# Distributed Racer

CPSC 416 Project 2

Kevin Mienata, Jonathan Kim, JiYoon Ok, Benjamin Gee
b9c9 p2v9a p7x8 v5e0b

## Introduction

A distributed “racing” game made up of 3 different types of nodes: server, cars and players. A single server node is responsible for hosting the game state.

Each player node (referred to as pnode) will connect to a car node (referred to as cnode) and will be part of a “team”. A car node will need to acquire a minimum number of pnodes before it is qualified to be part of the game. The server will allow the cnodes a fixed amount of time to acquire players before the game start. The cnodes that acquire enough pnodes in the alloted time will join the game by making a connection to the server, the others will be restricted to join. These settings will be inputted in our config files for both the car, player and server nodes. Prior to joining the server, the cnode will randomly designate one of their pnodes as the leader (driver), whose role will become important later. The game will start once it reaches the maximum number of cnodes allowed or if it has at least two cnodes after the alloted time is up.

**Figure 1** Race track with reward and multiple car nodes competing for the reward. Concurrency issue arises when two or more car nodes arrive at the reward at the same time. There is only one reward for one car node, so a majority voting solution between the other car nodes in the system will determine who receives the reward.

When the car node connects to the server, the server will send the track of the race to the car nodes and how the rewards are scattered in the track. Rewards on the track can only be given to one car node and that is decided by the car node that gets to the reward the fastest. To resolve the

problem where multiple car nodes reach the reward at the same time, player nodes in the other car nodes will vote for the car node that should retrieve the reward. The car nodes will take the majority of the votes and send it to the server. The server will then make the decision accordingly after tabulating the votes from the car nodes.

Once the game starts, no other cnodes or pnodes will be able to join their respective nodes. The objective of the game is to be the first team to reach the finish line by collectively answering trivia type questions provided by the server. Each cnode will periodically receive a question from the server and will disseminate the question to each of its pnodes. The question will be associated with a reward which indicates how many steps the cnode can move forward to the finish line if your team answers the question correctly. It is up to the pnodes (players) to answer the question as quickly and as correctly as possible and return the answer to their cnode. Once the cnode receives an answer from all of it's pnodes, it will select the most frequent answer. In the case of a tie, the cnode will choose one at random. Once the cnode comes up with a consensus answer, it will send that answer to the server.

The server will be listening to each cnode for answer. The first cnode that sends a correct answer back to the server will be rewarded their steps to advance forward in the race. In addition, the top 3 fastest cnodes to answer the question will be rewarded coins; 3 coins for first, 2 coins for second, 1 coin for third and 0 coins for the rest. The coins are used to purchase interruptions at any point in the game by first sending a request to the cnode, then to the server. These purchased interruptions can then be thrown at any other cnode in the network to interfere with their gameplay. The decision of when to purchase, what to purchase, the connection with the cnode and which cnode to direct the interruption to is all made by the driver of that respective cnode. Once you have advanced forward in the race, the next question is sent by the server and the process repeats itself.

In the case that the cnode sends an incorrect answer to the server, the server responds giving the cnode and all its players a second chance to answer the question. If again, the cnode answers incorrectly, a timeout penalty will be incurred on the cnode where the cnode is inactive for that time. After the penalty has been served, the server will send the next question for the team to answer and the process repeats itself.

Our mission for this project is to focus on the distributed aspect and leave all graphics and GUI to a minimum. The game is played entirely on the console and the questions, options and game updates will be displayed in text format for each pnode to read. The pnodes will answer and select options by simply typing out the answer or choice.

# System Topology and Nodes

The topology of the game application will be composed of three types of nodes: single server, N number of car nodes (referred to as cnodes), and M number of player nodes (referred to as pnodes). Below is a figure that illustrates the network topology between the nodes.

