Samuel Carroll
CptS 464
Project 2 Writeup
May 2nd, 2019

This project came with a lot of ups and downs as I made my way through it. I do think that overall I learned a lot by maneuvering my way through this project. Throughout this assignment I learned a lot about how to modify rti's system in order to track a lot of moving pieces all at once. I started by developing my PubThread files, which meant getting threads to work on rti. I chose to use c++'s standard thread library in order to handle my multiple PubThread instances. Once I figured out the threading of rti publishers I moved on to actually building my publisher instances. This was by far the most difficult part of this assignment, which possibly is because I did it first. I started by developing my helper functions, ones that determine how full a bus is, whether an accident occured, and the traffic conditions. These were pretty simple c++ function calls within my ThreadPub class. I then took my functions I'd written and placed them into my main loop in my publisher function. I set up my publisher to check these functions every time it loops and pass the various responses in the publish object. I needed an Accident publisher to be worked in with my Position publisher to ensure that both message types would get out. I ended up merging all of the set up and calls of an Accident publisher in with my position publisher in my ThreadPub class, so it all became a single super function that only sends an accident message when an accident occurs, and other than that sends a position message every loop (and sleeps between them based on if accident occured or the given traffic conditions. I suppose this looks less like what I learned and more just what I did, but the truth is all of this was new. This whole project, and the previous project were filled with new things I had never worked with before. Learning how to debug this entire system was a hugely important skill I gained throughout this project, I had to track where issues were occuring and make at times some very large adjustments to my project layout based on what I found. I'd never dealt with nesting publisher types in project 1 so that was a very interesting and new challenge I faced, I learned how to deal with this situation and make it work for this project such that I could have both my Position and Accident publishers present in my function. Similar to this I also had to get my subscribers working to receive both Position and Accident messages. I did this process after my publisher so I followed a similar method of nesting my subscribers in a single function, it was a different process because the publishers and subscribers do vary in their implementation, however overall it was a lot of the same in getting that to work like my publishers. With the subscriber side I had to figure out how to make sure every published message was read, so on all of my subscribers I removed sleep time. Now this brought in another important roadblock, how to ensure I was only reading the publish messages that were relevant to that subscriber (that's all for the operator but more specific for a passenger subscriber). I solved this with simply an if statement, once a passenger gets on a bus we store that buses name, now it will only print a statement when it receives a message from a bus name with the same name that the passenger is on. I considered letting my passenger subscribers sleep for the time between stops that is passed with each position message, but this left too much potential to miss a position being published for that bus (i.e. when I tested it I occasionally missed position

messages). From this I learned to just let it constantly read and parse out any irrelevant position messages via code. As I've stated I learned a lot from this assignment, it was a lot of fun to work through all of the different things that came up throughout this process. I think getting to work with so many moving parts in a simulated distributed environment is really beneficial to this course as it teaches you a lot and provides a lot of perspective on how to work with the things we learn in class.

I encountered quite a few challenges throughout this assignment. Two big problems I came across in this assignment were thread related, the first came when I started working, the provided makefiles would not let me pass function parameters into the c++ std::thread, I could thread a function call but if that function took parameters I couldn't call it. This took me a bit of digging to solve but the final solution ended up being quite simple, I just had to add "-std=c++11" to the end of the compiler flags in my makefile for my publisher (the only one that uses threads). This was actually my first modification to my makefile, it happened right at the start of my project and I'm glad that it did because it gave me the confidence moving forward to make other modifications to my makefile, which was a very necessary thing to do on this assignment. My other thread issue was much more rti related than that previous issue, this issue caused me to be unable to run more than four threads at a time but also thankfully provided an error message. When I searched the error message I found some rti community threads that included similar situations, essentially what was occuring was there's a QOS setting for resource_limits.max_objects_per_thread that was limiting how many threads could run for my publisher. I read through the community post I found and discovered that this QOS setting could only be modified in code (a fix to the xml parser had been sent in but no updates been made yet, as I did try going through xml). I ended up using code provided by the original commenter to fix this issue, simply 4 lines of code at the start of my PubThread publisher function was all it took to resolve this issue. My third big issue I hit while working through this assignment was a bit weird and I'm still not fully sure what the issue was. My original set up for my PubThread publisher was two separate publisher function (one for Positioin, one for Accident), the Accident publisher only went through once, and was called by the Position publisher when an accident had occured. For some reason this method actually sent no Accident messages at all, I tested with my Passenger and Operator subscribers, as well as with the provided Accident subscriber made from the idls, none of these would read accident messages, so I figured nothing was being written even though all of the code seemed fine. I eventually opted to adjust my approach, I put my separate accident publisher into my position publisher, but only called a message publish when an accident occured, this solution worked and is thus my final implementation on my threaded publisher.

All in all my system does work, I had two final things I could never seem to resolve and thus don't properly work for my final submission. In the idl's you provided for this assignment the time between stops value was set to a long type. This means that it can't pass decimal values, for this reason my time between stops values that gets passed only sends whole numbers in a message so I can only have 1, 2, 3, etc.. for time between stops. I attempted to solve this issue using the nanoseconds option in the DDS sleep call because I could use whole numbers and

still get decimal seconds doing this, unfortunately when I got too much heave traffic the value started coming out negative (I assume due to overflow). This issue meant that this solution would not be practical, thus I had to stick with using whole numbers of seconds and dealing with a less specific value for my time between stops. The other issue I couldnt resolve is when I first start all of my threads for my publisher. For some reason it seemed to have trouble sending the first stop as a message, the prints on the first one would all become overlapped and mix together, and I could maybe get one or two of my published first stops to be read by my operator subscriber. I never figured out why this was occuring, I assume it has to do with how the threads all start since it doesn't happen after the initial startup but that's why my images and outputs for this assignment don't show many values at stop one at the very beginning. It was certainly a weird issue but didn't have any impact on the overall project unless I tried to start a passenger at stop 1 from the first startup (stop 1 works fine after the first loop so it's not an issue with that stop value or anything). Outisde of these few errors the rest of my project works really well overall, the threaded publishers all work and are received by each subscriber, stops aren't missed as far as I can tell and the stops remaining count works as well so I think I had a successful overall implementation of this project.