William Casey Flynn, Mack Sutherland

Networks – Project Report

In our implementation of the game, we opted for two communication strategies; TCP connections to the game server, and UDP communication between clients. The decision to use TCP communication for the server was driven by the fact that TCP guarantees reliable data delivery which will enhance the user experience. The decision to use UDP communication between the clients was driven by a few factors, the most prevalent of which were to keep the communication overhead down, and to simplify the server game logic. There are, however, some pitfalls associated with using UDP, the most prevalent of which is the communication is not guaranteed. This may at some point lead to violations of the rules of the game (we, however, did not see this in testing). In the event that a message sent from one client to another is not received will only manifest in one of two violations to the game rules, as the communication between clients is only related to buzzing in, and sending answers.

The first violation is relatively trivial, if a player does not receive a buzz in from another player may also buzz in and answer the question. This scenario will allow two players to score, but will not interrupt the game cycle. The second issue that could be encountered is if a player receives a buzz, but does not receive the player answer. Without mitigation this would have led to a deadlock in the game until the player that did not receive the answer left the game. This, however was corrected by implementing a timeout while waiting on another player to answer the question while waiting. If this timeout occurs, the player discards the current question and awaits the next question.

We did identify a scenario in which the rules of the game could be violated due to high latency. In this scenario, if two players were to buzz in at relatively the same time (i.e. within milliseconds of each other) in the time it takes the packet to traverse the network, one player may answer the question without comparing the timestamp of their buzz relative to the opposing players buzz. We could have mitigated this by delaying the game by the maximum RTT of a packet or by an arbitrary amount of time (e.g. 1 second), however we opted to not address the issue, as we could not recreate it, and we believed the delay to be a hindrance to game play.

Scalability is always a concern in software projects. In the current implementation the game server can host up to 2^32 players, and run 2^24 concurrent games. These are obviously a theoretical value, and a more realistic estimates would be based on the hardware that the server is running on. The limiting factors would be the number of supportable TCP connections (running on server grade hardware, this would only be able to support several hundred concurrent connections as there is a thread associated with each connection) and the processing overhead of maintaining game threads (this would be on the order of hundreds, possibly a few thousand depending on the processor). We did not test these theoretical limits as the scope of the project did not include such testing.

In theory, if hosted on proper hardware, the game server should support 500 simultaneous clients, however, game function may be effected by processor latency due to the number of threads it must support. When considering this many clients, we would also have to consider the number of games running as there is one worker thread spawned for each game. The stack limit for a 32 bit instance of Windows 7 is 2GB, if we assume that there is 1MB of memory overhead per thread (this should be sufficient) then we would be able to support 2000 threads. This limit must be divided between clients and games.