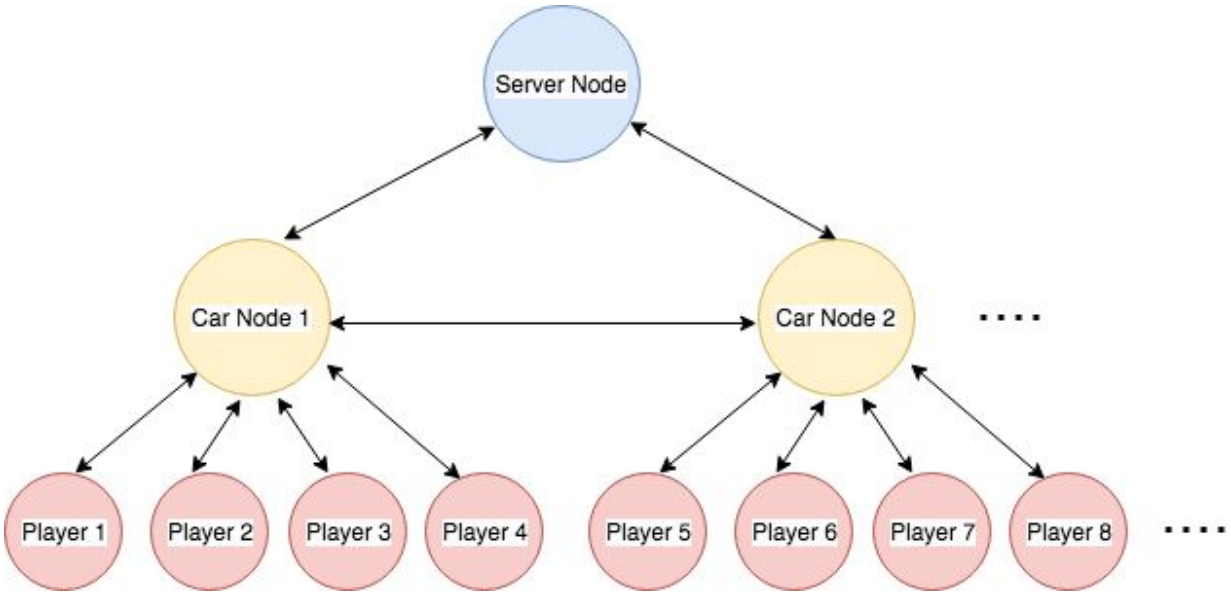


**Figure 2** System topology. Each player connects to the game and is assigned to a random car node. When the threshold of player nodes within a car node is reached, it can connect to the server and participate in the game.

## Server Node

Responsible for hosting the game. Each cnode connects to the server to begin the game. The server keeps track of the number of cnodes connected and works to connect cnodes with all other cnodes in the network. Once the game begins, the server is responsible for sending each cnode the trivia questions and verifying the answers received from the cnodes. Therefore, the server keeps a list of questions and answers in memory. In addition, it needs to keep track of which cnode is answering which question so that it can provide the exact same order of question to each cnode to keep fairness and consistency. Finally, the server will keep track of the “interruption store” where the cnodes can purchase interruptions to throw at other cnodes in the network.

### Car Node

A cnode is a supernode which will disseminate information to the pnodes and its own heartbeat using smart query flooding (similar to what Kazaa uses). Each cnode is connected to the server to retrieve information about list of questions, racetrack, and other cnodes connected in the network. A cnode is also connected to set of players that makes up the "team" and to the rest of the cnodes in the game they're competing against. A cnode is responsible for disseminating the question it receives from the server to its set of pnodes and listening for the answers. Once it receives an answer from each pnode, it will compute the majority answer (> 50%) or choose randomly in the case of a tie and send the consensus answer for verification on the server. The cnode-to-cnode connection is used as a way for one cnode to throw an interruption on behalf of another cnode in order to disrupt their gameplay and get an advantage. This is done once the respective cnode accumulates enough coins to purchase an interruption from the server.

### Player Node

Each pnode enters the game by connecting to a unique cnode. The other pnodes that are part of the cnode are a "team". Each pnode will receive a question from its respective cnode periodically once the game begins. The responsibility of the pnode is to answer the question as quickly and correctly as possible and return the answer to the cnode. Each pnode receives the question on the console and will answer by manually entering the answer on the console.

## APIs

1. Server to Car node
2. Car node to Car node
3. Car node to Player Node

### Server API

Server listens for connections from car nodes on a known TCP:IP port.

**Server - Car Node API**: A car node can invoke the following TCP RPCs to the server:

- gameSettings, err ← RegisterCar(address)
    - Registers a new car node with an address so other car nodes can connect to it once game starts.
    - If no error is returned, the game settings is returned.
    - Errors: *DisconnectedError*
- addressSet, err ← GetCars()
    - Get addresses of other cars currently connected to the server (joined the game).

