

**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

Copyright © 2022. Austin M. Keith.

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

	Page
ABSTRACT.....	1
DEDICATION.....	3
ACKNOWLEDGEMENTS.....	4
NOMENCLATURE.....	5
CHAPTERS	
1. INTRODUCTION.....	6
1.1 Traditional Networking.....	6
1.2 Software-Defined Networking.....	8
1.3 Software-Defined Radio.....	10
1.4 Complete System.....	12
2. METHODS.....	13
2.1 SDN Construction.....	13
2.2 SDR Communication.....	15
3. RESULTS.....	24
3.1 Operation of the Complete System.....	24
3.2 Effectiveness of Algorithms.....	25
4. CONCLUSION.....	29
4.1 Effectiveness of SDN-SDR Network.....	29
4.2 Real-World Applications.....	30
REFERENCES.....	33
APPENDIX A: PYTHON SOFTWARE WRITTEN FOR COMMUNICATION.....	35

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.

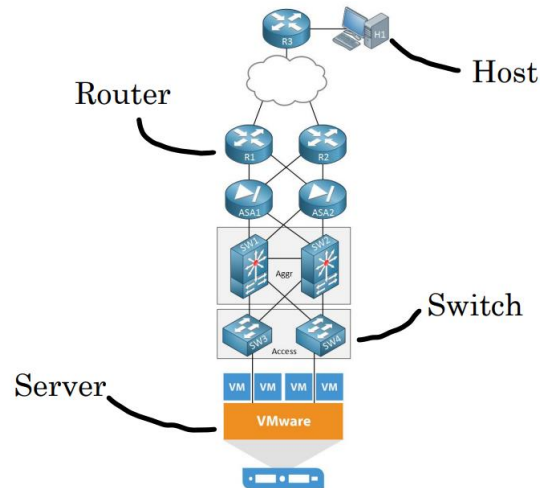


Figure 1.1: Network Components

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.

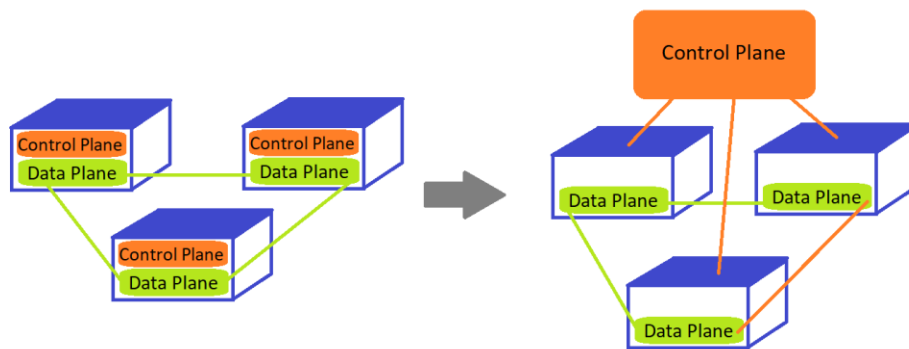


Figure 1.2: Control Plane Separation

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.

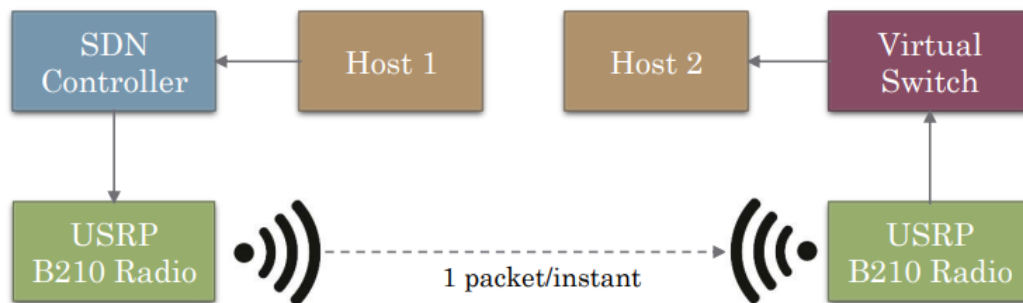


Figure 1.4: Complete Architecture of System

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.

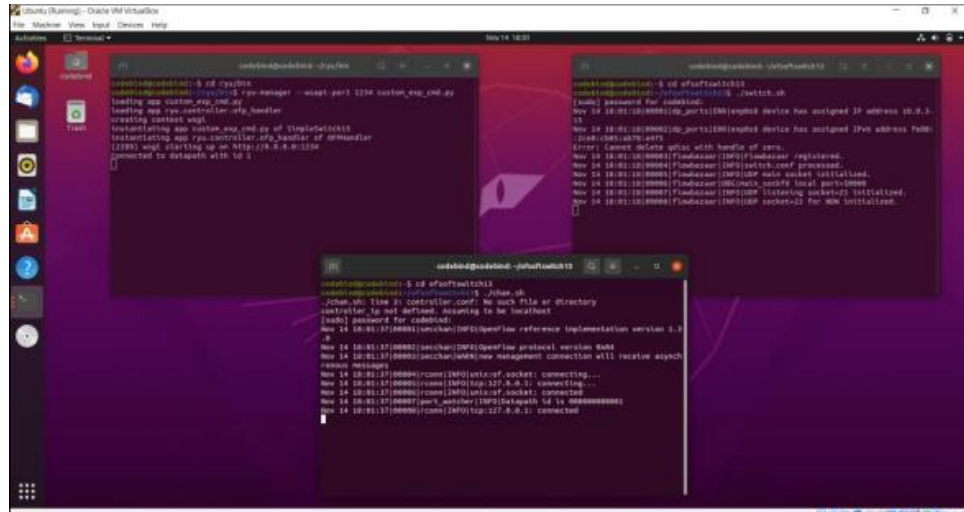


Figure 2.1: Terminal Interface for Switch and Controller

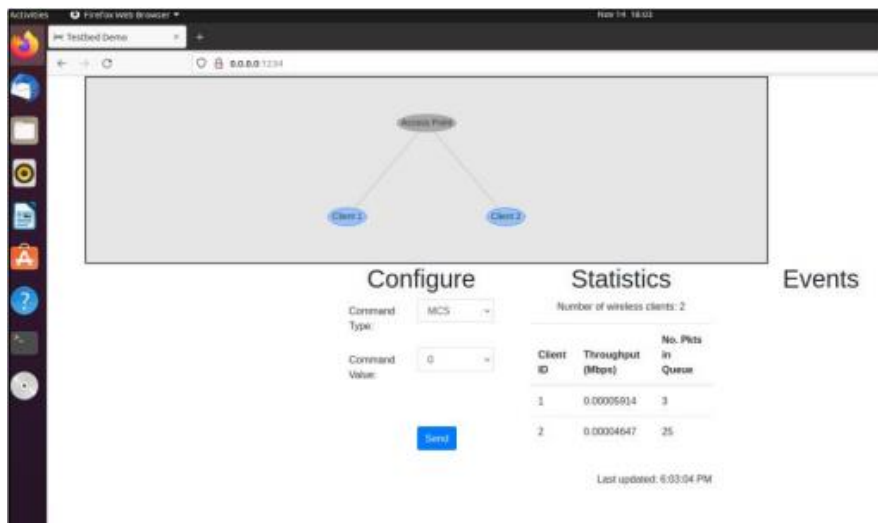


Figure 2.2: GUI Interface for SDN Controller

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.

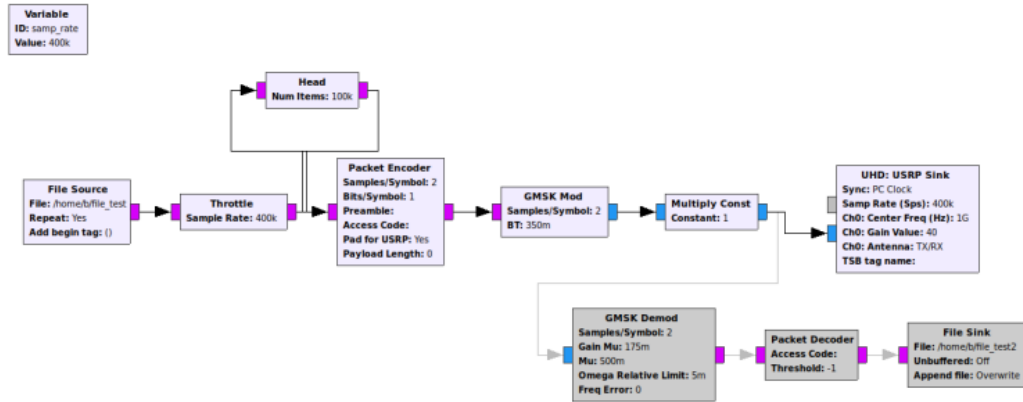


Figure 2.3: Transmission Block Diagram for One-Way Communication

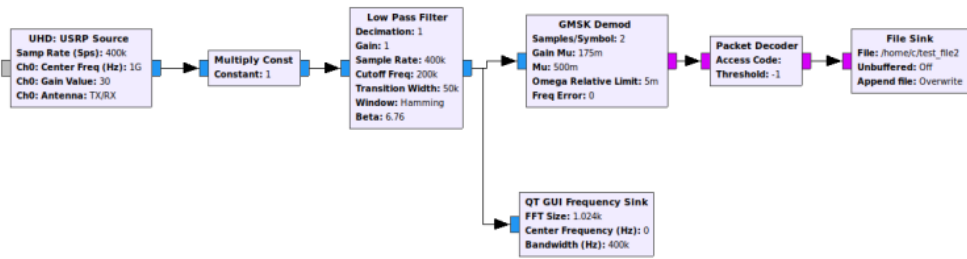


Figure 2.4: Reception Block Diagram for One-Way Communication

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.

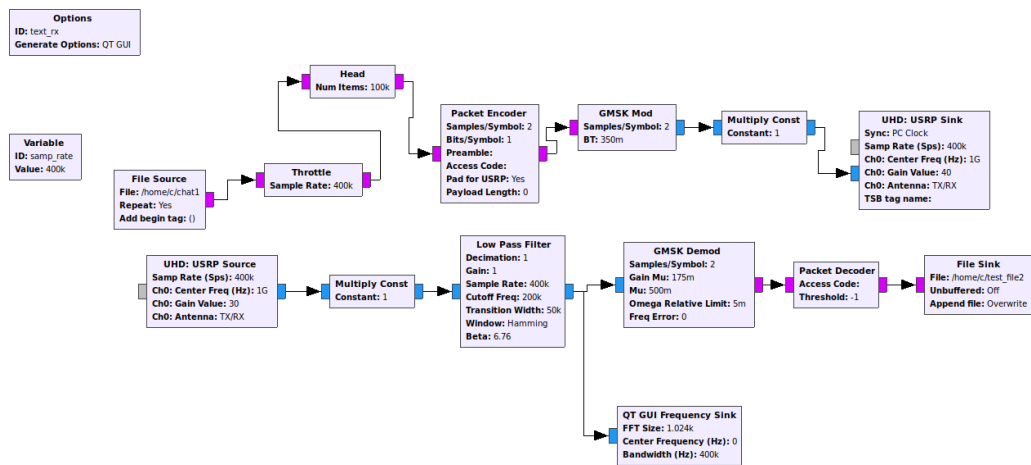


Figure 2.5: Two-Way Communication on Transmit Side

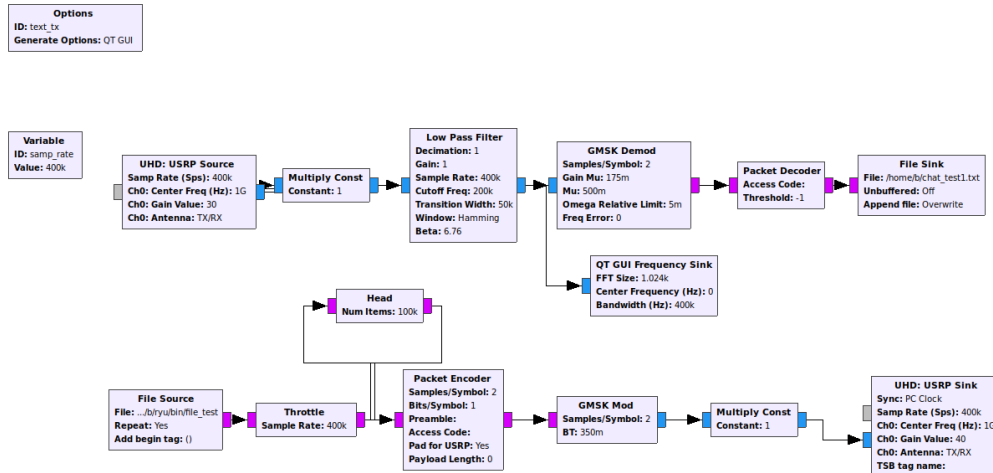


Figure 2.6: Two-Way Communication on Receive Side

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

```
1  #!/usr/bin/env python2
2  # -*- coding: utf-8 -*-
3  #####
4  # GNU Radio Python Flow Graph
5  # Title: Chat
6  # Generated: Thu Feb 10 09:30:28 2022
7  #####
8
9  if __name__ == '__main__':
10     import ctypes
11     import sys
12     if sys.platform.startswith('linux'):
13         try:
14             x11 = ctypes.cdll.LoadLibrary('libX11.so')
15             x11.XInitThreads()
16         except:
17             print "Warning: failed to XInitThreads()"
18
19     import ...
20
21 class chat(gr.top_block, Qt.QWidget):
22
23     def __init__(self):
24         gr.top_block.__init__(self, "Chat")
25         Qt.QWidget.__init__(self)
26         self.setWindowTitle("Chat")
27         qtgui.util.check_set_qps()
28         try:
29             self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-gre'))
30         except:
31             pass
32         self.top_scroll_layout = Qt.QVBoxLayout()
33         self.setLayout(self.top_scroll_layout)
34         self.top_scroll = Qt.QScrollArea()
35         self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
36         self.top_scroll_layout.addWidget(self.top_scroll)
37         self.top_scroll.setWidgetResizable(True)
38         self.top_widget = Qt.QWidget()
39         self.top_scroll.setWidget(self.top_widget)
40         self.top_layout = Qt.QVBoxLayout(self.top_widget)
41         self.top_grid_layout = Qt.QGridLayout()
42         self.top_layout.addLayout(self.top_grid_layout)
43
44         self.settings = Qt.QSettings("GNU Radio", "chat")
45         self.restoreGeometry(self.settings.value("geometry").toByteArray())
46
47         #####
48         # Variables
49         #####
50         self.samp_rate = samp_rate = 400000
51
52         while True:
53             weight1 = raw_input("Enter a value for the weight of sensor 1 (integer): ")
54             try:
55                 int(weight1)
56                 if (int(weight1) < 0):
57                     print "Value cannot be negative. Try again."
58                     continue
59             except ValueError:
60                 print "Value provided needs to be an integer. Try again."
61                 continue
62             break
63
64         while True:
65             weight2 = raw_input("Enter a value for the weight of sensor 2 (integer): ")
66             try:
67                 int(weight2)
68                 if (int(weight2) < 0):
69                     print "Value cannot be negative. Try again."
70                     continue
71             except ValueError:
```

(Image continued from previous page)

```
92         print "Value provided needs to be an integer. Try again."
93         continue
94     break
95
96     while True:
97         weight3 = raw_input("Enter a value for the weight of sensor 3 (integer): ")
98         try:
99             int(weight3)
100             if (int(weight3) < 0):
101                 print "Value cannot be negative. Try again."
102                 continue
103         except ValueError:
104             print "Value provided needs to be an integer. Try again."
105             continue
106     break
107
108     while True:
109         weight4 = raw_input("Enter a value for the weight of sensor 4 (integer): ")
110         try:
111             int(weight4)
112             if (int(weight4) < 0):
113                 print "Value cannot be negative. Try again."
114                 continue
115         except ValueError:
116             print "Value provided needs to be an integer. Try again."
117             continue
118     break
119
120     while True:
121         weight5 = raw_input("Enter a value for the weight of sensor 5 (integer): ")
122         try:
123             int(weight5)
124             if (int(weight5) < 0):
125                 print "Value cannot be negative. Try again."
126                 continue
127         except ValueError:
128             print "Value provided needs to be an integer. Try again."
129             continue
130     break
131
132     while True:
133         weight6 = raw_input("Enter a value for the weight of sensor 6 (integer): ")
134         try:
135             int(weight6)
136             if (int(weight6) < 0):
137                 print "Value cannot be negative. Try again."
138                 continue
139         except ValueError:
140             print "Value provided needs to be an integer. Try again."
141             continue
142     break
143
144     while True:
145         weight7 = raw_input("Enter a value for the weight of sensor 7 (integer): ")
146         try:
147             int(weight7)
148             if (int(weight7) < 0):
149                 print "Value cannot be negative. Try again."
150                 continue
151         except ValueError:
152             print "Value provided needs to be an integer. Try again."
153             continue
154     break
155
156     while True:
157         weight8 = raw_input("Enter a value for the weight of sensor 8 (integer): ")
158         try:
159             int(weight8)
160             if (int(weight8) < 0):
161                 print "Value cannot be negative. Try again."
162                 continue
163         except ValueError:
164             print "Value provided needs to be an integer. Try again."
165         continue
```

