SOFTWARE-DEFINED WIRELESS NETWORK FOR REAL-TIME SENSING

An Undergraduate Research Scholars Thesis

by

AUSTIN M. KEITH

Submitted to the LAUNCH: Undergraduate Research office at
Texas A&M University
in partial fulfillment of requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Faculty Research Advisor: Dr. I-Hong Hou

May 2022

Major: Electrical Engineering

RESEARCH COMPLIANCE CERTIFICATION

Research activities involving the use of human subjects, vertebrate animals, and/or biohazards must be reviewed and approved by the appropriate Texas A&M University regulatory research committee (i.e., IRB, IACUC, IBC) before the activity can commence. This requirement applies to activities conducted at Texas A&M and to activities conducted at non-Texas A&M facilities or institutions. In both cases, students are responsible for working with the relevant Texas A&M research compliance program to ensure and document that all Texas A&M compliance obligations are met before the study begins.

I, Austin M. Keith, certify that all research compliance requirements related to this Undergraduate Research Scholars thesis have been addressed with my Research Faculty Advisor prior to the collection of any data used in this final thesis submission.

This project did not require approval from the Texas A&M University Research Compliance & Biosafety office.

TABLE OF CONTENTS

ABSTRACT

Software-Defined Wireless Network for Real-Time Sensing

Austin M. Keith
Department of Electrical & Computer Engineering
Texas A&M University

Research Faculty Advisor: Dr. I-Hong Hou
Department of Electrical & Computer Engineering
Texas A&M University

Traditionally in the field of electronics, hardware is designed, developed, and improved on in various methods, whether it be increased storage capabilities or smaller models. Software applications lagged because of the hardware requirements to operate software, but increasingly, software tools are replacing technology that relied heavily on hardware components where applicable because of the abilities to both modify the technology easily and to consolidate tasks in an automated fashion. This research focuses on the networking space and aims to replace hardware architecture with software, as well as write algorithms to intelligently allocate incoming data.

To orchestrate this architecture and the algorithms, the modern tools of software-defined networking and software-defined radios were combined. This created a network capable of transmitting packets over-the-air, with the network itself having separated the data plane and control plane in the software-defined networking standard. The control plane is written entirely in software, allowing modifications to be made across the whole system relatively simply. In this research, two software-defined radios were used to represent a base station and a field multi-

sensor collector respectively. The field sensor transmits real sensor data from a web database that represents readings of the resistance of a gas over time from fourteen sensors. The base station radio can only receive a single packet at a time from the secondary radio due to bandwidth constraints, and so, using a software-defined controller, the various scheduling policies are compared to develop the most efficient means of processing the individual data packets.

The final algorithm started from basic round-robin before evolving into weighted round-robin, with measurable results in terms of root-mean-square error values for each sensor and one for the total transmission period. The weighted round-robin was upgraded a step further to have real-time weight updates at regular intervals based on the accuracy of prediction for the next value in the sequence, per sensor. The contrast between the three stages of development for the round-robin algorithms is plain to see, with steady improvement between basic round-robin and weighted round-robin, and drastic improvement between weighted round-robin and the smart algorithm. The results from the research project yield a final draft of communication between the software-defined radios that produced an effective and efficient manner of software-defined networking.

DEDICATION

To my family, for supporting me continuously. To my fiancé, for always being there. To my friends, for keeping me focused.

ACKNOWLEDGEMENTS

Contributors

I would like to thank my faculty advisor, Dr. I-Hong Hou, and my graduate student colleagues, Siqi Fan and Khaled Nakhleh, for their guidance and support throughout the course of this research.

Thanks also go to my friends and colleagues and the department faculty and staff for making my time at Texas A&M University a great experience.

Finally, thanks to my parents for their support and encouragement and to my fiancé for her patience and love.

The B210 software-defined radios used for over-the-air communication were provided by the Wright Brothers Institute and the Air Force Research Laboratory for participating in the Beyond 5G University Challenge for 2021-2022.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

No sources for funding were used for the duration of this research.

NOMENCLATURE

USRP – Universal Software Radio Peripheral

SDN – Software-Defined Network

SDR – Software-Defined Radio

GUI – Graphical User Interface

RF – Radio Frequency

GMSK – Gaussian Minimum Shift Keying

GHz – Gigahertz

RMSE – Root-Mean-Square-Error

UDP – User Datagram Protocol

WRR – Weighted Round-Robin

1. INTRODUCTION

The goal of this project is to have a software-defined network that takes advantage of two software-defined radios, one working as a base station and the other representing a series of field sensors, to transmit a variety of sensor data over-the-air. Because of bandwidth constraints, only a single data packet can transmit through the radios at a single instant of time. Algorithms written for the controller of the network allocate the incoming sensor data efficiently to reduce the time it takes to process the incoming data. Because much of the operation of the entire system can be modified through the controller software, the network can switch between various scheduling priorities to demonstrate the differences in efficiency between them.

1.1 Traditional Networking

To begin the understanding of this research project, one must understand how a traditional network works as well as the issues surrounding it. Networking itself is simply a connection of two or more devices that allows them to share information with each other. The number of connections and geographical range of the connections determines the categorization of the network, but there are common elements between all traditional network types. Each are built on connections between computers that are created with the use of switches and routers, as shown in Figure 1.1 below. Switches ensure that data packets are communicated and sent properly within a single network, while routers ensure that data packets are correctly transmitted between networks themselves. This creates a hierarchy of networks, routers, and switches that are highly dependent on physical hardware for the establishment of the network and for future modifications.

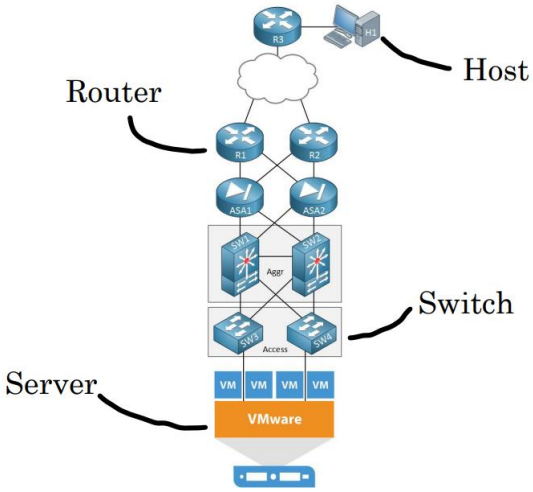

Figure 1.1: Network Components

An example of such a network on the larger side of the scale is the Internet, which is a giant web of interconnected networks so that any device can send information to any other device. Other examples include university campus networks, a network of computers in an office building, and a network of gamers playing online together. Each of these examples is highly dependent on hardware, limiting its flexibility and increasing its complexity as the number of users join.

1.1.1 Problems with Traditional Networking

Some of the issues associated with this style of network building and amendment have been hinted at. Most of the current problems with traditional networking can be categorized under either an issue of complexity or an issue of inefficiency.

The immediate problem with traditional networking is that the number of users has grown at a tremendous rate in the last few decades. Both the number of people that own devices has increased as well as the number of devices that a single individual owns. Regardless of the networks that these devices are connected to, the overall complexity of networking has increased

dramatically because of the number of connections that needs to be maintained has increased dramatically. Each additional connection requires the use of hardware to either make physical connections (with a cable) or virtual connections (with Wi-Fi, for example). This creates an enormous number of switches, routers, and other network architecture hardware that must be independently maintained to ensure the network operates as intended. Additionally, any changes in the way that devices are connected, including emergency changes if a critical part of the network fails, are slow to happen and difficult to execute. Network engineers must be well-trained and staffed at many companies just to accomplish this difficult work.

The inefficiencies of traditional networking may be obvious with many disciplines of technology turning towards software, even in fields that were dominated by hardware in the past. The limitations on flexibility with physical hardware in the environment of increased complexity create problems in areas of high traffic in a network in a live setting, and the potential for network connections to adapt in a highly efficient manner to the information that they communicate is quite high. However, even in an offline setting, changes to a network are limited because multiple parts of the network must be individually modified. The problem of multiple network engineers changing different switches and routers without communicating and causing conflicts with the network connections also occurs.

1.2 Software-Defined Networking

The newer type of networking that is starting to be employed is known as software-defined networking, or SDN. SDN differs largely from the traditional network due to an abstract layer of computer architecture known as the control plane which is separated from the data plane in SDN. Before, the data plane and control plane were paired together, meaning in practicality that each fixed piece of hardware in the network that is responsible for sending packets was

independently operated. With SDN, the control plane is separated into a centralized unit known as the controller, and the devices responsible for communication are directed by the controller, as shown in Figure 1.2. In this manner, a proper analogy is that traditional network architecture is orchestrated like a pack of wolves, where each individual animal makes decisions so that the pack can survive. A similar analogy can be made for SDNs which operate like a beehive, with individual worker units that have a hive mind and adapt to new situations collectively.

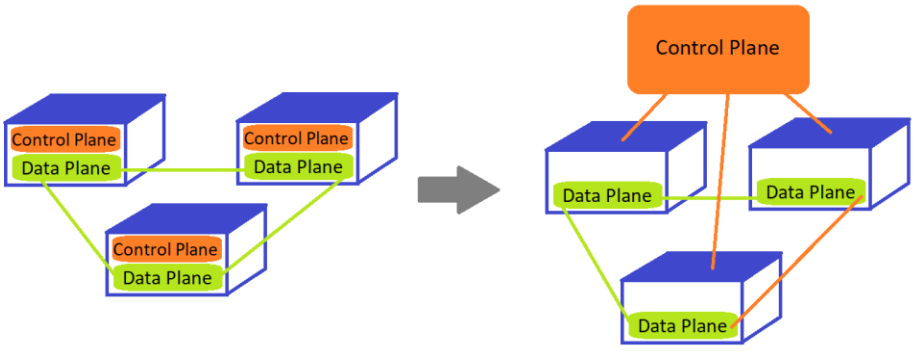

Figure 1.2: Control Plane Separation

Because the SDN controller is programmable and runs the network, separating the control plane and data plane has huge benefits to networking and addresses many of the core issues defined with traditional networking.

1.2.1 Benefits with SDN

There are many benefits associated with switching from a traditional network to a software-defined network. The first and most obvious benefit is that network operation is controlled by a single, programmable unit. This means that, rather than a network engineering modifying switches and packets at an individual level, a single network engineer can write code for a controller that modifies the entire system. This greatly reduces the complexity of managing a given network. Adjustments are automatically made within the network to adapt to the new

communication methods written in the controller software. The increased level of flexibility this provides cannot be understated. Adding multiple devices to a network becomes much simpler as the controller adjusts virtual switches and routers to accommodate new users. Not only can a single network be changed drastically in a short period of time, but an SDN can be repurposed for multiple different functions as well. Established virtual networks that are no longer needed can be modified to serve a new function without the steps necessary in traditional networking to rebuild the architecture.

Additionally, maintenance of a network using SDN technology is much simpler. The controller operates by changing the connection lines and general network architecture to address the conditions set by an engineer within the software. This allows automatic readjustments in connections between devices when errors occur. Network failures at critical traffic points are no longer a major concern while the network is manually repaired as done with traditional networking; instead, the SDN establishes new connections automatically to ensure operation continues as intended. While this implies benefits that address the complexity and flexibility issues described under the traditional networking infrastructure, it also improves the security of networks. Point-to-point connections that are attacked by malicious users can adapt with a new secure channel of communication once the network controller recognizes that an attack is happening.

1.3 Software-Defined Radio

In a similar line of thought, radios can have a huge increase in flexibility and a reduction in maintenance cost and complexity by moving some part of the physical functionality into the software domain. This precisely describes software-defined radios (SDRs), where some aspect of the physical elements of a traditional radio are controlled by programmable software. Some

physical components of a radio are necessary for transmission and signal processing, but the general idea of separating the control plane into software applies here just as done with SDN. There are a great variety of types of SDRs as their share of the market increased over the last few decades, and there are a multitude of benefits that come from using an SDR over a traditional-style radio. In Figure 1.3, the two radios that are used for the duration of this project are shown.

Figure 1.3: Two Software-Defined Radios (B210s)

1.3.1 Benefits of SDR

Several of the benefits associated with SDN over traditional networking are reflected in the benefits of choosing SDR over traditional radios. Modification of the purpose of the SDR is simple and only requires a change in the code that controls its operation. Similarly, maintenance of the radio is much less complicated when, instead of physically tampering with the device, new code can be uploaded that addresses the problem. Both simplifications massively reduce the cost of using the technology and provide greater flexibility than previously offered.

The added flexibility that comes with the programmable functions of the SDR is another benefit. A group of radios can be linked together and controlled through their software to do a variety of tasks that were unavailable previously. Commands or data packets that are sent over-

the-air using SDR technology allows devices in a network or similar infrastructure to be modified remotely. Radio frequency processing can be adjusted easily, allowing communication channels to rapidly change. This comes with increased security benefits as well, with emphasis on application within military technology due to the large use of traditional radios.

1.4 Complete System

The task of this project is to combine the new technologies and applications of SDR and SDN into a single system, and then to write algorithms that receive data packets and allocates them efficiently. This system is intended to be a software-defined network containing two SDRs that communicate to each other. Specifically, the controller for the SDN sends commands to a virtual switch by sending the command to the first SDR, which then gets transmitted over-the-air to the second SDR, and finally, the second SDR sends the command to the switch. The switch responds appropriately to the command, most of which were written previously by graduate students in this research field. In Figure 1.4, a block diagram of the complete architecture for the system is presented.

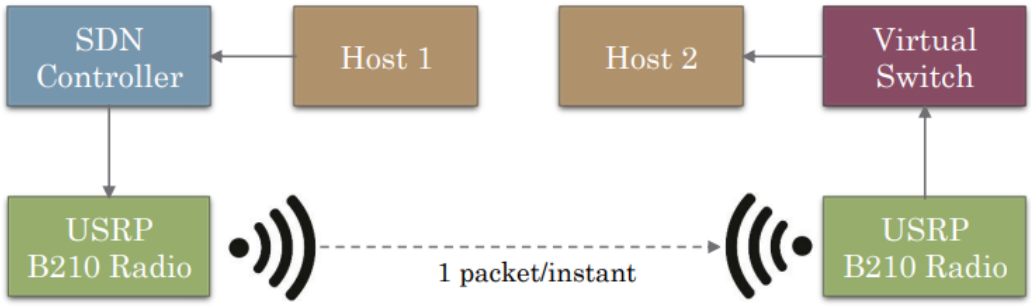

Figure 1.4: Complete Architecture of System

Some of the commands that are sent through this system change the scheduling priority of the SDN. Traditional scheduling methods, such as round-robin, are included, along with the other algorithms written for the purpose of this project.

2. METHODS

The approach to this research project is best described in two categories: the construction of the SDN and the development of the communication between the base station and field sensor SDRs. The first semester was primarily focused on the early work for implementing the SDN on the hosts and the research components of the project, while the winter break and second semester were instead focused on creating the block diagrams and Python files for the radio connections, writing the scheduling priority algorithms to allocate the data appropriately, and completing the infrastructure of the project.

2.1 SDN Construction

Much of the work involved in the initialization of the SDN was about reconstructing the basics from the work of previous graduate students using the open-source information from Ryu and OpenFlow (developed basic controller for network and methods for configuring settings). The terminal commands for constructing the first edition of the SDN were included in some documentation from the previous semesters, and using that information along with the public instructions, the SDN was first created on a single host running multiple virtual machines. The controller terminal, switch terminal, and a connection terminal were all running independently, and communication was confirmed with fake data being sent to a GUI that was included in the work done by the previous graduate students. Figures 2.1 and 2.2 show the early user interaction with the terminal and GUI when establishing the SDN.

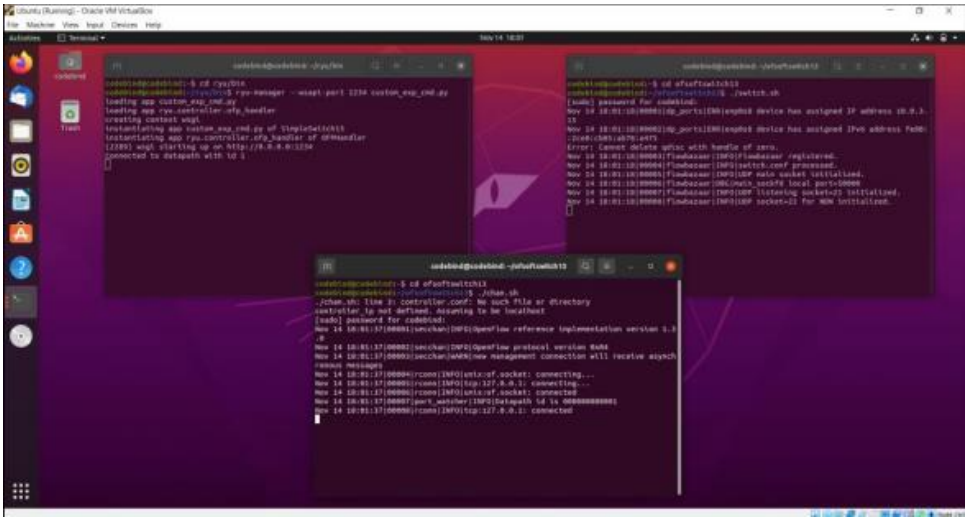

Figure 2.1: Terminal Interface for Switch and Controller

Figure 2.2: GUI Interface for SDN Controller

2.1.1 UDP Connection from USRPs to Localhosts

The original plan for the next step in SDN research was to create a UDP connection between the respective localhost and radio connected to the host. This was not known to be possible, so the alternative plan of improving the weighted round-robin algorithm with a smart algorithm that updates the weights of sensors in real-time was created. In the end, there was no

solution found on addressing this UDP connection, so the backup plan was implemented, and a smart algorithm was created. The SDN aspect of this project, because of this change, was less significant and primarily was relegated to research purposes.

2.2 SDR Communication

The construction of the architecture that allows communication between the two SDRs originally used the GNU Radio Companion software tool to construct block diagrams using the pre-built library meant for SDRs and other communication equipment. From there, the generated Python files from the block diagrams within GNU Radio Companion were modified by hand to develop the specific needs regarding this research. Each host device had its own radio, software, and Python files that were created collectively before individual modifications pertaining to the base station or the field sensors respectively were implemented.

2.2.1 One-Way Communication

The first block diagrams created were to establish some form of basic communication between the two radios. After researching the toolset within GNU Radio Companion and understanding the capabilities of the included functions, the transmission and reception files were developed on each respective radio's host device. The method of communication uses GMSK modulation on the transmit side, and GMSK demodulation on the receive side. The two B210 SDRs are both using a channel frequency of 1 GHz and a sample rate of 400k samples per second. These were chosen with the device characteristics and the nature of the project in mind. The two block diagrams that represent the files created to first establish one-way communication are displayed below in Figures 2.3 and 2.4.

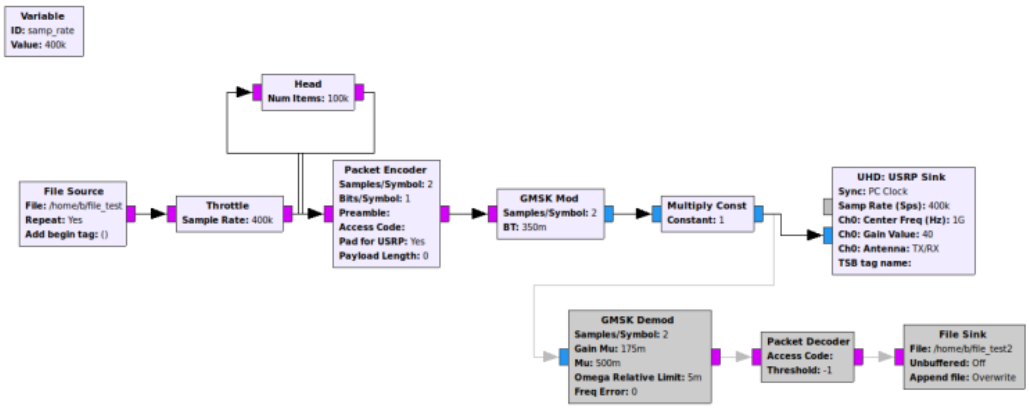

Figure 2.3: Transmission Block Diagram for One-Way Communication

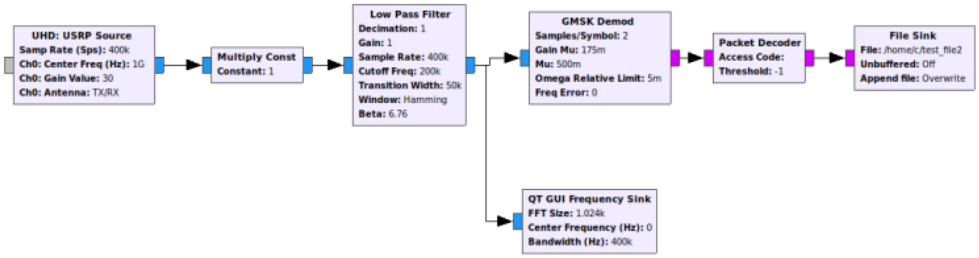

Figure 2.4: Reception Block Diagram for One-Way Communication

The actual data that was being sent from the first SDR to the second SDR came from a local text file on the transmission host with a series of fake data points created for the purpose of establishing one-way communication. The data is read from the text file, encoded into packets, modulated, and sent through the B210s before the process is reversed on the other side and the data is stored onto a new text file on the receiving host. This method of using a text file to send fake data is repeated until the SDN commands are implemented into the SDR's communication.

