Introduction

This document outlines the implementation details of a peer-to-peer file sharing simulation, divided into three parts. Each part introduces increasingly sophisticated strategies for managing file piece requests and uploads among peers in a simulated network environment.

1 Part 1: `momostd.py`

The `momostd.py` file implements the foundational peer-to-peer sharing algorithm, focusing on optimizing the way file pieces are requested and uploaded among peers.

1.1 `requests` Method

- **Purpose**: Determines the optimal pieces of the file to request from other peers, aiming to efficiently complete the file download.
- **Process**:

```python
# Identify needed pieces and shuffle them to prevent symmetry
needed_pieces = list(filter(needed, list(range(len(self.pieces)))))
random.shuffle(needed_pieces)

# Count the frequency of each available piece among peers
frequencies = {}
for peer in peers:
    for piece in peer.available_pieces:
        if piece in frequencies:
            frequencies[piece] += 1
        else:
            frequencies[piece] = 1
```

- The method first identifies the pieces that the peer still needs and then shuffles this list to randomize the request order. It implements a ”rarest first" strategy by counting the frequency of each piece among all peers, prioritizing requests for the rarest pieces.

1.2 `uploads` Method

- **Purpose**: Decides which peers to unchoke and share file pieces with, based on a combination of random selection and strategic decision-making.
- **Process**:

```python
# Randomly select peers to unchoke, with a twist for
↪  optimistic unchoking
chosen = [req.requester_id for req in requests]
bws = even_split(self.up_bw, len(chosen))

# Every 3 rounds, perform optimistic unchoking
if round % 3 == 0:
    optimistic_unchoke_peer = random.choice(
        [req.requester_id for req in requests if
        ↪  req.requester_id not in chosen])
    chosen.append(optimistic_unchoke_peer)
```

- This method selects peers to unchoke based on their requests, evenly distributing available bandwidth. Additionally, every three rounds, it performs optimistic unchoking to explore and potentially reward less cooperative peers.

2 Part 2: `momotyrant.py`

The `momotyrant.py` file enhances the peer selection strategy by incorporating contribution-based incentives, improving fairness and efficiency in the network.

2.1 Enhanced `requests` Method

- **Purpose**: Adjusts the piece request strategy to prioritize peers based on their download contributions, aiming to reward helpful peers and encourage reciprocity.

- **Process**:

```
# Calculate download contributions from each peer
download_contributions = {}
for download in history.downloads[-1]:
    if download.from_id in download_contributions:
        download_contributions[download.from_id] +=
         ↪  download.blocks
    else:
        download_contributions[download.from_id] =
         ↪  download.blocks

# Sort peers by their contribution
peers_by_contribution = sorted(peers, key=lambda peer:
 ↪  download_contributions.get(peer.id, 0), reverse=True)
```

- This method calculates and sorts peers based on their download contributions to the requesting peer, prioritizing those who have contributed more significantly to the peer's downloads in previous rounds.

2.2 Refined uploads Method

- **Purpose**: Enhances the upload decision process by considering peers' contribution levels, aiming to reward those who contribute more to the network.

- **Process**:

```
# Aggregate download contributions for the last round
download_contributions = {}
for download in history.downloads[-1]:
    peer_id = download.from_id
    if peer_id not in download_contributions:
        download_contributions[peer_id]
         ↪  =download\_contributions[peer\_id] +
         ↪  download.blocks
```

- This snippet aggregates the download contributions from each peer during the last round. It then uses this information to inform the decision on whom to unchoke, favoring peers that have contributed more to the peer's success in terms of downloaded blocks.

3 Part 3: `momopropshare.py`

The `momopropshare.py` file introduces a proportionate sharing mechanism, ensuring that resources are distributed more equitably among peers based on their contributions.

3.1 Proportionate `uploads` Method

- **Purpose**: Implements a bandwidth allocation strategy that distributes available bandwidth proportionally to peers based on their contribution levels.

- **Process**:

```
# Calculate the total downloaded contributions and
↪  allocate bandwidth proportionally
total_downloaded = sum(download_contributions.values())

for peer_id, contribution in
↪  download_contributions.items():
    if total_downloaded > 0:
        peer_bw = self.up_bw * (contribution /
        ↪  total_downloaded)
    else:
        peer_bw = self.up_bw //
        ↪  len(download_contributions)
```

- This method ensures that peers who have contributed more to the network, in terms of the amount of data they have provided to others, are rewarded with a greater share of the bandwidth when it comes to uploads. This fosters a cooperative network environment where contributions are directly linked to benefits.

4 Conclusion

This document has provided a detailed overview of the implementation of three distinct strategies for managing file piece requests and uploads in a simulated peer-to-peer network. By progressively enhancing the algorithms for piece selection and bandwidth allocation, these files aim to optimize network efficiency, fairness, and the overall performance of the file-sharing ecosystem.