Department of Electrical and Computer Engineering
Concordia University
Communication Networks and Protocols - COEN 366

Fall 2025

Project: Peer-to-Peer File Backup and Recovery System (P^{2}PBRS)

Designed by: Ahmed Bali

1. Introduction:

The **P**eer-**to**-**P**eer File **B**ackup and **R**ecovery **S**ystem (**P2PBRS**) introduces students to core concepts of reliable data transfer, error detection, and recovery in networked systems.

This project simulates a distributed backup platform where peers cooperate to store each other's files under the coordination of a central server. The server node never accesses the actual data directly; it only coordinates, monitors, and maintains metadata.

Note: For simplicity, we omit any mechanisms related to *storage awards*, *credits*, or *backup costs*. All peers in this system cooperate voluntarily, and no monetary or credit-based transactions are implemented.

Each peer can act as both a **client** (requesting file backup or recovery) and a **storage node** (holding backup chunks for other peers). The system uses both **UDP** and **TCP**:

- **UDP** → for control messages (registration, announcements, integrity checks).
- **TCP** → for reliable transfer of file chunks and recovery operations.

Any file type can be transferred, including `.txt, .pdf, .jpg, .mp4`, or binary executables. Files are treated purely as sequences of bytes.

Students will design and implement both the server and peer components in Java, Python, or C++, focusing on socket programming, concurrency, reliability, and persistence. Detailed protocol descriptions are provided in Section 2, and specific project requirements are outlined in Section 3.

2. Protocol of P^{2}PBRS System:

The system architecture consists of several peers and one coordination server. Peers may have different roles: Backup Owners who back up their data, and Storage Nodes who hold replicas for others. A peer may combine both roles.
Peers communicate directly only for actual file chunk transfers during backup and recovery; all other coordination is mediated through the server.

The server is always reachable at a known IP address and a fixed UDP socket (e.g., port 5000). It listens for incoming messages from clients for registration, backup requests, and heartbeat messages, while TCP is used for transferring chunks and restoring files.

Terminology**:** For simplicity, we assume **one** user per client (**peer**). Hence, the terms "user," "client," and "peer" are used interchangeably throughout this document.

2.1. Registration and De-registration (Communications through UDP):

Before participating, a peer must register with the server by sending:

The user sends the following details: their unique name, role (OWNER, STORAGE or BOTH), IP address, a UDP port number where they can be reached by the server, and a TCP port number where they can be reached by other clients for backup and recovery, and the available storage (e.g., in MB) offered to hold other peers' data.
If a peer registers as "OWNER" only, the server will use it only for backup requests, not for storing others' files. This allows flexibility in the system, for example, a node with limited capacity.

The registration request (message "REGISTER") is sent to the service via UDP.

REGISTER	RQ#	Name	Role	IP_Address	UDP_Port#	TCP_Port#	Storage_Capacity

Example:

```
REGISTER  01  Alice  BOTH  192.168.1.10  5001  6001  1024MB
```

Upon receiving this message, the server may either accept or deny the registration. For instance, the registration can be denied if the provided Name is already in use or when the server cannot handle additional clients.

If the registration is accepted, the server responds to the user with the following message:

REGISTERED	RQ#

If the registration is denied, the server will respond with the following message and provide the reason.

REGISTER-DENIED	RQ#	Reason

The RQ# is used to refer to which "REGISTER" message this confirmation or denial corresponds to. It is the same case of all the messages where RQ# is used.

A user can de-register by sending the following message to the server:

DE-REGISTER	RQ#	Name

The server removes the peer's record if found; otherwise, it ignores the request.

2.2. Backup Announcement and Chunk Distribution (UDP for control, TCP for data)

A registered peer requesting to back up a file first notifies the server by sending BACKUP_REQ request via UDP with the following pieces of information:

BACKUP_REQ	RQ#	File_Name	File_Size	Checksum

Where, Checksum, such as CRC32, is used to ensure integrity.

