WISCANET User Manual

Jacob Holtom
ASU WISCA
2020-05-22

Abstract

WISCANET, a Software Defined Radio Network (SDRN), provides the user with a highly configurable and flexible algorithm development platform to conduct wireless communications research. Each node utilizes an Ettus Research Universal Software Radio Peripheral (USRP) as the RF front-end, and MATLAB or Python for baseband processing. WISCANET emulates real-time over the air processing, by synchronizing radio transmit and receive operations, while pausing for a configurable time between each on-air frame to allow for slower processing chains. This design enables users to implement and test real-time over-the-air applications without optimizing processing. Algorithms can then be written entirely in MATLAB or Python, because of the wait between on-air frames, allowing for easier development and algorithmic debugging than C, C++, or an FPGA, despite the execution being slower. WISCANET controls the radios with minimal user input, requiring only straightforward radio parameters, such as frequency and sample rate, so that users can test without programming or debugging complex RF frontend hardware. The WISCANET approach enables the user to develop, test and validate new concepts and algorithms quickly over the air.

Contents

1 Architecture

1.1 Overview

The WISCA Software Defined Radio Network (WISCANET) is a software defined radio network that provides the user with a highly configurable and flexible algorithm development platform to conduct wireless communications research. It is composed of a configurable number of Radio Frequency (RF) frontends. WISCANET combines the flexibility of Software Defined Radio (SDR)s and the convenience and power of MATLAB or Python into a user-friendly package. The purpose of the system is to bridge the gap between simulation and experimental validation. It is designed so that the user can develop complex RF applications directly in MATLAB or Python and quickly implement them on an experimental platform without having detailed knowledge of the underlying software or hardware. By imposing no timing requirements, WISCANET also allows for baseband processing to take as long as necessary, thereby avoiding the need to optimize early for over-the-air experiments. WISCANET synchronizes all of the radios in the network to transmit and receive simultaneously by utilizing the Global Positioning System (GPS).

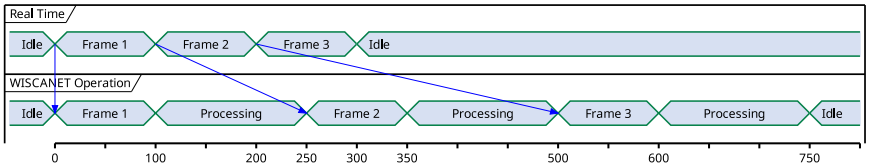


Figure 1: WISCANET transmits in chunks, one frame, and then processing as opposed to the traditional real-time RF over-the-air experiments where frames are transmitted continuously. This is enabled through GPS synchronization, as each frame is scheduled to happen on all radios, with configurable time left between for processing to be completed.

WISCANET's control software consists of the control node, edge node, uControl (USRP Control), and MATLAB MEX libraries which are written in C and C++. There is also a MATLAB class library that provides the user a simple interface to write their baseband processing algorithms against. There is also an equivalent Python class library.

