

PROTOCOL DESCRIPTION

1. Overview

The Micro-CDN system operates on a distributed architecture where multiple Content Servers host files, an Index Server acts as a central directory, and a Monitor Server ensures system health.

The communication protocol is designed using a hybrid transport layer approach:

- **TCP (Transmission Control Protocol):** Used for reliable control messages (registration, lookups) and file transfer.
- **UDP (User Datagram Protocol):** Used for lightweight, periodic health monitoring (heartbeats).

All messages are text-based (ASCII), encoded in UTF-8, and terminated by a newline character (\n). This ensures that the protocol is human-readable and easy to debug.

2. Message Formats (Syntax)

The system uses specific message formats for different interaction channels. Arguments are separated by spaces.

2.1. Content Server <-> Index Server (TCP)

Used by Content Servers to register themselves and synchronize their file lists upon startup.

Message Type	Direction	Syntax (Format)	Description
Registration	CS ->Index	REGISTER <server_id> <tcp_port> <udp_port>	Registers the server ID and listening ports.
Add File	CS ->Index	ADD_FILE <server_id> <filename> <size>	Informs the Index about a hosted file and its size (bytes).
End Sync	CS ->Index	DONE_FILES	Indicates that the file list transmission is complete.
Response	Index ->CS	OK REGISTERED or OK FILES_ADDED	Acknowledgment of successful operations.
Update Load	CS ->Index	UPDATE_LOAD <server_id> <load>	(Optional) Updates the current client count.

2.2. Client <-> Index Server (TCP)

Used by Clients to discover which server hosts a requested file.

Message Type	Direction	Syntax (Format)	Description
Handshake	Client ->Index	HELLO	Initiates the session.
Greeting	Index ->Client	WELCOME MICRO-CDN	Server accepts the connection.
Lookup	Client ->Index	GET <file_name>	Requests the location of a specific file.
Redirect	Index ->Client	SERVER <ip> <tcp_port> <server_id> <size>	Returns the details of the best available server.
Error	Index ->Client	ERROR FILE_NOT_FOUND	Returned if the file is not registered on any live server.

2.3. Client <-> Content Server (TCP)

Used for the actual file transfer.

Message Type	Direction	Syntax (Format)	Description
Download	Client ->CS	GET <file_name>	Requests the content of the file.
Success	CS ->Client	OK <file_size>	Followed immediately by the raw binary stream of the file.
Error	CS ->Client	ERROR FILE_NOT_FOUND	Returned if the file does not exist on disk.

2.4. Monitor Protocol (UDP & TCP)

Used for health checking and failure notification.

- **Heartbeat (UDP):** Content Server ->Monitor
 - **Format:** HEARTBEAT <server_id> <ip> <tcp_port> <load> <num_files>
 - **Frequency:** Sent every 3 seconds to prove liveness.
- **Failure Notification (TCP):** Monitor ->Index Server
 - **Format:** SERVER_DOWN <server_id> <timestamp>
 - **Trigger:** Sent immediately when a server is marked as "DEAD".
- **Index Registration (TCP):** Index Server ->Monitor
 - **Format:** REGISTER_INDEX <ip> <port>

- **Purpose:** The Index Server subscribes to receive failure notifications.

3. Interaction Sequences

The following sequences describe the logical flow of operations within the system.

Sequence A: System Startup & Registration

1. **Monitor Start:** The Monitor Server starts listening on UDP port 6000.
2. **Index Start:** The Index Server starts on TCP 5000 and immediately connects to the Monitor (TCP 6001) to register its notification listener port.
3. **Content Server Start:**
 - The Content Server scans its local directory to build a file map.
 - It opens a TCP connection to the Index Server.
 - It sends REGISTER followed by multiple ADD_FILE messages.
 - After sending DONE_FILES, it closes the connection.
 - Simultaneously, a background thread begins sending HEARTBEAT UDP packets to the Monitor every 3 seconds.