Running both files on each host at the same time resulted in the file being correctly transferred to the other side, with the only issue being a glitch that is built into GNU Radio

Companion that requires the message to repeat continuously. This is addressed in a future upgrade to the SDR communications.

2.2.2 Two-Way Communication

The first upgrade needed for the communication between the two SDRs is to establish two-way communication, or the ability for both radios to transmit and receive information. Once again, additional block diagrams are built in GNU Radio Companion for these modifications. Fortunately, much of the content of the blocks is the same as when establishing one-way communication, but the specific settings regarding the antennas being used and the timing when either actively sending or actively listening to a signal are changed to meet the new requirements. The block diagrams used for this point of the project are shown in Figures 2.5 and 2.6 below.

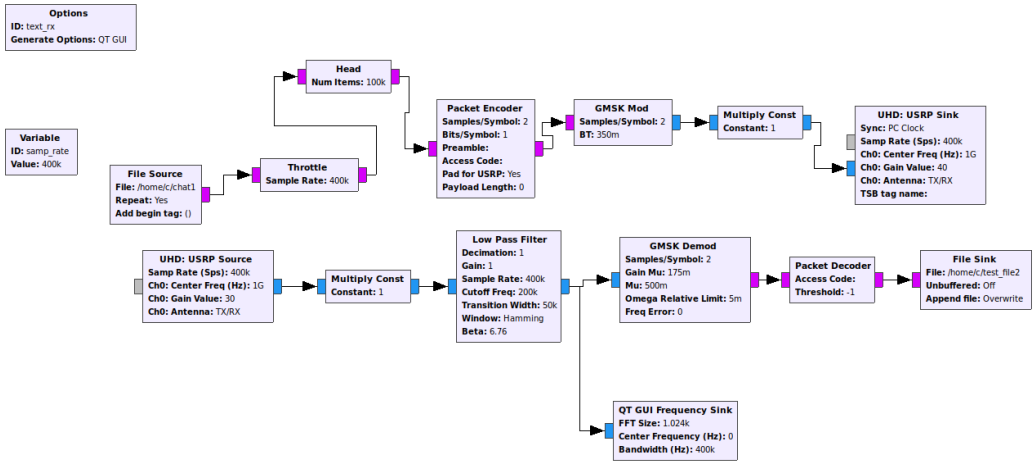

Figure 2.5: Two-Way Communication on Transmit Side

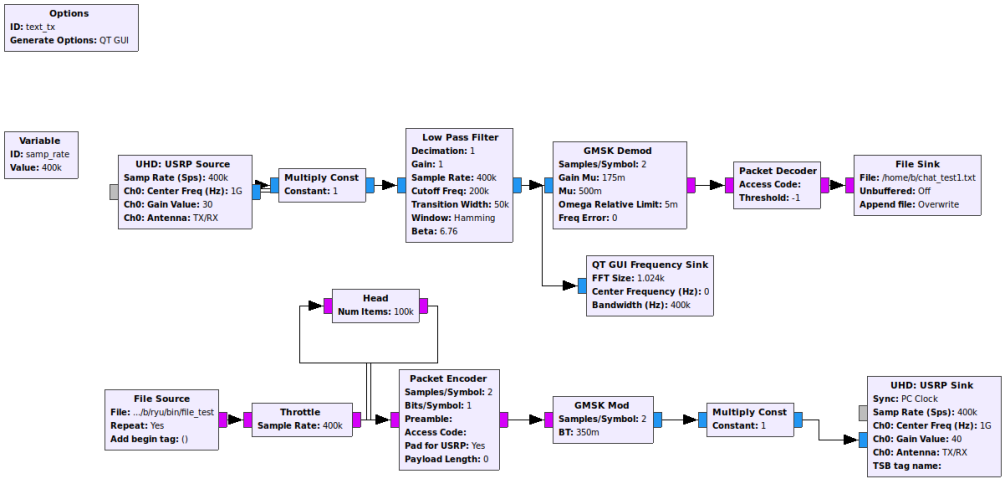

Figure 2.6: Two-Way Communication on Receive Side

After constructing the block diagrams for the radios, the generated Python files are modified to match the style of communication needed for this project. This means that timings and order are introduced to the devices; each respective device needs to be able to understand that it can only listen while it is not talking, and that while it is talking, it is not listening. These changes allow basic conversation between the SDRs to take place without interruption or other complications to the dialogue.

Once the Python files have been adjusted, the radios can send information back and forth to each other. The text file filled with fake data referenced previously for testing functionality of the radio communications is once again used here.

2.2.3 Transfer from Multiple Text Files

The next step in terms of communication between the radios is to add the capability to send from multiple options of text files on the field sensor side of the SDRs. In this model, the field sensor SDR has three text files, each filled with random data and identifier numbers for each data point as done previously. The base station SDR can request any combination of the

three text files, including just a single file, to be transmitted back towards the base station. The field sensor SDR will send each file completely, one-at-a-time, until the request is fulfilled. The repeat bug found earlier in GNU Radio Companion is not addressed at this stage of the project, and the consequences of its problems were addressed in this upgrade by having a short timer set for each file to prevent infinite repetition.

2.2.4 Round-Robin Communication

The further modifications made for the next iteration of communication between the two devices takes place entirely within the Python file. With the addition of two-way communication with multiple text files in the last upgrade, the changes here mostly relate to timing and addressing the repeat issue from GNU Radio Companion. Code was developed to address the repeat bug that takes advantage of the local text files on each host. The Python file will read from each file that is requested by the base station host, and record one line at a time from each file into a separate text file for transmission. The additional layer of reading from a file addresses the issue of repeating well, and it allows for this process to work (using the text file setup) without changing the base libraries of GNU Radio Companion.

Following the fix of the repeat bug, the code for a round-robin scheduling priority was implemented at this time in the research. Round-robin algorithms work by sending a single data point from multiple set options, and then going around each set sending the first single data point until you reach the second data point of the first set chosen. This process repeats, one-at-a-time for each data set (or text file, in this case) until all the data has been transmitted. This works in the Python code with timings set between the two SDRs to prevent transmission and reception on one device at the same time. Although this serves the purpose of establishing the first basic algorithm for this project, the next step in the code is to set up acknowledgements in the

communication, where after receiving any data or information, the receiving radio sends a receipt of data received back to the transmitting radio to ensure that no data was lost and needs to be sent again.

2.2.5 Weighted Round-Robin Communication

The next iteration of upgrade that was included was a series of smaller changes that resulted in a complete weighted round-robin algorithm, or a round-robin algorithm that takes in weight factors to give priority to the sensors that need it. As mentioned previously, the first of those changes was to implement acknowledgements. In this manner, after the field sensor radio sends a single data point to the base station radio, the base station responds with an ACK message to show that it has received the data properly. The field sensor radio waits until the ACK is acquired before sending the next piece of data in the list. This is important because it makes the communication channel more robust and decreases the likelihood of error in transmission. It also allows the algorithm to be less dependent on timing between the two radios, and instead allows them to be synced up like a conversation. At this point, much of the intentional delays added to ensure the timing of the radios were synced were removed and the communication process took much less time to complete.

Following this, the weights pertaining to each text file (and will later be each sensor) needed to be implemented. A user-input terminal request for weights was added, and after spending some time developing the code for a proper weighted round-robin algorithm for determining the order of files to send, the basics of weighted round-robin were present. The inclusion of the algorithm was successful and established the correct order for weighted round-robin, whether for two text files or for ten, by labeling each file in a list, calculating the proper

order based on the inputted weights, and ensuring that the next data point in the list to be sent had an identifier that matched the request from the algorithm.

After confirming the success of the ordering of data, the number of text files was extended to fourteen (still with fake data) to ensure readiness for the fourteen sensors with real data that are represented in this research. The real data was then included, replacing all the fake data that has been used up to this point, and additional code had to be written to address the formatting differences (all the real data was included in a single csv file). This was relatively easy to address compared with the other components of the algorithm.

For each sensor being represented, the next step to include for the weighted round-robin is a prediction scheme for guessing the next value that will be sent based on the data of the past. The prediction for the next value in each sensor was developed as the average of all of the data that has been sent previously, which was not the best prediction method but worked as a baseline. The significance of including this is that it allows a root-mean-square-error (RMSE) to be calculated and generated at the end of the communication between the radios for each sensor, and then added together to find a total RMSE. This is the metric that is used to evaluate the success of the models created. Final modifications are minor and include examples such as having the end of the file communicated to stop the communication, having the field sensor wait on the base station to be running before continuing operation, and others. The changes were successful, completing the requirements for the weighted round-robin algorithm that were desired.

2.2.6 Smart Weighted Round-Robin Communication

The last form up the algorithm developed is the smart weighted round-robin algorithm. The difference between the last version of this and the smart version is that the smart weighted

round-robin algorithm updates the weights in real time to get more data and make better predictions, resulting in a lower error rate. This final change required some restructuring of the previous Python codes to accommodate the changing weights, including additional communication back and forth between the radios among others.

On the base station radio side of the code, many updates were included. The first is the weight change calculations and parameters for making changes to the weight; this was accomplished with creative thinking and experimental testing of different techniques. The ending parameters established for weight changes are as follows: the weight of a respective sensor will increase by a factor of one if the difference between the prediction and real value is greater than two times the weight, and the weight will decrease by a factor of one if the difference is less than the weight. This allows a more dynamic change in weights than simply looking at the difference in prediction and real value; the weight factor implemented spreads priority more evenly over the sensors and prevents exponential growth or collapse in the weight changes. This method for changing the weights is quite stable, and by far gave the best results compared to other techniques. Lastly, the weights were transmitted back to the field sensor radio side of communication to implement the calculated changes.

The other major change to the base station code is the improved prediction model. As mentioned previously, the prediction for the next value at each sensor in the weighted round-robin algorithm was built on the average of all data that had already transmitted. This was changed at this stage of the research project to a much better model: the average difference between two data points for every pair sent thus far is added to the last number received. The shifting of the average from the real value to the difference in real values was remarkably successful at preventing wide errors in prediction.

On the field sensor radio side, the only changes made were taking the weight updates that were sent every twenty transmissions and incorporating them into the new weighted round-robin order that the scheduler makes. This required some creativity, not only for the difficulty of implementing the reorder in further transmission, but for recognizing the data that has already been sent. This problem was solved by including fillers of zero in the first parts of the order that had already been transmitted, and from there, the smart weighted round-robin worked perfectly.

3. RESULTS

3.1 Operation of the Complete System

The physical setup of the architecture as described in Figure 1.4 is modeled below in Figure 3.1. Each side of the radios has a host device that the user interacts with, and the radios are communicating with the various algorithms described over-the-air.

Figure 3.1: Overview of Complete System

There were complications in achieving a UDP connection between the localhost device and the USRP radios. This aspect was deemed to be beyond the scope of this research project, so instead the smart weighted round-robin algorithm was developed for this project. The Python codes for the base station and field sensor radios are contained in the hard drive of their respective host devices, and the public repository data used for the sensor data is stored on the field sensor radio's host. Otherwise, the files and information necessary for communication are created during the operation of the Python software files themselves.

3.2 Effectiveness of Algorithms

For data collection, the process was relatively simple. The algorithms were each ran with 150 data points total, meaning that there were 150 transmissions from the field sensor radio to the base station radio and 150 sent acknowledgements in return. Because there are fourteen sensors, this is adequate amount of information to compare the effectiveness of the various algorithms, which is measured with the RMSE values from each sensor as well as the total RMSE for a single iteration. For the algorithms with weights, the weights are varied over several iterations to get a solid set of data for comparison. Although the round-robin original algorithm did not include a measurement for prediction and RMSE values, entering a value of one for the weight of every sensor in the weighted round-robin algorithm will give the values for a simple round-robin iteration. The various information collected is displayed in Table 3.1 below.

Table 3.1: Final Data Results with Varying Weights

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Total
Weight	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-
WRR RMSE	37.958	48.737	25.271	9.933	24.029	23.744	24.61	17.505	5.069	9.488	12.225	9.097	5.805	4.411	257.883
Smart RMSE	8.769	6.5	15.955	2.557	12.362	4.981	9.725	30.095	6.935	12.817	27.549	6.728	3.615	2.242	150.83
Weight	1	6	4	8	3	2	6	1	4	6	3	5	5	1	-
WRR RMSE	7.92	88.451	25.271	16.044	19.298	12.833	30.203	13.796	5.069	13.341	11.136	10.563	8.784	0	262.708
Smart RMSE	8.393	6.498	16.211	4.274	12.12	4.981	10.54	26.98	6.935	13.248	27.549	6.111	3.623	0	147.462
Weight	3	2	5	2	1	3	3	5	2	4	1	2	3	4	-
WRR RMSE	42.058	29.871	75.265	7.861	8.229	23.744	24.61	22.83	4.316	12.268	7.437	6.379	6.939	6.142	277.95
Smart RMSE	8.769	6.498	16.365	2.478	10.537	4.983	10.214	34.058	6.819	13.248	21.375	5.453	3.473	5.526	149.794
Weight	2	4	2	1	7	1	4	5	3	6	1	6	2	3	-
WRR RMSE	21.353	63.283	12.653	3.249	36.015	5.843	25.565	19.743	4.499	15.805	7.437	13.2	3.133	4.219	235.996
Smart RMSE	8.748	6.401	15.955	2.26	13.259	4.677	10.214	34.058	6.935	13.287	22.631	6.823	3.206	1.099	149.553
Weight	6	3	3	5	2	2	1	3	4	1	2	4	1	2	-
WRR RMSE	90.045	56.005	29.443	14.159	19.298	18.668	12.515	17.67	6.739	6.346	11.02	11.486	2.174	3.006	298.573
Smart RMSE	8.77	6.498	16.211	4.077	12.12	4.981	9.167	31.966	6.935	12.065	27.133	5.453	3.206	1.099	149.681

The format of the table has sensors A – N (all fourteen sensors) in the column labels at the top, and in groups of three rows, it shows the weights chosen for that specific iteration along with the root-mean-square-error values from the weighted round-robin (WRR RMSE) and the smart weighted round-robin (Smart RMSE). On the rightmost column, the total RMSE for each was also included.

3.2.1 Round-Robin Algorithm

The only relevant data from Table 3.1 for the basic round-robin algorithm is the first iteration, where all the weights are set to one, meaning the priority is spread across each sensor equally, which results in going around one-by-one in the round-robin manner. The total error that was calculated in this algorithm run is 257.883 from adding each sensor's error together; without a base reference, this feels meaningless. However, comparing to the other (randomly chosen) weight values, 257.883 is reasonably within the bounds of error from the other weighted round-robin total errors. It is not the highest or lowest error value, which makes sense because the varying weights are going to impact the errors for each sensor differently which can have a positive or negative impact on the value. This is true for the total values, but also for each of the sensors. For the remaining algorithms, the analysis can be done with the individual sensor error values and the total error from the basic round-robin as a base value.

3.2.2 Weighted Round-Robin Algorithm

Changing the weights with randomly chosen values resulted in four iterations of data as shown in Table 3.1 above (excluding the first iteration, representing basic round-robin). Keeping the values from the round-robin algorithm run as base values to compare to, some changes in individual sensor weights add to the RMSE value, while others decrease it. The variations all remain within reasonable bounds from each other, such as the fact that all the total RMSE values are in the two-hundreds. Because the data set for each iteration is the same, with some careful analysis, the results from the weighted round-robin algorithm can be used and further iterations can be ran to discover ideal values for individual weights leading towards the much lower error rates. Fortunately, this is unnecessary because the next algorithm does this on its own without any need for user calculations.

3.2.3 Smart Weighted Round-Robin Algorithm

The smart weighted round-robin algorithm makes drastic improvements on the RMSE values for every single iteration tested. When comparing results to the base value from basic round-robin in Table 3.1, the smart algorithm yields a total RMSE of 150.830, compared to the original RMSE in round-robin of 257.883, a 41.5% reduction of error. The trend holds out for the various weights attempted; the second through fifth iterations yield reductions of 43.9%, 46.1%, 36.6%, and 50.0% respectively. These improvements are massive and prove that the smart weighted round-robin algorithm consistently outperforms the basic round-robin algorithm and weighted round-robin algorithm drastically.

For most of the individual sensor data points, the trend also holds, although there is some variation where the weighted round-robin outperforms specific sensors. This is primarily due to two reasons: the first is that there is an inherent bias towards the first few sensors in the order, and the second is that the actual data is fluctuating in different manners. The bias exists because of the manner of scheduling in weighted round-robin; even if the weights are distributed equally, the first sensors still transmit their data first. When the updates to the weights are implemented and the new schedule is created for the remaining data, if any two weights are equal, the first sensor in the original order is prioritized. This bias has been mostly reduced with the update to the next-value prediction model, but it is still present at a small scale and likely will be for all variations of real-time weighted round-robin scheduling. The second reason for specific outperformances by the weighted round-robin algorithm is that the real-time data from the public repository is fluctuating and inconsistent across all the sensors, so some predictions for the next-values will be more inaccurate than the mean of all data points when the data changes from increasing to decreasing or vice versa. This is inherently a property of the data used, and the

prediction would be more accurate over longer periods of time (meaning more data points in a transmission would reduce the chance of this occurring). Regardless, the total RMSE values are significantly lower than the other algorithms, so these edge cases are largely unimportant to the grand scheme of the communication.

4. CONCLUSION

4.1 Effectiveness of SDN-SDR Network

The combination of software-defined radios and software-defined networks into an experimental architecture such as done with this project is a task that is profound and fascinating in its development. Each technology serves to displace its predecessor by moving components into software, allowing greater flexibility with the construction of this network than previously. The radios themselves allow communication between the base station and the field sensors to take place over-the-air, and the elements of their communication controlled in software (frequency, sample size) can be changed in real-time. The software nature of the network itself allows code to be developed further for the virtual controller to make specific changes for real-world applications, such as a certain method of reading a set of sensors or changes to who can access the network.

Additionally, because everything other than the actual B210s takes place in software, once the radios are set up in permanent place as meant to be for the project, changes to the entire architecture can be made remotely. This is perhaps the most key benefit of the use of these newer technologies in the network construction – the ability to change settings remotely directly addresses the issues with traditional networking discussed in the introduction and increases the ease of use for the technology. For example, one can imagine a scenario where out of the thousands of sensors reading temperature, a critical sensor starts acting strange. With the SDR-SDN full setup, the sensor can be isolated, analyzed, fixed, and reincluded without a single shutdown of the entire network. The benefits of this are huge.

However, one critical aspect that should be discussed is the practicality of converting modern networks into the software domain using SDN and SDR technology. It is easy to imagine one or the other being used for its respective widening of the range of applications but implementing both at the same time is unlikely for a company or project to do, and there are potential benefits from that decision. The learning curve for creating and adapting this network, along with integrating the SDN and SDRs together on each host, is a huge cost to the construction of an SDR-SDN network that many projects will not be willing to undertake. The components of a traditional network, by comparison, are well-studied, well-understood, and widely applied so that there are infinite resources related to the construction of the network and network engineers that are trained to create them. With both software-defined networks and software-defined radios, each technology is documented decently well, but the user base is not large enough to have a comparable number of resources to use. When the technologies are combined, there are far fewer than there are independently.

Once the large costs for acquiring the equipment and understanding it to the point of development are undergone, however, the radios and network architecture as a whole are quite useful and applicable in the world. For projects that need specific flexibility and special attention, the SDN-SDR architecture can address those needs in a highly dynamic manner.

4.2 Real-World Applications

The real-world applications of SDNs, SDRs, and SDN-SDR systems are interesting to consider. As stated previously, the ability to control elements using software has been critical in the development of all sorts of modern technology – smart phones, cameras, laptops, and cars, to name a few examples. The changes to network and radio technologies are not unprecedented,

and the expectation is that the popularity of the other devices that use a combination of hardware and software elements will be repeated with these tools.

Software-defined networks, however, have had somewhat limited success in finding applications (Saleem 2016). For much of the world, the costs for implementing SDNs is much higher than the costs of operating a traditional network. Also, the surrounding network infrastructure has for a long time been underdeveloped to serve the purposes of SDNs, although the ground for this is changing as the demand for more network flexibility goes up. Despite the excitement for the technology, the rate of growth has not met expectations, and the SDN has not yet proven itself as a permanent upgrade to networking.

The software-defined radios have had more success in market implementation. The SDRs are applicable almost everywhere where radios were previously used, such as communication, measuring and monitoring data, and streaming information. The immediate bonus of using an SDR as opposed to a traditional radio is the ability to have multiple options for configuration for different scenarios. For testing equipment or analyzing data from sensors, the flexibility involved in being able to switch between predefined, custom settings allows much less radio technology to be used as many of their individual tasks have been taken instead by the SDR. For many industries and companies, particularly those involved with networks, SDRs are gaining popularity and have an optimistic future.

The real-world applications for SDN-SDR combinations are limited at the moment. For the industries that want to create networks with the highest flexibility, especially regarding sensor reading, the technology can be very useful. However, this has yet to occur beyond the research efforts behind the technology. One could imagine, however, that companies that are reading lots of sensor data in real-time would receive lots of benefits by investing in SDN-SDR

combinations for their data collection, as done in this research project. The modular nature of the project allows any number of sensors to be added, and each sensor can be remotely modified at any point in time. An industry such as weather prediction, with conditions that change rapidly and the requirement of a lot of sensor data, would benefit from using the SDN-SDR technology.

For now, the devices used in this research project are primarily at the research stage of their development, but the applications to real-world projects are promising.

