# Authentication:

One of the major parts of the project that was not completed by the grad student was a failure to authenticate connections. In order to improve the security in this manner, a shared key authentication system was used. First the key was put in a place where both the client, or entity connecting, and the server, the entity the client connected to, could access. In the specific file system that was in the main homework directory. The additional security was coded in TCPConn. The system uses a state machine to track what state it is in and what it should do in that state. Before the authentication model was used, there were 7 states. In order to perform the security algorithm, 6 states were added. The algorithm is found on page 514, in *Distributed Systems* (3rd ed), the class textbook. The first step of the algorithm is that the client connects and allows the server to know who is contacting it. In the implementation, that would be the SID transfer. However, since the connection is already established and we were given a code base to start with, I chose to start with the server sending a challenge and the client waiting for a challenge. This allows the code that already exists to still function with almost no modification. Additionally, there is a security benefit of less information about each side of the connection being shared before identities are confirmed. That being said, the server sends a challenge. For challenges, I coded a random generator that selects 16 random numbers from 0 to 255. The odds of that being the same between runs is small. This helps protect from attacks as listening to network traffic would not help attackers narrow down anything to gain useful information. After sending the challenge, the server goes to the next state to wait for the client’s response. The client then encrypts and returns the message. The client goes to waiting for a verification to continue from the server. The server first decrypts the response to the challenge. If it still matches what was sent, then the server sends a verification back and waits for the client’s challenge. If it was not correct, then the server terminates the connection. The client creates a challenge in the same way that the server did and then sends that and waits for a response. The server also encrypts the challenge and sends it back. The client attempts to decrypt and as long as they get the expected message they go to the next state, which is sending their SID. The server went to waiting for a SID from the client after sending its response to the challenge. The code then carries on with the rest of the information sharing. One of the primary attacks that had to be guarded against with this authentication models was a reflection attack. In the reflection attack, there is a change in the order of packets sent allowing for a clever attacker to get the information and fool the server. First, the attacker starts a session and sends his information and a challenge. In this model, the attacker has an advantage because they send the challenge first, making the server “prove” itself to them first. The server then responds with the encryption and its own challenge. The attacker can then open a second session and then send the server’s challenge from the first session as its challenge. The server would respond with the correct encryption which the attacker could then use back in the first session to bypass security. My implementation avoids this because the client that is connecting has to verify itself first before the server will ever send anything encrypted. By the time the connector has the information of an encryption, they have already been verified.

# Deduplication Solution:

Another problem that the system has is that it does not appropriately deal with duplication. This section will discuss how this was handled in the implementation following constraints. Other options that would be better designs will be discussed in another section. The initial problem was that the drone broadcasts its message to all the towers that are within a certain radius. These towers then have to replicate their data to the other servers to make sure that all the servers are up to date on what is going on and can track the drone. The problem is that there is no check and since the towers have different timestamps, there is no guarantee that the timestamps will match up. If they do not, they will mess up the drone’s flight path which would make it harder to calculate. In order to address that I use an algorithm that eliminates duplicates. It runs in O(n3) time with the worst case being all replicated data that is unique. Each time a piece of data is replicated in, the server compares it against every point that is in its current database. If there are no duplicates then it keeps it. If there are duplicates, then it picks whichever one was entered second and gets rid of it. It then scans the rest of the database to ensure no other duplicates happened. Comparing each plot against each other plot is an O(n2) task. This would be done at most n times if all the data was replicated. It has to be done this way because data picked up by the antenna that is paired with the server was not code that we were supposed to edit. Since we are unable to edit that code, we have to check the whole database to catch any duplications from an antenna injection that might have happened after a replication for that data already happened. One more check is also done at the end just to ensure that the data is correct. One such sweep would be sufficient if the data only had to match at the end. However, the data was supposed to be consistent so that duplicates were taken out as quickly as possible so waiting until the end to do so was not possible in this scenario. The code checked the plots so that it could handle different drones being in the same place and not remove the data. They also handled time differences. If the drone flew back along the path that it flew out on, as long as there is a 20 second difference between the coordinates matching, the code can differentiate between the two. I noticed this problem in my testing and found a way to fix it by adding an additional check. I selected 20 seconds because the system documentation states that drone timestamps are with 15 seconds of the global clock. Additionally, the severs are with +/- 3 seconds. Therefore, that means a max of 18 seconds difference, although the servers can only be up to 6 seconds different. I felt that this amount would be able to tolerate an acceptable amount of flexibility and could only potentially lose a couple points in a scenario where the drone is trying to immediately turn and return on a straight line.

All in all this part (and the time skew) probably took me 30+ hours to complete. I didn’t want to turn in a project that had not completed this part so I decided to keep working and finish it instead of turning in an incomplete project.

# Time Skew:

The third problem was related to the plots and it is that the servers are not synced on time but the server uses its time to timestamp each piece of information it receives. This leads to the drone seeming to speed up or slow down when the server’s points do not match. There are two steps to fixing this problem. The first is having a leader so that all the servers know which clock to base their final results around. Finding a leader is done through an election. In my code the election consists of getting all of the server’s names and then picking the one with the lowest ID. The instructions stated that this election should not send packets for coordination so it does not. The next step is correcting the time differences on each server. First, any time there is a duplicate that includes the local server, I calculate the difference between the local server timestamp and the other server’s timestamp. I store that information and use it with future points and to fix past points. I have to use the local time until the end because we are not supposed to touch the antenna simulation code so I can’t modify the injected points which means when I adjust them at the end, they all need to be based on the local time. At the end of the code before printing out, I run through the list again and reverse the difference between the local server and the leader and that puts the file in terms of the leader instead of local.

# Consistency Model

# Communications method

# Naming convention

# Coordination techniques: