This is a specification of the server synchronisation for *Focustro*.

I am not very experienced with TLA+, so this shouldn't be considered good style. Let me know if you think this can be improved.

MODULE *consistency*

EXTENDS *Integers*, *Sequences*, *TLC*

CONSTANTS *ClientIds*, *ObjectIds*, *PropNames*, *Values*, *NULL*, *MaxWrites*, *MaxNetworkFailures*
VARIABLES *clientStates*, *serverState*, *usedIds*, *writeCount*, *networkFailures*

$Init \triangleq$
Clients start with an empty set of objects and a *NULL lastTimestamp*
$\land\ clientStates = [$
$clientId \in ClientIds \mapsto$
$[$
$objects \mapsto [objectId \in ObjectIds \mapsto NULL],$
$writeQueue \mapsto \langle\rangle,$
$lastTimestamp \mapsto NULL,$
$inbox \mapsto \langle\rangle$
$]$
$]$
The server starts with an empty set of objects and a 0 timestamp
$\land\ serverState = [$
$objects \mapsto [objectId \in ObjectIds \mapsto NULL],$
$timestamp \mapsto 0,$
$inbox \mapsto \langle\rangle$
$]$
Simulate *UUIDs* by keeping track of used *IDs*
$\land\ usedIds = \{\}$
Keep track of the number of writes to limit the state space to *MaxWrites*
$\land\ writeCount = 0$
Keep track of the number of failures to limit the state space to *MaxNetworkFailures*
$\land\ networkFailures = 0$

Client actions

A client creates a new object using a unique *ID*
$CreateObject(clientId,\ objectId,\ object) \triangleq$
$\land\ writeCount < MaxWrites$
$\land\ objectId \notin usedIds$
$\land\ clientStates[clientId].objects[objectId] = NULL$
$\land\ clientStates' = [$
$clientStates$ EXCEPT
$![clientId].objects[objectId] = object,$
queue a 'create' operation
$![clientId].writeQueue = Append(@,$
$[$

```
                    writeId ↦ writeCount,
                    op ↦ "create",
                    id ↦ objectId,
                    object ↦ object
                ]
            )
        ]
    ∧ usedIds′ = usedIds ∪ {objectId}
    ∧ writeCount′ = writeCount + 1
    ∧ UNCHANGED ⟨serverState, networkFailures⟩

 A client updates a property of an existing object
ModifyProperty(clientId, objectId, property, value) ≜
    ∧ writeCount < MaxWrites
    ∧ clientStates[clientId].objects[objectId] ≠ NULL
    ∧ clientStates′ = [
            clientStates EXCEPT
            ![clientId].objects[objectId][property] = value,
             queue an 'update' operation
            ![clientId].writeQueue = Append(@,
                [
                    writeId ↦ writeCount,
                    op ↦ "update",
                    id ↦ objectId,
                    property ↦ property,
                    value ↦ value
                ]
            )
        ]
    ∧ writeCount′ = writeCount + 1
    ∧ UNCHANGED ⟨usedIds, serverState, networkFailures⟩


 A client should periodically send a sync request to the server
ClientSend(clientId) ≜
     To limit the state space, only send one message at a time to the server
    ∧ serverState.inbox = ⟨⟩
     Don't send if there are messages waiting to be processed from the server
    ∧ clientStates[clientId].inbox = ⟨⟩
    ∧ ∨ clientStates[clientId].writeQueue ≠ ⟨⟩
      ∨ clientStates[clientId].lastTimestamp ≠ serverState.timestamp
    ∧ serverState′ = [
            serverState EXCEPT
            !.inbox = Append(@, [
                clientId ↦ clientId,
                lastTimestamp ↦ clientStates[clientId].lastTimestamp,
```

```
                write ↦
                    IF clientStates[clientId].writeQueue ≠ ⟨⟩
                        THEN Head(clientStates[clientId].writeQueue)
                        ELSE NULL
            ])
        ]
    ∧ UNCHANGED ⟨clientStates, usedIds, writeCount, networkFailures⟩

The client receives a response from the server
ClientRecv(clientId) ≜
    ∧ clientStates[clientId].inbox ≠ ⟨⟩
    ∧ clientStates′ =
        LET msg ≜ Head(clientStates[clientId].inbox)
            writes ≜ clientStates[clientId].writeQueue
            check(write) ≜ write.writeId ≠ msg.ack
        IN [
            clientStates EXCEPT
            ![clientId].inbox = Tail(@),
            ![clientId].writeQueue = SelectSeq(writes, check),
            ![clientId].lastTimestamp = msg.timestamp,
            ![clientId].objects = [
                objectId ∈ ObjectIds ↦
                    IF objectId ∈ DOMAIN msg.updates
                        THEN msg.updates[objectId]
                        ELSE @[objectId]
            ]
        ]
    ∧ UNCHANGED ⟨serverState, usedIds, writeCount, networkFailures⟩

The client loses a response from the server
ClientLoseMessage(clientId) ≜
    ∧ clientStates[clientId].inbox ≠ ⟨⟩
    ∧ networkFailures < MaxNetworkFailures
    ∧ clientStates′ = [clientStates EXCEPT ![clientId].inbox = Tail(@)]
    ∧ networkFailures′ = networkFailures + 1
    ∧ UNCHANGED ⟨serverState, usedIds, writeCount⟩

Server helpers

ApplyWrite(write, serverObjects, timestamp) ≜
    IF write = NULL THEN serverObjects ELSE
    IF write.op = "create"
        THEN [serverObjects EXCEPT ![write.id] = [object ↦ write.object, updated ↦ timestamp]]
        ELSE [serverObjects EXCEPT ![write.id] = [
            object ↦ [@.object EXCEPT ![write.property] = write.value],
            updated ↦ timestamp
```

```
        ]]

UpdatedObjects(serverObjects, clientTimestamp) ≜
    IF clientTimestamp = NULL
     THEN LET S ≜ {objectId ∈ ObjectIds : serverObjects[objectId] ≠ NULL}
          IN  [objectId ∈ S ↦ serverObjects[objectId].object]
     ELSE LET S ≜ {objectId ∈ ObjectIds : ∧ serverObjects[objectId] ≠ NULL
                                          ∧ serverObjects[objectId].updated > clientTimestamp}
          IN  [objectId ∈ S ↦ serverObjects[objectId].object]

Server actions

The server receives a message from a client
ServerRecv ≜
    ∧ serverState.inbox ≠ ⟨⟩
    ∧ LET msg ≜ Head(serverState.inbox)
          newTimestamp ≜ IF msg.write = NULL
                         THEN serverState.timestamp
                         ELSE serverState.timestamp + 1
      IN
       ∧ serverState' = [
            serverState EXCEPT
            !.objects = ApplyWrite(msg.write, serverState.objects, newTimestamp),
            !.inbox = Tail(@),
            !.timestamp = newTimestamp
         ]
       ∧ clientStates' = [
            clientStates EXCEPT
            ![msg.clientId].inbox = Append(@,
               [
                  timestamp ↦ newTimestamp,
                  ack ↦ IF msg.write = NULL THEN NULL ELSE msg.write.writeId,
                  updates ↦ UpdatedObjects(serverState'.objects, msg.lastTimestamp)
               ]
            )
         ]
    ∧ UNCHANGED ⟨usedIds, writeCount, networkFailures⟩

The server loses a message from a client
ServerLoseMessage ≜
    ∧ serverState.inbox ≠ ⟨⟩
    ∧ networkFailures < MaxNetworkFailures
    ∧ serverState' = [
            serverState EXCEPT
            !.inbox = Tail(@)
      ]
```