REFERENCES

[1] "The Basics of SDN and the OpenFlow Network Architecture." NoviFlow, NoviFlow, 6 May 2021, noviflow.com/the-basics-of-sdn-and-the-openflow-network-architecture/.

[2] "What Is SDN?" *Juniper Networks*, Juniper Networks, https://www.juniper.net/us/en/research-topics/what-is-sdn.html.

[3] Akeela, Rami, and Behnam Dezfouli. "Software-Defined Radios: Architecture, State-of-the-Art, and Challenges." *Computer Communications*, Elsevier, 30 July 2018, https://www.sciencedirect.com/science/article/abs/pii/S0140366418302937.

[4] Cooney, Michael. "What Is SDN and Where Software-Defined Networking Is Going." Network World, Network World, 16 April. 2019, www.networkworld.com/article/3209131/what-sdn-is-and-where-its-going.html.

[5] Hamill, Stephanie. "Introduction to Software Defined Radio." Wireless Innovation Forum, 2021, www.wirelessinnovation.org/Introduction_to_SDR.

[6] IBM Cloud Education. "The Fundamentals of Networking." *IBM*, IBM, 17 Mar. 2021, https://www.ibm.com/cloud/learn/networking-a-complete-guide#toc-important--ez_DoRa_.

[7] Jain, Raj. "OpenFlow, Software Defined Networking (SDN) and Network Function Virtualization." McKelvey School of Engineering. Tutorial at 2014 IEEE 15th International Conference on High Performance Switching and Routing, 2014, Vancouver, Canada.

[8] Jena, Satyabrata. "Difference between Software Defined Network and Traditional Network." *GeeksforGeeks*, 21 Aug. 2020, https://www.geeksforgeeks.org/difference-between-software-defined-network-and-traditional-network/.

[9] Lessing, Marlese, and Connor Craven. "What Is an SDN Controller? Definition." SDxCentral, SDxCentral Studios, 2020, www.sdxcentral.com/networking/sdn/definitions/what-is-sdn-controller/.

[10] Molenaar, Rene. "Introduction to SDN (Software Defined Networking)."

NetworkLessons.com, 19 Dec. 2019, networklessons.com/cisco/ccna-routing-switching-icnd2-200-105/introduction-to-sdn-software-defined-networking.

[11] Saleem, Maab. "Software Defined Network: Use Cases From the Real World" RouterFreak.com, 14 July. 2016, https://www.routerfreak.com/software-defined-network-use-cases-from-the-real-world/.

APPENDIX A: PYTHON SOFTWARE WRITTEN FOR COMMUNICATION

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
##################################################
# GNU Radio Python Flow Graph
# Title: Chat
# Generated: Thu Feb 10 09:30:28 2022
##################################################

if __name__ == '__main__':
    import ctypes
    import sys
    if sys.platform.startswith('linux'):
        try:
            x11 = ctypes.cdll.LoadLibrary('libX11.so')
            x11.XInitThreads()
        except:
            print "Warning: failed to XInitThreads()"

import ...


class chat(gr.top_block, Qt.QWidget):

    def __init__(self):
        gr.top_block.__init__(self, "Chat")
        Qt.QWidget.__init__(self)
        self.setWindowTitle("Chat")
        qtgui.util.check_set_qss()
        try:
            self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
        except:
            pass
        self.top_scroll_layout = Qt.QVBoxLayout()
        self.setLayout(self.top_scroll_layout)
        self.top_scroll = Qt.QScrollArea()
        self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
        self.top_scroll_layout.addWidget(self.top_scroll)
        self.top_scroll.setWidgetResizable(True)
        self.top_widget = Qt.QWidget()
        self.top_scroll.setWidget(self.top_widget)
        self.top_layout = Qt.QVBoxLayout(self.top_widget)
        self.top_grid_layout = Qt.QGridLayout()
        self.top_layout.addLayout(self.top_grid_layout)

        self.settings = Qt.QSettings("GNU Radio", "chat")
        self.restoreGeometry(self.settings.value("geometry").toByteArray())


        ##################################################
        # Variables
        ##################################################
        self.samp_rate = samp_rate = 400000

        while True:
            weight1 = raw_input("Enter a value for the weight of sensor 1 (integer): ")
            try:
                int(weight1)
                if (int(weight1) < 0):
                    print "Value cannot be negative. Try again."
                    continue
            except ValueError:
                print "Value provided needs to be an integer. Try again."
                continue
            break


        while True:
            weight2 = raw_input("Enter a value for the weight of sensor 2 (integer): ")
            try:
                int(weight2)
                if (int(weight2) < 0):
                    print "Value cannot be negative. Try again."
                    continue
            except ValueError:
```

(Image continued from previous page)

```
            print "Value provided needs to be an integer. Try again."
            continue
        break

    while True:
        weight3 = raw_input("Enter a value for the weight of sensor 3 (integer): ")
        try:
            int(weight3)
            if (int(weight3) < 0):
                print "Value cannot be negative. Try again."
                continue
        except ValueError:
            print "Value provided needs to be an integer. Try again."
            continue
        break

    while True:
        weight4 = raw_input("Enter a value for the weight of sensor 4 (integer): ")
        try:
            int(weight4)
            if (int(weight4) < 0):
                print "Value cannot be negative. Try again."
                continue
        except ValueError:
            print "Value provided needs to be an integer. Try again."
            continue
        break

    while True:
        weight5 = raw_input("Enter a value for the weight of sensor 5 (integer): ")
        try:
            int(weight5)
            if (int(weight5) < 0):
                print "Value cannot be negative. Try again."
                continue
        except ValueError:
            print "Value provided needs to be an integer. Try again."
            continue
        break

    while True:
        weight6 = raw_input("Enter a value for the weight of sensor 6 (integer): ")
        try:
            int(weight6)
            if (int(weight6) < 0):
                print "Value cannot be negative. Try again."
                continue
        except ValueError:
            print "Value provided needs to be an integer. Try again."
            continue
        break

    while True:
        weight7 = raw_input("Enter a value for the weight of sensor 7 (integer): ")
        try:
            int(weight7)
            if (int(weight7) < 0):
                print "Value cannot be negative. Try again."
                continue
        except ValueError:
            print "Value provided needs to be an integer. Try again."
            continue
        break

    while True:
        weight8 = raw_input("Enter a value for the weight of sensor 8 (integer): ")
        try:
            int(weight8)
            if (int(weight8) < 0):
                print "Value cannot be negative. Try again."
                continue
        except ValueError:
            print "Value provided needs to be an integer. Try again."
            continue
```

(Image continued from previous page)

```
            break

        while True:
            weight9 = raw_input("Enter a value for the weight of sensor 9 (integer): ")
            try:
                int(weight9)
                if (int(weight9) < 0):
                    print "Value cannot be negative. Try again."
                    continue
            except ValueError:
                print "Value provided needs to be an integer. Try again."
                continue
            break

        while True:
            weight10 = raw_input("Enter a value for the weight of sensor 10 (integer): ")
            try:
                int(weight10)
                if (int(weight10) < 0):
                    print "Value cannot be negative. Try again."
                    continue
            except ValueError:
                print "Value provided needs to be an integer. Try again."
                continue
            break

        while True:
            weight11 = raw_input("Enter a value for the weight of sensor 11 (integer): ")
            try:
                int(weight11)
                if (int(weight11) < 0):
                    print "Value cannot be negative. Try again."
                    continue
            except ValueError:
                print "Value provided needs to be an integer. Try again."
                continue
            break

        while True:
            weight12 = raw_input("Enter a value for the weight of sensor 12 (integer): ")
            try:
                int(weight12)
                if (int(weight12) < 0):
                    print "Value cannot be negative. Try again."
                    continue
            except ValueError:
                print "Value provided needs to be an integer. Try again."
                continue
            break

        while True:
            weight13 = raw_input("Enter a value for the weight of sensor 13 (integer): ")
            try:
                int(weight13)
                if (int(weight13) < 0):
                    print "Value cannot be negative. Try again."
                    continue
            except ValueError:
                print "Value provided needs to be an integer. Try again."
                continue
            break

        while True:
            weight14 = raw_input("Enter a value for the weight of sensor 14 (integer): ")
            try:
                int(weight14)
                if (int(weight14) < 0):
                    print "Value cannot be negative. Try again."
                    continue
            except ValueError:
                print "Value provided needs to be an integer. Try again."
                continue
            break
```

(Image continued from previous page)

```
        weight_file = open('/home/c/chat3', 'w')

        weight_file.write(weight1 + ',')
        weight_file.write(weight2 + ',')
        weight_file.write(weight3 + ',')
        weight_file.write(weight4 + ',')
        weight_file.write(weight5 + ',')
        weight_file.write(weight6 + ',')
        weight_file.write(weight7 + ',')
        weight_file.write(weight8 + ',')
        weight_file.write(weight9 + ',')
        weight_file.write(weight10 + ',')
        weight_file.write(weight11 + ',')
        weight_file.write(weight12 + ',')
        weight_file.write(weight13 + ',')
        weight_file.write(weight14 + ',\n')

        weight_file.close()

        ##################################################
        # Blocks
        ##################################################
        self.uhd_usrp_sink_1 = uhd.usrp_sink(
            ",".join(("", "")),
            uhd.stream_args(
                cpu_format="fc32",
                channels=range(1),
            ),
        )
        self.uhd_usrp_sink_1.set_samp_rate(samp_rate)
        self.uhd_usrp_sink_1.set_time_now(uhd.time_spec(time.time()), uhd.ALL_MBOARDS)
        self.uhd_usrp_sink_1.set_center_freq(1000000000, 0)
        self.uhd_usrp_sink_1.set_gain(40, 0)
        self.uhd_usrp_sink_1.set_antenna('RX2', 0)
        self.digital_gmsk_mod_0 = digital.gmsk_mod(
            samples_per_symbol=2,
            bt=0.35,
            verbose=False,
            log=False,
        )
        self.blocks_throttle_0 = blocks.throttle(gr.sizeof_char*1, samp_rate,True)
        self.blocks_multiply_const_vxx_1 = blocks.multiply_const_vcc((1, ))
        self.blocks_head_0 = blocks.head(gr.sizeof_char*1, 100000)
        self.blocks_file_source_0 = blocks.file_source(gr.sizeof_char*1, '/home/c/chat3', True)
        self.blocks_file_source_0.set_begin_tag(pmt.PMT_NIL)
        self.blks2_packet_encoder_0 = grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
                samples_per_symbol=2,
                bits_per_symbol=1,
                preamble='',
                access_code='',
                pad_for_usrp=True,
            ),
            payload_length=0,
        )



        ##################################################
        # Connections
        ##################################################
        self.connect((self.blks2_packet_encoder_0, 0), (self.digital_gmsk_mod_0, 0))
        self.connect((self.blocks_file_source_0, 0), (self.blocks_throttle_0, 0))
        self.connect((self.blocks_head_0, 0), (self.blks2_packet_encoder_0, 0))
        self.connect((self.blocks_multiply_const_vxx_1, 0), (self.uhd_usrp_sink_1, 0))
        self.connect((self.blocks_throttle_0, 0), (self.blocks_head_0, 0))
        self.connect((self.digital_gmsk_mod_0, 0), (self.blocks_multiply_const_vxx_1, 0))

    def closeEvent(self, event):
        self.settings = Qt.QSettings("GNU Radio", "chat")
        self.settings.setValue("geometry", self.saveGeometry())
        event.accept()

    def get_samp_rate(self):
        return self.samp_rate
```

(Image continued from previous page)

```
314
315     def set_samp_rate(self, samp_rate):
316         self.samp_rate = samp_rate
317         self.uhd_usrp_sink_1.set_samp_rate(self.samp_rate)
318         self.blocks_throttle_0.set_sample_rate(self.samp_rate)
319
320
321 class text_rx(gr.top_block, Qt.QWidget):
322
323     def __init__(self):
324         gr.top_block.__init__(self, "Text Rx")
325         Qt.QWidget.__init__(self)
326         self.setWindowTitle("Text Rx")
327         qtgui.util.check_set_qss()
328         try:
329             self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
330         except:
331             pass
332         self.top_scroll_layout = Qt.QVBoxLayout()
333         self.setLayout(self.top_scroll_layout)
334         self.top_scroll = Qt.QScrollArea()
335         self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
336         self.top_scroll_layout.addWidget(self.top_scroll)
337         self.top_scroll.setWidgetResizable(True)
338         self.top_widget = Qt.QWidget()
339         self.top_scroll.setWidget(self.top_widget)
340         self.top_layout = Qt.QVBoxLayout(self.top_widget)
341         self.top_grid_layout = Qt.QGridLayout()
342         self.top_layout.addLayout(self.top_grid_layout)
343
344         self.settings = Qt.QSettings("GNU Radio", "text_rx")
345         self.restoreGeometry(self.settings.value("geometry").toByteArray())
346
347
348         ##################################################
349         # Variables
350         ##################################################
351         self.samp_rate = samp_rate = 400000
352
353         ##################################################
354         # Blocks
355         ##################################################
356         self.uhd_usrp_source_0 = uhd.usrp_source(
357             ",".join(("", "")),
358             uhd.stream_args(
359                 cpu_format="fc32",
360                 channels=range(1),
361             ),
362         )
363         self.uhd_usrp_source_0.set_samp_rate(samp_rate)
364         self.uhd_usrp_source_0.set_center_freq(1000000000, 0)
365         self.uhd_usrp_source_0.set_gain(30, 0)
366         self.uhd_usrp_source_0.set_antenna('RX2', 0)
367         self.qtgui_freq_sink_x_0 = qtgui.freq_sink_c(
368             1024, #size
369             firdes.WIN_BLACKMAN_hARRIS, #wintype
370             0, #fc
371             samp_rate, #bw
372             "", #name
373             1 #number of inputs
374         )
375         self.qtgui_freq_sink_x_0.set_update_time(0.10)
376         self.qtgui_freq_sink_x_0.set_y_axis(-140, 10)
377         self.qtgui_freq_sink_x_0.set_y_label('Relative Gain', 'dB')
378         self.qtgui_freq_sink_x_0.set_trigger_mode(qtgui.TRIG_MODE_FREE, 0.0, 0, "")
379         self.qtgui_freq_sink_x_0.enable_autoscale(False)
380         self.qtgui_freq_sink_x_0.enable_grid(False)
381         self.qtgui_freq_sink_x_0.set_fft_average(1.0)
382         self.qtgui_freq_sink_x_0.enable_axis_labels(True)
383         self.qtgui_freq_sink_x_0.enable_control_panel(False)
384
385         if not True:
386             self.qtgui_freq_sink_x_0.disable_legend()
```

(Image continued from previous page)

```
        if "complex" == "float" or "complex" == "msg_float":
            self.qtgui_freq_sink_x_0.set_plot_pos_half(not True)

        labels = ['', '', '', '', '',
                  '', '', '', '', '']
        widths = [1, 1, 1, 1, 1,
                  1, 1, 1, 1, 1]
        colors = ["blue", "red", "green", "black", "cyan",
                  "magenta", "yellow", "dark red", "dark green", "dark blue"]
        alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
                  1.0, 1.0, 1.0, 1.0, 1.0]
        for i in xrange(1):
            if len(labels[i]) == 0:
                self.qtgui_freq_sink_x_0.set_line_label(i, "Data {0}".format(i))
            else:
                self.qtgui_freq_sink_x_0.set_line_label(i, labels[i])
            self.qtgui_freq_sink_x_0.set_line_width(i, widths[i])
            self.qtgui_freq_sink_x_0.set_line_color(i, colors[i])
            self.qtgui_freq_sink_x_0.set_line_alpha(i, alphas[i])

        self._qtgui_freq_sink_x_0_win = sip.wrapinstance(self.qtgui_freq_sink_x_0.pyqwidget(), Qt.QWidget)
        self.top_grid_layout.addWidget(self._qtgui_freq_sink_x_0_win)
        self.low_pass_filter_0 = filter.fir_filter_ccf(1, firdes.low_pass(
            1, samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))
        self.digital_gmsk_demod_0 = digital.gmsk_demod(
            samples_per_symbol=2,
            gain_mu=0.175,
            mu=0.5,
            omega_relative_limit=0.005,
            freq_error=0.0,
            verbose=False,
            log=False,
        )

        clr_file = open('/home/c/test_file2', 'w')
        clr_file.close()

        self.blocks_multiply_const_vxx_0 = blocks.multiply_const_vcc((1, ))
        self.blocks_file_sink_0 = blocks.file_sink(gr.sizeof_char*1, '/home/c/test_file2', False)
        self.blocks_file_sink_0.set_unbuffered(False)
        self.blks2_packet_decoder_0 = grc_blks2.packet_demod_b(grc_blks2.packet_decoder(
                access_code='',
                threshold=-1,
                callback=lambda ok, payload: self.blks2_packet_decoder_0.recv_pkt(ok, payload),
            ),
        )




        ##################################################
        # Connections
        ##################################################
        self.connect((self.blks2_packet_decoder_0, 0), (self.blocks_file_sink_0, 0))
        self.connect((self.blocks_multiply_const_vxx_0, 0), (self.low_pass_filter_0, 0))
        self.connect((self.digital_gmsk_demod_0, 0), (self.blks2_packet_decoder_0, 0))
        self.connect((self.low_pass_filter_0, 0), (self.digital_gmsk_demod_0, 0))
        self.connect((self.low_pass_filter_0, 0), (self.qtgui_freq_sink_x_0, 0))
        self.connect((self.uhd_usrp_source_0, 0), (self.blocks_multiply_const_vxx_0, 0))


    def closeEvent(self, event):
        self.settings = Qt.QSettings("GNU Radio", "text_rx")
        self.settings.setValue("geometry", self.saveGeometry())
        event.accept()

    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.uhd_usrp_source_0.set_samp_rate(self.samp_rate)
        self.qtgui_freq_sink_x_0.set_frequency_range(0, self.samp_rate)
```

(Image continued from previous page)

```
        self.low_pass_filter_0.set_taps(firdes.low_pass(1, self.samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))


class ack(gr.top_block, Qt.QWidget):

    def __init__(self):
        gr.top_block.__init__(self, "Chat")
        Qt.QWidget.__init__(self)
        self.setWindowTitle("Chat")
        qtgui.util.check_set_qss()
        try:
            self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
        except:
            pass
        self.top_scroll_layout = Qt.QVBoxLayout()
        self.setLayout(self.top_scroll_layout)
        self.top_scroll = Qt.QScrollArea()
        self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
        self.top_scroll_layout.addWidget(self.top_scroll)
        self.top_scroll.setWidgetResizable(True)
        self.top_widget = Qt.QWidget()
        self.top_scroll.setWidget(self.top_widget)
        self.top_layout = Qt.QVBoxLayout(self.top_widget)
        self.top_grid_layout = Qt.QGridLayout()
        self.top_layout.addLayout(self.top_grid_layout)

        self.settings = Qt.QSettings("GNU Radio", "chat")
        self.restoreGeometry(self.settings.value("geometry").toByteArray())


        ##################################################
        # Variables
        ##################################################
        self.samp_rate = samp_rate = 400000

        weight_file = open('/home/c/chat3', 'w')
        weight_file.write("ACK\n")
        weight_file.close()

        ##################################################
        # Blocks
        ##################################################
        self.uhd_usrp_sink_1 = uhd.usrp_sink(
            ",".join(("", "")),
            uhd.stream_args(
                cpu_format="fc32",
                channels=range(1),
            ),
        )
        self.uhd_usrp_sink_1.set_samp_rate(samp_rate)
        self.uhd_usrp_sink_1.set_time_now(uhd.time_spec(time.time()), uhd.ALL_MBOARDS)
        self.uhd_usrp_sink_1.set_center_freq(1000000000, 0)
        self.uhd_usrp_sink_1.set_gain(40, 0)
        self.uhd_usrp_sink_1.set_antenna('TX/RX', 0)
        self.digital_gmsk_mod_0 = digital.gmsk_mod(
            samples_per_symbol=2,
            bt=0.35,
            verbose=False,
            log=False,
        )
        self.blocks_throttle_0 = blocks.throttle(gr.sizeof_char*1, samp_rate,True)
        self.blocks_multiply_const_vxx_1 = blocks.multiply_const_vcc((1, ))
        self.blocks_head_0 = blocks.head(gr.sizeof_char*1, 100000)
        self.blocks_file_source_0 = blocks.file_source(gr.sizeof_char*1, '/home/c/chat3', True)
        self.blocks_file_source_0.set_begin_tag(pmt.PMT_NIL)
        self.blks2_packet_encoder_0 = grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
                samples_per_symbol=2,
                bits_per_symbol=1,
                preamble='',
                access_code='',
                pad_for_usrp=True,
            ),
            payload_length=0,
        )
```

(Image continued from previous page)

```
        ##################################################
        # Connections
        ##################################################
        self.connect((self.blks2_packet_encoder_0, 0), (self.digital_gmsk_mod_0, 0))
        self.connect((self.blocks_file_source_0, 0), (self.blocks_throttle_0, 0))
        self.connect((self.blocks_head_0, 0), (self.blks2_packet_encoder_0, 0))
        self.connect((self.blocks_multiply_const_vxx_1, 0), (self.uhd_usrp_sink_1, 0))
        self.connect((self.blocks_throttle_0, 0), (self.blocks_head_0, 0))
        self.connect((self.digital_gmsk_mod_0, 0), (self.blocks_multiply_const_vxx_1, 0))

    def closeEvent(self, event):
        self.settings = Qt.QSettings("GNU Radio", "chat")
        self.settings.setValue("geometry", self.saveGeometry())
        event.accept()