(Image continued from previous page)

```
166         break
167
168     while True:
169         weight9 = raw_input("Enter a value for the weight of sensor 9 (integer): ")
170         try:
171             int(weight9)
172             if (int(weight9) < 0):
173                 print "Value cannot be negative. Try again."
174                 continue
175         except ValueError:
176             print "Value provided needs to be an integer. Try again."
177             continue
178         break
179
180     while True:
181         weight10 = raw_input("Enter a value for the weight of sensor 10 (integer): ")
182         try:
183             int(weight10)
184             if (int(weight10) < 0):
185                 print "Value cannot be negative. Try again."
186                 continue
187         except ValueError:
188             print "Value provided needs to be an integer. Try again."
189             continue
190         break
191
192     while True:
193         weight11 = raw_input("Enter a value for the weight of sensor 11 (integer): ")
194         try:
195             int(weight11)
196             if (int(weight11) < 0):
197                 print "Value cannot be negative. Try again."
198                 continue
199         except ValueError:
200             print "Value provided needs to be an integer. Try again."
201             continue
202         break
203
204     while True:
205         weight12 = raw_input("Enter a value for the weight of sensor 12 (integer): ")
206         try:
207             int(weight12)
208             if (int(weight12) < 0):
209                 print "Value cannot be negative. Try again."
210                 continue
211         except ValueError:
212             print "Value provided needs to be an integer. Try again."
213             continue
214         break
215
216     while True:
217         weight13 = raw_input("Enter a value for the weight of sensor 13 (integer): ")
218         try:
219             int(weight13)
220             if (int(weight13) < 0):
221                 print "Value cannot be negative. Try again."
222                 continue
223         except ValueError:
224             print "Value provided needs to be an integer. Try again."
225             continue
226         break
227
228     while True:
229         weight14 = raw_input("Enter a value for the weight of sensor 14 (integer): ")
230         try:
231             int(weight14)
232             if (int(weight14) < 0):
233                 print "Value cannot be negative. Try again."
234                 continue
235         except ValueError:
236             print "Value provided needs to be an integer. Try again."
237             continue
238         break
239
```


(Image continued from previous page)

```
240     weight_file = open('/home/c/chat3', 'w')
241
242     weight_file.write(weight1 + ',')
243     weight_file.write(weight2 + ',')
244     weight_file.write(weight3 + ',')
245     weight_file.write(weight4 + ',')
246     weight_file.write(weight5 + ',')
247     weight_file.write(weight6 + ',')
248     weight_file.write(weight7 + ',')
249     weight_file.write(weight8 + ',')
250     weight_file.write(weight9 + ',')
251     weight_file.write(weight10 + ',')
252     weight_file.write(weight11 + ',')
253     weight_file.write(weight12 + ',')
254     weight_file.write(weight13 + ',')
255     weight_file.write(weight14 + ',\n')
256
257     weight_file.close()
258
259     #####
260     # Blocks
261     #####
262     self.uhd_usrp_sink_1 = uhd.usrp_sink(
263         ", ".join("", ""),
264         uhd.stream_args(
265             cpu_format="fc32",
266             channels=range(1),
267         ),
268     )
269     self.uhd_usrp_sink_1.set_samp_rate(samp_rate)
270     self.uhd_usrp_sink_1.set_time_now(uhd.time_spec(time.time()), uhd.ALL_MBOARDS)
271     self.uhd_usrp_sink_1.set_center_freq(1000000000, 0)
272     self.uhd_usrp_sink_1.set_gain(40, 0)
273     self.uhd_usrp_sink_1.set_antenna('RX2', 0)
274     self.digital_gmsk_mod_0 = digital.gmsk_mod(
275         samples_per_symbol=2,
276         bt=0.35,
277         verbose=False,
278         log=False,
279     )
280     self.blocks_throttle_0 = blocks.throttle(gr.sizeof_char*1, samp_rate,True)
281     self.blocks_multiply_const_vxx_1 = blocks.multiply_const_vcc((1,))
282     self.blocks_head_0 = blocks.head(gr.sizeof_char*1, 100000)
283     self.blocks_file_source_0 = blocks.file_source(gr.sizeof_char*1, '/home/c/chat3', True)
284     self.blocks_file_source_0.set_begin_tag(pmt.PMT_NIL)
285     self.blks2_packet_encoder_0 = grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
286         samples_per_symbol=2,
287         bits_per_symbol=1,
288         preamble='',
289         access_code='',
290         pad_for_usrp=True,
291     ),
292         payload_length=0,
293     )
294
295
296
297     #####
298     # Connections
299     #####
300     self.connect((self.blks2_packet_encoder_0, 0), (self.digital_gmsk_mod_0, 0))
301     self.connect((self.blocks_file_source_0, 0), (self.blocks_throttle_0, 0))
302     self.connect((self.blocks_head_0, 0), (self.blks2_packet_encoder_0, 0))
303     self.connect((self.blocks_multiply_const_vxx_1, 0), (self.uhd_usrp_sink_1, 0))
304     self.connect((self.blocks_throttle_0, 0), (self.blocks_head_0, 0))
305     self.connect((self.digital_gmsk_mod_0, 0), (self.blocks_multiply_const_vxx_1, 0))
306
307     def closeEvent(self, event):
308         self.settings = Qt.QSettings("GNU Radio", "chat")
309         self.settings.setValue("geometry", self.saveGeometry())
310         event.accept()
311
312     def get_samp_rate(self):
313         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-gro'))
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, #vintype
370         0, #fc
371         samp_rate, #bv
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)

```
387
388
389     if "complex" == "float" or "complex" == "msg_float":
390         self.qtgui_freq_sink_x_0.set_plot_pos_half(not True)
391
392     labels = ['', '', '', '', '']
393     widths = [1, 1, 1, 1, 1]
394     colors = ["blue", "red", "green", "black", "cyan",
395             "magenta", "yellow", "dark red", "dark green", "dark blue"]
396     alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
397             1.0, 1.0, 1.0, 1.0, 1.0]
398
399     for i in xrange(1):
400         if len(labels[i]) == 0:
401             self.qtgui_freq_sink_x_0.set_line_label(i, "Data {0}".format(i))
402         else:
403             self.qtgui_freq_sink_x_0.set_line_label(i, labels[i])
404             self.qtgui_freq_sink_x_0.set_line_width(i, widths[i])
405             self.qtgui_freq_sink_x_0.set_line_color(i, colors[i])
406             self.qtgui_freq_sink_x_0.set_line_alpha(i, alphas[i])
407
408     self.qtgui_freq_sink_x_0_win = sip.wrapinstance(self.qtgui_freq_sink_x_0.pyqwidget(), Qt.QWidget)
409     self.top_grid_layout.addWidget(self.qtgui_freq_sink_x_0_win)
410     self.low_pass_filter_0 = filter.fir_filter_cof(1, firdes.low_pass(
411         1, samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))
412     self.digital_gmsk_demod_0 = digital.gmsk_demod(
413         samples_per_symbol=2,
414         gain_mu=0.175,
415         mu=0.5,
416         omega_relative_limit=0.005,
417         freq_error=0.0,
418         verbose=False,
419         log=False,
420     )
421
422     clr_file = open('/home/c/test_file2', 'w')
423     clr_file.close()
424
425     self.blocks_multiply_const_vxx_0 = blocks.multiply_const_vcc((1, ))
426     self.blocks_file_sink_0 = blocks.file_sink(gr.sizeof_char*1, '/home/c/test_file2', False)
427     self.blocks_file_sink_0.set_unbuffered(False)
428     self.blks2_packet_decoder_0 = grc_blks2.packet_demod_b(grc_blks2.packet_decoder(
429         access_code='',
430         threshold=-1,
431         callback=lambda ok, payload: self.blks2_packet_decoder_0.recv_pkt(ok, payload),
432     ),
433     )
434
435
436
437
438
439     #####
440     # Connections
441     #####
442     self.connect((self.blks2_packet_decoder_0, 0), (self.blocks_file_sink_0, 0))
443     self.connect((self.blocks_multiply_const_vxx_0, 0), (self.low_pass_filter_0, 0))
444     self.connect((self.digital_gmsk_demod_0, 0), (self.blks2_packet_decoder_0, 0))
445     self.connect((self.low_pass_filter_0, 0), (self.digital_gmsk_demod_0, 0))
446     self.connect((self.low_pass_filter_0, 0), (self.qtgui_freq_sink_x_0, 0))
447     self.connect((self.uhd_usrp_source_0, 0), (self.blocks_multiply_const_vxx_0, 0))
448
449
450     def closeEvent(self, event):
451         self.settings = Qt.QSettings("GNU Radio", "text_rx")
452         self.settings.setValue("geometry", self.saveGeometry())
453         event.accept()
454
455     def get_samp_rate(self):
456         return self.samp_rate
457
458     def set_samp_rate(self, samp_rate):
459         self.samp_rate = samp_rate
460         self.uhd_usrp_source_0.set_samp_rate(self.samp_rate)
461         self.qtgui_freq_sink_x_0.set_frequency_range(0, self.samp_rate)
```

(Image continued from previous page)

```
461 | self.low_pass_filter_0.set_taps(firdes.low_pass(1, self.samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))
462 |
463 |
464 | class ack(gr.top_block, Qt.QWidget):
465 |
466 |     def __init__(self):
467 |         gr.top_block.__init__(self, "Chat")
468 |         Qt.QWidget.__init__(self)
469 |         self.setWindowTitle("Chat")
470 |         QtGui.util.check_set_qss()
471 |         try:
472 |             self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
473 |         except:
474 |             pass
475 |         self.top_scroll_layout = Qt.QVBoxLayout()
476 |         self.setLayout(self.top_scroll_layout)
477 |         self.top_scroll = Qt.QScrollArea()
478 |         self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
479 |         self.top_scroll_layout.addWidget(self.top_scroll)
480 |         self.top_scroll.setWidgetResizable(True)
481 |         self.top_widget = Qt.QWidget()
482 |         self.top_scroll.setWidget(self.top_widget)
483 |         self.top_layout = Qt.QVBoxLayout(self.top_widget)
484 |         self.top_grid_layout = Qt.QGridLayout()
485 |         self.top_layout.addLayout(self.top_grid_layout)
486 |
487 |         self.settings = Qt.QSettings("GNU Radio", "chat")
488 |         self.restoreGeometry(self.settings.value("geometry").toByteArray())
489 |
490 |
491 |         #####
492 |         # Variables
493 |         #####
494 |         self.samp_rate = samp_rate = 400000
495 |
496 |         weight_file = open('/home/c/chat3', 'w')
497 |         weight_file.write("ACK\n")
498 |         weight_file.close()
499 |
500 |         #####
501 |         # Blocks
502 |         #####
503 |         self.uhd_usrp_sink_1 = uhd.usrp_sink(
504 |             ", ".join("", ""),
505 |             uhd.stream_args(
506 |                 cpu_format="fc32",
507 |                 channels=range(1),
508 |             ),
509 |         )
510 |         self.uhd_usrp_sink_1.set_samp_rate(samp_rate)
511 |         self.uhd_usrp_sink_1.set_time_now(uhd.time_spec(time.time()), uhd.ALL_MBOARDS)
512 |         self.uhd_usrp_sink_1.set_center_freq(1000000000, 0)
513 |         self.uhd_usrp_sink_1.set_gain(40, 0)
514 |         self.uhd_usrp_sink_1.set_antenna('TX/RX', 0)
515 |         self.digital_gmsk_mod_0 = digital.gmsk_mod(
516 |             samples_per_symbol=2,
517 |             bt=0.35,
518 |             verbose=False,
519 |             log=False,
520 |         )
521 |         self.blocks_throttle_0 = blocks.throttle(gr.sizeof_char*1, samp_rate, True)
522 |         self.blocks_multiply_const_vxx_1 = blocks.multiply_const_vcc((1,))
523 |         self.blocks_head_0 = blocks.head(gr.sizeof_char*1, 100000)
524 |         self.blocks_file_source_0 = blocks.file_source(gr.sizeof_char*1, '/home/c/chat3', True)
525 |         self.blocks_file_source_0.set_begin_tag(pmt.PMT_NIL)
526 |         self.blks2_packet_encoder_0 = grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
527 |             samples_per_symbol=2,
528 |             bits_per_symbol=1,
529 |             preamble='',
530 |             access_code='',
531 |             pad_for_usrp=True,
532 |         ),
533 |             payload_length=0,
534 |         )
```

(Image continued from previous page)

```
535
536
537
538 #####
539 # Connections
540 #####
541 self.connect((self.blks2_packet_encoder_0, 0), (self.digital_gmsk_mod_0, 0))
542 self.connect((self.blocks_file_source_0, 0), (self.blocks_throttle_0, 0))
543 self.connect((self.blocks_head_0, 0), (self.blks2_packet_encoder_0, 0))
544 self.connect((self.blocks_multiply_const_vxx_1, 0), (self.uhd_usrp_sink_1, 0))
545 self.connect((self.blocks_throttle_0, 0), (self.blocks_head_0, 0))
546 self.connect((self.digital_gmsk_mod_0, 0), (self.blocks_multiply_const_vxx_1, 0))
547
548 def closeEvent(self, event):
549     self.settings = Qt.QSettings("GNU Radio", "chat")
550     self.settings.setValue("geometry", self.saveGeometry())
551     event.accept()
552
553 def get_samp_rate(self):
554     return self.samp_rate
555
556 def set_samp_rate(self, samp_rate):
557     self.samp_rate = samp_rate
558     self.uhd_usrp_sink_1.set_samp_rate(self.samp_rate)
559     self.blocks_throttle_0.set_sample_rate(self.samp_rate)
560
561 def main(options=None):
562
563     from distutils.version import StrictVersion
564     if StrictVersion(Qt.qVersion()) >= StrictVersion("4.5.0"):
565         style = gr.prefs().get_string('qtgui', 'style', 'raster')
566         Qt.QApplication.setGraphicsSystem(style)
567     qapp = Qt.QApplication(sys.argv)
568
569
570 global weight1, weight2, weight3, weight4, weight5, weight6, weight7, weight8, weight9, weight10, weight11, weight12, weight13, weight14
571 a_num = 0
572 b_num = 0
573 c_num = 0
574 d_num = 0
575 e_num = 0
576 f_num = 0
577 g_num = 0
578 h_num = 0
579 i_num = 0
580 j_num = 0
581 k_num = 0
582 l_num = 0
583 m_num = 0
584 n_num = 0
585 sum_a = 0
586 sum_b = 0
587 sum_c = 0
588 sum_d = 0
589 sum_e = 0
590 sum_f = 0
591 sum_g = 0
592 sum_h = 0
593 sum_i = 0
594 sum_j = 0
595 sum_k = 0
596 sum_l = 0
597 sum_m = 0
598 sum_n = 0
599 predict_a = 0
600 predict_b = 0
601 predict_c = 0
602 predict_d = 0
603 predict_e = 0
604 predict_f = 0
605 predict_g = 0
606 predict_h = 0
607 predict_i = 0
608 predict_j = 0
```

(Image continued from previous page)

```
609 predict_k = 0
610 predict_l = 0
611 predict_m = 0
612 predict_n = 0
613 actual_a_list = []
614 actual_b_list = []
615 actual_c_list = []
616 actual_d_list = []
617 actual_e_list = []
618 actual_f_list = []
619 actual_g_list = []
620 actual_h_list = []
621 actual_i_list = []
622 actual_j_list = []
623 actual_k_list = []
624 actual_l_list = []
625 actual_m_list = []
626 actual_n_list = []
627 predict_a_list = []
628 predict_b_list = []
629 predict_c_list = []
630 predict_d_list = []
631 predict_e_list = []
632 predict_f_list = []
633 predict_g_list = []
634 predict_h_list = []
635 predict_i_list = []
636 predict_j_list = []
637 predict_k_list = []
638 predict_l_list = []
639 predict_m_list = []
640 predict_n_list = []
641
642 tb = chat()
643 tb.start()
644 time.sleep(1)
645 tb.stop()
646
647 final = open('/home/c/final', 'w')
648 final.close()
649
650 while True:
651     tb = text_rx()
652     tb.start()
653
654     data1 = open('/home/c/test_file2', 'r+')
655     final = open('/home/c/final', 'a')
656
657     while True:
658         data1.flush()
659         data1.readline()
660         file_content2 = data1.readline()
661         if (file_content2 == "eof\n"):
662             print(file_content2)
663             break
664         elif (file_content2 != ""):
665             final.write(file_content2)
666             split = file_content2.split(' ')
667             if (split[1].startswith('10') and (split[1].startswith('0', 2) == False)):
668                 a_num += 1
669                 actual_a_list.append(float(split[0]))
670                 sum_a += float(split[0])
671                 predict_a = sum_a/a_num
672                 print("Prediction for Next A: "+str(predict_a))
673                 predict_a_list.append(predict_a)
674             elif (split[1].startswith('20')):
675                 b_num += 1
676                 actual_b_list.append(float(split[0]))
677                 sum_b += float(split[0])
678                 predict_b = sum_b/b_num
679                 print("Prediction for Next B: "+str(predict_b))
680                 predict_b_list.append(predict_b)
681             elif (split[1].startswith('30')):
682                 c_num += 1
```

(Image continued from previous page)