    - Errors: *DisconnectedError*

## Car Node API

Car node listens to connection from other car nodes, server, and player on a known TCP:IP port.

**Car Node - Server API**: A server node can invoke the following TCP RPCs to a car node:

- answerString, err ← SolveQuestion(question)
    - Give next question in queue to be solved (by the players connected to this car node)
    - If no error is returned, answerString is returned
- err ← TakePenalty(timeout)
    - Cause interruption in car's game play with the given timeout period
    - Invoked when car node sends wrong answers more than given threshold
- err ← UpdatePosition()
    - Invokes rpcs to other car nodes to update the new position of current car node after the car answers the question correctly

**Car Node - Car Node API**: A car node can invoke the following TCP RPCs to another car node:

- err ← SendInterrupt(timeout)
    - Cause interruption in another car's game play with the given timeout period
- err ← UpdateMap(carNumber, newPosition)
    - Validates the new position of the given car node
    - Updates map with new position of the given car node

**Player Node - Car Node API**: A player node can invoke the following TCP RPCs to a car node:

- err ← registerPlayer(address)
    - Registers a new player node with an address so car node knows exactly who is part of its team
    - Errors: *PlayerAlreadyRegistered*

## Player Node API

Player node listens to connections from car node on a known TCP:IP port:

**Car Node - Player Node**: A car node can invoke the following TCP RPCs to a player node:

- answerString, err ← answerQuestion(questionString)
    - Gives question to a player, and receives answer back

# Distributed Systems Aspects and Challenges

## Synchronization

Each car must store the most up to date game state, such that when one car moves forward in the map, the other cars in the game should also update their game state. Once server checks that a car has provided a correct answer, the car node will advance its position, and let other car nodes know of its new position. Other cars will check the new position of the given car against that car's old position and validate if the move was valid. (i.e. valid move is when a car moves a set distance specified in game settings after answering a question correctly).

## Concurrency

When multiple car nodes reach a reward position at the same time the player nodes in the car node network will choose a winner and then decide which one of the car is awarded that reward. Then each car node will take the majority decision (i.e. car chosen by over 50% of players) and send it to the server. Server will then notify all car nodes of who is the winner. If there's a tie, then winner car is chosen at random.

## Scalability

The system has the ability to scale with the number of car nodes connected to the server and with the number of player nodes connected to each car node. Initially, the server will allow, for a period of time, for car nodes with the minimum number of players to join. It will accept as many "full teams" as possible in that allotted time. However, after the game has started it will not allow more car nodes to join.

## Integrity

When a malicious car node tries to send a location that is not consistent with other car nodes, there will be a validation process in which the other car nodes will deem it unfair. At that point the malicious car node will have its position frozen for a certain amount of time to prevent it from doing so again.

## Failure Handling

In our game, player nodes have a chance to be disconnected from the car node (from interruptions, etc). When a player node has disconnected, they will have an option of returning back into the race and reconnecting into their car node by solving a Proof of Work (difficulty will be set through a config file). Car nodes will always be receiving questions/moving in the race whenever it has at least one player node still connected to the car node. When all the player nodes are disconnected, the car node will freeze in its position and wait until a player node has

reconnected. In addition, every time the designated driver node of the car node is kicked out, a new player node will take its place. There will be a restriction for disconnected player nodes such that the player node can only go back into a car node in which they have disconnected from.

If a car node disconnects from the network (or crashes), it will take itself out of the game and disconnect all of the player nodes connected to it. Their position will be stuck at where they were before and they will no longer be able to return to the game or reconnect to the server (equivalent to quitting the game) until the game ends.

# Assumptions

- Server is always online and will never crash or disconnect.
- Topology from the server will ensure that all car nodes are connected to one another.
- Player nodes can only communicate with the car node it is connected to and no one else (including other players and the server).
- The server handles the list of questions, which will be consistent for all car nodes and in addition, it will be either multiple choice questions, true/false, or arithmetic questions.
- The server can handle infinite number of car nodes that register under a specific time interval.
- Car nodes will never deny a player before joining the race and never deny another car node connecting that tries to connect to it before the race starts.

# Testing Solution

In order to fully test the system we will need to have a set of cars, each with the same number of players.