1.2 Control Software

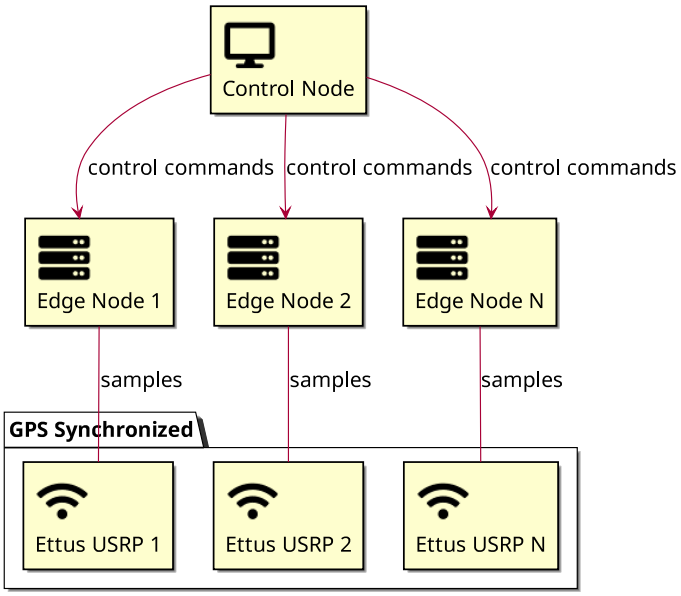


Figure 2: WISCANET Architecture

Control Node (cnode): The control node configures the edge nodes with the experiment parameters. When the experiment is executed, the control node commands each edge node to start the GPS synchronization process. Once this synchronization process is complete, the edge nodes start MATLAB at the control nodes commanded execution time. The control node runs two threads: one that draws the terminal interface (TUI), and another that runs the network functions.

Edge Node (enode): The edge nodes control the USRPs, communicate with the control node and launch the baseband processing software. The edge node connects to a control node on startup and becomes the minion of the control node, forming a star topology. When the edge nodes receives a run request, it starts up the USRP Controller (uControl) in another thread with the parameters provided by the configuration request from the control node. Once the USRP Controller has completed radio initialization and synchronization, the edge node software will start the MATLAB baseband processing code in the main thread, providing it the commanded start time as an argument to the MATLAB baseband processing code. The USRP controller has a setup time of 20 seconds to accommodate different USRP models. Commanded and scheduled times are stored and communicated in the UNIX timestamp format.

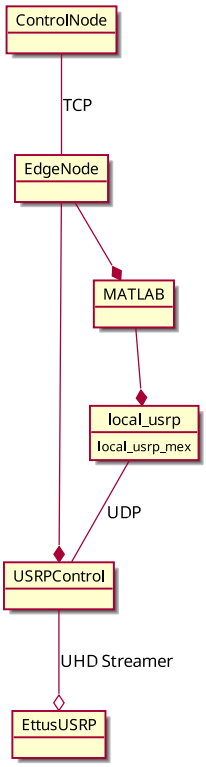


Figure 3: WISCANET Low-Level (Driver) Architecture

1.3 MATLAB

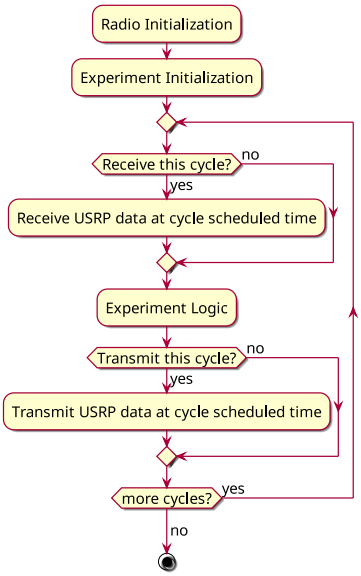


Figure 4: WISCANET MATLAB Baseband Control Flow

Local USRP Library: This MATLAB class (`local_usrp.m`) implements the interface between the

MATLAB baseband processing code and the WISCANET driver software (uControl). This interface is implemented utilizing a MATLAB MEX library called `local_usrp_mex.c`. The MEX library makes User Datagram Protocol (UDP) connections to the uControl process to send and recieve RF sample data.

1.4 Ettus USRPs

The Ettus Research USRP is a family of software defined radios. Ettus Research joined National Instruments (NI) in 2010. The USRPs interact with the host computer through the USRP Hardware Driver (UHD). WISCANET has so far been tested using the following hardware

- B210
- X310 + UBX160 daughterboard
- X300 + UBX160 daughterboard

USRP Control (uControl): The uControl driver program utilizes the UHD library from Ettus Research. It wraps access to and from the USRP with socket connections, so the MATLAB MEX function can provide and receive samples. It handles samples in packed complex format (real,imag,real,imag...) encoded as 16-bit signed integers. uControl utilizes the `uhd::usrp::multi_usrp` class to interact with the USRPs, which makes it immediately compatible with nearly all USRP models. This includes some more unique configurations such as multiple USRPs on a single host computer.