```
683     actual_c_list.append(float(split[0]))
684     sum_c += float(split[0])
685     predict_c = sum_c/c_num
686     print("Prediction for Next C: "+str(predict_c))
687     predict_c_list.append(predict_c)
688 elif (split[1].startswith('40')):
689     d_num += 1
690     actual_d_list.append(float(split[0]))
691     sum_d += float(split[0])
692     predict_d = sum_d/d_num
693     print("Prediction for Next D: "+str(predict_d))
694     predict_d_list.append(predict_d)
695 elif (split[1].startswith('50')):
696     e_num += 1
697     actual_e_list.append(float(split[0]))
698     sum_e += float(split[0])
699     predict_e = sum_e/e_num
700     print("Prediction for Next E: "+str(predict_e))
701     predict_e_list.append(predict_e)
702 elif (split[1].startswith('60')):
703     f_num += 1
704     actual_f_list.append(float(split[0]))
705     sum_f += float(split[0])
706     predict_f = sum_f/f_num
707     print("Prediction for Next F: "+str(predict_f))
708     predict_f_list.append(predict_f)
709 elif (split[1].startswith('70')):
710     g_num += 1
711     actual_g_list.append(float(split[0]))
712     sum_g += float(split[0])
713     predict_g = sum_g/g_num
714     print("Prediction for Next G: "+str(predict_g))
715     predict_g_list.append(predict_g)
716 elif (split[1].startswith('80')):
717     h_num += 1
718     actual_h_list.append(float(split[0]))
719     sum_h += float(split[0])
720     predict_h = sum_h/h_num
721     print("Prediction for Next H: "+str(predict_h))
722     predict_h_list.append(predict_h)
723 elif (split[1].startswith('90')):
724     i_num += 1
725     actual_i_list.append(float(split[0]))
726     sum_i += float(split[0])
727     predict_i = sum_i/i_num
728     print("Prediction for Next I: "+str(predict_i))
729     predict_i_list.append(predict_i)
730 elif (split[1].startswith('100')):
731     j_num += 1
732     actual_j_list.append(float(split[0]))
733     sum_j += float(split[0])
734     predict_j = sum_j/j_num
735     print("Prediction for Next J: "+str(predict_j))
736     predict_j_list.append(predict_j)
737 elif (split[1].startswith('110')):
738     k_num += 1
739     actual_k_list.append(float(split[0]))
740     sum_k += float(split[0])
741     predict_k = sum_k/k_num
742     print("Prediction for Next K: "+str(predict_k))
743     predict_k_list.append(predict_k)
744 elif (split[1].startswith('120')):
745     l_num += 1
746     actual_l_list.append(float(split[0]))
747     sum_l += float(split[0])
748     predict_l = sum_l/l_num
749     print("Prediction for Next L: "+str(predict_l))
750     predict_l_list.append(predict_l)
751 elif (split[1].startswith('130')):
752     m_num += 1
753     actual_m_list.append(float(split[0]))
754     sum_m += float(split[0])
755     predict_m = sum_m/m_num
756     print("Prediction for Next M: "+str(predict_m))
```

(Image continued from previous page)

```
757         predict_m_list.append(predict_m)
758     elif (split[1].startswith('140')):
759         n_num += 1
760         actual_n_list.append(float(split[0]))
761         sum_n += float(split[0])
762         predict_n = sum_n/n_num
763         print("Prediction for Next N: "+str(predict_n))
764         predict_n_list.append(predict_n)
765     print("Current Value: "+file_content2)
766     time.sleep(1)
767     sb = ack()
768     sb.start()
769     print("SENDING ACK")
770     time.sleep(1)
771     sb.stop()
772     break
773     time.sleep(1)
774
775     tb.stop()
776
777     if (file_content2 == "eof\n"):
778         datal.close()
779         final.close()
780         break
781
782     datal.close()
783     final.close()
784
785     tb.stop()
786
787     sum_sq_a = 0
788     for z in range(0, len(predict_a_list)-1):
789         square = (actual_a_list[z+1] - predict_a_list[z])**2
790         sum_sq_a += square
791     msqrterr_a = math.sqrt(sum_sq_a)
792     print("Root Mean Squared Error for Sensor 1: "+str(msqrterr_a))
793
794     sum_sq_b = 0
795     for z in range(0, len(predict_b_list)-1):
796         square = (actual_b_list[z+1] - predict_b_list[z])**2
797         sum_sq_b += square
798     msqrterr_b = math.sqrt(sum_sq_b)
799     print("Root Mean Squared Error for Sensor 2: "+str(msqrterr_b))
800
801     sum_sq_c = 0
802     for z in range(0, len(predict_c_list)-1):
803         square = (actual_c_list[z+1] - predict_c_list[z])**2
804         sum_sq_c += square
805     msqrterr_c = math.sqrt(sum_sq_c)
806     print("Root Mean Squared Error for Sensor 3: "+str(msqrterr_c))
807
808     sum_sq_d = 0
809     for z in range(0, len(predict_d_list)-1):
810         square = (actual_d_list[z+1] - predict_d_list[z])**2
811         sum_sq_d += square
812     msqrterr_d = math.sqrt(sum_sq_d)
813     print("Root Mean Squared Error for Sensor 4: "+str(msqrterr_d))
814
815     sum_sq_e = 0
816     for z in range(0, len(predict_e_list)-1):
817         square = (actual_e_list[z+1] - predict_e_list[z])**2
818         sum_sq_e += square
819     msqrterr_e = math.sqrt(sum_sq_e)
820     print("Root Mean Squared Error for Sensor 5: "+str(msqrterr_e))
821
822     sum_sq_f = 0
823     for z in range(0, len(predict_f_list)-1):
824         square = (actual_f_list[z+1] - predict_f_list[z])**2
825         sum_sq_f += square
826     msqrterr_f = math.sqrt(sum_sq_f)
827     print("Root Mean Squared Error for Sensor 6: "+str(msqrterr_f))
828
829     sum_sq_g = 0
830     for z in range(0, len(predict_g_list)-1):
```


(Image continued from previous page)

```
831     square = (actual_g_list[z+1] - predict_g_list[z])**2
832     sum_sq_g += square
833 msqrterr_g = math.sqrt(sum_sq_g)
834 print("Root Mean Squared Error for Sensor 7: "+str(msqrterr_g))
835
836     sum_sq_h = 0
837     for z in range(0, len(predict_h_list)-1):
838         square = (actual_h_list[z+1] - predict_h_list[z])**2
839         sum_sq_h += square
840 msqrterr_h = math.sqrt(sum_sq_h)
841 print("Root Mean Squared Error for Sensor 8: "+str(msqrterr_h))
842
843     sum_sq_i = 0
844     for z in range(0, len(predict_i_list)-1):
845         square = (actual_i_list[z+1] - predict_i_list[z])**2
846         sum_sq_i += square
847 msqrterr_i = math.sqrt(sum_sq_i)
848 print("Root Mean Squared Error for Sensor 9: "+str(msqrterr_i))
849
850     sum_sq_j = 0
851     for z in range(0, len(predict_j_list)-1):
852         square = (actual_j_list[z+1] - predict_j_list[z])**2
853         sum_sq_j += square
854 msqrterr_j = math.sqrt(sum_sq_j)
855 print("Root Mean Squared Error for Sensor 10: "+str(msqrterr_j))
856
857     sum_sq_k = 0
858     for z in range(0, len(predict_k_list)-1):
859         square = (actual_k_list[z+1] - predict_k_list[z])**2
860         sum_sq_k += square
861 msqrterr_k = math.sqrt(sum_sq_k)
862 print("Root Mean Squared Error for Sensor 11: "+str(msqrterr_k))
863
864     sum_sq_l = 0
865     for z in range(0, len(predict_l_list)-1):
866         square = (actual_l_list[z+1] - predict_l_list[z])**2
867         sum_sq_l += square
868 msqrterr_l = math.sqrt(sum_sq_l)
869 print("Root Mean Squared Error for Sensor 12: "+str(msqrterr_l))
870
871     sum_sq_m = 0
872     for z in range(0, len(predict_m_list)-1):
873         square = (actual_m_list[z+1] - predict_m_list[z])**2
874         sum_sq_m += square
875 msqrterr_m = math.sqrt(sum_sq_m)
876 print("Root Mean Squared Error for Sensor 13: "+str(msqrterr_m))
877
878     sum_sq_n = 0
879     for z in range(0, len(predict_n_list)-1):
880         square = (actual_n_list[z+1] - predict_n_list[z])**2
881         sum_sq_n += square
882 msqrterr_n = math.sqrt(sum_sq_n)
883 print("Root Mean Squared Error for Sensor 14: "+str(msqrterr_n))
884
885 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_
886 print("\nTotal Root Mean Squared Error: "+str(rmse_tot))
887
888 exit()
889
890 def quitting():
891     tb.stop()
892     tb.wait()
893 qapp.connect(qapp, Qt.SIGNAL("aboutToQuit()"), quitting)
894 qapp.exec_()
895
896
897 if __name__ == '__main__':
898     main()
```

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

```

1  #!/usr/bin/env python2
2  # -*- coding: utf-8 -*-
3  #####
4  # GNU Radio Python Flow Graph
5  # Title: Chat2
6  # Generated: Thu Feb 10 09:52:14 2022
7  #####
8
9  if __name__ == '__main__':
10     import ctypes
11     import sys
12     if sys.platform.startswith('linux'):
13         try:
14             x11 = ctypes.cdll.LoadLibrary('libX11.so')
15             x11.XInitThreads()
16         except:
17             print("Warning: failed to XInitThreads()")
18
19     import .....
39
40
41     class chat2(gr.top_block, Qt.QWidget):
42
43     def __init__(self):
44         gr.top_block.__init__(self, "Chat2")
45         Qt.QWidget.__init__(self)
46         self.setWindowTitle("Chat2")
47         qtgui.util.check_set_qss()
48         try:
49             self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-gre'))
50         except:
51             pass
52         self.top_scroll_layout = Qt.QVBoxLayout()
53         self.setLayout(self.top_scroll_layout)
54         self.top_scroll = Qt.QScrollArea()
55         self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
56         self.top_scroll_layout.addWidget(self.top_scroll)
57         self.top_scroll.setWidgetResizable(True)
58         self.top_widget = Qt.QWidget()
59         self.top_scroll.setWidget(self.top_widget)
60         self.top_layout = Qt.QVBoxLayout(self.top_widget)
61         self.top_grid_layout = Qt.QGridLayout()
62         self.top_layout.addLayout(self.top_grid_layout)
63
64         self.settings = Qt.QSettings("GNU Radio", "chat2")
65         self.restoreGeometry(self.settings.value("geometry").toByteArray())
66
67
68         #####
69         # Variables
70         #####
71         self.samp_rate = samp_rate = 400000
72
73         #####
74         # Blocks
75         #####
76         self.uhd_usrp_source_0 = uhd.usrp_source(
77             ", ".join("", ""),
78             uhd.stream_args(
79                 cpu_format="fc32",
80                 channels=range(1),
81             ),
82         )
83         self.uhd_usrp_source_0.set_samp_rate(samp_rate)
84         self.uhd_usrp_source_0.set_center_freq(1000000000, 0)
85         self.uhd_usrp_source_0.set_gain(30, 0)
86         self.uhd_usrp_source_0.set_antenna('TX/RX', 0)
87         self.qtgui_freq_sink_x_0 = qtgui.freq_sink_c(
88             1024, #size
89             firdes.WIN_BLACKMAN_hARRIS, #wintype
90             0, #fc
91             samp_rate, #bw
92             "", #name
93             1 #number of inputs

```

(Image continued from previous page)

```
94     )
95     self.qtgui_freq_sink_x_0.set_update_time(0.10)
96     self.qtgui_freq_sink_x_0.set_y_axis(-140, 10)
97     self.qtgui_freq_sink_x_0.set_y_label('Relative Gain', 'dB')
98     self.qtgui_freq_sink_x_0.set_trigger_mode(qtgui.TRIG_MODE_FREE, 0.0, 0, "")
99     self.qtgui_freq_sink_x_0.enable_autoscale(False)
100    self.qtgui_freq_sink_x_0.enable_grid(False)
101    self.qtgui_freq_sink_x_0.set_fft_average(1.0)
102    self.qtgui_freq_sink_x_0.enable_axis_labels(True)
103    self.qtgui_freq_sink_x_0.enable_control_panel(False)
104
105    if not True:
106        self.qtgui_freq_sink_x_0.disable_legend()
107
108    if "complex" == "float" or "complex" == "msg_float":
109        self.qtgui_freq_sink_x_0.set_plot_pos_half(not True)
110
111    labels = ['', '', '', '', '']
112            ['', '', '', '', '']
113    widths = [1, 1, 1, 1, 1,
114             1, 1, 1, 1, 1]
115    colors = ["blue", "red", "green", "black", "cyan",
116            "magenta", "yellow", "dark red", "dark green", "dark blue"]
117    alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
118            1.0, 1.0, 1.0, 1.0, 1.0]
119    for i in xrange(1):
120        if len(labels[i]) == 0:
121            self.qtgui_freq_sink_x_0.set_line_label(i, "Data {0}".format(i))
122        else:
123            self.qtgui_freq_sink_x_0.set_line_label(i, labels[i])
124            self.qtgui_freq_sink_x_0.set_line_width(i, widths[i])
125            self.qtgui_freq_sink_x_0.set_line_color(i, colors[i])
126            self.qtgui_freq_sink_x_0.set_line_alpha(i, alphas[i])
127
128    self._qtgui_freq_sink_x_0_win = sip.wrapinstance(self.qtgui_freq_sink_x_0.pyqwidget(), Qt.QWidget)
129    self.top_grid_layout.addWidget(self._qtgui_freq_sink_x_0_win)
130    self.low_pass_filter_0 = filter.fir_filter_ccf(1, firdes.low_pass(
131    1, samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))
132    self.digital_gmsk_demod_1 = digital.gmsk_demod(
133        samples_per_symbol=2,
134        gain_mu=0.175,
135        mu=0.5,
136        omega_relative_limit=0.005,
137        freq_error=0.0,
138        verbose=False,
139        log=False,
140    )
141    self.blocks_multiply_const_vxx_1 = blocks.multiply_const_vcc((1, ))
142    self.blocks_file_sink_1 = blocks.file_sink(gr.sizeof_char*1, '/home/b/chat_test1.txt', False)
143    self.blocks_file_sink_1.set_unbuffered(False)
144    self.blks2_packet_decoder_1 = gro_blks2.packet_demod_b(gro_blks2.packet_decoder(
145        access_code='',
146        threshold=1,
147        callback=lambda ok, payload: self.blks2_packet_decoder_1.recv_pkt(ok, payload),
148    )),
149    )
150
151
152
153    #####
154    # Connections
155    #####
156    self.connect((self.blks2_packet_decoder_1, 0), (self.blocks_file_sink_1, 0))
157    self.connect((self.blocks_multiply_const_vxx_1, 0), (self.low_pass_filter_0, 0))
158    self.connect((self.digital_gmsk_demod_1, 0), (self.blks2_packet_decoder_1, 0))
159    self.connect((self.low_pass_filter_0, 0), (self.digital_gmsk_demod_1, 0))
160    self.connect((self.low_pass_filter_0, 0), (self.qtgui_freq_sink_x_0, 0))
161    self.connect((self.uhd_usrp_source_0, 0), (self.blocks_multiply_const_vxx_1, 0))
162
163    def closeEvent(self, event):
164        self.settings = Qt.QSettings("GNU Radio", "chat2")
165        self.settings.setValue("geometry", self.saveGeometry())
166        event.accept()
167
```

(Image continued from previous page)

```
168 def get_samp_rate(self):
169     return self.samp_rate
170
171 def set_samp_rate(self, samp_rate):
172     self.samp_rate = samp_rate
173     self.uhd_usrp_source_0.set_samp_rate(self.samp_rate)
174     self.qtgui_freq_sink_x_0.set_frequency_range(0, self.samp_rate)
175     self.low_pass_filter_0.set_taps(firdes.low_pass(1, self.samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))
176
177
178 class text_tx(gr.top_block):
179
180     def __init__(self):
181         gr.top_block.__init__(self, "Text Tx")
182
183         #####
184         # Variables
185         #####
186         self.samp_rate = samp_rate = 400000
187         source = ''
188
189         #####
190         # Blocks
191         #####
192         self.uhd_usrp_sink_0 = uhd.usrp_sink(
193             ", ".join("", ""),
194             uhd.stream_args(
195                 cpu_format="fc32",
196                 channels=range(1),
197             ),
198         )
199         self.uhd_usrp_sink_0.set_samp_rate(samp_rate)
200         self.uhd_usrp_sink_0.set_time_now(uhd.time_spec(time.time()), uhd.ALL_MBOARDS)
201         self.uhd_usrp_sink_0.set_center_freq(1000000000, 0)
202         self.uhd_usrp_sink_0.set_gain(40, 0)
203         self.uhd_usrp_sink_0.set_antenna('TX/RX', 0)
204         self.digital_gmsk_mod_0 = digital.gmsk_mod(
205             samples_per_symbol=2,
206             bt=0.35,
207             verbose=False,
208             log=False,
209         )
210         self.blocks_throttle_0 = blocks.throttle(gr.sizeof_char*1, samp_rate, True)
211         self.blocks_multiply_const_vxx_0 = blocks.multiply_const_vcc((1, ))
212         self.blocks_head_0 = blocks.head(gr.sizeof_char*1, samp_rate/4)
213
214         send = open('/home/b/tx.txt', 'w')
215         if switch == 'a':
216             send.write(data0[a1])
217             print(data0[a1])
218         elif switch == 'b':
219             send.write(data1[b1])
220             print(data1[b1])
221         elif switch == 'c':
222             send.write(data2[c1])
223             print(data2[c1])
224         elif switch == 'd':
225             send.write(data3[d1])
226             print(data3[d1])
227         elif switch == 'e':
228             send.write(data4[e1])
229             print(data4[e1])
230         elif switch == 'f':
231             send.write(data5[f1])
232             print(data5[f1])
233         elif switch == 'g':
234             send.write(data6[g1])
235             print(data6[g1])
236         elif switch == 'h':
237             send.write(data7[h1])
238             print(data7[h1])
239         elif switch == 'i':
240             send.write(data8[i1])
241             print(data8[i1])
```

(Image continued from previous page)