    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.uhd_usrp_sink_1.set_samp_rate(self.samp_rate)
        self.blocks_throttle_0.set_sample_rate(self.samp_rate)


def main(options=None):

    from distutils.version import StrictVersion
    if StrictVersion(Qt.qVersion()) >= StrictVersion("4.5.0"):
        style = gr.prefs().get_string('qtgui', 'style', 'raster')
        Qt.QApplication.setGraphicsSystem(style)
    qapp = Qt.QApplication(sys.argv)

    global weight1, weight2, weight3, weight4, weight5, weight6, weight7, weight8, weight9, weight10, weight11, weight12, weight13, weight14
    a_num = 0
    b_num = 0
    c_num = 0
    d_num = 0
    e_num = 0
    f_num = 0
    g_num = 0
    h_num = 0
    i_num = 0
    j_num = 0
    k_num = 0
    l_num = 0
    m_num = 0
    n_num = 0
    sum_a = 0
    sum_b = 0
    sum_c = 0
    sum_d = 0
    sum_e = 0
    sum_f = 0
    sum_g = 0
    sum_h = 0
    sum_i = 0
    sum_j = 0
    sum_k = 0
    sum_l = 0
    sum_m = 0
    sum_n = 0
    predict_a = 0
    predict_b = 0
    predict_c = 0
    predict_d = 0
    predict_e = 0
    predict_f = 0
    predict_g = 0
    predict_h = 0
    predict_i = 0
    predict_j = 0
```

(Image continued from previous page)

```
    predict_k = 0
    predict_l = 0
    predict_m = 0
    predict_n = 0
    actual_a_list = []
    actual_b_list = []
    actual_c_list = []
    actual_d_list = []
    actual_e_list = []
    actual_f_list = []
    actual_g_list = []
    actual_h_list = []
    actual_i_list = []
    actual_j_list = []
    actual_k_list = []
    actual_l_list = []
    actual_m_list = []
    actual_n_list = []
    predict_a_list = []
    predict_b_list = []
    predict_c_list = []
    predict_d_list = []
    predict_e_list = []
    predict_f_list = []
    predict_g_list = []
    predict_h_list = []
    predict_i_list = []
    predict_j_list = []
    predict_k_list = []
    predict_l_list = []
    predict_m_list = []
    predict_n_list = []

    tb = chat()
    tb.start()
    time.sleep(1)
    tb.stop()

    final = open('/home/c/final', 'w')
    final.close()

    while True:
        tb = text_rx()
        tb.start()

        data1 = open('/home/c/test_file2', 'r+')
        final = open('/home/c/final', 'a')

        while True:
            data1.flush()
            data1.readline()
            file_content2 = data1.readline()
            if (file_content2 == "eof\n"):
                print(file_content2)
                break
            elif (file_content2 != ""):
                final.write(file_content2)
                split = file_content2.split(' ')
                if (split[1].startswith('10') and (split[1].startswith('0', 2) == False)):
                    a_num += 1
                    actual_a_list.append(float(split[0]))
                    sum_a += float(split[0])
                    predict_a = sum_a/a_num
                    print("Prediction for Next A: "+str(predict_a))
                    predict_a_list.append(predict_a)
                elif (split[1].startswith('20')):
                    b_num += 1
                    actual_b_list.append(float(split[0]))
                    sum_b += float(split[0])
                    predict_b = sum_b/b_num
                    print("Prediction for Next B: "+str(predict_b))
                    predict_b_list.append(predict_b)
                elif (split[1].startswith('30')):
                    c_num += 1
```

(Image continued from previous page)

```
                actual_c_list.append(float(split[0]))
                sum_c += float(split[0])
                predict_c = sum_c/c_num
                print("Prediction for Next C: "+str(predict_c))
                predict_c_list.append(predict_c)
            elif (split[1].startswith('40')):
                d_num += 1
                actual_d_list.append(float(split[0]))
                sum_d += float(split[0])
                predict_d = sum_d/d_num
                print("Prediction for Next D: "+str(predict_d))
                predict_d_list.append(predict_d)
            elif (split[1].startswith('50')):
                e_num += 1
                actual_e_list.append(float(split[0]))
                sum_e += float(split[0])
                predict_e = sum_e/e_num
                print("Prediction for Next E: "+str(predict_e))
                predict_e_list.append(predict_e)
            elif (split[1].startswith('60')):
                f_num += 1
                actual_f_list.append(float(split[0]))
                sum_f += float(split[0])
                predict_f = sum_f/f_num
                print("Prediction for Next F: "+str(predict_f))
                predict_f_list.append(predict_f)
            elif (split[1].startswith('70')):
                g_num += 1
                actual_g_list.append(float(split[0]))
                sum_g += float(split[0])
                predict_g = sum_g/g_num
                print("Prediction for Next G: "+str(predict_g))
                predict_g_list.append(predict_g)
            elif (split[1].startswith('80')):
                h_num += 1
                actual_h_list.append(float(split[0]))
                sum_h += float(split[0])
                predict_h = sum_h/h_num
                print("Prediction for Next H: "+str(predict_h))
                predict_h_list.append(predict_h)
            elif (split[1].startswith('90')):
                i_num += 1
                actual_i_list.append(float(split[0]))
                sum_i += float(split[0])
                predict_i = sum_i/i_num
                print("Prediction for Next I: "+str(predict_i))
                predict_i_list.append(predict_i)
            elif (split[1].startswith('100')):
                j_num += 1
                actual_j_list.append(float(split[0]))
                sum_j += float(split[0])
                predict_j = sum_j/j_num
                print("Prediction for Next J: "+str(predict_j))
                predict_j_list.append(predict_j)
            elif (split[1].startswith('110')):
                k_num += 1
                actual_k_list.append(float(split[0]))
                sum_k += float(split[0])
                predict_k = sum_k/k_num
                print("Prediction for Next K: "+str(predict_k))
                predict_k_list.append(predict_k)
            elif (split[1].startswith('120')):
                l_num += 1
                actual_l_list.append(float(split[0]))
                sum_l += float(split[0])
                predict_l = sum_l/l_num
                print("Prediction for Next L: "+str(predict_l))
                predict_l_list.append(predict_l)
            elif (split[1].startswith('130')):
                m_num += 1
                actual_m_list.append(float(split[0]))
                sum_m += float(split[0])
                predict_m = sum_m/m_num
                print("Prediction for Next M: "+str(predict_m))
```

(Image continued from previous page)

```
                predict_m_list.append(predict_m)
            elif (split[1].startswith('140')):
                n_num += 1
                actual_n_list.append(float(split[0]))
                sum_n += float(split[0])
                predict_n = sum_n/n_num
                print("Prediction for Next N: "+str(predict_n))
                predict_n_list.append(predict_n)
            print("Current Value: "+file_content2)
            time.sleep(1)
            sb = ack()
            sb.start()
            print "SENDING ACK"
            time.sleep(1)
            sb.stop()
            break
        time.sleep(1)

    tb.stop()

    if (file_content2 == "eof\n"):
        data1.close()
        final.close()
        break

    data1.close()
    final.close()

tb.stop()

sum_sq_a = 0
for z in range(0, len(predict_a_list)-1):
    square = (actual_a_list[z+1] - predict_a_list[z])**2
    sum_sq_a += square
msqrterr_a = math.sqrt(sum_sq_a)
print("Root Mean Squared Error for Sensor 1: "+str(msqrterr_a))

sum_sq_b = 0
for z in range(0, len(predict_b_list)-1):
    square = (actual_b_list[z+1] - predict_b_list[z])**2
    sum_sq_b += square
msqrterr_b = math.sqrt(sum_sq_b)
print("Root Mean Squared Error for Sensor 2: "+str(msqrterr_b))

sum_sq_c = 0
for z in range(0, len(predict_c_list)-1):
    square = (actual_c_list[z+1] - predict_c_list[z])**2
    sum_sq_c += square
msqrterr_c = math.sqrt(sum_sq_c)
print("Root Mean Squared Error for Sensor 3: "+str(msqrterr_c))

sum_sq_d = 0
for z in range(0, len(predict_d_list)-1):
    square = (actual_d_list[z+1] - predict_d_list[z])**2
    sum_sq_d += square
msqrterr_d = math.sqrt(sum_sq_d)
print("Root Mean Squared Error for Sensor 4: "+str(msqrterr_d))

sum_sq_e = 0
for z in range(0, len(predict_e_list)-1):
    square = (actual_e_list[z+1] - predict_e_list[z])**2
    sum_sq_e += square
msqrterr_e = math.sqrt(sum_sq_e)
print("Root Mean Squared Error for Sensor 5: "+str(msqrterr_e))

sum_sq_f = 0
for z in range(0, len(predict_f_list)-1):
    square = (actual_f_list[z+1] - predict_f_list[z])**2
    sum_sq_f += square
msqrterr_f = math.sqrt(sum_sq_f)
print("Root Mean Squared Error for Sensor 6: "+str(msqrterr_f))

sum_sq_g = 0
for z in range(0, len(predict_g_list)-1):
```

(Image continued from previous page)

```
        square = (actual_g_list[z+1] - predict_g_list[z])**2
        sum_sq_g += square
    msqrterr_g = math.sqrt(sum_sq_g)
    print("Root Mean Squared Error for Sensor 7: "+str(msqrterr_g))

    sum_sq_h = 0
    for z in range(0, len(predict_h_list)-1):
        square = (actual_h_list[z+1] - predict_h_list[z])**2
        sum_sq_h += square
    msqrterr_h = math.sqrt(sum_sq_h)
    print("Root Mean Squared Error for Sensor 8: "+str(msqrterr_h))

    sum_sq_i = 0
    for z in range(0, len(predict_i_list)-1):
        square = (actual_i_list[z+1] - predict_i_list[z])**2
        sum_sq_i += square
    msqrterr_i = math.sqrt(sum_sq_i)
    print("Root Mean Squared Error for Sensor 9: "+str(msqrterr_i))

    sum_sq_j = 0
    for z in range(0, len(predict_j_list)-1):
        square = (actual_j_list[z+1] - predict_j_list[z])**2
        sum_sq_j += square
    msqrterr_j = math.sqrt(sum_sq_j)
    print("Root Mean Squared Error for Sensor 10: "+str(msqrterr_j))

    sum_sq_k = 0
    for z in range(0, len(predict_k_list)-1):
        square = (actual_k_list[z+1] - predict_k_list[z])**2
        sum_sq_k += square
    msqrterr_k = math.sqrt(sum_sq_k)
    print("Root Mean Squared Error for Sensor 11: "+str(msqrterr_k))

    sum_sq_l = 0
    for z in range(0, len(predict_l_list)-1):
        square = (actual_l_list[z+1] - predict_l_list[z])**2
        sum_sq_l += square
    msqrterr_l = math.sqrt(sum_sq_l)
    print("Root Mean Squared Error for Sensor 12: "+str(msqrterr_l))

    sum_sq_m = 0
    for z in range(0, len(predict_m_list)-1):
        square = (actual_m_list[z+1] - predict_m_list[z])**2
        sum_sq_m += square
    msqrterr_m = math.sqrt(sum_sq_m)
    print("Root Mean Squared Error for Sensor 13: "+str(msqrterr_m))

    sum_sq_n = 0
    for z in range(0, len(predict_n_list)-1):
        square = (actual_n_list[z+1] - predict_n_list[z])**2
        sum_sq_n += square
    msqrterr_n = math.sqrt(sum_sq_n)
    print("Root Mean Squared Error for Sensor 14: "+str(msqrterr_n))

    rmse_tot = msqrterr_a+msqrterr_b+msqrterr_c+msqrterr_d+msqrterr_e+msqrterr_f+msqrterr_g+msqrterr_h+msqrterr_i+msqrterr_j+msqrterr_k+msqrterr_
    print("\nTotal Root Mean Squared Error: "+str(rmse_tot))

    exit()

    def quitting():
        tb.stop()
        tb.wait()
    qapp.connect(qapp, Qt.SIGNAL("aboutToQuit()"), quitting)
    qapp.exec_()


if __name__ == '__main__':
    main()
```

Figure A.1: Weighted Round-Robin Algorithm for Base Station Radio

```python
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
##################################################
# GNU Radio Python Flow Graph
# Title: Chat2
# Generated: Thu Feb 10 09:52:14 2022
##################################################

if __name__ == '__main__':
    import ctypes
    import sys
    if sys.platform.startswith('linux'):
        try:
            x11 = ctypes.cdll.LoadLibrary('libX11.so')
            x11.XInitThreads()
        except:
            print "Warning: failed to XInitThreads()"

import ....


class chat2(gr.top_block, Qt.QWidget):

    def __init__(self):
        gr.top_block.__init__(self, "Chat2")
        Qt.QWidget.__init__(self)
        self.setWindowTitle("Chat2")
        qtgui.util.check_set_qss()
        try:
            self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
        except:
            pass
        self.top_scroll_layout = Qt.QVBoxLayout()
        self.setLayout(self.top_scroll_layout)
        self.top_scroll = Qt.QScrollArea()
        self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
        self.top_scroll_layout.addWidget(self.top_scroll)
        self.top_scroll.setWidgetResizable(True)
        self.top_widget = Qt.QWidget()
        self.top_scroll.setWidget(self.top_widget)
        self.top_layout = Qt.QVBoxLayout(self.top_widget)
        self.top_grid_layout = Qt.QGridLayout()
        self.top_layout.addLayout(self.top_grid_layout)

        self.settings = Qt.QSettings("GNU Radio", "chat2")
        self.restoreGeometry(self.settings.value("geometry").toByteArray())


        ##################################################
        # Variables
        ##################################################
        self.samp_rate = samp_rate = 400000

        ##################################################
        # Blocks
        ##################################################
        self.uhd_usrp_source_0 = uhd.usrp_source(
            ",".join(("", "")),
            uhd.stream_args(
                cpu_format="fc32",
                channels=range(1),
            ),
        )
        self.uhd_usrp_source_0.set_samp_rate(samp_rate)
        self.uhd_usrp_source_0.set_center_freq(1000000000, 0)
        self.uhd_usrp_source_0.set_gain(30, 0)
        self.uhd_usrp_source_0.set_antenna('TX/RX', 0)
        self.qtgui_freq_sink_x_0 = qtgui.freq_sink_c(
            1024, #size
            firdes.WIN_BLACKMAN_hARRIS, #wintype
            0, #fc
            samp_rate, #bw
            "", #name
            1 #number of inputs
```

(Image continued from previous page)

```
        )
        self.qtgui_freq_sink_x_0.set_update_time(0.10)
        self.qtgui_freq_sink_x_0.set_y_axis(-140, 10)
        self.qtgui_freq_sink_x_0.set_y_label('Relative Gain', 'dB')
        self.qtgui_freq_sink_x_0.set_trigger_mode(qtgui.TRIG_MODE_FREE, 0.0, 0, "")
        self.qtgui_freq_sink_x_0.enable_autoscale(False)
        self.qtgui_freq_sink_x_0.enable_grid(False)
        self.qtgui_freq_sink_x_0.set_fft_average(1.0)
        self.qtgui_freq_sink_x_0.enable_axis_labels(True)
        self.qtgui_freq_sink_x_0.enable_control_panel(False)

        if not True:
          self.qtgui_freq_sink_x_0.disable_legend()

        if "complex" == "float" or "complex" == "msg_float":
          self.qtgui_freq_sink_x_0.set_plot_pos_half(not True)

        labels = ['', '', '', '', '',
                  '', '', '', '', '']
        widths = [1, 1, 1, 1, 1,
                  1, 1, 1, 1, 1]
        colors = ["blue", "red", "green", "black", "cyan",
                  "magenta", "yellow", "dark red", "dark green", "dark blue"]
        alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
                  1.0, 1.0, 1.0, 1.0, 1.0]
        for i in xrange(1):
            if len(labels[i]) == 0:
                self.qtgui_freq_sink_x_0.set_line_label(i, "Data {0}".format(i))
            else:
                self.qtgui_freq_sink_x_0.set_line_label(i, labels[i])
            self.qtgui_freq_sink_x_0.set_line_width(i, widths[i])
            self.qtgui_freq_sink_x_0.set_line_color(i, colors[i])
            self.qtgui_freq_sink_x_0.set_line_alpha(i, alphas[i])

        self._qtgui_freq_sink_x_0_win = sip.wrapinstance(self.qtgui_freq_sink_x_0.pyqwidget(), Qt.QWidget)
        self.top_grid_layout.addWidget(self._qtgui_freq_sink_x_0_win)
        self.low_pass_filter_0 = filter.fir_filter_ccf(1, firdes.low_pass(
            1, samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))
        self.digital_gmsk_demod_1 = digital.gmsk_demod(
            samples_per_symbol=2,
            gain_mu=0.175,
            mu=0.5,
            omega_relative_limit=0.005,
            freq_error=0.0,
            verbose=False,
            log=False,
        )
        self.blocks_multiply_const_vxx_1 = blocks.multiply_const_vcc((1, ))
        self.blocks_file_sink_1 = blocks.file_sink(gr.sizeof_char*1, '/home/b/chat_test1.txt', False)
        self.blocks_file_sink_1.set_unbuffered(False)
        self.blks2_packet_decoder_1 = grc_blks2.packet_demod_b(grc_blks2.packet_decoder(
                access_code='',
                threshold=-1,
                callback=lambda ok, payload: self.blks2_packet_decoder_1.recv_pkt(ok, payload),
            ),
        )



        ##################################################
        # Connections
        ##################################################
        self.connect((self.blks2_packet_decoder_1, 0), (self.blocks_file_sink_1, 0))
        self.connect((self.blocks_multiply_const_vxx_1, 0), (self.low_pass_filter_0, 0))
        self.connect((self.digital_gmsk_demod_1, 0), (self.blks2_packet_decoder_1, 0))
        self.connect((self.low_pass_filter_0, 0), (self.digital_gmsk_demod_1, 0))
        self.connect((self.low_pass_filter_0, 0), (self.qtgui_freq_sink_x_0, 0))
        self.connect((self.uhd_usrp_source_0, 0), (self.blocks_multiply_const_vxx_1, 0))

    def closeEvent(self, event):
        self.settings = Qt.QSettings("GNU Radio", "chat2")
        self.settings.setValue("geometry", self.saveGeometry())
        event.accept()

```

(Image continued from previous page)

```
    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.uhd_usrp_source_0.set_samp_rate(self.samp_rate)
        self.qtgui_freq_sink_x_0.set_frequency_range(0, self.samp_rate)
        self.low_pass_filter_0.set_taps(firdes.low_pass(1, self.samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))


class text_tx(gr.top_block):

    def __init__(self):
        gr.top_block.__init__(self, "Text Tx")

        ##################################################
        # Variables
        ##################################################
        self.samp_rate = samp_rate = 400000
        source = ''

        ##################################################
        # Blocks
        ##################################################
        self.uhd_usrp_sink_0 = uhd.usrp_sink(
            ",".join(("", "")),
            uhd.stream_args(
                cpu_format="fc32",
                channels=range(1),
            ),
        )
        self.uhd_usrp_sink_0.set_samp_rate(samp_rate)
        self.uhd_usrp_sink_0.set_time_now(uhd.time_spec(time.time()), uhd.ALL_MBOARDS)
        self.uhd_usrp_sink_0.set_center_freq(1000000000, 0)
        self.uhd_usrp_sink_0.set_gain(40, 0)
        self.uhd_usrp_sink_0.set_antenna('TX/RX', 0)
        self.digital_gmsk_mod_0 = digital.gmsk_mod(
            samples_per_symbol=2,
            bt=0.35,
            verbose=False,
            log=False,
        )
        self.blocks_throttle_0 = blocks.throttle(gr.sizeof_char*1, samp_rate,True)
        self.blocks_multiply_const_vxx_0 = blocks.multiply_const_vcc((1, ))
        self.blocks_head_0 = blocks.head(gr.sizeof_char*1, samp_rate/4)

        send = open('/home/b/tx.txt', 'w')
        if switch == 'a':
            send.write(data0[a1])
            print(data0[a1])
        elif switch == 'b':
            send.write(data1[b1])
            print(data1[b1])
        elif switch == 'c':
            send.write(data2[c1])
            print(data2[c1])
        elif switch == 'd':
            send.write(data3[d1])
            print(data3[d1])
        elif switch == 'e':
            send.write(data4[e1])
            print(data4[e1])
        elif switch == 'f':
            send.write(data5[f1])
            print(data5[f1])
        elif switch == 'g':
            send.write(data6[g1])
            print(data6[g1])
        elif switch == 'h':
            send.write(data7[h1])
            print(data7[h1])
        elif switch == 'i':
            send.write(data8[i1])
            print(data8[i1])
```

(Image continued from previous page)

```
        elif switch == 'j':
            send.write(data9[j1])
            print(data9[j1])
        elif switch == 'k':
            send.write(data10[k1])
            print(data10[k1])
        elif switch == 'l':
            send.write(data11[l1])
            print(data11[l1])
        elif switch == 'm':
            send.write(data12[m1])
            print(data12[m1])
        elif switch == 'n':
            send.write(data13[n1])
            print(data13[n1])
        elif switch == 'eof':
            send.write("eof\n")
            print "eof"
        send.close()

        self.blocks_file_source_0 = blocks.file_source(gr.sizeof_char*1, '/home/b/tx.txt', True)

        self.blocks_file_source_0.set_begin_tag(pmt.PMT_NIL)
        self.blks2_packet_encoder_0 = grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
                samples_per_symbol=2,
                bits_per_symbol=1,
                preamble='',
                access_code='',
                pad_for_usrp=True,
            ),
            payload_length=0,
        )