GPS Synchronization: The WISCANET system depends on a GPS Disciplined Oscillator (GPSDO) on each edge node USRP. This enables coarse time alignment ($\pm$ 50 ns) between the edge nodes. Each USRP's GPSDO needs an antenna with a good view of the sky. This will enable it to pick up the GPS constellation's signals and acquire a lock. If any single edge node USRP does not have GPS lock and synchronized time, that device will not complete initialization. If this happens, the experiment may continue, but that edge node will crash and will not function. This will usually corrupt the experiment.

1.5 Containerization and Podman

WISCANET can now be installed in a Podman (or Docker) container. This enables idempotence and easy replication of experiments without fighting drivers and other potential mismatched version bugs. This is done by installing the UHD driver, Boost, and all other WISCANET dependencies into a Fedora Latest Podman Container. It also includes an installation of MATLAB. This containerization technique also allows for single computers to easily run a complete WISCANET based network, utilizing multiple USRP devices. More information on this can be found in the `http://gitbliss.asu.edu/jholtom/wiscanet-docker` repository.

2 WISCANET Implementations

All implementation rely on a modern copy of MATLAB. The oldest tested is R2019b, but the typical installation currently uses R2020a. It can include as many or as few toolboxes as desired. The only requirements are that it

provides the MATLAB Coder and the base MATLAB. Other toolboxes may be required depending on experiments.

2.1 WISCANET Lab (Clustered)

In the BLISS Lab at ASU there are two clusters, one composed of B210s and one composed of X310s.

A distributed cluster implementation utilizes one computer for each USRP.

2.2 WISCANET-In-A-Box (WISCANET-Lite)

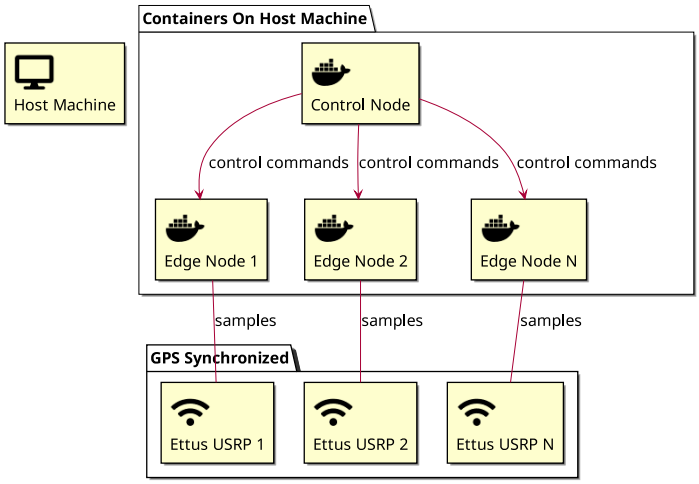


Figure 5: WISCANET-In-A-Box Architecture

The WISCANET-Lite system, utilizes Podman to create multiple edge nodes on a single machine. It also launches a control node in a container, providing a self-container and self-configuring solution. Multiple USRPs connected to the computer will need to be configured by the `<devaddr>` parameter in the appropriate usrconfig xml file.

The source code and a readme can be found at jholtom/wiscanet-docker on GITBliss.