Upon receiving BACKUP_REQ, the server:

1. Selects peers with available storage to host chunks.

Sends a notification to each selected storage node:

STORAGE_TASK	RQ#	File_Name	Chunk_Size	Owner_Name

The RQ# in this message may be different from RQ# of the initial BACKUP_REQ message as the first one is used to number a request generated by the Backup Owner and the second one is used to number a request generated by the server.

2. Replies to the requester:

If the request is accepted the server responds as follows:

BACKUP_PLAN	RQ#	File_Name	[Peer_List]	Chunk_Size

Example:

```
BACKUP_PLAN photo.jpg [PeerB, PeerC] 4096
```

Each peer in the Peer_List stores one or more chunks of the file; the Chunk_Size specifies the size of each individual chunk, not the total storage per peer.

If the BACKUP request is denied, the server sends a BACKUP-DENIED message containing the reason like the server cannot ensure the requested storage space.

BACKUP-DENIED	RQ#	Reason

3. Notifies each selected peer:

STORE_REQ	RQ#	File_Name	Chunk_ID	Owner_Name

Example:

```
STORE_REQ  RQ#1  report.pdf  1  Alice
```

The requesting peer (Backup Owner) then opens a **TCP** connection with each designated storage peer, sends chunks, and waits for acknowledgments:

SEND_CHUNK	RQ#	File_Name	Chunk_ID	Chunk_Size	Checksum
Data					

Example:

```
SEND_CHUNK 24 myphoto.jpg 0 4096\n
<binary data bytes>
```

After successful receipt, the storage peer:

1- Reads the header line and extracts metadata.
2- Reads exactly `Chunk_Size` bytes from the socket.
3- Verifies checksum and acknowledges:

CHUNK_OK	RQ#	File_Name	Chunk_ID

Example: `CHUNK_OK  24  myphoto.jpg  0`

If checksum mismatch occurs:

CHUNK_ERROR	RQ#	File_Name	Chunk_ID	Reason

Example: `CHUNK_ERROR  24  myphoto.jpg  0  Checksum_Mismatch`

The CHUNK_ERROR message is a negative acknowledgement that requires the requesting peer to retransmit the data frame. After three failed attempts, the Backup Owner cancels the transfer of that chunk and continues with the remaining chunks on other storage nodes.

After it receives CHUNK_OK from all storage peers, the Backup Owner notifies the server:

BACKUP_DONE	RQ#	File_Name

Which means that all peer-to-peer TCP transmissions completed successfully and the file is now fully backed up.

For more reliability, the server receives direct confirmation from each Storage Peer to the server:

STORE_ACK	RQ#	File_Name	Chunk_ID

This confirmation message is sent when the transmission is done and aligned with the same details received in the STORAGE_TASK message including the transmitted data size.

After a timeout, if the server does not receive a STORE_ACK message, it assumes that the storage of the specific chunk has not occurred successfully. In such cases, the server may trigger a reallocation of that chunk to another available node, which may lead to redundant chunk storage as a side effect.

The server monitors acknowledgments and updates its backup table mapping:
`{Owner_Name, File_Name → [Peers holding chunks with their IDs]}`

Notes:

- All control message, such as CHUNK_OK, CHUNK_ERROR, and BACKUP_DONE, are exchanged via UDP.
- Although TCP provides reliable, error-checked transmission, this project requires an additional checksum at the application level (e.g., CRC32, MD5, or SHA-1). This

checksum serves to verify end-to-end file integrity across backup, storage, and recovery phases. The purpose is to verify end-to-end data integrity.

2.3. Heartbeat and Integrity Check (UDP)

Each registered peer periodically reports its availability and stored chunk list:

HEARTBEAT	RQ#	Name	Number_Chunks#	Timestamp

If a heartbeat is missed for a defined timeout, the server assumes the peer failed and triggers recovery by instructing other peers holding replicas to create new copies elsewhere.