        ##################################################
        # Connections
        ##################################################
        self.connect((self.blks2_packet_encoder_0, 0), (self.digital_gmsk_mod_0, 0))
        self.connect((self.blocks_file_source_0, 0), (self.blocks_throttle_0, 0))
        self.connect((self.blocks_head_0, 0), (self.blks2_packet_encoder_0, 0))
        self.connect((self.blocks_multiply_const_vxx_0, 0), (self.uhd_usrp_sink_0, 0))
        self.connect((self.blocks_throttle_0, 0), (self.blocks_head_0, 0))
        self.connect((self.digital_gmsk_mod_0, 0), (self.blocks_multiply_const_vxx_0, 0))

    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.uhd_usrp_sink_0.set_samp_rate(self.samp_rate)
        self.blocks_throttle_0.set_sample_rate(self.samp_rate)
        self.blocks_head_0.set_length(self.samp_rate/4)


class WRRScheduler():

    cw = 0
    i = -1
    data_set = []
    max_s = None
    gcd_s = None
    len_s = None
    counter = {}

    def __init__(self, s = None):
        self._init_dataset(s)

    def _init_dataset(self, s):
        self.data_set = s
        self.max_s = max(s, key=lambda x: x[1])[1]
        self.gcd_s = reduce(fractions.gcd, [ weight for data, weight in s])
        self.len_s = len(s)

```

(Image continued from previous page)

```
    def schedule(self):
        while True:
            self.i = (self.i + 1) % self.len_s
            if self.i == 0:
                self.cw = self.cw - self.gcd_s
                if self.cw <= 0:
                    self.cw = self.max_s
                    if self.cw == 0:
                        return None
            if self.data_set[self.i][1] >= self.cw:
                self._inc_counter(self.data_set[self.i])
                return self.data_set[self.i]

    def _inc_counter(self, item):
        try:
            self.counter[item[0]] += 1
        except KeyError:
            self.counter[item[0]] = 1

    def set_data(self, s):
        self.reset()
        self._init_dataset(s)

    def reset_counter(self):
        self.counter = {}

    def reset(self):
        self.cw = 0
        self.i = -1
        self.data_set = []
        self.max_s = None
        self.gcd_s = None
        self.len_s = None
        self.reset_counter()

    def get_next(self, n = 1):
        if n > 1:
            return [ self.schedule() for i in range(0,n) ]
        return self.schedule()


class ack2(gr.top_block, Qt.QWidget):

    def __init__(self):
        gr.top_block.__init__(self, "Chat2")
        Qt.QWidget.__init__(self)
        self.setWindowTitle("Chat2")
        qtgui.util.check_set_qss()
        try:
            self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
        except:
            pass
        self.top_scroll_layout = Qt.QVBoxLayout()
        self.setLayout(self.top_scroll_layout)
        self.top_scroll = Qt.QScrollArea()
        self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
        self.top_scroll_layout.addWidget(self.top_scroll)
        self.top_scroll.setWidgetResizable(True)
        self.top_widget = Qt.QWidget()
        self.top_scroll.setWidget(self.top_widget)
        self.top_layout = Qt.QVBoxLayout(self.top_widget)
        self.top_grid_layout = Qt.QGridLayout()
        self.top_layout.addLayout(self.top_grid_layout)

        self.settings = Qt.QSettings("GNU Radio", "chat2")
        self.restoreGeometry(self.settings.value("geometry").toByteArray())


        ##################################################
        # Variables
        ##################################################
        self.samp_rate = samp_rate = 400000

        ##################################################
```

(Image continued from previous page)

```
        # Blocks
        ##################################################
        self.uhd_usrp_source_0 = uhd.usrp_source(
            ",".join(("", "")),
            uhd.stream_args(
                cpu_format="fc32",
                channels=range(1),
            ),
        )
        self.uhd_usrp_source_0.set_samp_rate(samp_rate)
        self.uhd_usrp_source_0.set_center_freq(1000000000, 0)
        self.uhd_usrp_source_0.set_gain(30, 0)
        self.uhd_usrp_source_0.set_antenna('TX/RX', 0)
        self.qtgui_freq_sink_x_0 = qtgui.freq_sink_c(
            1024, #size
            firdes.WIN_BLACKMAN_hARRIS, #wintype
            0, #fc
            samp_rate, #bw
            "", #name
            1 #number of inputs
        )
        self.qtgui_freq_sink_x_0.set_update_time(0.10)
        self.qtgui_freq_sink_x_0.set_y_axis(-140, 10)
        self.qtgui_freq_sink_x_0.set_y_label('Relative Gain', 'dB')
        self.qtgui_freq_sink_x_0.set_trigger_mode(qtgui.TRIG_MODE_FREE, 0.0, 0, "")
        self.qtgui_freq_sink_x_0.enable_autoscale(False)
        self.qtgui_freq_sink_x_0.enable_grid(False)
        self.qtgui_freq_sink_x_0.set_fft_average(1.0)
        self.qtgui_freq_sink_x_0.enable_axis_labels(True)
        self.qtgui_freq_sink_x_0.enable_control_panel(False)

        if not True:
          self.qtgui_freq_sink_x_0.disable_legend()

        if "complex" == "float" or "complex" == "msg_float":
          self.qtgui_freq_sink_x_0.set_plot_pos_half(not True)

        labels = ['', '', '', '', '',
                  '', '', '', '', '']
        widths = [1, 1, 1, 1, 1,
                  1, 1, 1, 1, 1]
        colors = ["blue", "red", "green", "black", "cyan",
                  "magenta", "yellow", "dark red", "dark green", "dark blue"]
        alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
                  1.0, 1.0, 1.0, 1.0, 1.0]
        for i in xrange(1):
            if len(labels[i]) == 0:
                self.qtgui_freq_sink_x_0.set_line_label(i, "Data {0}".format(i))
            else:
                self.qtgui_freq_sink_x_0.set_line_label(i, labels[i])
            self.qtgui_freq_sink_x_0.set_line_width(i, widths[i])
            self.qtgui_freq_sink_x_0.set_line_color(i, colors[i])
            self.qtgui_freq_sink_x_0.set_line_alpha(i, alphas[i])

        self._qtgui_freq_sink_x_0_win = sip.wrapinstance(self.qtgui_freq_sink_x_0.pyqwidget(), Qt.QWidget)
        self.top_grid_layout.addWidget(self._qtgui_freq_sink_x_0_win)
        self.low_pass_filter_0 = filter.fir_filter_ccf(1, firdes.low_pass(
            1, samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))
        self.digital_gmsk_demod_1 = digital.gmsk_demod(
            samples_per_symbol=2,
            gain_mu=0.175,
            mu=0.5,
            omega_relative_limit=0.005,
            freq_error=0.0,
            verbose=False,
            log=False,
        )
        self.blocks_multiply_const_vxx_1 = blocks.multiply_const_vcc((1, ))
        self.blocks_file_sink_1 = blocks.file_sink(gr.sizeof_char*1, '/home/b/chat_test2.txt', False)
        self.blocks_file_sink_1.set_unbuffered(False)
        self.blks2_packet_decoder_1 = grc_blks2.packet_demod_b(grc_blks2.packet_decoder(
                access_code='',
                threshold=-1,
                callback=lambda ok, payload: self.blks2_packet_decoder_1.recv_pkt(ok, payload),
```

(Image continued from previous page)

```
            ),
        )



        ##################################################
        # Connections
        ##################################################
        self.connect((self.blks2_packet_decoder_1, 0), (self.blocks_file_sink_1, 0))
        self.connect((self.blocks_multiply_const_vxx_1, 0), (self.low_pass_filter_0, 0))
        self.connect((self.digital_gmsk_demod_1, 0), (self.blks2_packet_decoder_1, 0))
        self.connect((self.low_pass_filter_0, 0), (self.digital_gmsk_demod_1, 0))
        self.connect((self.low_pass_filter_0, 0), (self.qtgui_freq_sink_x_0, 0))
        self.connect((self.uhd_usrp_source_0, 0), (self.blocks_multiply_const_vxx_1, 0))

    def closeEvent(self, event):
        self.settings = Qt.QSettings("GNU Radio", "chat2")
        self.settings.setValue("geometry", self.saveGeometry())
        event.accept()

    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.uhd_usrp_source_0.set_samp_rate(self.samp_rate)
        self.qtgui_freq_sink_x_0.set_frequency_range(0, self.samp_rate)
        self.low_pass_filter_0.set_taps(firdes.low_pass(1, self.samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))


def main(options=None):

    from distutils.version import StrictVersion
    if StrictVersion(Qt.qVersion()) >= StrictVersion("4.5.0"):
        style = gr.prefs().get_string('qtgui', 'style', 'raster')
        Qt.QApplication.setGraphicsSystem(style)
    qapp = Qt.QApplication(sys.argv)

    data_csv = open('/home/b/20161005_140846.csv', 'r')
    csvreader = csv.reader(data_csv)
    header = next(csvreader)
    data_rows = []
    for row in csvreader:
        data_rows.append(row)
    data_csv.close()

    for x in range(0,14):
        out = open('file' + str(x+1) + '.txt', 'w')
        for y in range(0, 100):
            out.write(data_rows[y][x+6]+' '+str(x+1)+'0'+str(y+1)+'\n')
        out.close()

    tb = chat2()
    tb.start()
    print "Ready to start..."
    while True:
        f = open('/home/b/chat_test1.txt', 'r')
        f.flush()
        f.readline()
        check = f.readline()
        if (check != ""):
            print "START"
            time.sleep(1)
            break
    tb.stop()

    weight_file = open('/home/b/chat_test1.txt', 'r')
    weight_file.flush()
    weight_file.readline()
    weights = weight_file.readline()
    w_list = weights.split(",")
    w_list.pop()
    for m in range(0, len(w_list)):
        w_list[m] = int(w_list[m])
```

(Image continued from previous page)

```
538
539     sensors = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N']
540
541     global i, a1, b1, c1, d1, e1, f1, g1, h1, i1, j1, k1, l1, m1, n1
542     i = 0
543     a1 = 0
544     b1 = 0
545     c1 = 0
546     d1 = 0
547     e1 = 0
548     f1 = 0
549     g1 = 0
550     h1 = 0
551     i1 = 0
552     j1 = 0
553     k1 = 0
554     l1 = 0
555     m1 = 0
556     n1 = 0
557
558     global data0, data1, data2, data3, data4, data5, data6, data7, data8, data9, data10, data11, data12, data13
559
560     read1 = open('/home/b/file1.txt', 'r')
561     data0 = read1.readlines()
562     length1 = len(data0)
563     read1.close()
564
565     read2 = open('/home/b/file2.txt', 'r')
566     data1 = read2.readlines()
567     length2 = len(data1)
568     read2.close()
569
570     read3 = open('/home/b/file3.txt', 'r')
571     data2 = read3.readlines()
572     length3 = len(data2)
573     read3.close()
574
575     read4 = open('/home/b/file4.txt', 'r')
576     data3 = read4.readlines()
577     length4 = len(data3)
578     read4.close()
579
580     read5 = open('/home/b/file5.txt', 'r')
581     data4 = read5.readlines()
582     length5 = len(data4)
583     read5.close()
584
585     read6 = open('/home/b/file6.txt', 'r')
586     data5 = read6.readlines()
587     length6 = len(data5)
588     read6.close()
589
590     read7 = open('/home/b/file7.txt', 'r')
591     data6 = read7.readlines()
592     length7 = len(data6)
593     read7.close()
594
595     read8 = open('/home/b/file8.txt', 'r')
596     data7 = read8.readlines()
597     length8 = len(data7)
598     read8.close()
599
600     read9 = open('/home/b/file9.txt', 'r')
601     data8 = read9.readlines()
602     length9 = len(data8)
603     read9.close()
604
605     read10 = open('/home/b/file10.txt', 'r')
606     data9 = read10.readlines()
607     length10 = len(data9)
608     read10.close()
609
610     read11 = open('/home/b/file11.txt', 'r')
611     data10 = read11.readlines()
```

(Image continued from previous page)

```
    length11 = len(data10)
    read11.close()

    read12 = open('/home/b/file12.txt', 'r')
    data11 = read12.readlines()
    length12 = len(data11)
    read12.close()

    read13 = open('/home/b/file13.txt', 'r')
    data12 = read13.readlines()
    length13 = len(data12)
    read13.close()

    read14 = open('/home/b/file14.txt', 'r')
    data13 = read14.readlines()
    length14 = len(data13)
    read14.close()

    data = list(zip(sensors, w_list))
    sched = WRRScheduler(data)

    global a_tot, b_tot, c_tot, d_tot, e_tot, f_tot, g_tot, h_tot, i_tot, j_tot, k_tot, l_tot, m_tot, n_tot, switch
    a_tot = 0
    b_tot = 0
    c_tot = 0
    d_tot = 0
    e_tot = 0
    f_tot = 0
    g_tot = 0
    h_tot = 0
    i_tot = 0
    j_tot = 0
    k_tot = 0
    l_tot = 0
    m_tot = 0
    n_tot = 0

    x = 0
    result = []
    while (x < 150):
        choose = sched.get_next()
        if choose[0] == 'A':
            a_tot += 1
            if (a_tot <= length1):
                result.append(choose[0])
                x += 1
        elif choose[0] == 'B':
            b_tot += 1
            if (b_tot <= length2):
                result.append(choose[0])
                x += 1
        elif choose[0] == 'C':
            c_tot += 1
            if (c_tot <= length3):
                result.append(choose[0])
                x += 1
        elif choose[0] == 'D':
            d_tot += 1
            if (d_tot <= length4):
                result.append(choose[0])
                x += 1
        elif choose[0] == 'E':
            e_tot += 1
            if (e_tot <= length5):
                result.append(choose[0])
                x += 1
        elif choose[0] == 'F':
            f_tot += 1
            if (f_tot <= length6):
                result.append(choose[0])
                x += 1
        elif choose[0] == 'G':
            g_tot += 1
            if (g_tot <= length7):
```

(Image continued from previous page)

```
                result.append(choose[0])
                x += 1
        elif choose[0] == 'H':
            h_tot += 1
            if (h_tot <= length8):
                result.append(choose[0])
                x += 1
        elif choose[0] == 'I':
            i_tot += 1
            if (i_tot <= length9):
                result.append(choose[0])
                x += 1
        elif choose[0] == 'J':
            j_tot += 1
            if (j_tot <= length10):
                result.append(choose[0])
                x += 1
        elif choose[0] == 'K':
            k_tot += 1
            if (k_tot <= length11):
                result.append(choose[0])
                x += 1
        elif choose[0] == 'L':
            l_tot += 1
            if (l_tot <= length12):
                result.append(choose[0])
                x += 1
        elif choose[0] == 'M':
            m_tot += 1
            if (m_tot <= length13):
                result.append(choose[0])
                x += 1
        elif choose[0] == 'N':
            n_tot += 1
            if (n_tot <= length14):
                result.append(choose[0])
                x += 1

    while i < (len(result)):
        if result[i] == 'A':
            switch = 'a'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            a1 += 1
        elif result[i] == 'B':
            switch = 'b'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            b1 += 1
        elif result[i] == 'C':
            switch = 'c'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            c1 += 1
        elif result[i] == 'D':
            switch = 'd'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            d1 += 1
        elif result[i] == 'E':
            switch = 'e'
            tb = text_tx()
```

(Image continued from previous page)

```
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            e1 += 1
        elif result[i] == 'F':
            switch = 'f'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            f1 += 1
        elif result[i] == 'G':
            switch = 'g'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            g1 += 1
        elif result[i] == 'H':
            switch = 'h'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            h1 += 1
        elif result[i] == 'I':
            switch = 'i'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            i1 += 1
        elif result[i] == 'J':
            switch = 'j'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            j1 += 1
        elif result[i] == 'K':
            switch = 'k'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            k1 += 1
        elif result[i] == 'L':
            switch = 'l'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            l1 += 1
        elif result[i] == 'M':
            switch = 'm'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            m1 += 1
        elif result[i] == 'N':
            switch = 'n'
            tb = text_tx()
            tb.start()
            time.sleep(1)
```

(Image continued from previous page)

```
            tb.stop()
            i += 1
            n1 += 1
        else:
            print "Error: Empty Result"
        tb = ack2()
        tb.start()
        while True:
            ack_file = open('/home/b/chat_test2.txt', 'r')
            ack_file.flush()
            ack_file.readline()
            ack0 = ack_file.readline()
            if (ack0 == 'ACK\n'):
                print "RECEIVING ACK"
                time.sleep(1)
                break
            ack_file.close()
            time.sleep(1)
        tb.stop()

    switch = 'eof'
    tb = text_tx()
    tb.start()
    time.sleep(1)
    tb.stop()

    print('done')
    exit()

    def quitting():
        tb.stop()
        tb.wait()
    qapp.connect(qapp, Qt.SIGNAL("aboutToQuit()"), quitting)
    qapp.exec_()


if __name__ == '__main__':
    main()
```

Figure A.2: Weighted Round-Robin Algorithm for Field Sensor Radio

```python
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
##################################################
# GNU Radio Python Flow Graph
# Title: Chat
# Generated: Thu Feb 10 09:30:28 2022
##################################################

if __name__ == '__main__':
    import ctypes
    import sys
    if sys.platform.startswith('linux'):
        try:
            x11 = ctypes.cdll.LoadLibrary('libX11.so')
            x11.XInitThreads()
        except:
            print "Warning: failed to XInitThreads()"

import ...


class chat(gr.top_block, Qt.QWidget):

    def __init__(self):
        gr.top_block.__init__(self, "Chat")
        Qt.QWidget.__init__(self)
        self.setWindowTitle("Chat")
        qtgui.util.check_set_qss()
        try:
            self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
        except:
            pass
        self.top_scroll_layout = Qt.QVBoxLayout()
        self.setLayout(self.top_scroll_layout)
        self.top_scroll = Qt.QScrollArea()
        self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
        self.top_scroll_layout.addWidget(self.top_scroll)
        self.top_scroll.setWidgetResizable(True)
        self.top_widget = Qt.QWidget()
        self.top_scroll.setWidget(self.top_widget)
        self.top_layout = Qt.QVBoxLayout(self.top_widget)
        self.top_grid_layout = Qt.QGridLayout()
        self.top_layout.addLayout(self.top_grid_layout)

        self.settings = Qt.QSettings("GNU Radio", "chat")
        self.restoreGeometry(self.settings.value("geometry").toByteArray())


        ##################################################
        # Variables
        ##################################################
        self.samp_rate = samp_rate = 400000

        ##################################################
        # Blocks
        ##################################################
        self.uhd_usrp_sink_1 = uhd.usrp_sink(
            ",".join(("", "")),
            uhd.stream_args(
                cpu_format="fc32",
                channels=range(1),
            ),
        )
        self.uhd_usrp_sink_1.set_samp_rate(samp_rate)
        self.uhd_usrp_sink_1.set_time_now(uhd.time_spec(time.time()), uhd.ALL_MBOARDS)
        self.uhd_usrp_sink_1.set_center_freq(1000000000, 0)
        self.uhd_usrp_sink_1.set_gain(40, 0)
        self.uhd_usrp_sink_1.set_antenna('RX2', 0)
        self.digital_gmsk_mod_0 = digital.gmsk_mod(
            samples_per_symbol=2,
            bt=0.35,
            verbose=False,
            log=False,
        )
```

(Image continued from previous page)

```
        self.blocks_throttle_0 = blocks.throttle(gr.sizeof_char*1, samp_rate,True)
        self.blocks_multiply_const_vxx_1 = blocks.multiply_const_vcc((1, ))
        self.blocks_head_0 = blocks.head(gr.sizeof_char*1, 100000)
        self.blocks_file_source_0 = blocks.file_source(gr.sizeof_char*1, '/home/c/chat3', True)
        self.blocks_file_source_0.set_begin_tag(pmt.PMT_NIL)
        self.blks2_packet_encoder_0 = grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
                samples_per_symbol=2,
                bits_per_symbol=1,
                preamble='',
                access_code='',
                pad_for_usrp=True,
            ),
            payload_length=0,
        )



        ##################################################
        # Connections
        ##################################################
        self.connect((self.blks2_packet_encoder_0, 0), (self.digital_gmsk_mod_0, 0))
        self.connect((self.blocks_file_source_0, 0), (self.blocks_throttle_0, 0))
        self.connect((self.blocks_head_0, 0), (self.blks2_packet_encoder_0, 0))
        self.connect((self.blocks_multiply_const_vxx_1, 0), (self.uhd_usrp_sink_1, 0))
        self.connect((self.blocks_throttle_0, 0), (self.blocks_head_0, 0))
        self.connect((self.digital_gmsk_mod_0, 0), (self.blocks_multiply_const_vxx_1, 0))

    def closeEvent(self, event):
        self.settings = Qt.QSettings("GNU Radio", "chat")
        self.settings.setValue("geometry", self.saveGeometry())
        event.accept()

    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.uhd_usrp_sink_1.set_samp_rate(self.samp_rate)
        self.blocks_throttle_0.set_sample_rate(self.samp_rate)


class text_rx(gr.top_block, Qt.QWidget):

    def __init__(self):
        gr.top_block.__init__(self, "Text Rx")
        Qt.QWidget.__init__(self)
        self.setWindowTitle("Text Rx")
        qtgui.util.check_set_qss()
        try:
            self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
        except:
            pass
        self.top_scroll_layout = Qt.QVBoxLayout()
        self.setLayout(self.top_scroll_layout)
        self.top_scroll = Qt.QScrollArea()
        self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
        self.top_scroll_layout.addWidget(self.top_scroll)
        self.top_scroll.setWidgetResizable(True)
        self.top_widget = Qt.QWidget()
        self.top_scroll.setWidget(self.top_widget)
        self.top_layout = Qt.QVBoxLayout(self.top_widget)
        self.top_grid_layout = Qt.QGridLayout()
        self.top_layout.addLayout(self.top_grid_layout)

        self.settings = Qt.QSettings("GNU Radio", "text_rx")
        self.restoreGeometry(self.settings.value("geometry").toByteArray())