```
242 elif switch == 'j':
243     send.write(data9[j1])
244     print(data9[j1])
245 elif switch == 'k':
246     send.write(data10[k1])
247     print(data10[k1])
248 elif switch == 'l':
249     send.write(data11[l1])
250     print(data11[l1])
251 elif switch == 'm':
252     send.write(data12[m1])
253     print(data12[m1])
254 elif switch == 'n':
255     send.write(data13[n1])
256     print(data13[n1])
257 elif switch == 'eof':
258     send.write("eof\n")
259     print("eof")
260 send.close()
261
262 self.blocks_file_source_0 = blocks.file_source(gr.sizeof_char*1, '/home/b/tx.txt', True)
263
264 self.blocks_file_source_0.set_begin_tag(pmt.PMT_NIL)
265 self.blks2_packet_encoder_0 = grc_blks2_packet_mod_b(grc_blks2_packet_encoder(
266     samples_per_symbol=2,
267     bits_per_symbol=1,
268     preamble='',
269     access_code='',
270     pad_for_usrp=True,
271 ),
272     payload_length=0,
273 )
274
275
276
277 #####
278 # Connections
279 #####
280 self.connect((self.blks2_packet_encoder_0, 0), (self.digital_gmsk_mod_0, 0))
281 self.connect((self.blocks_file_source_0, 0), (self.blocks_throttle_0, 0))
282 self.connect((self.blocks_head_0, 0), (self.blks2_packet_encoder_0, 0))
283 self.connect((self.blocks_multiply_const_vxx_0, 0), (self.uhd_usrp_sink_0, 0))
284 self.connect((self.blocks_throttle_0, 0), (self.blocks_head_0, 0))
285 self.connect((self.digital_gmsk_mod_0, 0), (self.blocks_multiply_const_vxx_0, 0))
286
287 def get_samp_rate(self):
288     return self.samp_rate
289
290 def set_samp_rate(self, samp_rate):
291     self.samp_rate = samp_rate
292     self.uhd_usrp_sink_0.set_samp_rate(self.samp_rate)
293     self.blocks_throttle_0.set_sample_rate(self.samp_rate)
294     self.blocks_head_0.set_length(self.samp_rate/4)
295
296
297 class WRRScheduler():
298
299     cw = 0
300     i = -1
301     data_set = []
302     max_s = None
303     gcd_s = None
304     len_s = None
305     counter = {}
306
307     def __init__(self, s = None):
308         self._init_dataset(s)
309
310     def _init_dataset(self, s):
311         self.data_set = s
312         self.max_s = max(s, key=lambda x: x[1])[1]
313         self.gcd_s = reduce(fractions.gcd, [weight for data, weight in s])
314         self.len_s = len(s)
315
```

(Image continued from previous page)

```
316 def schedule(self):
317     while True:
318         self.i = (self.i + 1) % self.len_s
319         if self.i == 0:
320             self.cw = self.cw - self.gcd_s
321             if self.cw <= 0:
322                 self.cw = self.max_s
323                 if self.cw == 0:
324                     return None
325             if self.data_set[self.i][1] >= self.cw:
326                 self._inc_counter(self.data_set[self.i])
327                 return self.data_set[self.i]
328
329 def _inc_counter(self, item):
330     try:
331         self.counter[item[0]] += 1
332     except KeyError:
333         self.counter[item[0]] = 1
334
335 def set_data(self, s):
336     self.reset()
337     self._init_dataset(s)
338
339 def reset_counter(self):
340     self.counter = {}
341
342 def reset(self):
343     self.cw = 0
344     self.i = -1
345     self.data_set = []
346     self.max_s = None
347     self.gcd_s = None
348     self.len_s = None
349     self.reset_counter()
350
351 def get_next(self, n = 1):
352     if n > 1:
353         return [ self.schedule() for i in range(0,n) ]
354     return self.schedule()
355
356
357 class ack2(gr.top_block, Qt.QWidget):
358
359     def __init__(self):
360         gr.top_block.__init__(self, "Chat2")
361         Qt.QWidget.__init__(self)
362         self.setWindowTitle("Chat2")
363         qtgui.util.check_set_qss()
364         try:
365             self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-gro'))
366         except:
367             pass
368         self.top_scroll_layout = Qt.QVBoxLayout()
369         self.setLayout(self.top_scroll_layout)
370         self.top_scroll = Qt.QScrollArea()
371         self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
372         self.top_scroll_layout.addWidget(self.top_scroll)
373         self.top_scroll.setWidgetResizable(True)
374         self.top_widget = Qt.QWidget()
375         self.top_scroll.setWidget(self.top_widget)
376         self.top_layout = Qt.QVBoxLayout(self.top_widget)
377         self.top_grid_layout = Qt.QGridLayout()
378         self.top_layout.addLayout(self.top_grid_layout)
379
380         self.settings = Qt.QSettings("GNU Radio", "chat2")
381         self.restoreGeometry(self.settings.value("geometry").toByteArray())
382
383
384     #####
385     # Variables
386     #####
387     self.samp_rate = samp_rate = 400000
388
389     #####
```

(Image continued from previous page)

```
390 # Blocks
391 #####
392 self.uhd_usrp_source_0 = uhd.usrp_source(
393     ", ".join("", ""),
394     uhd.stream_args(
395         cpu_format="fc32",
396         channels=range(1),
397     ),
398 )
399 self.uhd_usrp_source_0.set_samp_rate(samp_rate)
400 self.uhd_usrp_source_0.set_center_freq(1000000000, 0)
401 self.uhd_usrp_source_0.set_gain(30, 0)
402 self.uhd_usrp_source_0.set_antenna('TX/RX', 0)
403 self.qtgui_freq_sink_x_0 = qtgui.freq_sink_c(
404     1024, #size
405     firdes.WIN_BLACKMAN_hARRIS, #wintype
406     0, #fc
407     samp_rate, #bw
408     "", #name
409     1 #number of inputs
410 )
411 self.qtgui_freq_sink_x_0.set_update_time(0.10)
412 self.qtgui_freq_sink_x_0.set_y_axis(-140, 10)
413 self.qtgui_freq_sink_x_0.set_y_label('Relative Gain', 'dB')
414 self.qtgui_freq_sink_x_0.set_trigger_mode(qtgui.TRIG_MODE_FREE, 0.0, 0, "")
415 self.qtgui_freq_sink_x_0.enable_autoscale(False)
416 self.qtgui_freq_sink_x_0.enable_grid(False)
417 self.qtgui_freq_sink_x_0.set_fft_average(1.0)
418 self.qtgui_freq_sink_x_0.enable_axis_labels(True)
419 self.qtgui_freq_sink_x_0.enable_control_panel(False)
420
421 if not True:
422     self.qtgui_freq_sink_x_0.disable_legend()
423
424 if "complex" == "float" or "complex" == "msg_float":
425     self.qtgui_freq_sink_x_0.set_plot_pos_half(not True)
426
427 labels = ['', '', '', '', '',
428           '', '', '', '', '']
429 widths = [1, 1, 1, 1, 1,
430           1, 1, 1, 1, 1]
431 colors = ["blue", "red", "green", "black", "cyan",
432           "magenta", "yellow", "dark red", "dark green", "dark blue"]
433 alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
434           1.0, 1.0, 1.0, 1.0, 1.0]
435 for i in xrange(1):
436     if len(labels[i]) == 0:
437         self.qtgui_freq_sink_x_0.set_line_label(i, "Data {0}".format(i))
438     else:
439         self.qtgui_freq_sink_x_0.set_line_label(i, labels[i])
440         self.qtgui_freq_sink_x_0.set_line_width(i, widths[i])
441         self.qtgui_freq_sink_x_0.set_line_color(i, colors[i])
442         self.qtgui_freq_sink_x_0.set_line_alpha(i, alphas[i])
443
444 self.qtgui_freq_sink_x_0.win = sip.wrapinstance(self.qtgui_freq_sink_x_0.pyqwidget(), Qt.QWidget)
445 self.top_grid_layout.addWidget(self.qtgui_freq_sink_x_0.win)
446 self.low_pass_filter_0 = filter.fir_filter_ccf(1, firdes.low_pass(
447     1, samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))
448 self.digital_gmsk_demod_1 = digital.gmsk_demod(
449     samples_per_symbol=2,
450     gain_mu=0.175,
451     mu=0.5,
452     omega_relative_limit=0.005,
453     freq_error=0.0,
454     verbose=False,
455     log=False,
456 )
457 self.blocks_multiply_const_vxx_1 = blocks.multiply_const_vcc((1, ))
458 self.blocks_file_sink_1 = blocks.file_sink(gr.sizeof_char*1, '/home/b/chat_test2.txt', False)
459 self.blocks_file_sink_1.set_unbuffered(False)
460 self.blks2_packet_decoder_1 = grc_blks2.packet_demod_b(grc_blks2.packet_decoder(
461     access_code='',
462     threshold=-1,
463     callback=lambda ok, payload: self.blks2_packet_decoder_1.recv_pkt(ok, payload),
```

(Image continued from previous page)

```
464         ),
465     )
466
467
468
469     #####
470     # Connections
471     #####
472     self.connect((self.blks2_packet_decoder_1, 0), (self.blocks_file_sink_1, 0))
473     self.connect((self.blocks_multiply_const_vxx_1, 0), (self.low_pass_filter_0, 0))
474     self.connect((self.digital_gmsk_demod_1, 0), (self.blks2_packet_decoder_1, 0))
475     self.connect((self.low_pass_filter_0, 0), (self.digital_gmsk_demod_1, 0))
476     self.connect((self.low_pass_filter_0, 0), (self.qtgui_freq_sink_x_0, 0))
477     self.connect((self.uhd_usrp_source_0, 0), (self.blocks_multiply_const_vxx_1, 0))
478
479     def closeEvent(self, event):
480         self.settings = Qt.QSettings("GNU Radio", "chat2")
481         self.settings.setValue("geometry", self.saveGeometry())
482         event.accept()
483
484     def get_samp_rate(self):
485         return self.samp_rate
486
487     def set_samp_rate(self, samp_rate):
488         self.samp_rate = samp_rate
489         self.uhd_usrp_source_0.set_samp_rate(self.samp_rate)
490         self.qtgui_freq_sink_x_0.set_frequency_range(0, self.samp_rate)
491         self.low_pass_filter_0.set_taps(firdes.low_pass(1, self.samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))
492
493
494     def main(options=None):
495
496         from distutils.version import StrictVersion
497         if StrictVersion(Qt.qVersion()) >= StrictVersion("4.5.0"):
498             style = gr.prefs().get_string('qtgui', 'style', 'raster')
499             Qt.QApplication.setGraphicsSystem(style)
500         qapp = Qt.QApplication(sys.argv)
501
502         data_csv = open('/home/b/20161005_140846.csv', 'r')
503         csvreader = csv.reader(data_csv)
504         header = next(csvreader)
505         data_rows = []
506         for row in csvreader:
507             data_rows.append(row)
508         data_csv.close()
509
510         for x in range(0,14):
511             out = open('file' + str(x+1) + '.txt', 'w')
512             for y in range(0, 100):
513                 out.write(data_rows[y][x+6]+' '+str(x+1)+'0'+str(y+1)+'\n')
514             out.close()
515
516         tb = chat2()
517         tb.start()
518         print_"Ready to start..."
519         while True:
520             f = open('/home/b/chat_test1.txt', 'r')
521             f.flush()
522             f.readline()
523             check = f.readline()
524             if (check != ""):
525                 print_"START"
526                 time.sleep(1)
527                 break
528         tb.stop()
529
530         weight_file = open('/home/b/chat_test1.txt', 'r')
531         weight_file.flush()
532         weight_file.readline()
533         weights = weight_file.readline()
534         w_list = weights.split(",")
535         w_list.pop()
536         for m in range(0, len(w_list)):
537             w_list[m] = int(w_list[m])
```


(Image continued from previous page)

```
538 sensors = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N']
539
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)

```
612     length1 = len(data10)
613     read11.close()
614
615     read12 = open('/home/b/file12.txt', 'r')
616     data11 = read12.readlines()
617     length2 = len(data11)
618     read12.close()
619
620     read13 = open('/home/b/file13.txt', 'r')
621     data12 = read13.readlines()
622     length3 = len(data12)
623     read13.close()
624
625     read14 = open('/home/b/file14.txt', 'r')
626     data13 = read14.readlines()
627     length4 = len(data13)
628     read14.close()
629
630     data = list(zip(sensors, w_list))
631     sched = WRRScheduler(data)
632
633     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
634     a_tot = 0
635     b_tot = 0
636     c_tot = 0
637     d_tot = 0
638     e_tot = 0
639     f_tot = 0
640     g_tot = 0
641     h_tot = 0
642     i_tot = 0
643     j_tot = 0
644     k_tot = 0
645     l_tot = 0
646     m_tot = 0
647     n_tot = 0
648
649     x = 0
650     result = []
651     while (x < 150):
652         choose = sched.get_next()
653         if choose[0] == 'A':
654             a_tot += 1
655             if (a_tot <= length1):
656                 result.append(choose[0])
657                 x += 1
658         elif choose[0] == 'B':
659             b_tot += 1
660             if (b_tot <= length2):
661                 result.append(choose[0])
662                 x += 1
663         elif choose[0] == 'C':
664             c_tot += 1
665             if (c_tot <= length3):
666                 result.append(choose[0])
667                 x += 1
668         elif choose[0] == 'D':
669             d_tot += 1
670             if (d_tot <= length4):
671                 result.append(choose[0])
672                 x += 1
673         elif choose[0] == 'E':
674             e_tot += 1
675             if (e_tot <= length5):
676                 result.append(choose[0])
677                 x += 1
678         elif choose[0] == 'F':
679             f_tot += 1
680             if (f_tot <= length6):
681                 result.append(choose[0])
682                 x += 1
683         elif choose[0] == 'G':
684             g_tot += 1
685             if (g_tot <= length7):
```

(Image continued from previous page)

```
686         result.append(choose[0])
687         x += 1
688     elif choose[0] == 'H':
689         h_tot += 1
690         if (h_tot <= length8):
691             result.append(choose[0])
692             x += 1
693     elif choose[0] == 'I':
694         i_tot += 1
695         if (i_tot <= length9):
696             result.append(choose[0])
697             x += 1
698     elif choose[0] == 'J':
699         j_tot += 1
700         if (j_tot <= length10):
701             result.append(choose[0])
702             x += 1
703     elif choose[0] == 'K':
704         k_tot += 1
705         if (k_tot <= length11):
706             result.append(choose[0])
707             x += 1
708     elif choose[0] == 'L':
709         l_tot += 1
710         if (l_tot <= length12):
711             result.append(choose[0])
712             x += 1
713     elif choose[0] == 'M':
714         m_tot += 1
715         if (m_tot <= length13):
716             result.append(choose[0])
717             x += 1
718     elif choose[0] == 'N':
719         n_tot += 1
720         if (n_tot <= length14):
721             result.append(choose[0])
722             x += 1
723
724     while i < (len(result)):
725         if result[i] == 'A':
726             switch = 'a'
727             tb = text_tx()
728             tb.start()
729             time.sleep(1)
730             tb.stop()
731             i += 1
732             a1 += 1
733         elif result[i] == 'B':
734             switch = 'b'
735             tb = text_tx()
736             tb.start()
737             time.sleep(1)
738             tb.stop()
739             i += 1
740             b1 += 1
741         elif result[i] == 'C':
742             switch = 'c'
743             tb = text_tx()
744             tb.start()
745             time.sleep(1)
746             tb.stop()
747             i += 1
748             c1 += 1
749         elif result[i] == 'D':
750             switch = 'd'
751             tb = text_tx()
752             tb.start()
753             time.sleep(1)
754             tb.stop()
755             i += 1
756             d1 += 1
757         elif result[i] == 'E':
758             switch = 'e'
759             tb = text_tx()
```

(Image continued from previous page)

```
760     tb.start()
761     time.sleep(1)
762     tb.stop()
763     i += 1
764     e1 += 1
765     elif result[i] == 'F':
766         switch = 'f'
767         tb = text_tx()
768         tb.start()
769         time.sleep(1)
770         tb.stop()
771         i += 1
772         f1 += 1
773     elif result[i] == 'G':
774         switch = 'g'
775         tb = text_tx()
776         tb.start()
777         time.sleep(1)
778         tb.stop()
779         i += 1
780         g1 += 1
781     elif result[i] == 'H':
782         switch = 'h'
783         tb = text_tx()
784         tb.start()
785         time.sleep(1)
786         tb.stop()
787         i += 1
788         h1 += 1
789     elif result[i] == 'I':
790         switch = 'i'
791         tb = text_tx()
792         tb.start()
793         time.sleep(1)
794         tb.stop()
795         i += 1
796         i1 += 1
797     elif result[i] == 'J':
798         switch = 'j'
799         tb = text_tx()
800         tb.start()
801         time.sleep(1)
802         tb.stop()
803         i += 1
804         j1 += 1
805     elif result[i] == 'K':
806         switch = 'k'
807         tb = text_tx()
808         tb.start()
809         time.sleep(1)
810         tb.stop()
811         i += 1
812         k1 += 1
813     elif result[i] == 'L':
814         switch = 'l'
815         tb = text_tx()
816         tb.start()
817         time.sleep(1)
818         tb.stop()
819         i += 1
820         l1 += 1
821     elif result[i] == 'M':
822         switch = 'm'
823         tb = text_tx()
824         tb.start()
825         time.sleep(1)
826         tb.stop()
827         i += 1
828         m1 += 1
829     elif result[i] == 'N':
830         switch = 'n'
831         tb = text_tx()
832         tb.start()
833         time.sleep(1)
```

(Image continued from previous page)

```
834         tb.stop()
835         i += 1
836         n1 += 1
837     else:
838         print("Error: Empty Result")
839     tb = ack2()
840     tb.start()
841     while True:
842         ack_file = open('/home/b/chat_test2.txt', 'r')
843         ack_file.flush()
844         ack_file.readline()
845         ack0 = ack_file.readline()
846         if (ack0 == 'ACK\n'):
847             print("RECEIVING ACK")
848             time.sleep(1)
849             break
850         ack_file.close()
851         time.sleep(1)
852     tb.stop()
853
854     switch = 'eof'
855     tb = text_tx()
856     tb.start()
857     time.sleep(1)
858     tb.stop()
859
860     print('done')
861     exit()
862
863     def quitting():
864         tb.stop()
865         tb.wait()
866     qapp.connect(qapp, Qt.SIGNAL("aboutToQuit()"), quitting)
867     qapp.exec_()
868
869
870 ► if __name__ == '__main__':
871     main()
```

