**Assignment 2**

1. **Completion of the missing parts in the code:**
2. Node.find\_block\_to\_back\_up():

def find\_block\_to\_back\_up(self):

for block\_id, (held\_locally, peer) in enumerate(zip(self.local\_blocks, self.backed\_up\_blocks)):

if held\_locally and peer is None:

return block\_id

This method looks for this node blocks which are not saved locally in our node (held\_locally) and are not backed up in the other nodes (peer)

1. Node.schedule\_next\_upload():

def schedule\_next\_upload(self, sim: Backup):

assert self.online # we can only upload if we're online

# we don't want to upload if we're already uploading something

if self.current\_upload is not None:

return

# 1. first find if we have a backup that a remote node needs

for peer, block\_id in self.remote\_blocks\_held.items():

# peer.online - if the remote peer is online\

# peer.current\_download is None - if the remote peer is not downloading anything

# not peer.local\_blocks[block\_id] - if the remote peer does not have his block locally

if peer.online and peer.current\_download is None and not peer.local\_blocks[block\_id]: # added

sim.schedule\_transfer(self, peer, block\_id, restore=True) # added

return # it means that that peer lost the block and we are uploading it to him

# 2. if other nodes do not require repair,

# we look for our own blocks which are not backed up yet

block\_id = self.find\_block\_to\_back\_up()

if block\_id is None:

return

# 3. collecting the nodes that have at least one our block backed up

remote\_owners = set(node for node in self.backed\_up\_blocks if node is not None)

# nodes having one block

for peer in sim.nodes:

# 4. we look for a node:

# if the peer is not self, is online, is not among the remote owners, has enough space # and is not

# downloading anything currently, schedule the backup of block\_id from self to peer

if (peer is not self and peer.online and peer not in remote\_owners and peer.current\_download is None # added

and peer.free\_space >= self.block\_size): # added

sim.schedule\_transfer(self, peer, block\_id, restore=False) # added

return

This method defines whether it is necessary to upload something:

* To recover a block for the remote node
* To make a backup of this node blocks on other nodes

1. Node.schedule\_next\_download

def schedule\_next\_download(self, sim: Backup):

# we don't want to download if we're already downloading something

if self.current\_download is not None:

return

# 1. find if we have a block which is on a remote node and not on our node

for block\_id, (held\_locally, peer) in enumerate(zip(self.local\_blocks, self.backed\_up\_blocks)):

if not held\_locally and peer is not None and peer.online and peer.current\_upload is None: # +TODO

sim.schedule\_transfer(peer, self, block\_id, restore=True) # +TODO

return # it means that we are downloading the block from the # remote peer

# 2. we look for a node

for peer in sim.nodes:

# if the peer is not self, is online, has no current upload, is not among the remote owners,

# has enough space and is not downloading anything currently, schedule the backup of block\_id from self to peer

if (peer is not self and peer.online and peer.current\_upload is None and peer not in self.remote\_blocks\_held # +TODO

and self.free\_space >= peer.block\_size):

block\_id = peer.find\_block\_to\_back\_up() # added

if block\_id is not None:

sim.schedule\_transfer(peer, self, block\_id, restore=False)

return

This method defines whether it is necessary to download something:

* recover a block of this node which saved as a backup on another node
* to make a backup of the other node block of this node

1. Online.process()

def process(self, sim: Backup):

node = self.node

# we do nothing if the node is already online or failed

if node.online or node.failed:

return

# otherwise set the node as online

node.online = True

# the node looks for which blocks are possible to back up

# and which blocks are possible to download

node.schedule\_next\_upload(sim) # class Node # added

node.schedule\_next\_download(sim) # class Node # added

# schedule the next offline event

sim.schedule(exp\_rv(node.average\_uptime), Offline(node))

1. BlockRestoreComplete.update\_block\_state()

class BlockRestoreComplete(TransferComplete):

def update\_block\_state(self):

owner = self.downloader

# now the owner knows that his block is held locally

owner.local\_blocks[self.block\_id] = True

# if all the k blocks are held locally - all the data is restored

if sum(owner.local\_blocks) == owner.k: # we have exactly k local blocks, we have all of them then

pass # +TODO