        ##################################################
        # Variables
        ##################################################
        self.samp_rate = samp_rate = 400000

        ##################################################
```

(Image continued from previous page)

```
        # Blocks
        ##################################################
        self.uhd_usrp_source_0 = uhd.usrp_source(
            ",".join(("", "")),
            uhd.stream_args(
                cpu_format="fc32",
                channels=range(1),
            ),
        )
        self.uhd_usrp_source_0.set_samp_rate(samp_rate)
        self.uhd_usrp_source_0.set_center_freq(1000000000, 0)
        self.uhd_usrp_source_0.set_gain(30, 0)
        self.uhd_usrp_source_0.set_antenna('RX2', 0)
        self.qtgui_freq_sink_x_0 = qtgui.freq_sink_c(
            1024, #size
            firdes.WIN_BLACKMAN_hARRIS, #wintype
            0, #fc
            samp_rate, #bw
            "", #name
            1 #number of inputs
        )
        self.qtgui_freq_sink_x_0.set_update_time(0.10)
        self.qtgui_freq_sink_x_0.set_y_axis(-140, 10)
        self.qtgui_freq_sink_x_0.set_y_label('Relative Gain', 'dB')
        self.qtgui_freq_sink_x_0.set_trigger_mode(qtgui.TRIG_MODE_FREE, 0.0, 0, "")
        self.qtgui_freq_sink_x_0.enable_autoscale(False)
        self.qtgui_freq_sink_x_0.enable_grid(False)
        self.qtgui_freq_sink_x_0.set_fft_average(1.0)
        self.qtgui_freq_sink_x_0.enable_axis_labels(True)
        self.qtgui_freq_sink_x_0.enable_control_panel(False)

        if not True:
          self.qtgui_freq_sink_x_0.disable_legend()

        if "complex" == "float" or "complex" == "msg_float":
          self.qtgui_freq_sink_x_0.set_plot_pos_half(not True)

        labels = ['', '', '', '', '',
                  '', '', '', '', '']
        widths = [1, 1, 1, 1, 1,
                  1, 1, 1, 1, 1]
        colors = ["blue", "red", "green", "black", "cyan",
                  "magenta", "yellow", "dark red", "dark green", "dark blue"]
        alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
                  1.0, 1.0, 1.0, 1.0, 1.0]
        for i in xrange(1):
            if len(labels[i]) == 0:
                self.qtgui_freq_sink_x_0.set_line_label(i, "Data {0}".format(i))
            else:
                self.qtgui_freq_sink_x_0.set_line_label(i, labels[i])
            self.qtgui_freq_sink_x_0.set_line_width(i, widths[i])
            self.qtgui_freq_sink_x_0.set_line_color(i, colors[i])
            self.qtgui_freq_sink_x_0.set_line_alpha(i, alphas[i])

        self._qtgui_freq_sink_x_0_win = sip.wrapinstance(self.qtgui_freq_sink_x_0.pyqwidget(), Qt.QWidget)
        self.top_grid_layout.addWidget(self._qtgui_freq_sink_x_0_win)
        self.low_pass_filter_0 = filter.fir_filter_ccf(1, firdes.low_pass(
            1, samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))
        self.digital_gmsk_demod_0 = digital.gmsk_demod(
            samples_per_symbol=2,
            gain_mu=0.175,
            mu=0.5,
            omega_relative_limit=0.005,
            freq_error=0.0,
            verbose=False,
            log=False,
        )

        clr_file = open('/home/c/test_file2', 'w')
        clr_file.close()

        self.blocks_multiply_const_vxx_0 = blocks.multiply_const_vcc((1, ))
        self.blocks_file_sink_0 = blocks.file_sink(gr.sizeof_char*1, '/home/c/test_file2', False)
        self.blocks_file_sink_0.set_unbuffered(False)
```

(Image continued from previous page)

```
        self.blks2_packet_decoder_0 = grc_blks2.packet_demod_b(grc_blks2.packet_decoder(
                access_code='',
                threshold=-1,
                callback=lambda ok, payload: self.blks2_packet_decoder_0.recv_pkt(ok, payload),
            ),
        )




        ##################################################
        # Connections
        ##################################################
        self.connect((self.blks2_packet_decoder_0, 0), (self.blocks_file_sink_0, 0))
        self.connect((self.blocks_multiply_const_vxx_0, 0), (self.low_pass_filter_0, 0))
        self.connect((self.digital_gmsk_demod_0, 0), (self.blks2_packet_decoder_0, 0))
        self.connect((self.low_pass_filter_0, 0), (self.digital_gmsk_demod_0, 0))
        self.connect((self.low_pass_filter_0, 0), (self.qtgui_freq_sink_x_0, 0))
        self.connect((self.uhd_usrp_source_0, 0), (self.blocks_multiply_const_vxx_0, 0))


    def closeEvent(self, event):
        self.settings = Qt.QSettings("GNU Radio", "text_rx")
        self.settings.setValue("geometry", self.saveGeometry())
        event.accept()

    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.uhd_usrp_source_0.set_samp_rate(self.samp_rate)
        self.qtgui_freq_sink_x_0.set_frequency_range(0, self.samp_rate)
        self.low_pass_filter_0.set_taps(firdes.low_pass(1, self.samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))


class ack(gr.top_block, Qt.QWidget):

    def __init__(self):
        gr.top_block.__init__(self, "Chat")
        Qt.QWidget.__init__(self)
        self.setWindowTitle("Chat")
        qtgui.util.check_set_qss()
        try:
            self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
        except:
            pass
        self.top_scroll_layout = Qt.QVBoxLayout()
        self.setLayout(self.top_scroll_layout)
        self.top_scroll = Qt.QScrollArea()
        self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
        self.top_scroll_layout.addWidget(self.top_scroll)
        self.top_scroll.setWidgetResizable(True)
        self.top_widget = Qt.QWidget()
        self.top_scroll.setWidget(self.top_widget)
        self.top_layout = Qt.QVBoxLayout(self.top_widget)
        self.top_grid_layout = Qt.QGridLayout()
        self.top_layout.addLayout(self.top_grid_layout)

        self.settings = Qt.QSettings("GNU Radio", "chat")
        self.restoreGeometry(self.settings.value("geometry").toByteArray())


        ##################################################
        # Variables
        ##################################################
        self.samp_rate = samp_rate = 400000

        weight_file = open('/home/c/chat4', 'w')
        weight_file.write("ACK\n")
        weight_file.close()

        ##################################################
        # Blocks
```

(Image continued from previous page)

```
        ##################################################
        self.uhd_usrp_sink_1 = uhd.usrp_sink(
            ",".join(("", "")),
            uhd.stream_args(
                cpu_format="fc32",
                channels=range(1),
            ),
        )
        self.uhd_usrp_sink_1.set_samp_rate(samp_rate)
        self.uhd_usrp_sink_1.set_time_now(uhd.time_spec(time.time()), uhd.ALL_MBOARDS)
        self.uhd_usrp_sink_1.set_center_freq(1000000000, 0)
        self.uhd_usrp_sink_1.set_gain(40, 0)
        self.uhd_usrp_sink_1.set_antenna('TX/RX', 0)
        self.digital_gmsk_mod_0 = digital.gmsk_mod(
            samples_per_symbol=2,
            bt=0.35,
            verbose=False,
            log=False,
        )
        self.blocks_throttle_0 = blocks.throttle(gr.sizeof_char*1, samp_rate,True)
        self.blocks_multiply_const_vxx_1 = blocks.multiply_const_vcc((1, ))
        self.blocks_head_0 = blocks.head(gr.sizeof_char*1, 100000)
        self.blocks_file_source_0 = blocks.file_source(gr.sizeof_char*1, '/home/c/chat4', True)
        self.blocks_file_source_0.set_begin_tag(pmt.PMT_NIL)
        self.blks2_packet_encoder_0 = grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
                samples_per_symbol=2,
                bits_per_symbol=1,
                preamble='',
                access_code='',
                pad_for_usrp=True,
            ),
            payload_length=0,
        )



        ##################################################
        # Connections
        ##################################################
        self.connect((self.blks2_packet_encoder_0, 0), (self.digital_gmsk_mod_0, 0))
        self.connect((self.blocks_file_source_0, 0), (self.blocks_throttle_0, 0))
        self.connect((self.blocks_head_0, 0), (self.blks2_packet_encoder_0, 0))
        self.connect((self.blocks_multiply_const_vxx_1, 0), (self.uhd_usrp_sink_1, 0))
        self.connect((self.blocks_throttle_0, 0), (self.blocks_head_0, 0))
        self.connect((self.digital_gmsk_mod_0, 0), (self.blocks_multiply_const_vxx_1, 0))

    def closeEvent(self, event):
        self.settings = Qt.QSettings("GNU Radio", "chat")
        self.settings.setValue("geometry", self.saveGeometry())
        event.accept()

    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.uhd_usrp_sink_1.set_samp_rate(self.samp_rate)
        self.blocks_throttle_0.set_sample_rate(self.samp_rate)


def main(options=None):

    from distutils.version import StrictVersion
    if StrictVersion(Qt.qVersion()) >= StrictVersion("4.5.0"):
        style = gr.prefs().get_string('qtgui', 'style', 'raster')
        Qt.QApplication.setGraphicsSystem(style)
    qapp = Qt.QApplication(sys.argv)

    a_num = 0
    b_num = 0
    c_num = 0
    d_num = 0
    e_num = 0
    f_num = 0
```

(Image continued from previous page)

```
    g_num = 0
    h_num = 0
    i_num = 0
    j_num = 0
    k_num = 0
    l_num = 0
    m_num = 0
    n_num = 0
    sum_a = 0
    sum_b = 0
    sum_c = 0
    sum_d = 0
    sum_e = 0
    sum_f = 0
    sum_g = 0
    sum_h = 0
    sum_i = 0
    sum_j = 0
    sum_k = 0
    sum_l = 0
    sum_m = 0
    sum_n = 0
    predict_a = 0
    predict_b = 0
    predict_c = 0
    predict_d = 0
    predict_e = 0
    predict_f = 0
    predict_g = 0
    predict_h = 0
    predict_i = 0
    predict_j = 0
    predict_k = 0
    predict_l = 0
    predict_m = 0
    predict_n = 0
    actual_a_list = []
    actual_b_list = []
    actual_c_list = []
    actual_d_list = []
    actual_e_list = []
    actual_f_list = []
    actual_g_list = []
    actual_h_list = []
    actual_i_list = []
    actual_j_list = []
    actual_k_list = []
    actual_l_list = []
    actual_m_list = []
    actual_n_list = []
    predict_a_list = []
    predict_b_list = []
    predict_c_list = []
    predict_d_list = []
    predict_e_list = []
    predict_f_list = []
    predict_g_list = []
    predict_h_list = []
    predict_i_list = []
    predict_j_list = []
    predict_k_list = []
    predict_l_list = []
    predict_m_list = []
    predict_n_list = []

    while True:
        weight1 = raw_input("Enter a value for the weight of sensor 1 (integer): ")
        try:
            int(weight1)
            if (int(weight1) < 0):
                print "Value cannot be negative. Try again."
                continue
        except ValueError:
            print "Value provided needs to be an integer. Try again."
```

(Image continued from previous page)

```
            continue
        break

    while True:
        weight2 = raw_input("Enter a value for the weight of sensor 2 (integer): ")
        try:
            int(weight2)
            if (int(weight2) < 0):
                print "Value cannot be negative. Try again."
                continue
        except ValueError:
            print "Value provided needs to be an integer. Try again."
            continue
        break

    while True:
        weight3 = raw_input("Enter a value for the weight of sensor 3 (integer): ")
        try:
            int(weight3)
            if (int(weight3) < 0):
                print "Value cannot be negative. Try again."
                continue
        except ValueError:
            print "Value provided needs to be an integer. Try again."
            continue
        break

    while True:
        weight4 = raw_input("Enter a value for the weight of sensor 4 (integer): ")
        try:
            int(weight4)
            if (int(weight4) < 0):
                print "Value cannot be negative. Try again."
                continue
        except ValueError:
            print "Value provided needs to be an integer. Try again."
            continue
        break

    while True:
        weight5 = raw_input("Enter a value for the weight of sensor 5 (integer): ")
        try:
            int(weight5)
            if (int(weight5) < 0):
                print "Value cannot be negative. Try again."
                continue
        except ValueError:
            print "Value provided needs to be an integer. Try again."
            continue
        break

    while True:
        weight6 = raw_input("Enter a value for the weight of sensor 6 (integer): ")
        try:
            int(weight6)
            if (int(weight6) < 0):
                print "Value cannot be negative. Try again."
                continue
        except ValueError:
            print "Value provided needs to be an integer. Try again."
            continue
        break

    while True:
        weight7 = raw_input("Enter a value for the weight of sensor 7 (integer): ")
        try:
            int(weight7)
            if (int(weight7) < 0):
                print "Value cannot be negative. Try again."
                continue
        except ValueError:
            print "Value provided needs to be an integer. Try again."
            continue
        break
```

(Image continued from previous page)

```
536
537     while True:
538         weight8 = raw_input("Enter a value for the weight of sensor 8 (integer): ")
539         try:
540             int(weight8)
541             if (int(weight8) < 0):
542                 print "Value cannot be negative. Try again."
543                 continue
544         except ValueError:
545             print "Value provided needs to be an integer. Try again."
546             continue
547         break
548
549     while True:
550         weight9 = raw_input("Enter a value for the weight of sensor 9 (integer): ")
551         try:
552             int(weight9)
553             if (int(weight9) < 0):
554                 print "Value cannot be negative. Try again."
555                 continue
556         except ValueError:
557             print "Value provided needs to be an integer. Try again."
558             continue
559         break
560
561     while True:
562         weight10 = raw_input("Enter a value for the weight of sensor 10 (integer): ")
563         try:
564             int(weight10)
565             if (int(weight10) < 0):
566                 print "Value cannot be negative. Try again."
567                 continue
568         except ValueError:
569             print "Value provided needs to be an integer. Try again."
570             continue
571         break
572
573     while True:
574         weight11 = raw_input("Enter a value for the weight of sensor 11 (integer): ")
575         try:
576             int(weight11)
577             if (int(weight11) < 0):
578                 print "Value cannot be negative. Try again."
579                 continue
580         except ValueError:
581             print "Value provided needs to be an integer. Try again."
582             continue
583         break
584
585     while True:
586         weight12 = raw_input("Enter a value for the weight of sensor 12 (integer): ")
587         try:
588             int(weight12)
589             if (int(weight12) < 0):
590                 print "Value cannot be negative. Try again."
591                 continue
592         except ValueError:
593             print "Value provided needs to be an integer. Try again."
594             continue
595         break
596
597     while True:
598         weight13 = raw_input("Enter a value for the weight of sensor 13 (integer): ")
599         try:
600             int(weight13)
601             if (int(weight13) < 0):
602                 print "Value cannot be negative. Try again."
603                 continue
604         except ValueError:
605             print "Value provided needs to be an integer. Try again."
606             continue
607         break
608
609     while True:
```

(Image continued from previous page)

```
        weight14 = raw_input("Enter a value for the weight of sensor 14 (integer): ")
        try:
            int(weight14)
            if (int(weight14) < 0):
                print "Value cannot be negative. Try again."
                continue
        except ValueError:
            print "Value provided needs to be an integer. Try again."
            continue
        break

    weight_file = open('/home/c/chat3', 'w')

    weight_file.write(weight1 + ',')
    weight_file.write(weight2 + ',')
    weight_file.write(weight3 + ',')
    weight_file.write(weight4 + ',')
    weight_file.write(weight5 + ',')
    weight_file.write(weight6 + ',')
    weight_file.write(weight7 + ',')
    weight_file.write(weight8 + ',')
    weight_file.write(weight9 + ',')
    weight_file.write(weight10 + ',')
    weight_file.write(weight11 + ',')
    weight_file.write(weight12 + ',')
    weight_file.write(weight13 + ',')
    weight_file.write(weight14 + ',\n')

    weight_file.close()

    tb = chat()
    tb.start()
    time.sleep(1)
    tb.stop()

    final = open('/home/c/final', 'w')
    final.close()

    packet_tot = 0

    change0_list = []
    change1_list = []
    change2_list = []
    change3_list = []
    change4_list = []
    change5_list = []
    change6_list = []
    change7_list = []
    change8_list = []
    change9_list = []
    change10_list = []
    change11_list = []
    change12_list = []
    change13_list = []
    sum_change0 = 0
    sum_change1 = 0
    sum_change2 = 0
    sum_change3 = 0
    sum_change4 = 0
    sum_change5 = 0
    sum_change6 = 0
    sum_change7 = 0
    sum_change8 = 0
    sum_change9 = 0
    sum_change10 = 0
    sum_change11 = 0
    sum_change12 = 0
    sum_change13 = 0

    while True:
        tb = text_rx()
        tb.start()
        data1 = open('/home/c/test_file2', 'r+')
        final = open('/home/c/final', 'a')
```

(Image continued from previous page)

```
    while True:
        diff = 0
        data1.flush()
        data1.readline()
        file_content2 = data1.readline()
        if (file_content2 == "eof\n"):
            print(file_content2)
            break
        elif (file_content2 != ""):
            packet_tot += 1
            final.write(file_content2)
            split = file_content2.split(' ')
            if (split[1].startswith('10') and (split[1].startswith('0', 2) == False)):
                a_num += 1
                actual_a_list.append(float(split[0]))
                if (a_num > 1):
                    diff = abs(predict_a_list[-1]-actual_a_list[-1])
                    print(diff)
                    if (diff > 2*int(weight1)):
                        weight1 = int(weight1)+1
                    elif (int(weight1) > 1) and (diff < int(weight1)):
                        weight1 = int(weight1)-1
                    print(weight1)
                if (a_num > 1):
                    change0 = actual_a_list[-1]-actual_a_list[-2]
                    sum_change0 += change0
                    predict_a = actual_a_list[-1] + sum_change0/a_num
                else:
                    predict_a = actual_a_list[-1]
                print("Prediction for Next A: "+str(predict_a))
                predict_a_list.append(predict_a)
            elif (split[1].startswith('20')):
                b_num += 1
                actual_b_list.append(float(split[0]))
                if (b_num > 1):
                    diff = abs(predict_b_list[-1]-actual_b_list[-1])
                    print(diff)
                    if (diff > 2*int(weight2)):
                        weight2 = int(weight2)+1
                    elif (int(weight2) > 1) and (diff < int(weight2)):
                        weight2 = int(weight2)-1
                    print(weight2)
                if (b_num > 1):
                    change1 = actual_b_list[-1]-actual_b_list[-2]
                    sum_change1 += change1
                    predict_b = actual_b_list[-1] + sum_change1/b_num
                else:
                    predict_b = actual_b_list[-1]
                print("Prediction for Next B: "+str(predict_b))
                predict_b_list.append(predict_b)
            elif (split[1].startswith('30')):
                c_num += 1
                actual_c_list.append(float(split[0]))
                if (c_num > 1):
                    diff = abs(predict_c_list[-1]-actual_c_list[-1])
                    print(diff)
                    if (diff > 2*int(weight3)):
                        weight3 = int(weight3)+1
                    elif (int(weight3) > 1) and (diff < int(weight3)):
                        weight3 = int(weight3)-1
                    print(weight3)
                if (c_num > 1):
                    change2 = actual_c_list[-1]-actual_c_list[-2]
                    sum_change2 += change2
                    predict_c = actual_c_list[-1] + sum_change2/c_num
                else:
                    predict_c = actual_c_list[-1]
                print("Prediction for Next C: "+str(predict_c))
                predict_c_list.append(predict_c)
            elif (split[1].startswith('40')):
                d_num += 1
                actual_d_list.append(float(split[0]))
                if (d_num > 1):
                    diff = abs(predict_d_list[-1]-actual_d_list[-1])
```

(Image continued from previous page)

