Our code uses multithreading for handling the actions of the client as well as incoming messages from peers. One thread runs for the listening socket and there is an additional thread that runs for each peer that has a connection to our client, as well as a main thread. The listening socket is in a state of perpetual listening for a new peer to attempt to establish a connection by way of sending a handshake message. When it receives the handshake, our peer creates a new peer thread for that handshaking peer and responds with a handshake according to BitTorrent protocol. Our client then receives a bitfield for the peer and responds with its own. At this point, our client and the peer enter into a self-looping state which we implement using a while loop that runs indefinitely. Note that our main thread also, periodically, receives messages from the tracker that indicate if new peers have joined the network—if we receive such a message indicating a new peer, our code attempts to connect to that peer by sending a handshake. This is the other way that we start peer threads and enter into the looping action state.

The role of the looping action state runs in the context of a connection with a single peer and handles all file exchange between our client and that peer. It uses 4 flags: *unchoked*, *unchoking*, *interested*, and *requested*. When *unchoked* is true, our client is able to receive file chunks from the peer, and when it is false (as it is initially), our client must send an "interested" message to that peer and receive an "unchoke" response. When *unchoking* is true, our client is able to send file chunks to the peer; when it is false (as it is initially), it must receive an "interested" message, in response to which it sends an "unchoke" message and marks *unchoking* as true. *Interested* (initially false) gets set to true when the client sends out an "interested"

message and resets to false when it receives an "unchoke" message from the peer. *Requested* (initially false) indicates that the client has requested a chunk from the peer and has not yet received a corresponding "piece" message from the peer.

The action state first checks if it should look for missing pieces. If it is not waiting on an "unchoke" message for an "interested" message that it has already sent (*interested == false*) and it is not waiting for a "piece" response for a "request" message it has sent (*requested == false*), it will determine if there is a piece it wants from the peer. It iterates through a structure that determines what pieces the peer has as well as one that holds which pieces for the client are "unlocked". A piece is "unlocked" if the client does not have it downloaded and has not yet requested it from any peer (without yet receiving a corresponding "piece" message). If the client discovers that there is such a piece that it could request from the peer and it is still choked, it sends an "interested" message to the peer and sets *interested = true*. If it is unchoked, the client simply sends a request for that piece and sets *requested = true*.

The next step of the action state is to listen for a message from the peer. At this point we know that one of the following is true: the client is waiting for an "unchoke" message from the peer, the client is waiting for a "piece" message from the peer, or the peer has no pieces that the client needs. With this knowledge, we know that the client does not need to take action until it receives a message from the peer. If the client receives an "unchoke" message, it sets *interested* to 0 and *unchoked* to 1 and continues to the top of the action state. If the client receives an "interested" message, it responds with an "unchoke" message, sets *unchoking* to 1,

and continues. If the client receives a “piece” message, it checks the hash of that piece against the one provided by the tracker—if it is the correct piece, it marks that piece as done and adds it to the file (using a file lock); if it is not the correct piece, it unlocks that piece so that other peer threads can request it, discards the piece, and continues. If the client receives a “have” message, it updates the metainfo for that peer and continues. Finally, if the client receives a “request” message, it checks if *unchoking == 1*. If so, it sends that piece to the peer and continues.