
      Payload Payload {
            ChannelHeader ChannelHeader {
                  ChannelId string
                  ChaincodeName string
                  ChaincodeVersion string
                  TxId string
            }

            SignatureHeader SignatureHeader {
                  Creator []byte
                  Nonce []byte
            }

            TransactionAction TransactionAction {
```

```
                Args [][]byte

                ProposalResponsePayload ProposalResponsePayload {
                      ProposalHash []byte
                      ChaincodeName string
                      ChaincodeVersion string
                      RWSet []byte
                      RV []byte
                }

                Endorsements []Endorsement {
                      Endorser []byte
                      Signature []byte
                }
          }
      }

      Signature []byte
}
```

Notable fields include:

- `ChannelHeader` and `SignatureHeader` are supposed to be the same as in the `Proposal` message.
- Args is supposed to be the set of arguments originally supplied to the chaincode invocation.
- `ProposalResponsePayload` comes from the field of the same name in the `ProposalResponse` message.
- The `Endorsements` array is filled from the field of the same name in the `ProposalResponse` message. Note that more than one endorsement can be collected from as many `ProposalResponse` messages.

Step 5

The orderer uses its internal algorithms to create an ordered sequence of transactions.

Step 6

The peer receives a `Block` message from the orderer, which is defined as follows

```
Block {
      BlockHeader BlockHeader {
            Number       uint64
            PreviousHash []byte
            DataHash     []byte
      }
```

```
      OrdererSignature OrdererSignature {
            SignatureHeader SignatureHeader {
                  Creator []byte
                  Nonce []byte
            }

            Signature       []byte
      }

      Transactions []Transaction {
            Payload Payload {
                  ChannelHeader ChannelHeader {
                        ChannelId string
                        ChaincodeName string
                        ChaincodeVersion string
                        TxId string
                  }

                  SignatureHeader SignatureHeader {
                        Creator []byte
                        Nonce []byte
                  }

                  TransactionAction TransactionAction {
                        Args [][]byte

                        ProposalResponsePayload ProposalResponsePayload {
                              ProposalHash []byte
                              ChaincodeName string
                              ChaincodeVersion string
                              RWSet []byte
                              RV []byte
                        }

                        Endorsements []Endorsement {
                              Endorser []byte
                              Signature []byte
                        }
                  }
            }

            Signature []byte
      }
}
```

Notable fields include:

- `Number` is a block counter.
- `PreviousHash` is the hash of the previous block.
- `DataHash` is the hash of the data portion of this block.
- `OrdererSignature` contains a signature from the orderer over BlockHeader and `OrdererSignature.SignatureHeader`.
- `Transactions` is an array of `Transaction` messages.

The peer performs a number of security checks, including:

- There is no other committed transaction in the ledger with the same `TxId`.
- The `Endorsements` array complies with the endorsement policy of the specified chaincode on the specified channel.
- Each of the signatures in `Endorsements[i].Signature` is a valid signature from `Endorsements[i].Endorser` over `ProposalResponsePayload` and `Endorser`
- `ProposalHash` is the hash of the concatenation of `Payload.ChannelHeader`, `Payload.SignatureHeader` and `Payload.TransactionAction.Args`.

Appendix

Sequence diagram sources (http://sdedit.sourceforge.net)

```
client:client
peer:peer
chaincode:chaincode
orderer:orderer

client:ProposalResponse=peer.handleProposal(SignedProposal)
peer:RV, RWSet=chaincode.simulateExecution(Args)
peer:ProposalResponse=peer.Endorse(SignedProposal, RV, RWSet, SK_peer)

client:TX=client.BuildTransaction(ProposalResponse, SK_client)&
client:orderer.Broadcast(TX)
orderer:peer.Deliver(Block)
peer:peer.ValidateAndCommit(Block)
```