```
                print(diff)
                if (diff > 2*int(weight4)):
                    weight4 = int(weight4)+1
                elif (int(weight4) > 1) and (diff < int(weight4)):
                    weight4 = int(weight4)-1
                print(weight4)
            if (d_num > 1):
                change3 = actual_d_list[-1]-actual_d_list[-2]
                sum_change3 += change3
                predict_d = actual_d_list[-1] + sum_change3/d_num
            else:
                predict_d = actual_d_list[-1]
            print("Prediction for Next D: "+str(predict_d))
            predict_d_list.append(predict_d)
        elif (split[1].startswith('50')):
            e_num += 1
            actual_e_list.append(float(split[0]))
            if (e_num > 1):
                diff = abs(predict_e_list[-1]-actual_e_list[-1])
                print(diff)
                if (diff > 2*int(weight5)):
                    weight5 = int(weight5)+1
                elif (int(weight5) > 1) and (diff < int(weight5)):
                    weight5 = int(weight5)-1
                print(weight5)
            if (e_num > 1):
                change4 = actual_e_list[-1]-actual_e_list[-2]
                sum_change4 += change4
                predict_e = actual_e_list[-1] + sum_change4/e_num
            else:
                predict_e = actual_e_list[-1]
            print("Prediction for Next E: "+str(predict_e))
            predict_e_list.append(predict_e)
        elif (split[1].startswith('60')):
            f_num += 1
            actual_f_list.append(float(split[0]))
            if (f_num > 1):
                diff = abs(predict_f_list[-1]-actual_f_list[-1])
                print(diff)
                if (diff > 2*int(weight6)):
                    weight6 = int(weight6)+1
                elif (int(weight6) > 1) and (diff < int(weight6)):
                    weight6 = int(weight6)-1
                print(weight6)
            if (f_num > 1):
                change5 = actual_f_list[-1]-actual_f_list[-2]
                sum_change5 += change5
                predict_f = actual_f_list[-1] + sum_change5/f_num
            else:
                predict_f = actual_f_list[-1]
            print("Prediction for Next F: "+str(predict_f))
            predict_f_list.append(predict_f)
        elif (split[1].startswith('70')):
            g_num += 1
            actual_g_list.append(float(split[0]))
            if (g_num > 1):
                diff = abs(predict_g_list[-1]-actual_g_list[-1])
                print(diff)
                if (diff > 2*int(weight7)):
                    weight7 = int(weight7)+1
                elif (int(weight7) > 1) and (diff < int(weight7)):
                    weight7 = int(weight7)-1
                print(weight7)
            if (g_num > 1):
                change6 = actual_g_list[-1]-actual_g_list[-2]
                sum_change6 += change6
                predict_g = actual_g_list[-1] + sum_change6/g_num
            else:
                predict_g = actual_g_list[-1]
            print("Prediction for Next G: "+str(predict_g))
            predict_g_list.append(predict_g)
        elif (split[1].startswith('80')):
            h_num += 1
            actual_h_list.append(float(split[0]))
```

(Image continued from previous page)

```
            if (h_num > 1):
                diff = abs(predict_h_list[-1]-actual_h_list[-1])
                print(diff)
                if (diff > 2*int(weight8)):
                    weight8 = int(weight8)+1
                elif (int(weight8) > 1) and (diff < int(weight8)):
                    weight8 = int(weight8)-1
                print(weight8)
            if (h_num > 1):
                change7 = actual_h_list[-1]-actual_h_list[-2]
                sum_change7 += change7
                predict_h = actual_h_list[-1] + sum_change7/h_num
            else:
                predict_h = actual_h_list[-1]
            print("Prediction for Next H: "+str(predict_h))
            predict_h_list.append(predict_h)
        elif (split[1].startswith('90')):
            i_num += 1
            actual_i_list.append(float(split[0]))
            if (i_num > 1):
                diff = abs(predict_i_list[-1]-actual_i_list[-1])
                print(diff)
                if (diff > 2*int(weight9)):
                    weight9 = int(weight9)+1
                elif (int(weight9) > 1) and (diff < int(weight9)):
                    weight9 = int(weight9)-1
                print(weight9)
            if (i_num > 1):
                change8 = actual_i_list[-1]-actual_i_list[-2]
                sum_change8 += change8
                predict_i = actual_i_list[-1] + sum_change8/i_num
            else:
                predict_i = actual_i_list[-1]
            print("Prediction for Next I: "+str(predict_i))
            predict_i_list.append(predict_i)
        elif (split[1].startswith('100')):
            j_num += 1
            actual_j_list.append(float(split[0]))
            if (j_num > 1):
                diff = abs(predict_j_list[-1]-actual_j_list[-1])
                print(diff)
                if (diff > 2*int(weight10)):
                    weight10 = int(weight10)+1
                elif (int(weight10) > 1) and (diff < int(weight10)):
                    weight10 = int(weight10)-1
                print(weight10)
            if (j_num > 1):
                change9 = actual_j_list[-1]-actual_j_list[-2]
                sum_change9 += change9
                predict_j = actual_j_list[-1] + sum_change9/j_num
            else:
                predict_j = actual_j_list[-1]
            print("Prediction for Next J: "+str(predict_j))
            predict_j_list.append(predict_j)
        elif (split[1].startswith('110')):
            k_num += 1
            actual_k_list.append(float(split[0]))
            if (k_num > 1):
                diff = abs(predict_k_list[-1]-actual_k_list[-1])
                print(diff)
                if (diff > 2*int(weight11)):
                    weight11 = int(weight11)+1
                elif (int(weight11) > 1) and (diff < int(weight11)):
                    weight11 = int(weight11)-1
                print(weight11)
            if (k_num > 1):
                change10 = actual_k_list[-1]-actual_k_list[-2]
                sum_change10 += change10
                predict_k = actual_k_list[-1] + sum_change10/k_num
            else:
                predict_k = actual_k_list[-1]
            print("Prediction for Next K: "+str(predict_k))
            predict_k_list.append(predict_k)
        elif (split[1].startswith('120')):
```

(Image continued from previous page)

```
                l_num += 1
                actual_l_list.append(float(split[0]))
                if (l_num > 1):
                    diff = abs(predict_l_list[-1]-actual_l_list[-1])
                    print(diff)
                    if (diff > 2*int(weight12)):
                        weight12 = int(weight12)+1
                    elif (int(weight12) > 1) and (diff < int(weight12)):
                        weight12 = int(weight12)-1
                    print(weight12)
                if (l_num > 1):
                    change11 = actual_l_list[-1]-actual_l_list[-2]
                    sum_change11 += change11
                    predict_l = actual_l_list[-1] + sum_change11/l_num
                else:
                    predict_l = actual_l_list[-1]
                print("Prediction for Next L: "+str(predict_l))
                predict_l_list.append(predict_l)
            elif (split[1].startswith('130')):
                m_num += 1
                actual_m_list.append(float(split[0]))
                if (m_num > 1):
                    diff = abs(predict_m_list[-1]-actual_m_list[-1])
                    print(diff)
                    if (diff > 2*int(weight13)):
                        weight13 = int(weight13)+1
                    elif (int(weight13) > 1) and (diff < int(weight13)):
                        weight13 = int(weight13)-1
                    print(weight13)
                if (m_num > 1):
                    change12 = actual_m_list[-1]-actual_m_list[-2]
                    sum_change12 += change12
                    predict_m = actual_m_list[-1] + sum_change12/m_num
                else:
                    predict_m = actual_m_list[-1]
                print("Prediction for Next M: "+str(predict_m))
                predict_m_list.append(predict_m)
            elif (split[1].startswith('140')):
                n_num += 1
                actual_n_list.append(float(split[0]))
                if (n_num > 1):
                    diff = abs(predict_n_list[-1]-actual_n_list[-1])
                    print(diff)
                    if (diff > 2*int(weight14)):
                        weight14 = int(weight14)+1
                    elif (int(weight14) > 1) and (diff < int(weight14)):
                        weight14 = int(weight14)-1
                    print(weight14)
                if (n_num > 1):
                    change13 = actual_n_list[-1]-actual_n_list[-2]
                    sum_change13 += change13
                    predict_n = actual_n_list[-1] + sum_change13/n_num
                else:
                    predict_n = actual_n_list[-1]
                print("Prediction for Next N: "+str(predict_n))
                predict_n_list.append(predict_n)
            print("Current Value: "+file_content2)
            time.sleep(1)
            sb = ack()
            sb.start()
            print "SENDING ACK"
            time.sleep(1)
            sb.stop()
            if (packet_tot > 28) and (packet_tot % 20 == 0):
                weight_file = open('/home/c/chat3', 'w')
                weight_file.write(str(weight1) + ',')
                weight_file.write(str(weight2) + ',')
                weight_file.write(str(weight3) + ',')
                weight_file.write(str(weight4) + ',')
                weight_file.write(str(weight5) + ',')
                weight_file.write(str(weight6) + ',')
                weight_file.write(str(weight7) + ',')
                weight_file.write(str(weight8) + ',')
                weight_file.write(str(weight9) + ',')
```

(Image continued from previous page)

```
                    weight_file.write(str(weight10) + ',')
                    weight_file.write(str(weight11) + ',')
                    weight_file.write(str(weight12) + ',')
                    weight_file.write(str(weight13) + ',')
                    weight_file.write(str(weight14) + ',\n')
                    weight_file.close()
                    up = chat()
                    up.start()
                    time.sleep(1)
                    up.stop()
                break
            time.sleep(1)

        tb.stop()

        if (file_content2 == "eof\n"):
            data1.close()
            final.close()
            break

        data1.close()
        final.close()

    tb.stop()

    sum_sq_a = 0
    for z in range(0, len(predict_a_list)-1):
        square = (actual_a_list[z+1] - predict_a_list[z])**2
        sum_sq_a += square
    msqrterr_a = math.sqrt(sum_sq_a)
    print("Root Mean Squared Error for Sensor 1: "+str(msqrterr_a))

    sum_sq_b = 0
    for z in range(0, len(predict_b_list)-1):
        square = (actual_b_list[z+1] - predict_b_list[z])**2
        sum_sq_b += square
    msqrterr_b = math.sqrt(sum_sq_b)
    print("Root Mean Squared Error for Sensor 2: "+str(msqrterr_b))

    sum_sq_c = 0
    for z in range(0, len(predict_c_list)-1):
        square = (actual_c_list[z+1] - predict_c_list[z])**2
        sum_sq_c += square
    msqrterr_c = math.sqrt(sum_sq_c)
    print("Root Mean Squared Error for Sensor 3: "+str(msqrterr_c))

    sum_sq_d = 0
    for z in range(0, len(predict_d_list)-1):
        square = (actual_d_list[z+1] - predict_d_list[z])**2
        sum_sq_d += square
    msqrterr_d = math.sqrt(sum_sq_d)
    print("Root Mean Squared Error for Sensor 4: "+str(msqrterr_d))

    sum_sq_e = 0
    for z in range(0, len(predict_e_list)-1):
        square = (actual_e_list[z+1] - predict_e_list[z])**2
        sum_sq_e += square
    msqrterr_e = math.sqrt(sum_sq_e)
    print("Root Mean Squared Error for Sensor 5: "+str(msqrterr_e))

    sum_sq_f = 0
    for z in range(0, len(predict_f_list)-1):
        square = (actual_f_list[z+1] - predict_f_list[z])**2
        sum_sq_f += square
    msqrterr_f = math.sqrt(sum_sq_f)
    print("Root Mean Squared Error for Sensor 6: "+str(msqrterr_f))

    sum_sq_g = 0
    for z in range(0, len(predict_g_list)-1):
        square = (actual_g_list[z+1] - predict_g_list[z])**2
        sum_sq_g += square
    msqrterr_g = math.sqrt(sum_sq_g)
    print("Root Mean Squared Error for Sensor 7: "+str(msqrterr_g))

```

(Image continued from previous page)

```
    sum_sq_h = 0
    for z in range(0, len(predict_h_list)-1):
        square = (actual_h_list[z+1] - predict_h_list[z])**2
        sum_sq_h += square
    msqrterr_h = math.sqrt(sum_sq_h)
    print("Root Mean Squared Error for Sensor 8: "+str(msqrterr_h))

    sum_sq_i = 0
    for z in range(0, len(predict_i_list)-1):
        square = (actual_i_list[z+1] - predict_i_list[z])**2
        sum_sq_i += square
    msqrterr_i = math.sqrt(sum_sq_i)
    print("Root Mean Squared Error for Sensor 9: "+str(msqrterr_i))

    sum_sq_j = 0
    for z in range(0, len(predict_j_list)-1):
        square = (actual_j_list[z+1] - predict_j_list[z])**2
        sum_sq_j += square
    msqrterr_j = math.sqrt(sum_sq_j)
    print("Root Mean Squared Error for Sensor 10: "+str(msqrterr_j))

    sum_sq_k = 0
    for z in range(0, len(predict_k_list)-1):
        square = (actual_k_list[z+1] - predict_k_list[z])**2
        sum_sq_k += square
    msqrterr_k = math.sqrt(sum_sq_k)
    print("Root Mean Squared Error for Sensor 11: "+str(msqrterr_k))

    sum_sq_l = 0
    for z in range(0, len(predict_l_list)-1):
        square = (actual_l_list[z+1] - predict_l_list[z])**2
        sum_sq_l += square
    msqrterr_l = math.sqrt(sum_sq_l)
    print("Root Mean Squared Error for Sensor 12: "+str(msqrterr_l))

    sum_sq_m = 0
    for z in range(0, len(predict m list)-1):
        square = (actual_m_list[z+1] - predict_m_list[z])**2
        sum_sq_m += square
    msqrterr_m = math.sqrt(sum_sq_m)
    print("Root Mean Squared Error for Sensor 13: "+str(msqrterr_m))

    sum_sq_n = 0
    for z in range(0, len(predict_n_list)-1):
        square = (actual_n_list[z+1] - predict_n_list[z])**2
        sum_sq_n += square
    msqrterr_n = math.sqrt(sum_sq_n)
    print("Root Mean Squared Error for Sensor 14: "+str(msqrterr_n))

    rmse_tot = msqrterr_a+msqrterr_b+msqrterr_c+msqrterr_d+msqrterr_e+msqrterr_f+msqrterr_g+msqrterr_h+msqrterr_i+msqrterr_j+msqrterr_k+msqrterr
    print("\nTotal Root Mean Squared Error: "+str(rmse_tot))

    exit()

    def quitting():
        tb.stop()
        tb.wait()
    qapp.connect(qapp, Qt.SIGNAL("aboutToQuit()"), quitting)
    qapp.exec_()


if __name__ == '__main__':
    main()
```

Figure A.3: Smart Weighted Round-Robin Algorithm for Base Station Radio

```python
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
##################################################
# GNU Radio Python Flow Graph
# Title: Chat2
# Generated: Thu Feb 10 09:52:14 2022
##################################################

if __name__ == '__main__':
    import ctypes
    import sys
    if sys.platform.startswith('linux'):
        try:
            x11 = ctypes.cdll.LoadLibrary('libX11.so')
            x11.XInitThreads()
        except:
            print "Warning: failed to XInitThreads()"

import ...


class chat2(gr.top_block, Qt.QWidget):

    def __init__(self):
        gr.top_block.__init__(self, "Chat2")
        Qt.QWidget.__init__(self)
        self.setWindowTitle("Chat2")
        qtgui.util.check_set_qss()
        try:
            self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
        except:
            pass
        self.top_scroll_layout = Qt.QVBoxLayout()
        self.setLayout(self.top_scroll_layout)
        self.top_scroll = Qt.QScrollArea()
        self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
        self.top_scroll_layout.addWidget(self.top_scroll)
        self.top_scroll.setWidgetResizable(True)
        self.top_widget = Qt.QWidget()
        self.top_scroll.setWidget(self.top_widget)
        self.top_layout = Qt.QVBoxLayout(self.top_widget)
        self.top_grid_layout = Qt.QGridLayout()
        self.top_layout.addLayout(self.top_grid_layout)

        self.settings = Qt.QSettings("GNU Radio", "chat2")
        self.restoreGeometry(self.settings.value("geometry").toByteArray())


        ##################################################
        # Variables
        ##################################################
        self.samp_rate = samp_rate = 400000

        ##################################################
        # Blocks
        ##################################################
        self.uhd_usrp_source_0 = uhd.usrp_source(
            ",".join(("", "")),
            uhd.stream_args(
                cpu_format="fc32",
                channels=range(1),
            ),
        )
        self.uhd_usrp_source_0.set_samp_rate(samp_rate)
        self.uhd_usrp_source_0.set_center_freq(1000000000, 0)
        self.uhd_usrp_source_0.set_gain(30, 0)
        self.uhd_usrp_source_0.set_antenna('TX/RX', 0)
        self.qtgui_freq_sink_x_0 = qtgui.freq_sink_c(
            1024, #size
            firdes.WIN_BLACKMAN_hARRIS, #wintype
            0, #fc
            samp_rate, #bw
            "", #name
            1 #number of inputs
```

(Image continued from previous page)

```
        )
        self.qtgui_freq_sink_x_0.set_update_time(0.10)
        self.qtgui_freq_sink_x_0.set_y_axis(-140, 10)
        self.qtgui_freq_sink_x_0.set_y_label('Relative Gain', 'dB')
        self.qtgui_freq_sink_x_0.set_trigger_mode(qtgui.TRIG_MODE_FREE, 0.0, 0, "")
        self.qtgui_freq_sink_x_0.enable_autoscale(False)
        self.qtgui_freq_sink_x_0.enable_grid(False)
        self.qtgui_freq_sink_x_0.set_fft_average(1.0)
        self.qtgui_freq_sink_x_0.enable_axis_labels(True)
        self.qtgui_freq_sink_x_0.enable_control_panel(False)

        if not True:
          self.qtgui_freq_sink_x_0.disable_legend()

        if "complex" == "float" or "complex" == "msg_float":
          self.qtgui_freq_sink_x_0.set_plot_pos_half(not True)

        labels = ['', '', '', '', '',
                  '', '', '', '', '']
        widths = [1, 1, 1, 1, 1,
                  1, 1, 1, 1, 1]
        colors = ["blue", "red", "green", "black", "cyan",
                  "magenta", "yellow", "dark red", "dark green", "dark blue"]
        alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
                  1.0, 1.0, 1.0, 1.0, 1.0]
        for i in xrange(1):
            if len(labels[i]) == 0:
                self.qtgui_freq_sink_x_0.set_line_label(i, "Data {0}".format(i))
            else:
                self.qtgui_freq_sink_x_0.set_line_label(i, labels[i])
            self.qtgui_freq_sink_x_0.set_line_width(i, widths[i])
            self.qtgui_freq_sink_x_0.set_line_color(i, colors[i])
            self.qtgui_freq_sink_x_0.set_line_alpha(i, alphas[i])

        self._qtgui_freq_sink_x_0_win = sip.wrapinstance(self.qtgui_freq_sink_x_0.pyqwidget(), Qt.QWidget)
        self.top_grid_layout.addWidget(self._qtgui_freq_sink_x_0_win)
        self.low_pass_filter_0 = filter.fir_filter_ccf(1, firdes.low_pass(
            1, samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))
        self.digital_gmsk_demod_1 = digital.gmsk_demod(
            samples_per_symbol=2,
            gain_mu=0.175,
            mu=0.5,
            omega_relative_limit=0.005,
            freq_error=0.0,
            verbose=False,
            log=False,
        )
        self.blocks_multiply_const_vxx_1 = blocks.multiply_const_vcc((1, ))
        self.blocks_file_sink_1 = blocks.file_sink(gr.sizeof_char*1, '/home/b/chat_test1.txt', False)
        self.blocks_file_sink_1.set_unbuffered(False)
        self.blks2_packet_decoder_1 = grc_blks2.packet_demod_b(grc_blks2.packet_decoder(
                access_code='',
                threshold=-1,
                callback=lambda ok, payload: self.blks2_packet_decoder_1.recv_pkt(ok, payload),
            ),
        )



        ##################################################
        # Connections
        ##################################################
        self.connect((self.blks2_packet_decoder_1, 0), (self.blocks_file_sink_1, 0))
        self.connect((self.blocks_multiply_const_vxx_1, 0), (self.low_pass_filter_0, 0))
        self.connect((self.digital_gmsk_demod_1, 0), (self.blks2_packet_decoder_1, 0))
        self.connect((self.low_pass_filter_0, 0), (self.digital_gmsk_demod_1, 0))
        self.connect((self.low_pass_filter_0, 0), (self.qtgui_freq_sink_x_0, 0))
        self.connect((self.uhd_usrp_source_0, 0), (self.blocks_multiply_const_vxx_1, 0))

    def closeEvent(self, event):
        self.settings = Qt.QSettings("GNU Radio", "chat2")
        self.settings.setValue("geometry", self.saveGeometry())
        event.accept()

```

(Image continued from previous page)

```
    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.uhd_usrp_source_0.set_samp_rate(self.samp_rate)
        self.qtgui_freq_sink_x_0.set_frequency_range(0, self.samp_rate)
        self.low_pass_filter_0.set_taps(firdes.low_pass(1, self.samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))


class text_tx(gr.top_block):

    def __init__(self):
        gr.top_block.__init__(self, "Text Tx")

        ##################################################
        # Variables
        ##################################################
        self.samp_rate = samp_rate = 400000
        source = ''

        ##################################################
        # Blocks
        ##################################################
        self.uhd_usrp_sink_0 = uhd.usrp_sink(
            ",".join(("", "")),
            uhd.stream_args(
                cpu_format="fc32",
                channels=range(1),
            ),
        )
        self.uhd_usrp_sink_0.set_samp_rate(samp_rate)
        self.uhd_usrp_sink_0.set_time_now(uhd.time_spec(time.time()), uhd.ALL_MBOARDS)
        self.uhd_usrp_sink_0.set_center_freq(1000000000, 0)
        self.uhd_usrp_sink_0.set_gain(40, 0)
        self.uhd_usrp_sink_0.set_antenna('TX/RX', 0)
        self.digital_gmsk_mod_0 = digital.gmsk_mod(
            samples_per_symbol=2,
            bt=0.35,
            verbose=False,
            log=False,
        )
        self.blocks_throttle_0 = blocks.throttle(gr.sizeof_char*1, samp_rate,True)
        self.blocks_multiply_const_vxx_0 = blocks.multiply_const_vcc((1, ))
        self.blocks_head_0 = blocks.head(gr.sizeof_char*1, samp_rate/4)

        send = open('/home/b/tx.txt', 'w')
        if switch == 'a':
            send.write(data0[a1])
            print(data0[a1])
        elif switch == 'b':
            send.write(data1[b1])
            print(data1[b1])
        elif switch == 'c':
            send.write(data2[c1])
            print(data2[c1])
        elif switch == 'd':
            send.write(data3[d1])
            print(data3[d1])
        elif switch == 'e':
            send.write(data4[e1])
            print(data4[e1])
        elif switch == 'f':
            send.write(data5[f1])
            print(data5[f1])
        elif switch == 'g':
            send.write(data6[g1])
            print(data6[g1])
        elif switch == 'h':
            send.write(data7[h1])
            print(data7[h1])
        elif switch == 'i':
            send.write(data8[i1])
            print(data8[i1])
```

(Image continued from previous page)