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

```

1  #!/usr/bin/env python2
2  # -*- coding: utf-8 -*-
3  #####
4  # GNU Radio Python Flow Graph
5  # Title: Chat
6  # Generated: Thu Feb 10 09:30:28 2022
7  #####
8
9  if __name__ == '__main__':
10     import ctypes
11     import sys
12     if sys.platform.startswith('linux'):
13         try:
14             x11 = ctypes.cdll.LoadLibrary('libX11.so')
15             x11.XInitThreads()
16         except:
17             print, "Warning: failed to XInitThreads()"
18
19     import ...
20
21     class chat(gr.top_block, Qt.QWidget):
22
23     def __init__(self):
24         gr.top_block.__init__(self, "Chat")
25         Qt.QWidget.__init__(self)
26         self.setWindowTitle("Chat")
27         qtgui.util.check_set_qss()
28         try:
29             self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-gro'))
30         except:
31             pass
32         self.top_scroll_layout = Qt.QVBoxLayout()
33         self.setLayout(self.top_scroll_layout)
34         self.top_scroll = Qt.QScrollArea()
35         self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
36         self.top_scroll_layout.addWidget(self.top_scroll)
37         self.top_scroll.setWidgetResizable(True)
38         self.top_widget = Qt.QWidget()
39         self.top_scroll.setWidget(self.top_widget)
40         self.top_layout = Qt.QVBoxLayout(self.top_widget)
41         self.top_grid_layout = Qt.QGridLayout()
42         self.top_layout.addLayout(self.top_grid_layout)
43
44         self.settings = Qt.QSettings("GNU Radio", "chat")
45         self.restoreGeometry(self.settings.value("geometry").toByteArray())
46
47         #####
48         # Variables
49         #####
50         self.samp_rate = samp_rate = 400000
51
52         #####
53         # Blocks
54         #####
55         self.uhd_usrp_sink_1 = uhd.usrp_sink(
56             ", ".join("", ""),
57             uhd.stream_args(
58                 cpu_format="fc32",
59                 channels=range(1),
60             ),
61         )
62         self.uhd_usrp_sink_1.set_samp_rate(samp_rate)
63         self.uhd_usrp_sink_1.set_time_now(uhd.time_spec(time.time()), uhd.ALL_MBOARDS)
64         self.uhd_usrp_sink_1.set_center_freq(1000000000, 0)
65         self.uhd_usrp_sink_1.set_gain(40, 0)
66         self.uhd_usrp_sink_1.set_antenna('RX2', 0)
67         self.digital_gmsk_mod_0 = digital.gmsk_mod(
68             samples_per_symbol=2,
69             bt=0.35,
70             verbose=False,
71             log=False,
72         )
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91

```

(Image continued from previous page)

```
92 self.blocks_throttle_0 = blocks.throttle(gr.sizeof_char*1, samp_rate,True)
93 self.blocks_multiply_const_vxx_1 = blocks.multiply_const_vcc((1, ))
94 self.blocks_head_0 = blocks.head(gr.sizeof_char*1, 100000)
95 self.blocks_file_source_0 = blocks.file_source(gr.sizeof_char*1, '/home/c/chat3', True)
96 self.blocks_file_source_0.set_begin_tag(pmt.PMT_NIL)
97 self.blks2_packet_encoder_0 = grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
98     samples_per_symbol=2,
99     bits_per_symbol=1,
100     preamble='',
101     access_code='',
102     pad_for_usrp=True,
103     ),
104     payload_length=0,
105 )
106
107
108
109 #####
110 # Connections
111 #####
112 self.connect((self.blks2_packet_encoder_0, 0), (self.digital_gmsk_mod_0, 0))
113 self.connect((self.blocks_file_source_0, 0), (self.blocks_throttle_0, 0))
114 self.connect((self.blocks_head_0, 0), (self.blks2_packet_encoder_0, 0))
115 self.connect((self.blocks_multiply_const_vxx_1, 0), (self.uhd_usrp_sink_1, 0))
116 self.connect((self.blocks_throttle_0, 0), (self.blocks_head_0, 0))
117 self.connect((self.digital_gmsk_mod_0, 0), (self.blocks_multiply_const_vxx_1, 0))
118
119 def closeEvent(self, event):
120     self.settings = Qt.QSettings("GNU Radio", "chat")
121     self.settings.setValue("geometry", self.saveGeometry())
122     event.accept()
123
124 def get_samp_rate(self):
125     return self.samp_rate
126
127 def set_samp_rate(self, samp_rate):
128     self.samp_rate = samp_rate
129     self.uhd_usrp_sink_1.set_samp_rate(self.samp_rate)
130     self.blocks_throttle_0.set_sample_rate(self.samp_rate)
131
132
133 class text_rx(gr.top_block, Qt.QWidget):
134
135     def __init__(self):
136         gr.top_block.__init__(self, "Text Rx")
137         Qt.QWidget.__init__(self)
138         self.setWindowTitle("Text Rx")
139         qtgui.util.check_set_qss()
140         try:
141             self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
142         except:
143             pass
144         self.top_scroll_layout = Qt.QVBoxLayout()
145         self.setLayout(self.top_scroll_layout)
146         self.top_scroll = Qt.QScrollArea()
147         self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
148         self.top_scroll_layout.addWidget(self.top_scroll)
149         self.top_scroll.setWidgetResizable(True)
150         self.top_widget = Qt.QWidget()
151         self.top_scroll.setWidget(self.top_widget)
152         self.top_layout = Qt.QVBoxLayout(self.top_widget)
153         self.top_grid_layout = Qt.QGridLayout()
154         self.top_layout.addLayout(self.top_grid_layout)
155
156         self.settings = Qt.QSettings("GNU Radio", "text_rx")
157         self.restoreGeometry(self.settings.value("geometry").toByteArray())
158
159
160 #####
161 # Variables
162 #####
163 self.samp_rate = samp_rate = 400000
164
165 #####
```

(Image continued from previous page)

```
166 # Blocks
167 #####
168 self.uhd_usrp_source_0 = uhd.usrp_source(
169     ",".join((" ", "")),
170     uhd.stream_args(
171         cpu_format="fc32",
172         channels=range(1),
173     ),
174 )
175 self.uhd_usrp_source_0.set_samp_rate(samp_rate)
176 self.uhd_usrp_source_0.set_center_freq(1000000000, 0)
177 self.uhd_usrp_source_0.set_gain(30, 0)
178 self.uhd_usrp_source_0.set_antenna('RX2', 0)
179 self.qtgui_freq_sink_x_0 = qtgui.freq_sink_c(
180     1024, #size
181     firdes.WIN_BLACKMAN_HARRIS, #wintype
182     0, #fc
183     samp_rate, #bw
184     "", #name
185     1 #number of inputs
186 )
187 self.qtgui_freq_sink_x_0.set_update_time(0.10)
188 self.qtgui_freq_sink_x_0.set_y_axis(-140, 10)
189 self.qtgui_freq_sink_x_0.set_y_label('Relative Gain', 'dB')
190 self.qtgui_freq_sink_x_0.set_trigger_mode(qtgui.TRIG_MODE_FREE, 0.0, 0, "")
191 self.qtgui_freq_sink_x_0.enable_autoscale(False)
192 self.qtgui_freq_sink_x_0.enable_grid(False)
193 self.qtgui_freq_sink_x_0.set_fft_average(1.0)
194 self.qtgui_freq_sink_x_0.enable_axis_labels(True)
195 self.qtgui_freq_sink_x_0.enable_control_panel(False)
196
197 if not True:
198     self.qtgui_freq_sink_x_0.disable_legend()
199
200 if "complex" == "float" or "complex" == "msg_float":
201     self.qtgui_freq_sink_x_0.set_plot_pos_half(not True)
202
203 labels = ['', '', '', '', '',
204           '', '', '', '', '']
205 widths = [1, 1, 1, 1, 1,
206           1, 1, 1, 1, 1]
207 colors = ["blue", "red", "green", "black", "cyan",
208           "magenta", "yellow", "dark red", "dark green", "dark blue"]
209 alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
210           1.0, 1.0, 1.0, 1.0, 1.0]
211 for i in xrange(1):
212     if len(labels[i]) == 0:
213         self.qtgui_freq_sink_x_0.set_line_label(i, "Data (0)".format(i))
214     else:
215         self.qtgui_freq_sink_x_0.set_line_label(i, labels[i])
216         self.qtgui_freq_sink_x_0.set_line_width(i, widths[i])
217         self.qtgui_freq_sink_x_0.set_line_color(i, colors[i])
218         self.qtgui_freq_sink_x_0.set_line_alpha(i, alphas[i])
219
220 self.qtgui_freq_sink_x_0_win = sip.wrapinstance(self.qtgui_freq_sink_x_0.pyqwidget(), Qt.QWidget)
221 self.top_grid_layout.addWidget(self.qtgui_freq_sink_x_0_win)
222 self.low_pass_filter_0 = filter.fir_filter_ccf(1, firdes.low_pass(
223     1, samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))
224 self.digital_gmsk_demod_0 = digital.gmsk_demod(
225     samples_per_symbol=2,
226     gain_mu=0.175,
227     mu=0.5,
228     omega_relative_limit=0.005,
229     freq_error=0.0,
230     verbose=False,
231     log=False,
232 )
233
234 clr_file = open('/home/c/test_file2', 'w')
235 clr_file.close()
236
237 self.blocks_multiply_const_vxx_0 = blocks.multiply_const_vcc((1, ))
238 self.blocks_file_sink_0 = blocks.file_sink(gr.sizeof_char*1, '/home/c/test_file2', False)
239 self.blocks_file_sink_0.set_unbuffered(False)
```


(Image continued from previous page)

```
240     self.blks2_packet_decoder_0 = grc_blks2.packet_demod_b(grc_blks2.packet_decoder(
241         access_code='',
242         threshold=-1,
243         callback=lambda ok, payload: self.blks2_packet_decoder_0.recv_pkt(ok, payload),
244     ),
245 )
246
247
248
249
250     #####
251     # Connections
252     #####
253     self.connect((self.blks2_packet_decoder_0, 0), (self.blocks_file_sink_0, 0))
254     self.connect((self.blocks_multiply_const_vxx_0, 0), (self.low_pass_filter_0, 0))
255     self.connect((self.digital_gmsk_demod_0, 0), (self.blks2_packet_decoder_0, 0))
256     self.connect((self.low_pass_filter_0, 0), (self.digital_gmsk_demod_0, 0))
257     self.connect((self.low_pass_filter_0, 0), (self.qtgui_freq_sink_x_0, 0))
258     self.connect((self.uhd_usrp_source_0, 0), (self.blocks_multiply_const_vxx_0, 0))
259
260
261     def closeEvent(self, event):
262         self.settings = Qt.QSettings("GNU Radio", "text_rx")
263         self.settings.setValue("geometry", self.saveGeometry())
264         event.accept()
265
266     def get_samp_rate(self):
267         return self.samp_rate
268
269     def set_samp_rate(self, samp_rate):
270         self.samp_rate = samp_rate
271         self.uhd_usrp_source_0.set_samp_rate(self.samp_rate)
272         self.qtgui_freq_sink_x_0.set_frequency_range(0, self.samp_rate)
273         self.low_pass_filter_0.set_taps(firdes.low_pass(1, self.samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))
274
275
276     class ack(gr.top_block, Qt.QWidget):
277
278         def __init__(self):
279             gr.top_block.__init__(self, "Chat")
280             Qt.QWidget.__init__(self)
281             self.setWindowTitle("Chat")
282             QtGui.UTIL.check_set_qss()
283             try:
284                 self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-gro'))
285             except:
286                 pass
287             self.top_scroll_layout = Qt.QVBoxLayout()
288             self.setLayout(self.top_scroll_layout)
289             self.top_scroll = Qt.QScrollArea()
290             self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
291             self.top_scroll_layout.addWidget(self.top_scroll)
292             self.top_scroll.setWidgetResizable(True)
293             self.top_widget = Qt.QWidget()
294             self.top_scroll.setWidget(self.top_widget)
295             self.top_layout = Qt.QVBoxLayout(self.top_widget)
296             self.top_grid_layout = Qt.QGridLayout()
297             self.top_layout.addLayout(self.top_grid_layout)
298
299             self.settings = Qt.QSettings("GNU Radio", "chat")
300             self.restoreGeometry(self.settings.value("geometry").toByteArray())
301
302
303     #####
304     # Variables
305     #####
306     self.samp_rate = samp_rate = 400000
307
308     weight_file = open('/home/cj/chat4', 'w')
309     weight_file.write("ACK\n")
310     weight_file.close()
311
312     #####
313     # Blocks
```

(Image continued from previous page)

```
314 #####
315 self.uhd_usrp_sink_1 = uhd.usrp_sink(
316     ", ".join("", ""),
317     uhd.stream_args(
318         cpu_format="fc32",
319         channels=range(1),
320     ),
321 )
322 self.uhd_usrp_sink_1.set_samp_rate(samp_rate)
323 self.uhd_usrp_sink_1.set_time_now(uhd.time_spec(time.time()), uhd.ALL_MBOARDS)
324 self.uhd_usrp_sink_1.set_center_freq(1000000000, 0)
325 self.uhd_usrp_sink_1.set_gain(40, 0)
326 self.uhd_usrp_sink_1.set_antenna('TX/RX', 0)
327 self.digital_gmsk_mod_0 = digital.gmsk_mod(
328     samples_per_symbol=2,
329     bt=0.35,
330     verbose=False,
331     log=False,
332 )
333 self.blocks_throttle_0 = blocks.throttle(gr.sizeof_char*1, samp_rate, True)
334 self.blocks_multiply_const_vxx_1 = blocks.multiply_const_vcc((1,))
335 self.blocks_head_0 = blocks.head(gr.sizeof_char*1, 100000)
336 self.blocks_file_source_0 = blocks.file_source(gr.sizeof_char*1, '/home/c/chat4', True)
337 self.blocks_file_source_0.set_begin_tag(pmt.FMT_NIL)
338 self.blks2_packet_encoder_0 = grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
339     samples_per_symbol=2,
340     bits_per_symbol=1,
341     preamble='',
342     access_code='',
343     pad_for_usrp=True,
344 ),
345     payload_length=0,
346 )
347
348
349
350 #####
351 # Connections
352 #####
353 self.connect((self.blks2_packet_encoder_0, 0), (self.digital_gmsk_mod_0, 0))
354 self.connect((self.blocks_file_source_0, 0), (self.blocks_throttle_0, 0))
355 self.connect((self.blocks_head_0, 0), (self.blks2_packet_encoder_0, 0))
356 self.connect((self.blocks_multiply_const_vxx_1, 0), (self.uhd_usrp_sink_1, 0))
357 self.connect((self.blocks_throttle_0, 0), (self.blocks_head_0, 0))
358 self.connect((self.digital_gmsk_mod_0, 0), (self.blocks_multiply_const_vxx_1, 0))
359
360 def closeEvent(self, event):
361     self.settings = Qt.QSettings("GNU Radio", "chat")
362     self.settings.setValue("geometry", self.saveGeometry())
363     event.accept()
364
365 def get_samp_rate(self):
366     return self.samp_rate
367
368 def set_samp_rate(self, samp_rate):
369     self.samp_rate = samp_rate
370     self.uhd_usrp_sink_1.set_samp_rate(self.samp_rate)
371     self.blocks_throttle_0.set_sample_rate(self.samp_rate)
372
373
374 def main(options=None):
375
376     from distutils.version import StrictVersion
377     if StrictVersion(Qt.qVersion()) >= StrictVersion("4.5.0"):
378         style = gr.prefs().get_string('qtgui', 'style', 'raster')
379         Qt.QApplication.setGraphicsSystem(style)
380     qapp = Qt.QApplication(sys.argv)
381
382     a_num = 0
383     b_num = 0
384     c_num = 0
385     d_num = 0
386     e_num = 0
387     f_num = 0
```

(Image continued from previous page)

```
388 g_num = 0
389 h_num = 0
390 i_num = 0
391 j_num = 0
392 k_num = 0
393 l_num = 0
394 m_num = 0
395 n_num = 0
396 sum_a = 0
397 sum_b = 0
398 sum_c = 0
399 sum_d = 0
400 sum_e = 0
401 sum_f = 0
402 sum_g = 0
403 sum_h = 0
404 sum_i = 0
405 sum_j = 0
406 sum_k = 0
407 sum_l = 0
408 sum_m = 0
409 sum_n = 0
410 predict_a = 0
411 predict_b = 0
412 predict_c = 0
413 predict_d = 0
414 predict_e = 0
415 predict_f = 0
416 predict_g = 0
417 predict_h = 0
418 predict_i = 0
419 predict_j = 0
420 predict_k = 0
421 predict_l = 0
422 predict_m = 0
423 predict_n = 0
424 actual_a_list = []
425 actual_b_list = []
426 actual_c_list = []
427 actual_d_list = []
428 actual_e_list = []
429 actual_f_list = []
430 actual_g_list = []
431 actual_h_list = []
432 actual_i_list = []
433 actual_j_list = []
434 actual_k_list = []
435 actual_l_list = []
436 actual_m_list = []
437 actual_n_list = []
438 predict_a_list = []
439 predict_b_list = []
440 predict_c_list = []
441 predict_d_list = []
442 predict_e_list = []
443 predict_f_list = []
444 predict_g_list = []
445 predict_h_list = []
446 predict_i_list = []
447 predict_j_list = []
448 predict_k_list = []
449 predict_l_list = []
450 predict_m_list = []
451 predict_n_list = []
452
453 while True:
454     weight1 = raw_input("Enter a value for the weight of sensor 1 (integer): ")
455     try:
456         int(weight1)
457         if int(weight1) <= 0:
458             print "Value cannot be negative. Try again."
459             continue
460     except ValueError:
461         print "Value provided needs to be an integer. Try again."
```