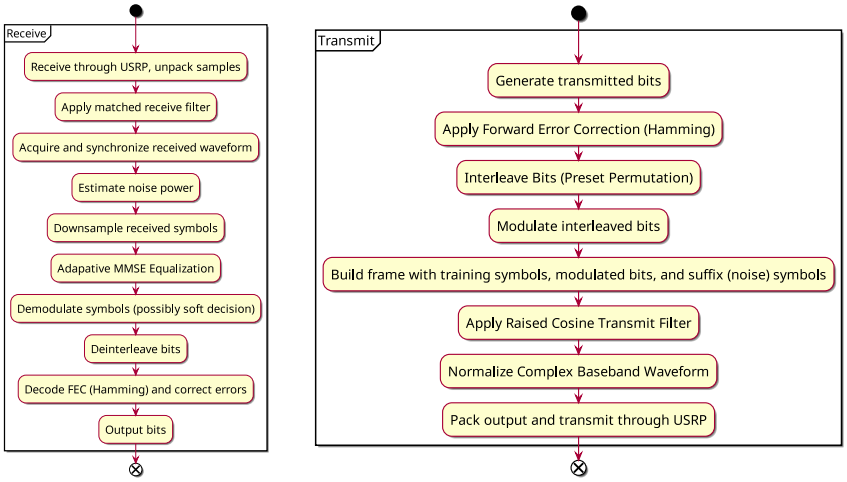
2.2.1 Limitations

This is potentially limited by your computers bandwidth, and multiprocessing ability. This may impose an upper limit on sample rate.

3 Applications

3.1 Sine (UMAC_sin)

This application transmit a sine tone (100kHz by default) at complex baseband, for a quarter of the configured samples, which is transmitted at the configured carrier frequency of the network on one USRP. Another node (or



(a) Ping Pong Receive Chain (b) Ping Pong Transmit Chain

Figure 6: Ping Pong Processing Chains

multiple) is configured to receive the tone and save the samples for analysis. A simple analysis program is provided that looks at the power spectral density, and the time domain waveform. This is a great test application to determine SNR and test radio behavior.

3.2 Ping Pong (UMAC_sdrn)

The Ping Pong (Software Defined Radio Network) demonstration sends messages between a configurable number of nodes, who in turn pass them on to the next node in a random sequence.

3.3 Dynamic Ping Pong (UMAC_sdrn_dynamic)

The Dynamic Ping Pong demonstration sends messages between a configurable number of nodes. The major difference is that it uses the PDK (Protocol Development Kit) to dynamically choose appropriate waveforms for an environment.

3.4 Joint Comms Radar (UMAC_jcr)

4 Workflow

! → From here on out, this manual assumes a working knowledge of basic UNIX/Linux command line operations, including creating, copying, moving, making folders, editing files, launching programs and more. If you do not feel immediately comfortable performing these operations, the authors recommend looking through the course at: MIT CSAIL Missing Semester. The authors suggest lectures from 1/13, 1/14, 1/21, 1/22 as most relevant.

4.1 Starting WISCANET software

: Log into every node that you are planning to use in your experiment (control node, and all edge nodes)

: On the control node, change directories to `~/wdemo/run/cnode/bin` and execute `./cnode`.

: On each edge node, change directories to `~/wdemo/run/enode/bin` and execute `./enode`.

! → : On the control node, as you start the edge nodes, you should see registrations coming in and being acknowledged. If not, something is potentially wrong with enode configuration, or your network.

: Check that all of the edge nodes have been connected and registered. This can be done using the option `1. Print enode status`.

4.2 Adding and configuring your experiment

4.2.1 Configuration Files

Your experiment will need to configure every edge node it uses. An XML (eXtensible Markup Language) file is used. It is called `usrconfig_ipaddr.xml` and placed in the `~/wdemo/run/usr/cfg/` directory. The ipaddr is replaced by the IP address of the edge node it configures. The control node knows which of the edge node configurations to select from the file `~/wdemo/run/usr/cfg/iplist`. Place the IP address of each edge node your experiment uses on a line in this file.