- The server will run first, and then will accept connections from any even number of car nodes
- Once there are at least 2 or more car nodes then a player node can then connect to any car of their choice using the command line to specify which IP address and port to which to connect to.
- There must be the same number of player nodes across all the car nodes
- Once the connections are all initiated then the gameplay can be tested by trying out the feature in which the lanes shrink in size and then the system will have to decide who to put in the front in the case in which two or more car nodes try to get into the same position

- Need to test out the interrupts and exceptions for when a player node has claimed enough rewards in order to slow down other players of choice
- We will test the system to see how the nodes fail and connect or disconnect, to see the game state to see whether the positions are correct.
- By using Azure we can make a number of virtual machines to get an even amount of cars and a number of players to connect to and a server to test out the system completely.

# Threat Models

There can be a number of threats that can compromise the system from its intended uses.
The following will outline some of the issues:

- There is a case where a player runs a bunch of player node apps connected to either the same car node, or different car nodes in order to answer questions faster or ruin other team's majority decision.
    - This can be solved by checking for each player's ip address that tries to connect to a car and ensuring that no other cars currently have this ip address stored.
- The car could potentially lie about its position and send out wrong information to the other cars to try to game the system and get ahead of other player nodes.
    - The validation between peer to peer will solve this issue by each car node having its own map of positions of every other car node in the system and will disseminate information about their own positions to be validated by other car nodes
- Car nodes in the race can buy interruption items from the Server by using the rewards/points they have accumulated by answering questions quickly. The car node can potentially give the Server the wrong amount of points that they have and buy an interruption item even though they do not have enough points.
    - Server keeps tracks of the amount of rewards/points given to all the players in a map to check to see if any one of the car nodes are trying to buy an interruption inconsistent with its reward amount.
- Very rare case but the server could potentially get hacked into where a player gains access to all the answers and then uses that to win the game easily.
    - We will assume that this attack will not occur on the system.

# Azure Deployment

The project will be deployed on Azure, where each virtual machine will include either the server, car or player node code. Each node will be deployed on a unique virtual machine on Azure.

# SWOT Analysis

## Internals

*Strengths:*

- Group members are familiar with one others coding practices and also have communication experience with one another making it easier to coordinate
- Team members are hard working, intelligent, and well versed in the systems we intend to design.
- Members have experience programming a peer-to-peer system with error handling from Project 1.
- Able to identify the areas that need improvement and then collaborate and collaborate together to implement solutions.
- Members in the group all have knowledge on the proposed project and have an idea of what is going on in every aspect of the project.

*Weaknesses:*

- Not being able to schedule meetings could become an issue with people having other courses to work on.
- Complexities that may hinder the robustness of the program should be avoided while still maintaining a level of complexity as to satisfy the end-user.
- Having to use the golang testing framework which is not familiar to any of the group members.

## Externals

*Opportunities:*

- Project idea is neat solution to a game which involves many clients in a peer-to-peer system
- Construction of a system that applies aspects of distributed systems to the best of our abilities
- Using a new tool for everyone that is Azure (although we already used in project 1 it is still a learning process).

*Threats:*

- Time constraints on project deadlines not being met.
- Adding features to the system to best showcase aspects of distributed systems without being too simple.
- Being able to fully test the system and debug most of the key bugs affecting the system before rolling out on the final deadline

# Timeline and Task Breakdown

The proposal draft being due on Mar 2nd is not included in the deadlines as this report encompasses that date

| Date | Event |
|---|---|
| March 9 | Final Proposal for report is due (This is done as a group effort so each member made a contribution) |
| March 13 | Finalize all implementation as well as divide up the tasks and being the initial coding of system<br>Ben - Server<br>Kevin - Car<br>Yooni - Player<br>Jonathan - Error Handling / Validation |
| March 18 | Implement the car, player and server so that the car can talk to the server and the players can communicate with the player.<br><br>Ben, Kevin, Yooni, Jonathan (It will be a team effort to construct the system from the ground up and then combine all our repesective work) |
| March 23 | Schedule a meeting with the TA |
| March 24 | Coordinate system for the track, and features involving concurrency between cars, interrupts and exceptions for car-to-car functionality.<br><br>Ben - Mapping coordinate system for track<br>Yooni - Concurrency between car<br>Kevin - Interruption throwing from car-to-car<br>Jonathan - Interrupts |
| March 31 | Finalize all the details, debug the system and start the testing process<br><br>It will be a combined group effort to finalize all the code and to debug the system as well as testing |
| April 6 | Project code freeze and all testing finished |
| April 15 | Demos and peer review<br>Group effort |