(Image continued from previous page)

```
462         continue
463     break
464
465     while True:
466         weight2 = raw_input("Enter a value for the weight of sensor 2 (integer): ")
467         try:
468             int(weight2)
469             if (int(weight2) < 0):
470                 print "Value cannot be negative. Try again."
471                 continue
472         except ValueError:
473             print "Value provided needs to be an integer. Try again."
474             continue
475     break
476
477     while True:
478         weight3 = raw_input("Enter a value for the weight of sensor 3 (integer): ")
479         try:
480             int(weight3)
481             if (int(weight3) < 0):
482                 print "Value cannot be negative. Try again."
483                 continue
484         except ValueError:
485             print "Value provided needs to be an integer. Try again."
486             continue
487     break
488
489     while True:
490         weight4 = raw_input("Enter a value for the weight of sensor 4 (integer): ")
491         try:
492             int(weight4)
493             if (int(weight4) < 0):
494                 print "Value cannot be negative. Try again."
495                 continue
496         except ValueError:
497             print "Value provided needs to be an integer. Try again."
498             continue
499     break
500
501     while True:
502         weight5 = raw_input("Enter a value for the weight of sensor 5 (integer): ")
503         try:
504             int(weight5)
505             if (int(weight5) < 0):
506                 print "Value cannot be negative. Try again."
507                 continue
508         except ValueError:
509             print "Value provided needs to be an integer. Try again."
510             continue
511     break
512
513     while True:
514         weight6 = raw_input("Enter a value for the weight of sensor 6 (integer): ")
515         try:
516             int(weight6)
517             if (int(weight6) < 0):
518                 print "Value cannot be negative. Try again."
519                 continue
520         except ValueError:
521             print "Value provided needs to be an integer. Try again."
522             continue
523     break
524
525     while True:
526         weight7 = raw_input("Enter a value for the weight of sensor 7 (integer): ")
527         try:
528             int(weight7)
529             if (int(weight7) < 0):
530                 print "Value cannot be negative. Try again."
531                 continue
532         except ValueError:
533             print "Value provided needs to be an integer. Try again."
534             continue
535     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)

```
610     weight14 = raw_input("Enter a value for the weight of sensor 14 (integer): ")
611     try:
612         int(weight14)
613         if int(weight14) < 0:
614             print "Value cannot be negative. Try again."
615             continue
616     except ValueError:
617         print "Value provided needs to be an integer. Try again."
618         continue
619     break
620
621     weight_file = open('/home/c/chat3', 'w')
622
623     weight_file.write(weight1 + ',')
624     weight_file.write(weight2 + ',')
625     weight_file.write(weight3 + ',')
626     weight_file.write(weight4 + ',')
627     weight_file.write(weight5 + ',')
628     weight_file.write(weight6 + ',')
629     weight_file.write(weight7 + ',')
630     weight_file.write(weight8 + ',')
631     weight_file.write(weight9 + ',')
632     weight_file.write(weight10 + ',')
633     weight_file.write(weight11 + ',')
634     weight_file.write(weight12 + ',')
635     weight_file.write(weight13 + ',')
636     weight_file.write(weight14 + ',\n')
637
638     weight_file.close()
639
640     tb = chat()
641     tb.start()
642     time.sleep(1)
643     tb.stop()
644
645     final = open('/home/c/final', 'w')
646     final.close()
647
648     packet_tot = 0
649
650     change0_list = []
651     change1_list = []
652     change2_list = []
653     change3_list = []
654     change4_list = []
655     change5_list = []
656     change6_list = []
657     change7_list = []
658     change8_list = []
659     change9_list = []
660     change10_list = []
661     change11_list = []
662     change12_list = []
663     change13_list = []
664     sum_change0 = 0
665     sum_change1 = 0
666     sum_change2 = 0
667     sum_change3 = 0
668     sum_change4 = 0
669     sum_change5 = 0
670     sum_change6 = 0
671     sum_change7 = 0
672     sum_change8 = 0
673     sum_change9 = 0
674     sum_change10 = 0
675     sum_change11 = 0
676     sum_change12 = 0
677     sum_change13 = 0
678
679     while True:
680         tb = text_rx()
681         tb.start()
682         data1 = open('/home/c/test_file2', 'r+')
683         final = open('/home/c/final', 'a')
```

(Image continued from previous page)

```
684 while True:
685     diff = 0
686     data1.flush()
687     data1.readline()
688     file_content2 = data1.readline()
689     if (file_content2 == "eof\n"):
690         print(file_content2)
691         break
692     elif (file_content2 != ""):
693         packet_tot += 1
694         final.write(file_content2)
695         split = file_content2.split(' ')
696         if (split[1].startswith('10') and (split[1].startswith('0', 2) == False)):
697             a_num += 1
698             actual_a_list.append(float(split[0]))
699             if (a_num > 1):
700                 diff = abs(predict_a_list[-1]-actual_a_list[-1])
701                 print(diff)
702                 if (diff > 2*int(weight1)):
703                     weight1 = int(weight1)+1
704                 elif (int(weight1) > 1) and (diff < int(weight1)):
705                     weight1 = int(weight1)-1
706                 print(weight1)
707             if (a_num > 1):
708                 change0 = actual_a_list[-1]-actual_a_list[-2]
709                 sum_change0 += change0
710                 predict_a = actual_a_list[-1] + sum_change0/a_num
711             else:
712                 predict_a = actual_a_list[-1]
713             print("Prediction for Next A: "+str(predict_a))
714             predict_a_list.append(predict_a)
715         elif (split[1].startswith('20')):
716             b_num += 1
717             actual_b_list.append(float(split[0]))
718             if (b_num > 1):
719                 diff = abs(predict_b_list[-1]-actual_b_list[-1])
720                 print(diff)
721                 if (diff > 2*int(weight2)):
722                     weight2 = int(weight2)+1
723                 elif (int(weight2) > 1) and (diff < int(weight2)):
724                     weight2 = int(weight2)-1
725                 print(weight2)
726             if (b_num > 1):
727                 change1 = actual_b_list[-1]-actual_b_list[-2]
728                 sum_change1 += change1
729                 predict_b = actual_b_list[-1] + sum_change1/b_num
730             else:
731                 predict_b = actual_b_list[-1]
732             print("Prediction for Next B: "+str(predict_b))
733             predict_b_list.append(predict_b)
734         elif (split[1].startswith('30')):
735             c_num += 1
736             actual_c_list.append(float(split[0]))
737             if (c_num > 1):
738                 diff = abs(predict_c_list[-1]-actual_c_list[-1])
739                 print(diff)
740                 if (diff > 2*int(weight3)):
741                     weight3 = int(weight3)+1
742                 elif (int(weight3) > 1) and (diff < int(weight3)):
743                     weight3 = int(weight3)-1
744                 print(weight3)
745             if (c_num > 1):
746                 change2 = actual_c_list[-1]-actual_c_list[-2]
747                 sum_change2 += change2
748                 predict_c = actual_c_list[-1] + sum_change2/c_num
749             else:
750                 predict_c = actual_c_list[-1]
751             print("Prediction for Next C: "+str(predict_c))
752             predict_c_list.append(predict_c)
753         elif (split[1].startswith('40')):
754             d_num += 1
755             actual_d_list.append(float(split[0]))
756             if (d_num > 1):
757                 diff = abs(predict d list[-1]-actual d list[-1])
```

(Image continued from previous page)

```
758     print(diff)
759     if (diff > 2*int(weight4)):
760         weight4 = int(weight4)+1
761     elif (int(weight4) > 1) and (diff < int(weight4)):
762         weight4 = int(weight4)-1
763     print(weight4)
764
765     if (d_num > 1):
766         change3 = actual_d_list[-1]-actual_d_list[-2]
767         sum_change3 += change3
768         predict_d = actual_d_list[-1] + sum_change3/d_num
769     else:
770         predict_d = actual_d_list[-1]
771     print("Prediction for Next D: "+str(predict_d))
772     predict_d_list.append(predict_d)
773 elif (split[1].startswith('50')):
774     e_num += 1
775     actual_e_list.append(float(split[0]))
776     if (e_num > 1):
777         diff = abs(predict_e_list[-1]-actual_e_list[-1])
778         print(diff)
779         if (diff > 2*int(weight5)):
780             weight5 = int(weight5)+1
781         elif (int(weight5) > 1) and (diff < int(weight5)):
782             weight5 = int(weight5)-1
783         print(weight5)
784     if (e_num > 1):
785         change4 = actual_e_list[-1]-actual_e_list[-2]
786         sum_change4 += change4
787         predict_e = actual_e_list[-1] + sum_change4/e_num
788     else:
789         predict_e = actual_e_list[-1]
790     print("Prediction for Next E: "+str(predict_e))
791     predict_e_list.append(predict_e)
792 elif (split[1].startswith('60')):
793     f_num += 1
794     actual_f_list.append(float(split[0]))
795     if (f_num > 1):
796         diff = abs(predict_f_list[-1]-actual_f_list[-1])
797         print(diff)
798         if (diff > 2*int(weight6)):
799             weight6 = int(weight6)+1
800         elif (int(weight6) > 1) and (diff < int(weight6)):
801             weight6 = int(weight6)-1
802         print(weight6)
803     if (f_num > 1):
804         change5 = actual_f_list[-1]-actual_f_list[-2]
805         sum_change5 += change5
806         predict_f = actual_f_list[-1] + sum_change5/f_num
807     else:
808         predict_f = actual_f_list[-1]
809     print("Prediction for Next F: "+str(predict_f))
810     predict_f_list.append(predict_f)
811 elif (split[1].startswith('70')):
812     g_num += 1
813     actual_g_list.append(float(split[0]))
814     if (g_num > 1):
815         diff = abs(predict_g_list[-1]-actual_g_list[-1])
816         print(diff)
817         if (diff > 2*int(weight7)):
818             weight7 = int(weight7)+1
819         elif (int(weight7) > 1) and (diff < int(weight7)):
820             weight7 = int(weight7)-1
821         print(weight7)
822     if (g_num > 1):
823         change6 = actual_g_list[-1]-actual_g_list[-2]
824         sum_change6 += change6
825         predict_g = actual_g_list[-1] + sum_change6/g_num
826     else:
827         predict_g = actual_g_list[-1]
828     print("Prediction for Next G: "+str(predict_g))
829     predict_g_list.append(predict_g)
830 elif (split[1].startswith('80')):
831     h_num += 1
832     actual_h_list.append(float(split[0]))
```


(Image continued from previous page)

```
832 if (h_num > 1):
833     diff = abs(predict_h_list[-1]-actual_h_list[-1])
834     print(diff)
835     if (diff > 2*int(weight8)):
836         weight8 = int(weight8)+1
837     elif (int(weight8) > 1) and (diff < int(weight8)):
838         weight8 = int(weight8)-1
839     print(weight8)
840 if (h_num > 1):
841     change7 = actual_h_list[-1]-actual_h_list[-2]
842     sum_change7 += change7
843     predict_h = actual_h_list[-1] + sum_change7/h_num
844 else:
845     predict_h = actual_h_list[-1]
846 print("Prediction for Next H: "+str(predict_h))
847 predict_h_list.append(predict_h)
848 elif (split[1].startswith('90')):
849     i_num += 1
850     actual_i_list.append(float(split[0]))
851 if (i_num > 1):
852     diff = abs(predict_i_list[-1]-actual_i_list[-1])
853     print(diff)
854     if (diff > 2*int(weight9)):
855         weight9 = int(weight9)+1
856     elif (int(weight9) > 1) and (diff < int(weight9)):
857         weight9 = int(weight9)-1
858     print(weight9)
859 if (i_num > 1):
860     change8 = actual_i_list[-1]-actual_i_list[-2]
861     sum_change8 += change8
862     predict_i = actual_i_list[-1] + sum_change8/i_num
863 else:
864     predict_i = actual_i_list[-1]
865 print("Prediction for Next I: "+str(predict_i))
866 predict_i_list.append(predict_i)
867 elif (split[1].startswith('100')):
868     j_num += 1
869     actual_j_list.append(float(split[0]))
870 if (j_num > 1):
871     diff = abs(predict_j_list[-1]-actual_j_list[-1])
872     print(diff)
873     if (diff > 2*int(weight10)):
874         weight10 = int(weight10)+1
875     elif (int(weight10) > 1) and (diff < int(weight10)):
876         weight10 = int(weight10)-1
877     print(weight10)
878 if (j_num > 1):
879     change9 = actual_j_list[-1]-actual_j_list[-2]
880     sum_change9 += change9
881     predict_j = actual_j_list[-1] + sum_change9/j_num
882 else:
883     predict_j = actual_j_list[-1]
884 print("Prediction for Next J: "+str(predict_j))
885 predict_j_list.append(predict_j)
886 elif (split[1].startswith('110')):
887     k_num += 1
888     actual_k_list.append(float(split[0]))
889 if (k_num > 1):
890     diff = abs(predict_k_list[-1]-actual_k_list[-1])
891     print(diff)
892     if (diff > 2*int(weight11)):
893         weight11 = int(weight11)+1
894     elif (int(weight11) > 1) and (diff < int(weight11)):
895         weight11 = int(weight11)-1
896     print(weight11)
897 if (k_num > 1):
898     change10 = actual_k_list[-1]-actual_k_list[-2]
899     sum_change10 += change10
900     predict_k = actual_k_list[-1] + sum_change10/k_num
901 else:
902     predict_k = actual_k_list[-1]
903 print("Prediction for Next K: "+str(predict_k))
904 predict_k_list.append(predict_k)
905 elif (split[1].startswith('120')):
```

(Image continued from previous page)

```
906 l_num += 1
907 actual_l_list.append(float(split[0]))
908 if (l_num > 1):
909     diff = abs(predict_l_list[-1]-actual_l_list[-1])
910     print(diff)
911     if (diff > 2*int(weight12)):
912         weight12 = int(weight12)+1
913     elif (int(weight12) > 1) and (diff < int(weight12)):
914         weight12 = int(weight12)-1
915     print(weight12)
916 if (l_num > 1):
917     change11 = actual_l_list[-1]-actual_l_list[-2]
918     sum_change11 += change11
919     predict_l = actual_l_list[-1] + sum_change11/l_num
920 else:
921     predict_l = actual_l_list[-1]
922 print("Prediction for Next L: "+str(predict_l))
923 predict_l_list.append(predict_l)
924 elif (split[1].startswith('130')):
925     m_num += 1
926     actual_m_list.append(float(split[0]))
927     if (m_num > 1):
928         diff = abs(predict_m_list[-1]-actual_m_list[-1])
929         print(diff)
930         if (diff > 2*int(weight13)):
931             weight13 = int(weight13)+1
932         elif (int(weight13) > 1) and (diff < int(weight13)):
933             weight13 = int(weight13)-1
934         print(weight13)
935     if (m_num > 1):
936         change12 = actual_m_list[-1]-actual_m_list[-2]
937         sum_change12 += change12
938         predict_m = actual_m_list[-1] + sum_change12/m_num
939     else:
940         predict_m = actual_m_list[-1]
941     print("Prediction for Next M: "+str(predict_m))
942     predict_m_list.append(predict_m)
943 elif (split[1].startswith('140')):
944     n_num += 1
945     actual_n_list.append(float(split[0]))
946     if (n_num > 1):
947         diff = abs(predict_n_list[-1]-actual_n_list[-1])
948         print(diff)
949         if (diff > 2*int(weight14)):
950             weight14 = int(weight14)+1
951         elif (int(weight14) > 1) and (diff < int(weight14)):
952             weight14 = int(weight14)-1
953         print(weight14)
954     if (n_num > 1):
955         change13 = actual_n_list[-1]-actual_n_list[-2]
956         sum_change13 += change13
957         predict_n = actual_n_list[-1] + sum_change13/n_num
958     else:
959         predict_n = actual_n_list[-1]
960     print("Prediction for Next N: "+str(predict_n))
961     predict_n_list.append(predict_n)
962 print("Current Value: "+file_content2)
963 time.sleep(1)
964 sb = ack()
965 sb.start()
966 print("SENDING ACK")
967 time.sleep(1)
968 sb.stop()
969 if (packet_tot > 28) and (packet_tot % 20 == 0):
970     weight_file = open('/home/c/ghat3', 'w')
971     weight_file.write(str(weight1) + ',')
972     weight_file.write(str(weight2) + ',')
973     weight_file.write(str(weight3) + ',')
974     weight_file.write(str(weight4) + ',')
975     weight_file.write(str(weight5) + ',')
976     weight_file.write(str(weight6) + ',')
977     weight_file.write(str(weight7) + ',')
978     weight_file.write(str(weight8) + ',')
979     weight_file.write(str(weight9) + ',')
```

(Image continued from previous page)

```
980         weight_file.write(str(weight10) + ',')
981         weight_file.write(str(weight11) + ',')
982         weight_file.write(str(weight12) + ',')
983         weight_file.write(str(weight13) + ',')
984         weight_file.write(str(weight14) + ',\n')
985         weight_file.close()
986         up = chat()
987         up.start()
988         time.sleep(1)
989         up.stop()
990     break
991     time.sleep(1)
992
993     tb.stop()
994
995     if (file_content2 == "eof\n"):
996         data1.close()
997         final.close()
998         break
999
1000     data1.close()
1001     final.close()
1002
1003     tb.stop()
1004
1005     sum_sq_a = 0
1006     for z in range(0, len(predict_a_list)-1):
1007         square = (actual_a_list[z+1] - predict_a_list[z])**2
1008         sum_sq_a += square
1009     msqrterr_a = math.sqrt(sum_sq_a)
1010     print("Root Mean Squared Error for Sensor 1: "+str(msqrterr_a))
1011
1012     sum_sq_b = 0
1013     for z in range(0, len(predict_b_list)-1):
1014         square = (actual_b_list[z+1] - predict_b_list[z])**2
1015         sum_sq_b += square
1016     msqrterr_b = math.sqrt(sum_sq_b)
1017     print("Root Mean Squared Error for Sensor 2: "+str(msqrterr_b))
1018
1019     sum_sq_c = 0
1020     for z in range(0, len(predict_c_list)-1):
1021         square = (actual_c_list[z+1] - predict_c_list[z])**2
1022         sum_sq_c += square
1023     msqrterr_c = math.sqrt(sum_sq_c)
1024     print("Root Mean Squared Error for Sensor 3: "+str(msqrterr_c))
1025
1026     sum_sq_d = 0
1027     for z in range(0, len(predict_d_list)-1):
1028         square = (actual_d_list[z+1] - predict_d_list[z])**2
1029         sum_sq_d += square
1030     msqrterr_d = math.sqrt(sum_sq_d)
1031     print("Root Mean Squared Error for Sensor 4: "+str(msqrterr_d))
1032
1033     sum_sq_e = 0
1034     for z in range(0, len(predict_e_list)-1):
1035         square = (actual_e_list[z+1] - predict_e_list[z])**2
1036         sum_sq_e += square
1037     msqrterr_e = math.sqrt(sum_sq_e)
1038     print("Root Mean Squared Error for Sensor 5: "+str(msqrterr_e))
1039
1040     sum_sq_f = 0
1041     for z in range(0, len(predict_f_list)-1):
1042         square = (actual_f_list[z+1] - predict_f_list[z])**2
1043         sum_sq_f += square
1044     msqrterr_f = math.sqrt(sum_sq_f)
1045     print("Root Mean Squared Error for Sensor 6: "+str(msqrterr_f))
1046
1047     sum_sq_g = 0
1048     for z in range(0, len(predict_g_list)-1):
1049         square = (actual_g_list[z+1] - predict_g_list[z])**2
1050         sum_sq_g += square
1051     msqrterr_g = math.sqrt(sum_sq_g)
1052     print("Root Mean Squared Error for Sensor 7: "+str(msqrterr_g))
1053
```

(Image continued from previous page)

