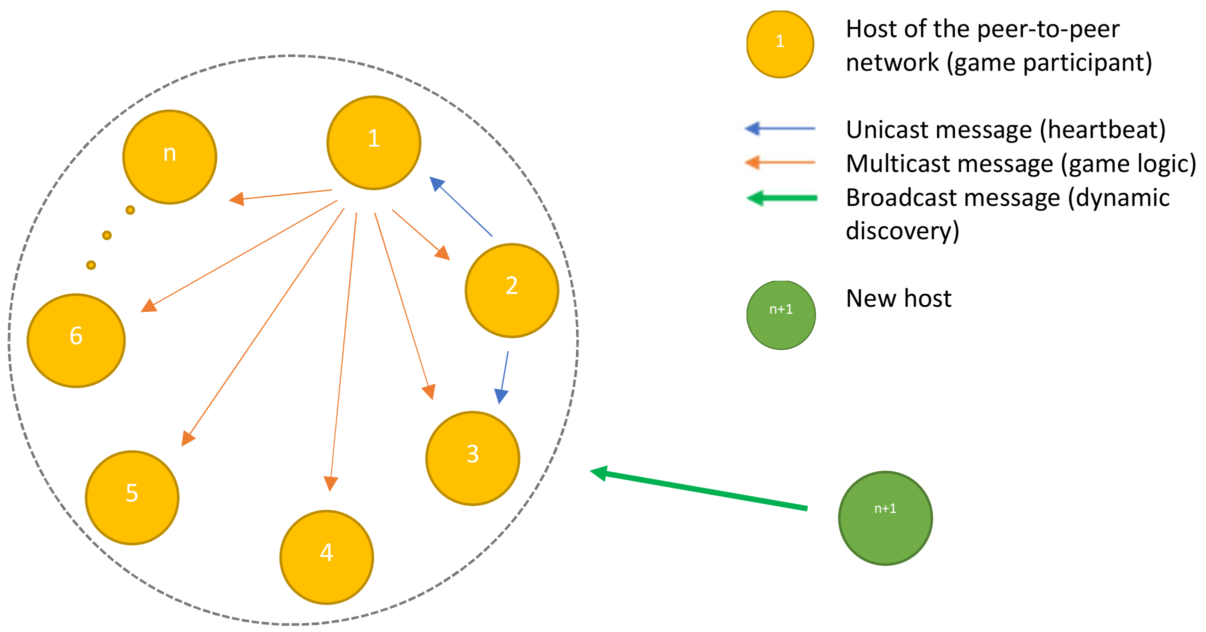
Project Requirement Implementation Description

Introduction

We propose the design and implementation of a massive multiplayer online game using core and advanced principles of distributed system development. The game will be based on the idea of a game known as “Simon says” with the slight difference that what “Simon says” is not to be executed in motions by the participating players; in our case “Simon” distributes a string of characters per round to the players. These have to reproduce this string through their input device and then send a message containing said input back for evaluation. The first player to return the correct response string gets 10 points, 2nd player gets 5 points and so on. After each round a new “Simon” is elected among players that returned a faulty response or at random if not applicable.

Architectural Description


1
Host of the peer-to-peer network (game participant)
Unicast message (heartbeat)
Multicast message (game logic)
Broadcast message (dynamic discovery)
n+1
New host
n
1
2
3
4
5
6
n+1

- Every node has a UUID and contains a list of all other nodes, including their respective UUIDs
- We declare the next highest and next lowest UUIDs as neighbors to a node, so that we implicitly form a ring
- We use a P2P Network to build the game, which contains one elected Master and a variable amount of players

Dynamic Discovery of Hosts

- The procedure to join a game will be implemented like this:
    - New host sends a broadcast message asking to join
    - The master node (Simon) is always listening for new host
    - When the master recognizes a join request message, he answers with a join command and the current score board of players, this contains all the player's UUIDs and scores
    - The master also sends the UUID-IP dictionary, so that the newly connected peer knows the IP-address of all the other peers
    - The master sends the UUID and IP of the newly connected node to all other hosts, so that they can update their player list
    - With this information the new peer searches for it's neighbors
    - Additional logic is required to ensure, that every peer has the same Group View (score board and UUID-IP dictionary)
    - When the new node receives the join message it has all the information it needs to participate in the game
    - It is possibly to have multiple rooms (on different broadcast ports)

Crash Fault Tolerance

- Every node contains the score board and the UUID-IP dictionary, so it is only necessary that one node is functioning correctly to contain a valid state of the game
- After every game round the winners are communicated to everyone in the game, so that all the nodes can update their score board accordingly and contain the same information
- The nodes send heartbeat messages to their neighbors, to verify their availability

- If a neighbor isn't available, the noticing node informs all peers, scoreboards are updated, and it chooses a new neighbor.
- If the crashed node was the master, then a new voting period to elect a new master among the remaining nodes is triggered.

Voting

- The Bully Algorithm based on the UUIDs is implemented for leader election/voting
- After every game round or if the current master crashes a new master gets elected
- Voting can be proved correct through manipulating the UUIDs and running tests

Ordered Reliable Multicast

We need Total Ordering

- Every peer gets the messages (delivered) in the exact same order

Why do we need this?

- We do not have a central server, that stores the truth (of the game status)
- We need a way to make sure that every peer has the exact same game status
- If each peer receives the messages in the same order and we have a definitive algorithm that derives the game state from the messages, we can ensure that everyone has the same game state stored locally.

How do we achieve this?

- We use the ISIS (Intermediate System to Intermediate System) Protocol

How do we implement this?

- We build a Unicast to multiple receivers through our middleware, this Unicast implements a TCP connection, so that we have agreements and no arbitrary delays

Byzantine Fault Tolerance

- The heartbeat is extended to not only ping a node's neighbors, but also query their current state, to verify the neighbor is working properly
- If two neighbors do not agree on a specific game of state, then a global consensus mechanism routine is triggered to find an agreement and resolution for the detected byzantine fault
- To reach consensus the sum of the faulty nodes must be smaller than a third of the sum of all nodes