2.4. File Recovery and Reconstruction (UDP for control, TCP for data)

When a peer loses a file, it requests restoration from the server:

RESTORE_REQ	RQ#	File_Name

The server responds with the list of peers that host the chunks:

RESTORE_PLAN	RQ#	File_Name	[Peer_List]

The requester establishes TCP connections to retrieve chunks:

GET_CHUNK	RQ#	File_Name	Chunk_ID

Each storage peer replies:

CHUNK_DATA	RQ#	File_Name	Chunk_ID	Checksum

After all chunks are received and reassembled, the client verifies integrity using the overall file checksum provided in `BACKUP_REQ`.

If successful, it reports:

RESTORE_OK	RQ#	File_Name

Otherwise:

RESTORE_FAIL	RQ#	File_Name	Reason

An example of a restoration failure is when one or more storage nodes go down, preventing the file from being reassembled.

The server does not directly observe peer-to-peer TCP transfers. It infers completion through control acknowledgments, such as `RESTORE_OK` and `BACKUP_DONE`.

2.5. Failure Handling and Recovery (UDP + TCP)

If a peer fails during backup or recovery:

- The server detects the failure via missed heartbeat.
- It redistributes chunks by instructing other active peers:

REPLICATE_REQ	RQ#	File_Name	Chunk_ID	Target_Peer

Peers perform replication over TCP and confirm success.

Note: Students may simulate peer failure by intentionally stopping a node process to observe the server's redundancy recovery behavior.

3. Requirements

The project must be completed in groups of three (**3**) to four (**4**) students. You are required to use the **Group self-selection** feature on Moodle to create or join a group. The project report must include a dedicated section detailing the specific contributions of each team member. If you experience any issues with the Moodle system, please contact me at ahmed.bali@concordia.ca before the deadline.

Design and implement the client (peer) and server that follow the protocol(s) aforementioned.
In your report, you must also discuss and justify your **design choices**, including how messages are encoded (e.g., simple text messages), how concurrency is handled, and which checksum or integrity verification technique (e.g., CRC32, MD5, SHA-1) was selected.

Both the client (peer) and server must be multi-threaded.

The information stored in the server should be persistent. In the event of a server crash and restoration, the server must recover all pre-existing information.

Reporting: Server and clients should be reporting their communications to the users of the system using a log file or printing directly on the screen. In other words, during the demonstration, the marker can see the messages sent and received, progress and failures.

Assumptions/Error/Exception Handling
You should be aware that the description as it is does not state everything. For instance, what happens if a storage node receives a request with a file owner that does not correspond to the storage request received from the server? Or if a client that is not registered attempts to send a storage request to the server. How to handle duplicate registrations. What to do if a chunk transfer fails mid-way. How checksum mismatches are treated (e.g., re-send).

State and document clearly any assumption you make beyond the assumptions made by the instructors.

Extra Features (For Bonus Marks): Notice that in this project we do not require authentication of users. Extra marks will be given to students who add an authentication scheme to the proposed system. In addition, other services may be considered like Compression / Encryption of chunks. Also, a GUI is not required as long as we can run the clients and the server and see what is going on with the messages. Extra marks will be given if you decide to build a GUI.

Recommended Schedule

We strongly recommend the following schedule to avoid the rush of the end of the term:

- **Phase 1:** Registration / De-registration via UDP. (to be completed by the last week of October)
- **Phase 2:** Backup & Heartbeat management (UDP + TCP). (to be completed by mid-November)
- **Phase 3:** Recovery and Integrity Validation (TCP). (to be completed by the last week of November)

By following this schedule, you will have the opportunity to showcase your progress to the Lab TAs and resolve potential issues before the final submission and demonstration.

Deliverables

Project Report (Week 13): You should hand in a report, by Week 13, where you clearly document your assumptions, design decisions, code and experiments. You should also clearly state the contributions of every member of the group. Every student must contribute **technically** (designing and implementing the protocol) to the project.

Demonstration (Week 13): A demo will be held during Week 13. During the demo the members of the group must all be present and ready to answer questions.

During the demo, we may also examine the code itself and the report.

Notes:

Extra clarifications or adjustments may be announced on Moodle.

Students are encouraged to test their system under multiple peers and simulated failures to validate reliability and recovery behavior.

The project will be discussed further during lectures.