```
        elif switch == 'j':
            send.write(data9[j1])
            print(data9[j1])
        elif switch == 'k':
            send.write(data10[k1])
            print(data10[k1])
        elif switch == 'l':
            send.write(data11[l1])
            print(data11[l1])
        elif switch == 'm':
            send.write(data12[m1])
            print(data12[m1])
        elif switch == 'n':
            send.write(data13[n1])
            print(data13[n1])
        elif switch == 'eof':
            send.write("eof\n")
            print "eof"
        send.close()

        self.blocks_file_source_0 = blocks.file_source(gr.sizeof_char*1, '/home/b/tx.txt', True)

        self.blocks_file_source_0.set_begin_tag(pmt.PMT_NIL)
        self.blks2_packet_encoder_0 = grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
                samples_per_symbol=2,
                bits_per_symbol=1,
                preamble='',
                access_code='',
                pad_for_usrp=True,
            ),
            payload_length=0,
        )



        ##################################################
        # Connections
        ##################################################
        self.connect((self.blks2_packet_encoder_0, 0), (self.digital_gmsk_mod_0, 0))
        self.connect((self.blocks_file_source_0, 0), (self.blocks_throttle_0, 0))
        self.connect((self.blocks_head_0, 0), (self.blks2_packet_encoder_0, 0))
        self.connect((self.blocks_multiply_const_vxx_0, 0), (self.uhd_usrp_sink_0, 0))
        self.connect((self.blocks_throttle_0, 0), (self.blocks_head_0, 0))
        self.connect((self.digital_gmsk_mod_0, 0), (self.blocks_multiply_const_vxx_0, 0))

    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.uhd_usrp_sink_0.set_samp_rate(self.samp_rate)
        self.blocks_throttle_0.set_sample_rate(self.samp_rate)
        self.blocks_head_0.set_length(self.samp_rate/4)


class WRRScheduler():

    cw = 0
    i = -1
    data_set = []
    max_s = None
    gcd_s = None
    len_s = None
    counter = {}

    def __init__(self, s = None):
        self._init_dataset(s)

    def _init_dataset(self, s):
        self.data_set = s
        self.max_s = max(s, key=lambda x: x[1])[1]
        self.gcd_s = reduce(fractions.gcd, [ weight for data, weight in s])
        self.len_s = len(s)

```

(Image continued from previous page)

```
    def schedule(self):
        while True:
            self.i = (self.i + 1) % self.len_s
            if self.i == 0:
                self.cw = self.cw - self.gcd_s
                if self.cw <= 0:
                    self.cw = self.max_s
                    if self.cw == 0:
                        return None
            if self.data_set[self.i][1] >= self.cw:
                self._inc_counter(self.data_set[self.i])
                return self.data_set[self.i]

    def _inc_counter(self, item):
        try:
            self.counter[item[0]] += 1
        except KeyError:
            self.counter[item[0]] = 1

    def set_data(self, s):
        self.reset()
        self._init_dataset(s)

    def reset_counter(self):
        self.counter = {}

    def reset(self):
        self.cw = 0
        self.i = -1
        self.data_set = []
        self.max_s = None
        self.gcd_s = None
        self.len_s = None
        self.reset_counter()

    def get_next(self, n = 1):
        if n > 1:
            return [ self.schedule() for i in range(0,n) ]
        return self.schedule()


class ack2(gr.top_block, Qt.QWidget):

    def __init__(self):
        gr.top_block.__init__(self, "Chat2")
        Qt.QWidget.__init__(self)
        self.setWindowTitle("Chat2")
        qtgui.util.check_set_qss()
        try:
            self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
        except:
            pass
        self.top_scroll_layout = Qt.QVBoxLayout()
        self.setLayout(self.top_scroll_layout)
        self.top_scroll = Qt.QScrollArea()
        self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
        self.top_scroll_layout.addWidget(self.top_scroll)
        self.top_scroll.setWidgetResizable(True)
        self.top_widget = Qt.QWidget()
        self.top_scroll.setWidget(self.top_widget)
        self.top_layout = Qt.QVBoxLayout(self.top_widget)
        self.top_grid_layout = Qt.QGridLayout()
        self.top_layout.addLayout(self.top_grid_layout)

        self.settings = Qt.QSettings("GNU Radio", "chat2")
        self.restoreGeometry(self.settings.value("geometry").toByteArray())


        ##################################################
        # Variables
        ##################################################
        self.samp_rate = samp_rate = 400000

        ##################################################
```

(Image continued from previous page)

```
        # Blocks
        ##################################################
        self.uhd_usrp_source_0 = uhd.usrp_source(
            ",".join(("", "")),
            uhd.stream_args(
                cpu_format="fc32",
                channels=range(1),
            ),
        )
        self.uhd_usrp_source_0.set_samp_rate(samp_rate)
        self.uhd_usrp_source_0.set_center_freq(1000000000, 0)
        self.uhd_usrp_source_0.set_gain(30, 0)
        self.uhd_usrp_source_0.set_antenna('TX/RX', 0)
        self.qtgui_freq_sink_x_0 = qtgui.freq_sink_c(
            1024, #size
            firdes.WIN_BLACKMAN_hARRIS, #wintype
            0, #fc
            samp_rate, #bw
            "", #name
            1 #number of inputs
        )
        self.qtgui_freq_sink_x_0.set_update_time(0.10)
        self.qtgui_freq_sink_x_0.set_y_axis(-140, 10)
        self.qtgui_freq_sink_x_0.set_y_label('Relative Gain', 'dB')
        self.qtgui_freq_sink_x_0.set_trigger_mode(qtgui.TRIG_MODE_FREE, 0.0, 0, "")
        self.qtgui_freq_sink_x_0.enable_autoscale(False)
        self.qtgui_freq_sink_x_0.enable_grid(False)
        self.qtgui_freq_sink_x_0.set_fft_average(1.0)
        self.qtgui_freq_sink_x_0.enable_axis_labels(True)
        self.qtgui_freq_sink_x_0.enable_control_panel(False)

        if not True:
          self.qtgui_freq_sink_x_0.disable_legend()

        if "complex" == "float" or "complex" == "msg_float":
          self.qtgui_freq_sink_x_0.set_plot_pos_half(not True)

        labels = ['', '', '', '', '',
                  '', '', '', '', '']
        widths = [1, 1, 1, 1, 1,
                  1, 1, 1, 1, 1]
        colors = ["blue", "red", "green", "black", "cyan",
                  "magenta", "yellow", "dark red", "dark green", "dark blue"]
        alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
                  1.0, 1.0, 1.0, 1.0, 1.0]
        for i in xrange(1):
            if len(labels[i]) == 0:
                self.qtgui_freq_sink_x_0.set_line_label(i, "Data {0}".format(i))
            else:
                self.qtgui_freq_sink_x_0.set_line_label(i, labels[i])
            self.qtgui_freq_sink_x_0.set_line_width(i, widths[i])
            self.qtgui_freq_sink_x_0.set_line_color(i, colors[i])
            self.qtgui_freq_sink_x_0.set_line_alpha(i, alphas[i])

        self._qtgui_freq_sink_x_0_win = sip.wrapinstance(self.qtgui_freq_sink_x_0.pyqwidget(), Qt.QWidget)
        self.top_grid_layout.addWidget(self._qtgui_freq_sink_x_0_win)
        self.low_pass_filter_0 = filter.fir_filter_ccf(1, firdes.low_pass(
            1, samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))
        self.digital_gmsk_demod_1 = digital.gmsk_demod(
            samples_per_symbol=2,
            gain_mu=0.175,
            mu=0.5,
            omega_relative_limit=0.005,
            freq_error=0.0,
            verbose=False,
            log=False,
        )
        self.blocks_multiply_const_vxx_1 = blocks.multiply_const_vcc((1, ))
        self.blocks_file_sink_1 = blocks.file_sink(gr.sizeof_char*1, '/home/b/chat_test2.txt', False)
        self.blocks_file_sink_1.set_unbuffered(False)
        self.blks2_packet_decoder_1 = grc_blks2.packet_demod_b(grc_blks2.packet_decoder(
                access_code='',
                threshold=-1,
                callback=lambda ok, payload: self.blks2_packet_decoder_1.recv_pkt(ok, payload),
```

(Image continued from previous page)

```
            ),
        )



        ##################################################
        # Connections
        ##################################################
        self.connect((self.blks2_packet_decoder_1, 0), (self.blocks_file_sink_1, 0))
        self.connect((self.blocks_multiply_const_vxx_1, 0), (self.low_pass_filter_0, 0))
        self.connect((self.digital_gmsk_demod_1, 0), (self.blks2_packet_decoder_1, 0))
        self.connect((self.low_pass_filter_0, 0), (self.digital_gmsk_demod_1, 0))
        self.connect((self.low_pass_filter_0, 0), (self.qtgui_freq_sink_x_0, 0))
        self.connect((self.uhd_usrp_source_0, 0), (self.blocks_multiply_const_vxx_1, 0))

    def closeEvent(self, event):
        self.settings = Qt.QSettings("GNU Radio", "chat2")
        self.settings.setValue("geometry", self.saveGeometry())
        event.accept()

    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.uhd_usrp_source_0.set_samp_rate(self.samp_rate)
        self.qtgui_freq_sink_x_0.set_frequency_range(0, self.samp_rate)
        self.low_pass_filter_0.set_taps(firdes.low_pass(1, self.samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))


def main(options=None):

    from distutils.version import StrictVersion
    if StrictVersion(Qt.qVersion()) >= StrictVersion("4.5.0"):
        style = gr.prefs().get_string('qtgui', 'style', 'raster')
        Qt.QApplication.setGraphicsSystem(style)
    qapp = Qt.QApplication(sys.argv)

    data_csv = open('/home/b/20161005_140846.csv', 'r')
    csvreader = csv.reader(data_csv)
    header = next(csvreader)
    data_rows = []
    for row in csvreader:
        data_rows.append(row)
    data_csv.close()

    for x in range(0,14):
        out = open('file' + str(x+1) + '.txt', 'w')
        for y in range(0, 100):
            out.write(data_rows[y][x+6]+' '+str(x+1)+'0'+str(y+1)+'\n')
        out.close()

    tb = chat2()
    tb.start()
    print "Ready to start..."
    while True:
        f = open('/home/b/chat_test1.txt', 'r')
        f.flush()
        f.readline()
        check = f.readline()
        if (check != ""):
            print "START"
            time.sleep(1)
            break
    tb.stop()

    weight_file = open('/home/b/chat_test1.txt', 'r')
    weight_file.flush()
    weight_file.readline()
    weights = weight_file.readline()
    w_list = weights.split(",")
    w_list.pop()
    for m in range(0, len(w_list)):
        w_list[m] = int(w_list[m])
```

(Image continued from previous page)

```
538
539     sensors = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N']
540
541     global i, a1, b1, c1, d1, e1, f1, g1, h1, i1, j1, k1, l1, m1, n1
542     i = 0
543     a1 = 0
544     b1 = 0
545     c1 = 0
546     d1 = 0
547     e1 = 0
548     f1 = 0
549     g1 = 0
550     h1 = 0
551     i1 = 0
552     j1 = 0
553     k1 = 0
554     l1 = 0
555     m1 = 0
556     n1 = 0
557
558     global data0, data1, data2, data3, data4, data5, data6, data7, data8, data9, data10, data11, data12, data13
559
560     read1 = open('/home/b/file1.txt', 'r')
561     data0 = read1.readlines()
562     length1 = len(data0)
563     read1.close()
564
565     read2 = open('/home/b/file2.txt', 'r')
566     data1 = read2.readlines()
567     length2 = len(data1)
568     read2.close()
569
570     read3 = open('/home/b/file3.txt', 'r')
571     data2 = read3.readlines()
572     length3 = len(data2)
573     read3.close()
574
575     read4 = open('/home/b/file4.txt', 'r')
576     data3 = read4.readlines()
577     length4 = len(data3)
578     read4.close()
579
580     read5 = open('/home/b/file5.txt', 'r')
581     data4 = read5.readlines()
582     length5 = len(data4)
583     read5.close()
584
585     read6 = open('/home/b/file6.txt', 'r')
586     data5 = read6.readlines()
587     length6 = len(data5)
588     read6.close()
589
590     read7 = open('/home/b/file7.txt', 'r')
591     data6 = read7.readlines()
592     length7 = len(data6)
593     read7.close()
594
595     read8 = open('/home/b/file8.txt', 'r')
596     data7 = read8.readlines()
597     length8 = len(data7)
598     read8.close()
599
600     read9 = open('/home/b/file9.txt', 'r')
601     data8 = read9.readlines()
602     length9 = len(data8)
603     read9.close()
604
605     read10 = open('/home/b/file10.txt', 'r')
606     data9 = read10.readlines()
607     length10 = len(data9)
608     read10.close()
609
610     read11 = open('/home/b/file11.txt', 'r')
611     data10 = read11.readlines()
```

(Image continued from previous page)

```
    length11 = len(data10)
    read11.close()

    read12 = open('/home/b/file12.txt', 'r')
    data11 = read12.readlines()
    length12 = len(data11)
    read12.close()

    read13 = open('/home/b/file13.txt', 'r')
    data12 = read13.readlines()
    length13 = len(data12)
    read13.close()

    read14 = open('/home/b/file14.txt', 'r')
    data13 = read14.readlines()
    length14 = len(data13)
    read14.close()

    data = list(zip(sensors, w_list))
    sched = WRRScheduler(data)

    global a_tot, b_tot, c_tot, d_tot, e_tot, f_tot, g_tot, h_tot, i_tot, j_tot, k_tot, l_tot, m_tot, n_tot, switch
    a_tot = 0
    b_tot = 0
    c_tot = 0
    d_tot = 0
    e_tot = 0
    f_tot = 0
    g_tot = 0
    h_tot = 0
    i_tot = 0
    j_tot = 0
    k_tot = 0
    l_tot = 0
    m_tot = 0
    n_tot = 0

    packet_num = 0
    iterations = 150
    result = []
    while (packet_num < iterations):
        choose = sched.get_next()
        if choose[0] == 'A':
            a_tot += 1
            if (a_tot <= length1):
                result.append(choose[0])
                packet_num += 1
        elif choose[0] == 'B':
            b_tot += 1
            if (b_tot <= length2):
                result.append(choose[0])
                packet_num += 1
        elif choose[0] == 'C':
            c_tot += 1
            if (c_tot <= length3):
                result.append(choose[0])
                packet_num += 1
        elif choose[0] == 'D':
            d_tot += 1
            if (d_tot <= length4):
                result.append(choose[0])
                packet_num += 1
        elif choose[0] == 'E':
            e_tot += 1
            if (e_tot <= length5):
                result.append(choose[0])
                packet_num += 1
        elif choose[0] == 'F':
            f_tot += 1
            if (f_tot <= length6):
                result.append(choose[0])
                packet_num += 1
        elif choose[0] == 'G':
            g_tot += 1
```

(Image continued from previous page)

```
            if (g_tot <= length7):
                result.append(choose[0])
                packet_num += 1
        elif choose[0] == 'H':
            h_tot += 1
            if (h_tot <= length8):
                result.append(choose[0])
                packet_num += 1
        elif choose[0] == 'I':
            i_tot += 1
            if (i_tot <= length9):
                result.append(choose[0])
                packet_num += 1
        elif choose[0] == 'J':
            j_tot += 1
            if (j_tot <= length10):
                result.append(choose[0])
                packet_num += 1
        elif choose[0] == 'K':
            k_tot += 1
            if (k_tot <= length11):
                result.append(choose[0])
                packet_num += 1
        elif choose[0] == 'L':
            l_tot += 1
            if (l_tot <= length12):
                result.append(choose[0])
                packet_num += 1
        elif choose[0] == 'M':
            m_tot += 1
            if (m_tot <= length13):
                result.append(choose[0])
                packet_num += 1
        elif choose[0] == 'N':
            n_tot += 1
            if (n_tot <= length14):
                result.append(choose[0])
                packet_num += 1

    while i < iterations:
        if result[i] == 'A':
            switch = 'a'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            a1 += 1
        elif result[i] == 'B':
            switch = 'b'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            b1 += 1
        elif result[i] == 'C':
            switch = 'c'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            c1 += 1
        elif result[i] == 'D':
            switch = 'd'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            d1 += 1
        elif result[i] == 'E':
            switch = 'e'
```

(Image continued from previous page)

```
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            e1 += 1
        elif result[i] == 'F':
            switch = 'f'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            f1 += 1
        elif result[i] == 'G':
            switch = 'g'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            g1 += 1
        elif result[i] == 'H':
            switch = 'h'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            h1 += 1
        elif result[i] == 'I':
            switch = 'i'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            i1 += 1
        elif result[i] == 'J':
            switch = 'j'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            j1 += 1
        elif result[i] == 'K':
            switch = 'k'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            k1 += 1
        elif result[i] == 'L':
            switch = 'l'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            l1 += 1
        elif result[i] == 'M':
            switch = 'm'
            tb = text_tx()
            tb.start()
            time.sleep(1)
            tb.stop()
            i += 1
            m1 += 1
        elif result[i] == 'N':
            switch = 'n'
            tb = text_tx()
            tb.start()
```

(Image continued from previous page)

```
            time.sleep(1)
            tb.stop()
            i += 1
            n1 += 1
        else:
            print "Error: Empty Result"
        tb = ack2()
        tb.start()
        while True:
            ack_file = open('/home/b/chat_test2.txt', 'r')
            ack_file.flush()
            ack_file.readline()
            ack0 = ack_file.readline()
            if (ack0 == 'ACK\n'):
                print "RECEIVING ACK"
                time.sleep(1)
                break
            ack_file.close()
            time.sleep(1)
        tb.stop()
        if (i > 28) and (i % 20 == 0):
            tb = chat2()
            tb.start()
            while True:
                f = open('/home/b/chat_test1.txt', 'r')
                f.flush()
                f.readline()
                check = f.readline()
                if (check != ""):
                    time.sleep(1)
                    break
            tb.stop()
            weight_file = open('/home/b/chat_test1.txt', 'r')
            weight_file.flush()
            weight_file.readline()
            weights = weight_file.readline()
            w_list = weights.split(",")
            w_list.pop()
            for m in range(0, len(w_list)):
                w_list[m] = int(w_list[m])
            data = list(zip(sensors, w_list))
            sched = WRRScheduler(data)
            count = 0
            new_result = []
        while (count < (iterations-i)):
            new_choose = sched.get_next()
        if new_choose[0] == 'A':
            a_tot += 1
            if (a_tot <= length1):
                new_result.append(new_choose[0])
                count += 1
        elif new_choose[0] == 'B':
            b_tot += 1
            if (b_tot <= length2):
                new_result.append(new_choose[0])
                count += 1
        elif new_choose[0] == 'C':
            c_tot += 1
            if (c_tot <= length3):
                new_result.append(new_choose[0])
                count += 1
        elif new_choose[0] == 'D':
            d_tot += 1
            if (d_tot <= length4):
                new_result.append(new_choose[0])
                count += 1
        elif new_choose[0] == 'E':
            e_tot += 1
            if (e_tot <= length5):
                new_result.append(new_choose[0])
                count += 1
        elif new_choose[0] == 'F':
            f_tot += 1
            if (f_tot <= length6):
```

(Image continued from previous page)

```
                new_result.append(new_choose[0])
                count += 1
        elif new_choose[0] == 'G':
            g_tot += 1
            if (g_tot <= length7):
                new_result.append(new_choose[0])
                count += 1
        elif new_choose[0] == 'H':
            h_tot += 1
            if (h_tot <= length8):
                new_result.append(new_choose[0])
                count += 1
        elif new_choose[0] == 'I':
            i_tot += 1
            if (i_tot <= length9):
                new_result.append(new_choose[0])
                count += 1
        elif new_choose[0] == 'J':
            j_tot += 1
            if (j_tot <= length10):
                new_result.append(new_choose[0])
                count += 1
        elif new_choose[0] == 'K':
            k_tot += 1
            if (k_tot <= length11):
                new_result.append(new_choose[0])
                count += 1
        elif new_choose[0] == 'L':
            l_tot += 1
            if (l_tot <= length12):
                new_result.append(new_choose[0])
                count += 1
        elif new_choose[0] == 'M':
            m_tot += 1
            if (m_tot <= length13):
                new_result.append(new_choose[0])
                count += 1
        elif new_choose[0] == 'N':
            n_tot += 1
            if (n_tot <= length14):
                new_result.append(new_choose[0])
                count += 1
            if (len(new_result) < iterations):
                for v in range(iterations-len(new_result)):
                    new_result.insert(0,0)
            result = new_result
            choose = new_choose

    switch = 'eof'
    tb = text_tx()
    tb.start()
    time.sleep(1)
    tb.stop()

    print('done')
    exit()

    def quitting():
        tb.stop()
        tb.wait()
    qapp.connect(qapp, Qt.SIGNAL("aboutToQuit()"), quitting)
    qapp.exec_()


if __name__ == '__main__':
    main()
```

Figure A.4: Smart Weighted Round-Robin Algorithm for Field Sensor Radio