Sequence B: Client File Download

1. **Discovery:**
 - The Client connects to the Index Server.
 - Client sends: GET shared_document.txt.
 - Index Server Logic: It looks up the file, filters for "alive" servers, and selects the one with the lowest load.
 - Index Server responds: SERVER localhost 7001 CS1 600.
 - Client closes the connection.
2. **Transfer:**
 - The Client connects to the specific Content Server (CS1) at port 7001.
 - Client sends: GET shared_document.txt.
 - CS1 responds: OK 600.
 - CS1 streams 600 bytes of data.
 - Client reads exactly 600 bytes, writes to disk, and closes the connection.

Sequence C: Failure Detection & Recovery

1. **Failure:** A Content Server (e.g., CS1) crashes or is terminated. Heartbeats stop.

2. **Timeout:** The Monitor Server detects that no heartbeat has been received from CS1 for more than HEARTBEAT_TIMEOUT (8 seconds).
3. **State Change:** Monitor marks CS1 as "DEAD".
4. **Notification:** Monitor opens a TCP connection to the Index Server and sends SERVER_DOWN CS1.
5. **Update:** Index Server updates its internal table, marking CS1 as "dead".
6. **Routing:** Future client requests for files on CS1 are automatically routed to other available replicas (e.g., CS2), or an error is returned if no replicas exist.

4. State Machines

This section defines the valid states and transitions for each component.

4.1. Monitor Server State Machine

- **STATE_LISTENING:** The default state. It listens for UDP heartbeats. Upon receiving a packet, it updates the last_seen timestamp of the sender.
- **STATE_CHECKING:** triggered periodically (every 2 seconds). It iterates through the list of registered servers.
 - *Transition:* If $(\text{current_time} - \text{last_seen}) > 8 \text{ seconds}$ -> Go to STATE_DEAD.
- **STATE_DEAD:** The server is marked as unavailable.
 - *Action:* Trigger a TCP notification to the Index Server.
- **STATE_RECOVERED:** If a heartbeat is received from a server previously marked "DEAD", it is marked "ALIVE" again.

4.2. Index Server State Machine

- **STATE_IDLE:** Waiting for incoming TCP connections.
- **STATE_REGISTRATION:** Entered when a message starts with REGISTER. The server updates its topology map and file index.
- **STATE_QUERY_PROCESSING:** Entered when a message starts with GET. The server performs Load Balancing selection.
- **STATE_TOPOLOGY_UPDATE:** Entered when a SERVER_DOWN message is received from the Monitor. The server removes the dead node from active routing.

4.3. Content Server State Machine

- **STATE_INITIALIZING:** Scanning local storage and preparing sockets.
- **STATE_SYNC:** Communicating with the Index Server to register files.
- **STATE_RUNNING:**
 - *Sub-state A (TCP):* Blocking accept loop waiting for client download requests.
 - *Sub-state B (UDP):* Infinite loop sleeping for 3 seconds and sending Heartbeats.

5. Error Handling Rules

The protocol is designed to handle common network and application errors gracefully.

1. Unknown Commands:

- If any server receives a malformed string or an unknown command header, it responds with ERROR UNKNOWN_COMMAND or ERROR INVALID_FORMAT and closes the connection to prevent undefined behavior.

2. UDP Packet Loss:

- The system assumes UDP is unreliable. A single missed heartbeat does not trigger a failure. The 8-second timeout provides a buffer for lost packets, ensuring a server is only declared dead if roughly 2-3 consecutive heartbeats are missed.

3. File Not Found:

- **Index Level:** If the Index Server cannot find a registered file, it returns ERROR FILE_NOT_FOUND.
- **Content Level:** If the Content Server cannot find the physical file (e.g., it was deleted after registration), it returns ERROR FILE_NOT_FOUND instead of crashing.

4. Client Disconnection / Partial Reads:

- The Client expects exactly <file_size> bytes. If the TCP connection drops before all bytes are received, the Client detects the size mismatch and logs a warning ("Downloaded X bytes, expected Y bytes"). It does not save the corrupted file as valid.

5. Monitor Unreachability:

- If the Index Server cannot connect to the Monitor during startup, it logs the error but continues to function (graceful degradation), though automatic failure updates will be disabled.