```
1054     sum_sq_h = 0
1055     for z in range(0, len(predict_h_list)-1):
1056         square = (actual_h_list[z+1] - predict_h_list[z])**2
1057         sum_sq_h += square
1058     msqrterr_h = math.sqrt(sum_sq_h)
1059     print("Root Mean Squared Error for Sensor 8: "+str(msqrterr_h))
1060
1061     sum_sq_i = 0
1062     for z in range(0, len(predict_i_list)-1):
1063         square = (actual_i_list[z+1] - predict_i_list[z])**2
1064         sum_sq_i += square
1065     msqrterr_i = math.sqrt(sum_sq_i)
1066     print("Root Mean Squared Error for Sensor 9: "+str(msqrterr_i))
1067
1068     sum_sq_j = 0
1069     for z in range(0, len(predict_j_list)-1):
1070         square = (actual_j_list[z+1] - predict_j_list[z])**2
1071         sum_sq_j += square
1072     msqrterr_j = math.sqrt(sum_sq_j)
1073     print("Root Mean Squared Error for Sensor 10: "+str(msqrterr_j))
1074
1075     sum_sq_k = 0
1076     for z in range(0, len(predict_k_list)-1):
1077         square = (actual_k_list[z+1] - predict_k_list[z])**2
1078         sum_sq_k += square
1079     msqrterr_k = math.sqrt(sum_sq_k)
1080     print("Root Mean Squared Error for Sensor 11: "+str(msqrterr_k))
1081
1082     sum_sq_l = 0
1083     for z in range(0, len(predict_l_list)-1):
1084         square = (actual_l_list[z+1] - predict_l_list[z])**2
1085         sum_sq_l += square
1086     msqrterr_l = math.sqrt(sum_sq_l)
1087     print("Root Mean Squared Error for Sensor 12: "+str(msqrterr_l))
1088
1089     sum_sq_m = 0
1090     for z in range(0, len(predict_m_list)-1):
1091         square = (actual_m_list[z+1] - predict_m_list[z])**2
1092         sum_sq_m += square
1093     msqrterr_m = math.sqrt(sum_sq_m)
1094     print("Root Mean Squared Error for Sensor 13: "+str(msqrterr_m))
1095
1096     sum_sq_n = 0
1097     for z in range(0, len(predict_n_list)-1):
1098         square = (actual_n_list[z+1] - predict_n_list[z])**2
1099         sum_sq_n += square
1100     msqrterr_n = math.sqrt(sum_sq_n)
1101     print("Root Mean Squared Error for Sensor 14: "+str(msqrterr_n))
1102
1103     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_l+msqrterr_m+msqrterr_n
1104     print("\nTotal Root Mean Squared Error: "+str(rmse_tot))
1105
1106     exit()
1107
1108     def quitting():
1109         tb.stop()
1110         tb.wait()
1111     qapp.connect(qapp, Qt.SIGNAL("aboutToQuit()"), quitting)
1112     qapp.exec_()
1113
1114
1115     if __name__ == '__main__':
1116         main()
```

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

```

1  #!/usr/bin/env python2
2  # -*- coding: utf-8 -*-
3  #####
4  # GNU Radio Python Flow Graph
5  # Title: Chat2
6  # Generated: Thu Feb 10 09:52:14 2022
7  #####
8
9  if __name__ == '__main__':
10     import ctypes
11     import sys
12     if sys.platform.startswith('linux'):
13         try:
14             x11 = ctypes.cdll.LoadLibrary('libX11.so')
15             x11.XInitThreads()
16         except:
17             print "Warning: failed to XInitThreads()"
18
19     import ...
20
21     class chat2(gr.top_block, Qt.QWidget):
22
23     def __init__(self):
24         gr.top_block.__init__(self, "Chat2")
25         Qt.QWidget.__init__(self)
26         self.setWindowTitle("Chat2")
27         qtgui.util.check_set_qss()
28         try:
29             self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
30         except:
31             pass
32         self.top_scroll_layout = Qt.QVBoxLayout()
33         self.setLayout(self.top_scroll_layout)
34         self.top_scroll = Qt.QScrollArea()
35         self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
36         self.top_scroll_layout.addWidget(self.top_scroll)
37         self.top_scroll.setWidgetResizable(True)
38         self.top_widget = Qt.QWidget()
39         self.top_scroll.setWidget(self.top_widget)
40         self.top_layout = Qt.QVBoxLayout(self.top_widget)
41         self.top_grid_layout = Qt.QGridLayout()
42         self.top_layout.addLayout(self.top_grid_layout)
43
44         self.settings = Qt.QSettings("GNU Radio", "chat2")
45         self.restoreGeometry(self.settings.value("geometry").toByteArray())
46
47         #####
48         # Variables
49         #####
50         self.samp_rate = samp_rate = 400000
51
52         #####
53         # Blocks
54         #####
55         self.uhd_usrp_source_0 = uhd.usrp_source(
56             ", ".join("", ""),
57             uhd.stream_args(
58                 cpu_format="fc32",
59                 channels=range(1),
60             ),
61         )
62         self.uhd_usrp_source_0.set_samp_rate(samp_rate)
63         self.uhd_usrp_source_0.set_center_freq(1000000000, 0)
64         self.uhd_usrp_source_0.set_gain(30, 0)
65         self.uhd_usrp_source_0.set_antenna('TX/RX', 0)
66         self.qtgui_freq_sink_x_0 = qtgui.freq_sink_c(
67             1024, #size
68             firdes.WIN_BLACKMAN_HARRIS, #vintype
69             0, #fc
70             samp_rate, #bw
71             "", #name
72             1 #number of inputs

```

(Image continued from previous page)

```
94     )
95     self.qtgui_freq_sink_x_0.set_update_time(0.10)
96     self.qtgui_freq_sink_x_0.set_y_axis(-140, 10)
97     self.qtgui_freq_sink_x_0.set_y_label('Relative Gain', 'dB')
98     self.qtgui_freq_sink_x_0.set_trigger_mode(qtgui.TRIG_MODE_FREE, 0.0, 0, "")
99     self.qtgui_freq_sink_x_0.enable_autoscale(False)
100    self.qtgui_freq_sink_x_0.enable_grid(False)
101    self.qtgui_freq_sink_x_0.set_fft_average(1.0)
102    self.qtgui_freq_sink_x_0.enable_axis_labels(True)
103    self.qtgui_freq_sink_x_0.enable_control_panel(False)
104
105    if not True:
106        self.qtgui_freq_sink_x_0.disable_legend()
107
108    if "complex" == "float" or "complex" == "msg_float":
109        self.qtgui_freq_sink_x_0.set_plot_pos_half(not True)
110
111    labels = ['', '', '', '', '',
112             '', '', '', '', '']
113    widths = [1, 1, 1, 1, 1,
114             1, 1, 1, 1, 1]
115    colors = ["blue", "red", "green", "black", "cyan",
116             "magenta", "yellow", "dark red", "dark green", "dark blue"]
117    alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
118             1.0, 1.0, 1.0, 1.0, 1.0]
119    for i in xrange(1):
120        if len(labels[i]) == 0:
121            self.qtgui_freq_sink_x_0.set_line_label(i, "Data (0)".format(i))
122        else:
123            self.qtgui_freq_sink_x_0.set_line_label(i, labels[i])
124        self.qtgui_freq_sink_x_0.set_line_width(i, widths[i])
125        self.qtgui_freq_sink_x_0.set_line_color(i, colors[i])
126        self.qtgui_freq_sink_x_0.set_line_alpha(i, alphas[i])
127
128    self.qtgui_freq_sink_x_0_win = sip.wrapinstance(self.qtgui_freq_sink_x_0.pyqwidget(), Qt.CWidget)
129    self.top_grid_layout.addWidget(self.qtgui_freq_sink_x_0_win)
130    self.low_pass_filter_0 = filter.fir_filter_ccf(1, firdes.low_pass(
131        1, samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))
132    self.digital_gmsk_demod_1 = digital.gmsk_demod(
133        samples_per_symbol=2,
134        gain_mu=0.175,
135        mu=0.5,
136        omega_relative_limit=0.005,
137        freq_error=0.0,
138        verbose=False,
139        log=False,
140    )
141    self.blocks_multiply_const_vxx_1 = blocks.multiply_const_vcc((1, ))
142    self.blocks_file_sink_1 = blocks.file_sink(gr.sizeof_char*1, '/home/b/chat_test1.txt', False)
143    self.blocks_file_sink_1.set_unbuffered(False)
144    self.blks2_packet_decoder_1 = grc_blks2.packet_demod_b(grc_blks2.packet_decoder(
145        access_code='',
146        threshold=-1,
147        callback=lambda ok, payload: self.blks2_packet_decoder_1.recv_pkt(ok, payload),
148    ),
149    ),
150    )
151
152
153    #####
154    # Connections
155    #####
156    self.connect((self.blks2_packet_decoder_1, 0), (self.blocks_file_sink_1, 0))
157    self.connect((self.blocks_multiply_const_vxx_1, 0), (self.low_pass_filter_0, 0))
158    self.connect((self.digital_gmsk_demod_1, 0), (self.blks2_packet_decoder_1, 0))
159    self.connect((self.low_pass_filter_0, 0), (self.digital_gmsk_demod_1, 0))
160    self.connect((self.low_pass_filter_0, 0), (self.qtgui_freq_sink_x_0, 0))
161    self.connect((self.uhd_usrp_source_0, 0), (self.blocks_multiply_const_vxx_1, 0))
162
163    def closeEvent(self, event):
164        self.settings = Qt.QSettings("GNU Radio", "chat2")
165        self.settings.setValue("geometry", self.saveGeometry())
166        event.accept()
167
```

(Image continued from previous page)

```
168 def get_samp_rate(self):
169     return self.samp_rate
170
171 def set_samp_rate(self, samp_rate):
172     self.samp_rate = samp_rate
173     self.uhd_usrp_source_0.set_samp_rate(self.samp_rate)
174     self.qtgui_freq_sink_x_0.set_frequency_range(0, self.samp_rate)
175     self.low_pass_filter_0.set_taps(firdes.low_pass(1, self.samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))
176
177
178 class text_tx(gr.top_block):
179
180     def __init__(self):
181         gr.top_block.__init__(self, "Text Tx")
182
183         #####
184         # Variables
185         #####
186         self.samp_rate = samp_rate = 400000
187         source = ''
188
189         #####
190         # Blocks
191         #####
192         self.uhd_usrp_sink_0 = uhd.usrp_sink(
193             ", ".join("", ""),
194             uhd.stream_args(
195                 cpu_format="fc32",
196                 channels=range(1),
197             ),
198         )
199         self.uhd_usrp_sink_0.set_samp_rate(samp_rate)
200         self.uhd_usrp_sink_0.set_time_now(uhd.time_spec(time.time()), uhd.ALL_MBOARDS)
201         self.uhd_usrp_sink_0.set_center_freq(1000000000, 0)
202         self.uhd_usrp_sink_0.set_gain(40, 0)
203         self.uhd_usrp_sink_0.set_antenna('TX/RX', 0)
204         self.digital_gmsk_mod_0 = digital.gmsk_mod(
205             samples_per_symbol=2,
206             bt=0.35,
207             verbose=False,
208             log=False,
209         )
210         self.blocks_throttle_0 = blocks.throttle(gr.sizeof_char*1, samp_rate,True)
211         self.blocks_multiply_const_vxx_0 = blocks.multiply_const_vcc((1, ))
212         self.blocks_head_0 = blocks.head(gr.sizeof_char*1, samp_rate/4)
213
214         send = open('/home/b/tx.txt', 'w')
215         if switch == 'a':
216             send.write(data0[a1])
217             print(data0[a1])
218         elif switch == 'b':
219             send.write(data1[b1])
220             print(data1[b1])
221         elif switch == 'c':
222             send.write(data2[c1])
223             print(data2[c1])
224         elif switch == 'd':
225             send.write(data3[d1])
226             print(data3[d1])
227         elif switch == 'e':
228             send.write(data4[e1])
229             print(data4[e1])
230         elif switch == 'f':
231             send.write(data5[f1])
232             print(data5[f1])
233         elif switch == 'g':
234             send.write(data6[g1])
235             print(data6[g1])
236         elif switch == 'h':
237             send.write(data7[h1])
238             print(data7[h1])
239         elif switch == 'i':
240             send.write(data8[i1])
241             print(data8[i1])
```

(Image continued from previous page)

```
242 elif switch == 'j':
243     send.write(data9[j1])
244     print(data9[j1])
245 elif switch == 'k':
246     send.write(data10[k1])
247     print(data10[k1])
248 elif switch == 'l':
249     send.write(data11[l1])
250     print(data11[l1])
251 elif switch == 'm':
252     send.write(data12[m1])
253     print(data12[m1])
254 elif switch == 'n':
255     send.write(data13[n1])
256     print(data13[n1])
257 elif switch == 'eof':
258     send.write("eof\n")
259     print,"eof"
260 send.close()
261
262 self.blocks_file_source_0 = blocks.file_source(gr.sizeof_char*1, '/home/b/tx.txt', True)
263
264 self.blocks_file_source_0.set_begin_tag(pmt.PMT_NIL)
265 self.blks2_packet_encoder_0 = grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
266     samples_per_symbol=2,
267     bits_per_symbol=1,
268     preamble='',
269     access_code='',
270     pad_for_usrp=True,
271 ),
272     payload_length=0,
273 )
274
275
276
277 #####
278 # Connections
279 #####
280 self.connect((self.blks2_packet_encoder_0, 0), (self.digital_gmsk_mod_0, 0))
281 self.connect((self.blocks_file_source_0, 0), (self.blocks_throttle_0, 0))
282 self.connect((self.blocks_head_0, 0), (self.blks2_packet_encoder_0, 0))
283 self.connect((self.blocks_multiply_const_vxx_0, 0), (self.uhd_usrp_sink_0, 0))
284 self.connect((self.blocks_throttle_0, 0), (self.blocks_head_0, 0))
285 self.connect((self.digital_gmsk_mod_0, 0), (self.blocks_multiply_const_vxx_0, 0))
286
287 def get_samp_rate(self):
288     return self.samp_rate
289
290 def set_samp_rate(self, samp_rate):
291     self.samp_rate = samp_rate
292     self.uhd_usrp_sink_0.set_samp_rate(self.samp_rate)
293     self.blocks_throttle_0.set_sample_rate(self.samp_rate)
294     self.blocks_head_0.set_length(self.samp_rate/4)
295
296 class WRRScheduler():
297
298     cw = 0
299     i = -1
300     data_set = []
301     max_s = None
302     gcd_s = None
303     len_s = None
304     counter = {}
305
306 def __init__(self, s = None):
307     self._init_dataset(s)
308
309 def _init_dataset(self, s):
310     self.data_set = s
311     self.max_s = max(s, key=lambda x: x[1])[1]
312     self.gcd_s = reduce(fractions.gcd, [ weight for data, weight in s])
313     self.len_s = len(s)
```


(Image continued from previous page)

```
316 def schedule(self):
317     while True:
318         self.i = (self.i + 1) % self.len_s
319         if self.i == 0:
320             self.cw = self.cw - self.gcd_s
321             if self.cw <= 0:
322                 self.cw = self.max_s
323                 if self.cw == 0:
324                     return None
325             if self.data_set[self.i][1] >= self.cw:
326                 self._inc_counter(self.data_set[self.i])
327                 return self.data_set[self.i]
328
329 def _inc_counter(self, item):
330     try:
331         self.counter[item[0]] += 1
332     except KeyError:
333         self.counter[item[0]] = 1
334
335 def set_data(self, s):
336     self.reset()
337     self._init_dataset(s)
338
339 def reset_counter(self):
340     self.counter = {}
341
342 def reset(self):
343     self.cw = 0
344     self.i = -1
345     self.data_set = []
346     self.max_s = None
347     self.gcd_s = None
348     self.len_s = None
349     self.reset_counter()
350
351 def get_next(self, n = 1):
352     if n > 1:
353         return [ self.schedule() for i in range(0,n) ]
354     return self.schedule()
355
356
357 class ack2(gr.top_block, Qt.QWidget):
358
359     def __init__(self):
360         gr.top_block.__init__(self, "Chat2")
361         Qt.QWidget.__init__(self)
362         self.setWindowTitle("Chat2")
363         qtgui.util.check_set_qss()
364         try:
365             self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-gre'))
366         except:
367             pass
368         self.top_scroll_layout = Qt.QVBoxLayout()
369         self.setLayout(self.top_scroll_layout)
370         self.top_scroll = Qt.QScrollArea()
371         self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
372         self.top_scroll_layout.addWidget(self.top_scroll)
373         self.top_scroll.setWidgetResizable(True)
374         self.top_widget = Qt.QWidget()
375         self.top_scroll.setWidget(self.top_widget)
376         self.top_layout = Qt.QVBoxLayout(self.top_widget)
377         self.top_grid_layout = Qt.QGridLayout()
378         self.top_layout.addLayout(self.top_grid_layout)
379
380         self.settings = Qt.QSettings("GNU Radio", "chat2")
381         self.restoreGeometry(self.settings.value("geometry").toByteArray())
382
383
384     #####
385     # Variables
386     #####
387     self.samp_rate = samp_rate = 400000
388
389     #####
```

(Image continued from previous page)