$\land$ $networkFailures' = networkFailures + 1$
$\land$ UNCHANGED $\langle clientStates,\ usedIds,\ writeCount \rangle$

System actions

$$
\begin{aligned}
Next \triangleq\ & \lor\ \exists\, clientId \in ClientIds : \\
& \quad \exists\, objectId \in ObjectIds : \\
& \quad \exists\, object \in [PropNames \to Values] : \\
& \quad CreateObject(clientId,\ objectId,\ object) \\
& \lor\ \exists\, clientId \in ClientIds : \\
& \quad \exists\, objectId \in ObjectIds : \\
& \quad \exists\, prop \in PropNames : \\
& \quad \exists\, value \in Values : \\
& \quad ModifyProperty(clientId,\ objectId,\ prop,\ value) \\
& \lor\ \exists\, clientId \in ClientIds : ClientSend(clientId) \\
& \lor\ \exists\, clientId \in ClientIds : ClientRecv(clientId) \\
& \lor\ \exists\, clientId \in ClientIds : ClientLoseMessage(clientId) \\
& \lor\ ServerRecv \\
& \lor\ ServerLoseMessage
\end{aligned}
$$

$AllVars \triangleq \langle clientStates,\ usedIds,\ serverState,\ writeCount,\ networkFailures \rangle$

Temporal properties

Assume that the client and server will eventually send/receive requests, even if the network is temporarily down

$$
\begin{aligned}
Fairness \triangleq\ & \\
& \land\ \forall\, clientId \in ClientIds : \\
& \quad \land\ \mathrm{WF}_{AllVars}(ClientRecv(clientId)) \\
& \quad \land\ \mathrm{WF}_{AllVars}(ClientSend(clientId)) \\
& \land\ \mathrm{WF}_{AllVars}(ServerRecv)
\end{aligned}
$$

$Spec \triangleq Init \land \Box[Next]_{AllVars} \land Fairness$

Properties to be checked

$$
\begin{aligned}
Consistent \triangleq\ & \\
& \exists\, clientId1 \in ClientIds : \\
& \forall\, clientId2 \in ClientIds : \\
& clientStates[clientId1].objects = clientStates[clientId2].objects
\end{aligned}
$$

$$
\begin{aligned}
Properties \triangleq\ & \\
& \Diamond\Box\, Consistent
\end{aligned}
$$