Project Information

Group ID	9	**Semester** *(WS2?)*
1. Student Name	Muhammad Amin Guluzade	WS25
2. Student Name	Eric Sîrbu	WS25
3. Student Name	Mykhailo Zelia	WS25
4. Student Name		
Project Title	Distributed Stock Exchange System	

Project Description	Our project aims to design and implement a Distributed Stock Exchange System that provides consistent and fault-tolerant order execution across multiple networked servers Traders (clients) connect to the exchange to submit buy and sell orders, while a cluster of distributed servers cooperatively maintains a globally consistent order book and execute trades.
Architectural Model	The system follows a client–server architecture. Clients submit trade requests and receive market updates from servers in the cluster. Each server exposes a client-facing interface and acts as both a front end (proxying client requests) and a replica manager maintaining a copy of the order book. Servers are organized in a leader–follower configuration. A single leader is responsible for globally sequencing incoming orders anc performing deterministic order matching. Follower servers maintain synchronized replicas of the order book and forward client orders to the leader.
Dynamic Discovery	A new trader (client) discovers the server that is quickest to grant access to create orders. Each server periodically broadcasts a “hello” message on a predefined UDP channel. Clients listen for these announcements and automatically connect to the nearest server. New servers instead connect to all existing servers and retrieve the orde book from the nearest neighbor, triggering a new leader election. Clients have a periodic timeout after which they check which server is the best to connect to and reconnect to a different server if necessary.

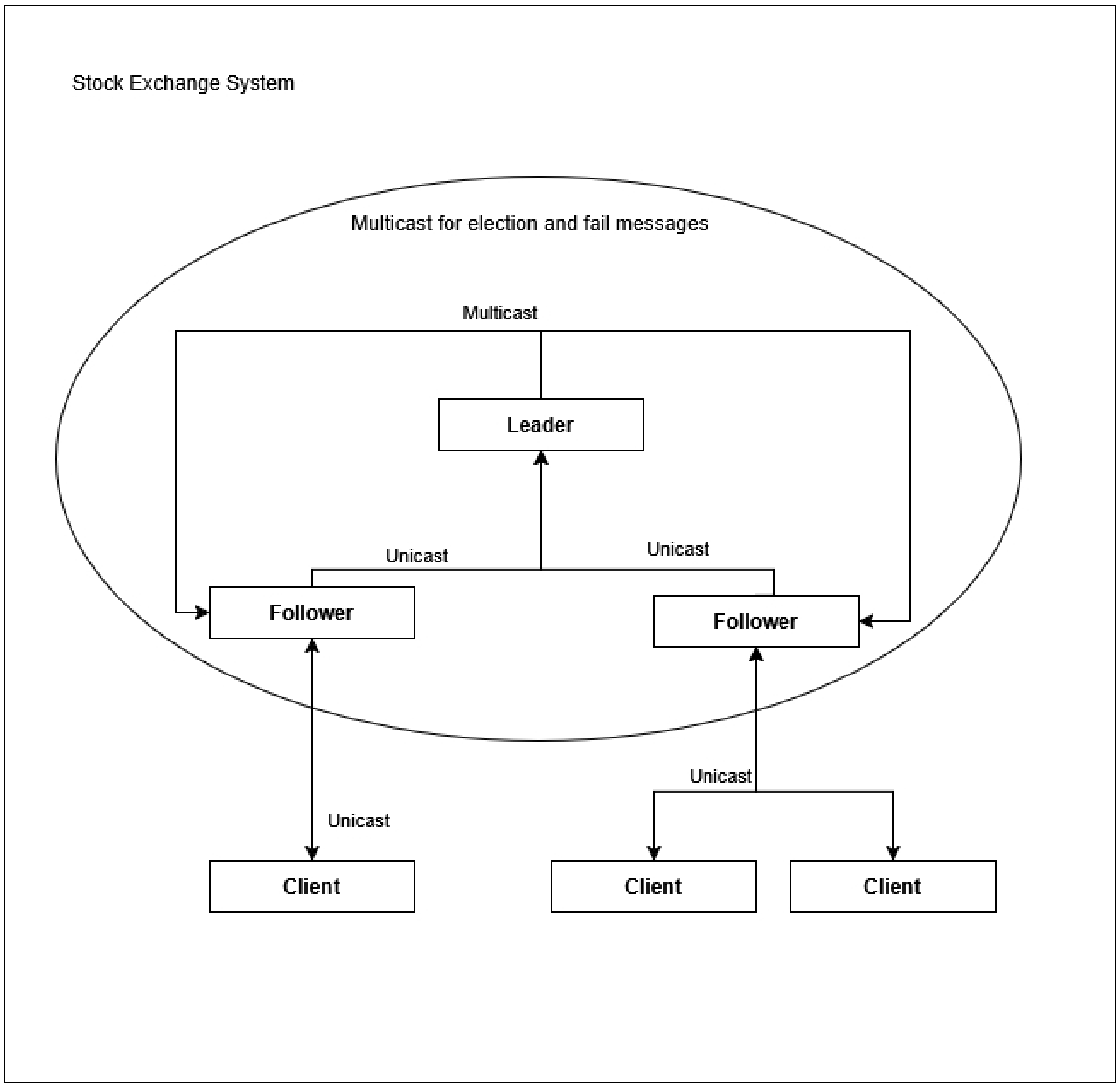
Fault Tolerance	The system assumes a fail-stop failure model. If any server fails it sends a message to the other servers, which then triggers a new election. The newly elected leader resumes processing from the most recent state among any of the followers. A stopped server follows the procedure of a newly joined server. Clients connected to a failed server receive a message from the server and reconnect to another server. A crashed client can rejoin at any time without affecting the system integrity.

Voting	Leader election provides coordination among servers. We plan to implement a voting algorithm, where the node that has the lowest average response time to each server becomes the leader. Every server pings every other server to determine the latencies between each other. The servers then use reliable multicast to send their average response time to every other server. Everyone will have the same global view and the server with the lowest average response time can be elected. Their IP addresses are used as a tiebreaker and all nodes agree on the same leader. A stopped server is easily detected and every other server can discard their corresponding response time.

Reliable Ordered Multicast	Clients and servers communicate via TCP sockets. Followers forward client requests to the leader via unicast communication. The leader receives these orders, matches them, assigns them within a global order, and then multicasts these updates reliably to all follower servers. This way the leader acts as a sequencer in addition to calculating the matching and appropriate responses to clients. Each follower maintains a history of the trades and applies incoming matchings in the order specified by the leader.

System Architecture Desin

Include a clear diagram of your system architecture (key components and connections between them).


Stock Exchange System
Multicast for election and fail messages
Multicast
Leader
Unicast
Unicast
Follower
Follower
Unicast
Unicast
Client
Client
Client