```
390 # Blocks
391 #####
392 self.uhd_usrp_source_0 = uhd.usrp_source(
393     ", ".join("", ""),
394     uhd.stream_args(
395         cpu_format="fc32",
396         channels=range(1),
397     ),
398 )
399 self.uhd_usrp_source_0.set_samp_rate(samp_rate)
400 self.uhd_usrp_source_0.set_center_freq(1000000000, 0)
401 self.uhd_usrp_source_0.set_gain(30, 0)
402 self.uhd_usrp_source_0.set_antenna('TX/RX', 0)
403 self.qtgui_freq_sink_x_0 = qtgui.freq_sink_c(
404     1024, #size
405     firdes.WIN_BLACKMAN_HARRIS, #wintype
406     0, #fc
407     samp_rate, #bw
408     "", #name
409     1 #number of inputs
410 )
411 self.qtgui_freq_sink_x_0.set_update_time(0.10)
412 self.qtgui_freq_sink_x_0.set_y_axis(-140, 10)
413 self.qtgui_freq_sink_x_0.set_y_label('Relative Gain', 'dB')
414 self.qtgui_freq_sink_x_0.set_trigger_mode(qtgui.TRIG_MODE_FREE, 0.0, 0, "")
415 self.qtgui_freq_sink_x_0.enable_autoscale(False)
416 self.qtgui_freq_sink_x_0.enable_grid(False)
417 self.qtgui_freq_sink_x_0.set_fft_average(1.0)
418 self.qtgui_freq_sink_x_0.enable_axis_labels(True)
419 self.qtgui_freq_sink_x_0.enable_control_panel(False)
420
421 if not True:
422     self.qtgui_freq_sink_x_0.disable_legend()
423
424 if "complex" == "float" or "complex" == "msg_float":
425     self.qtgui_freq_sink_x_0.set_plot_pos_half(not True)
426
427 labels = ['', '', '', '', '',
428           '', '', '', '', '']
429 widths = [1, 1, 1, 1, 1,
430           1, 1, 1, 1, 1]
431 colors = ["blue", "red", "green", "black", "cyan",
432           "magenta", "yellow", "dark red", "dark green", "dark blue"]
433 alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
434           1.0, 1.0, 1.0, 1.0, 1.0]
435 for i in xrange(1):
436     if len(labels[i]) == 0:
437         self.qtgui_freq_sink_x_0.set_line_label(i, "Data (0)".format(i))
438     else:
439         self.qtgui_freq_sink_x_0.set_line_label(i, labels[i])
440         self.qtgui_freq_sink_x_0.set_line_width(i, widths[i])
441         self.qtgui_freq_sink_x_0.set_line_color(i, colors[i])
442         self.qtgui_freq_sink_x_0.set_line_alpha(i, alphas[i])
443
444 self._qtgui_freq_sink_x_0_win = sip.wrapinstance(self.qtgui_freq_sink_x_0.pyqwidget(), Qt.QWidget)
445 self.top_grid_layout.addWidget(self._qtgui_freq_sink_x_0_win)
446 self.low_pass_filter_0 = filter.fir_filter_ccf(1, firdes.low_pass(
447     1, samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))
448 self.digital_gmsk_demod_1 = digital.gmsk_demod(
449     samples_per_symbol=2,
450     gain_mu=0.175,
451     mu=0.5,
452     omega_relative_limit=0.005,
453     freq_error=0.0,
454     verbose=False,
455     log=False,
456 )
457 self.blocks_multiply_const_vxx_1 = blocks.multiply_const_vcc((1, ))
458 self.blocks_file_sink_1 = blocks.file_sink(gr.sizeof_char*1, '/home/b/chat_test2.txt', False)
459 self.blocks_file_sink_1.set_unbuffered(False)
460 self.blks2_packet_decoder_1 = grc_blks2.packet_demod_b(grc_blks2.packet_decoder(
461     access_code='',
462     threshold=-1,
463     callback=lambda ok, payload: self.blks2_packet_decoder_1.recv_pkt(ok, payload),
```

(Image continued from previous page)

```
464         ),
465     )
466
467
468
469     #####
470     # Connections
471     #####
472     self.connect((self.blks2_packet_decoder_1, 0), (self.blocks_file_sink_1, 0))
473     self.connect((self.blocks_multiply_const_vxx_1, 0), (self.low_pass_filter_0, 0))
474     self.connect((self.digital_gmsk_demod_1, 0), (self.blks2_packet_decoder_1, 0))
475     self.connect((self.low_pass_filter_0, 0), (self.digital_gmsk_demod_1, 0))
476     self.connect((self.low_pass_filter_0, 0), (self.qtgui_freq_sink_x_0, 0))
477     self.connect((self.uhd_usrp_source_0, 0), (self.blocks_multiply_const_vxx_1, 0))
478
479     def closeEvent(self, event):
480         self.settings = Qt.QSettings("GNU Radio", "chat2")
481         self.settings.setValue("geometry", self.saveGeometry())
482         event.accept()
483
484     def get_samp_rate(self):
485         return self.samp_rate
486
487     def set_samp_rate(self, samp_rate):
488         self.samp_rate = samp_rate
489         self.uhd_usrp_source_0.set_samp_rate(self.samp_rate)
490         self.qtgui_freq_sink_x_0.set_frequency_range(0, self.samp_rate)
491         self.low_pass_filter_0.set_taps(firdes.low_pass(1, self.samp_rate, 200000, 50000, firdes.WIN_HAMMING, 6.76))
492
493
494     def main(options=None):
495
496         from distutils.version import StrictVersion
497         if StrictVersion(Qt.qVersion()) >= StrictVersion("4.5.0"):
498             style = gr.prefs().get_string('qtgui', 'style', 'raster')
499             Qt.QApplication.setGraphicsSystem(style)
500         qapp = Qt.QApplication(sys.argv)
501
502         data_csv = open('/home/b/20161005_140846.csv', 'r')
503         csvreader = csv.reader(data_csv)
504         header = next(csvreader)
505         data_rows = []
506         for row in csvreader:
507             data_rows.append(row)
508         data_csv.close()
509
510         for x in range(0,14):
511             out = open('file' + str(x+1) + '.txt', 'w')
512             for y in range(0, 100):
513                 out.write(data_rows[y][x+6]+' '+str(x+1)+'0'+str(y+1)+'\n')
514             out.close()
515
516         tb = chat2()
517         tb.start()
518         print("Ready to start...")
519         while True:
520             f = open('/home/b/chat_test1.txt', 'r')
521             f.flush()
522             f.readline()
523             check = f.readline()
524             if (check != ""):
525                 print("START")
526                 time.sleep(1)
527             break
528         tb.stop()
529
530         weight_file = open('/home/b/chat_test1.txt', 'r')
531         weight_file.flush()
532         weight_file.readline()
533         weights = weight_file.readline()
534         w_list = weights.split(",")
535         w_list.pop()
536         for m in range(0, len(w_list)):
537             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)

```
612     length1 = len(data10)
613     read11.close()
614
615     read12 = open('/home/b/file12.txt', 'r')
616     data11 = read12.readlines()
617     length2 = len(data11)
618     read12.close()
619
620     read13 = open('/home/b/file13.txt', 'r')
621     data12 = read13.readlines()
622     length3 = len(data12)
623     read13.close()
624
625     read14 = open('/home/b/file14.txt', 'r')
626     data13 = read14.readlines()
627     length4 = len(data13)
628     read14.close()
629
630     data = list(zip(sensors, w_list))
631     sched = WRRScheduler(data)
632
633     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
634     a_tot = 0
635     b_tot = 0
636     c_tot = 0
637     d_tot = 0
638     e_tot = 0
639     f_tot = 0
640     g_tot = 0
641     h_tot = 0
642     i_tot = 0
643     j_tot = 0
644     k_tot = 0
645     l_tot = 0
646     m_tot = 0
647     n_tot = 0
648
649     packet_num = 0
650     iterations = 150
651     result = []
652     while (packet_num < iterations):
653         choose = sched.get_next()
654         if choose[0] == 'A':
655             a_tot += 1
656             if (a_tot <= length1):
657                 result.append(choose[0])
658                 packet_num += 1
659         elif choose[0] == 'B':
660             b_tot += 1
661             if (b_tot <= length2):
662                 result.append(choose[0])
663                 packet_num += 1
664         elif choose[0] == 'C':
665             c_tot += 1
666             if (c_tot <= length3):
667                 result.append(choose[0])
668                 packet_num += 1
669         elif choose[0] == 'D':
670             d_tot += 1
671             if (d_tot <= length4):
672                 result.append(choose[0])
673                 packet_num += 1
674         elif choose[0] == 'E':
675             e_tot += 1
676             if (e_tot <= length5):
677                 result.append(choose[0])
678                 packet_num += 1
679         elif choose[0] == 'F':
680             f_tot += 1
681             if (f_tot <= length6):
682                 result.append(choose[0])
683                 packet_num += 1
684         elif choose[0] == 'G':
685             g_tot += 1
```

(Image continued from previous page)

```
686         if (q_tot <= length7):
687             result.append(choose[0])
688             packet_num += 1
689     elif choose[0] == 'H':
690         h_tot += 1
691         if (h_tot <= length8):
692             result.append(choose[0])
693             packet_num += 1
694     elif choose[0] == 'I':
695         i_tot += 1
696         if (i_tot <= length9):
697             result.append(choose[0])
698             packet_num += 1
699     elif choose[0] == 'J':
700         j_tot += 1
701         if (j_tot <= length10):
702             result.append(choose[0])
703             packet_num += 1
704     elif choose[0] == 'K':
705         k_tot += 1
706         if (k_tot <= length11):
707             result.append(choose[0])
708             packet_num += 1
709     elif choose[0] == 'L':
710         l_tot += 1
711         if (l_tot <= length12):
712             result.append(choose[0])
713             packet_num += 1
714     elif choose[0] == 'M':
715         m_tot += 1
716         if (m_tot <= length13):
717             result.append(choose[0])
718             packet_num += 1
719     elif choose[0] == 'N':
720         n_tot += 1
721         if (n_tot <= length14):
722             result.append(choose[0])
723             packet_num += 1
724
725     while i < iterations:
726         if result[i] == 'A':
727             switch = 'a'
728             tb = text_tx()
729             tb.start()
730             time.sleep(1)
731             tb.stop()
732             i += 1
733             a1 += 1
734         elif result[i] == 'B':
735             switch = 'b'
736             tb = text_tx()
737             tb.start()
738             time.sleep(1)
739             tb.stop()
740             i += 1
741             b1 += 1
742         elif result[i] == 'C':
743             switch = 'c'
744             tb = text_tx()
745             tb.start()
746             time.sleep(1)
747             tb.stop()
748             i += 1
749             c1 += 1
750         elif result[i] == 'D':
751             switch = 'd'
752             tb = text_tx()
753             tb.start()
754             time.sleep(1)
755             tb.stop()
756             i += 1
757             d1 += 1
758         elif result[i] == 'E':
759             switch = 'e'
```

(Image continued from previous page)

```
760     tb = text_tx()
761     tb.start()
762     time.sleep(1)
763     tb.stop()
764     i += 1
765     e1 += 1
766     elif result[i] == 'F':
767         switch = 'f'
768         tb = text_tx()
769         tb.start()
770         time.sleep(1)
771         tb.stop()
772         i += 1
773         f1 += 1
774     elif result[i] == 'G':
775         switch = 'g'
776         tb = text_tx()
777         tb.start()
778         time.sleep(1)
779         tb.stop()
780         i += 1
781         g1 += 1
782     elif result[i] == 'H':
783         switch = 'h'
784         tb = text_tx()
785         tb.start()
786         time.sleep(1)
787         tb.stop()
788         i += 1
789         h1 += 1
790     elif result[i] == 'I':
791         switch = 'i'
792         tb = text_tx()
793         tb.start()
794         time.sleep(1)
795         tb.stop()
796         i += 1
797         i1 += 1
798     elif result[i] == 'J':
799         switch = 'j'
800         tb = text_tx()
801         tb.start()
802         time.sleep(1)
803         tb.stop()
804         i += 1
805         j1 += 1
806     elif result[i] == 'K':
807         switch = 'k'
808         tb = text_tx()
809         tb.start()
810         time.sleep(1)
811         tb.stop()
812         i += 1
813         k1 += 1
814     elif result[i] == 'L':
815         switch = 'l'
816         tb = text_tx()
817         tb.start()
818         time.sleep(1)
819         tb.stop()
820         i += 1
821         l1 += 1
822     elif result[i] == 'M':
823         switch = 'm'
824         tb = text_tx()
825         tb.start()
826         time.sleep(1)
827         tb.stop()
828         i += 1
829         m1 += 1
830     elif result[i] == 'N':
831         switch = 'n'
832         tb = text_tx()
833         tb.start()
```

(Image continued from previous page)

```
834         time.sleep(1)
835         tb.stop()
836         i += 1
837         n1 += 1
838     else:
839         print_,"Error: Empty Result"
840     tb = ack2()
841     tb.start()
842     while True:
843         ack_file = open('/home/b/chat_test2.txt', 'r')
844         ack_file.flush()
845         ack_file.readline()
846         ack0 = ack_file.readline()
847         if (ack0 == 'ACK\n'):
848             print_,"RECEIVING ACK"
849             time.sleep(1)
850             break
851         ack_file.close()
852         time.sleep(1)
853     tb.stop()
854     if (i > 28) and (i % 20 == 0):
855         tb = chat2()
856         tb.start()
857         while True:
858             f = open('/home/b/chat_test1.txt', 'r')
859             f.flush()
860             f.readline()
861             check = f.readline()
862             if (check != ""):
863                 time.sleep(1)
864                 break
865         tb.stop()
866         weight_file = open('/home/b/chat_test1.txt', 'r')
867         weight_file.flush()
868         weight_file.readline()
869         weights = weight_file.readline()
870         w_list = weights.split(",")
871         w_list.pop()
872         for m in range(0, len(w_list)):
873             w_list[m] = int(w_list[m])
874         data = list(zip(sensors, w_list))
875         sched = WRRScheduler(data)
876         count = 0
877         new_result = []
878         while (count < (iterations-i)):
879             new_choose = sched.get_next()
880             if new_choose[0] == 'A':
881                 a_tot += 1
882                 if (a_tot <= length1):
883                     new_result.append(new_choose[0])
884                     count += 1
885             elif new_choose[0] == 'B':
886                 b_tot += 1
887                 if (b_tot <= length2):
888                     new_result.append(new_choose[0])
889                     count += 1
890             elif new_choose[0] == 'C':
891                 c_tot += 1
892                 if (c_tot <= length3):
893                     new_result.append(new_choose[0])
894                     count += 1
895             elif new_choose[0] == 'D':
896                 d_tot += 1
897                 if (d_tot <= length4):
898                     new_result.append(new_choose[0])
899                     count += 1
900             elif new_choose[0] == 'E':
901                 e_tot += 1
902                 if (e_tot <= length5):
903                     new_result.append(new_choose[0])
904                     count += 1
905             elif new_choose[0] == 'F':
906                 f_tot += 1
907                 if (f_tot <= length6):
```


(Image continued from previous page)

```
908         new_result.append(new_choose[0])
909         count += 1
910     elif new_choose[0] == 'G':
911         g_tot += 1
912         if (g_tot <= length7):
913             new_result.append(new_choose[0])
914             count += 1
915     elif new_choose[0] == 'H':
916         h_tot += 1
917         if (h_tot <= length8):
918             new_result.append(new_choose[0])
919             count += 1
920     elif new_choose[0] == 'I':
921         i_tot += 1
922         if (i_tot <= length9):
923             new_result.append(new_choose[0])
924             count += 1
925     elif new_choose[0] == 'J':
926         j_tot += 1
927         if (j_tot <= length10):
928             new_result.append(new_choose[0])
929             count += 1
930     elif new_choose[0] == 'K':
931         k_tot += 1
932         if (k_tot <= length11):
933             new_result.append(new_choose[0])
934             count += 1
935     elif new_choose[0] == 'L':
936         l_tot += 1
937         if (l_tot <= length12):
938             new_result.append(new_choose[0])
939             count += 1
940     elif new_choose[0] == 'M':
941         m_tot += 1
942         if (m_tot <= length13):
943             new_result.append(new_choose[0])
944             count += 1
945     elif new_choose[0] == 'N':
946         n_tot += 1
947         if (n_tot <= length14):
948             new_result.append(new_choose[0])
949             count += 1
950         if (len(new_result) < iterations):
951             for v in range(iterations-len(new_result)):
952                 new_result.insert(0,0)
953         result = new_result
954         choose = new_choose
955
956     switch = 'eof'
957     tb = text_tx()
958     tb.start()
959     time.sleep(1)
960     tb.stop()
961
962     print('done')
963     exit()
964
965     def quitting():
966         tb.stop()
967         tb.wait()
968     qapp.connect(qapp, Qt.SIGNAL("aboutToQuit()"), quitting)
969     qapp.exec_()
970
971
972     if __name__ == '__main__':
973         main()
974
```

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