Example IP List:

```
10.88.0.10
10.88.0.11
10.88.0.12
```

Example Edge Config:

```
---
logic_id: 2 # Logic ID, unique for each node
op_mode: TX/RX # TX or RX or TX/RX
mac_mode: UMAC # Only option
time_slot: 2 # Time slot (unused, set to logic_id by default)
lang: MATLAB # Runtime Language, can be MATLAB or 'python'
matlab_dir: UMAC_sdrn/usrpFunc # Directory under run/usr/mat/ that your baseband (function file) is in
matlab_func: usrpQPSKNode # Baseband MATLAB or Python Function
matlab_log: "NULL" # Log Analysis Function, typically unused and so set to NULL
num_samples: 50000 # Number of samples per channel to be collected.  Locked to 50k
sample_rate: 1000000.0 # Sample Rate in Hz
subdevice: "A:A" # USRP Subdevice (including frontend) spec
freq: 907000000.0 # Center Frequency for both RX and TX
tx_gain: 50.0 # Transmit Gain in dB
rx_gain: 40.0 # Receive Gain in dB
bandwidth: 100000000.0 # Bandwidth in Hz
device_addr: serial=30F419C # This is the UHD Device Spec
channels: "0,1,2,3" # This is the channel string. Typically count up from 0 to as many channels as your configuration supports
antennas: TX/RX # This is the UHD Antenna Spec, typically TX/RX or RX2
```

Example Edge Config (Old XML Format):

```
<? xml version="1.0" ?>
<UsrConfig>
    <LogicId> 1 </LogicId>
    <OpMode> TX/RX </OpMode>
    <MacMode> UMAC </MacMode>
```

```
    <Tslot> 1 </Tslot>
    <matDir> UMAC_sdrn/usrpFunc </matDir> <!-- This is the directory under
    ↪ run/usr/mat/ that your baseband code is in -->
    <BbMatlab> usrpQPSKNode </BbMatlab> <!-- This is the baseband MATLAB
    ↪ function -->
    <LogMatlab> NULL </LogMatlab>
    <nsamps> 50000  </nsamps> <!-- this is the number of samples per channel
    ↪ to be collected -->
    <rate> 20e6 </rate> <!-- This is the sample rate -->
    <subdev> A:0 </subdev> <!-- This is the USRP subdevice (including
    ↪ frontend) spec -->
    <freq> 0.907E9 </freq> <!-- this is the center frequency, for both RX and
    ↪ Tx -->
    <txgain> 30 </txgain> <!-- Transmit Gain in dB -->
    <rxgain> 30 </rxgain> <!-- Receive Gain in dB -->
    <bw> 100E6 </bw> <!-- Bandwidth in Hz -->
    <devaddr> addr=192.168.10.7 </devaddr> <!-- This is the UHD device
    ↪ specification -->
    <channels> "0,1,2,3" </channels> <!-- This is the specific channel
    ↪ string.  Typically count up from 0 to as many channels as you may need
    ↪ -->
    <antennas> "TX/RX" </antennas> <!-- This is the UHD antenna spec,
    ↪ typically TX/RX or RX2 -->
</UsrConfig>
```

The inputs in each of these fields expect the following units

- LogicId/Tslot: Integer (≥ 0)
- OpMode: TX or RX or TX/RX
- MacMode: Must be set to UMAC
- nsamps: Integer that currently must be $= 50000$
- rate: Integer in Hertz (20e6, 1e6, 0.1e9, 0.2e9 all acceptable)
- freq: Double in Hertz (907E6, 0.907E9, 2.4E9 all acceptable)
- txgain/rxgain: Integer in dB
- bw: Integer in Hertz (100E6, 4E6 all acceptable)

4.2.2 Baseband Software

Your baseband software goes in the `~/wdemo/run/usr/mat/` folder. Your baseband software should live in its own folder, such as `UMAC_sdrn` or `UMAC_sin`. In the edge node configuration files, configure the `matDir` tag to be that folder, or a subfolder, containing the actual baseband MATLAB script. Configure the `BbMatlab` tag to be the function name that launches your experiments baseband execution loop. Configure any other options, including a `LogMatlab` file that does analysis on the outputs of your baseband execution loop. More information on what these options control can be found in the documentation of the WISCANET source code.

Example Baseband (MATLAB):

```
function err = welcomeToWiscanet(sys_start_time)
close all;
addpath(genpath('../lib'));

% First start time for the experiment
start_time = sys_start_time;
```

```
7
8  %% Experiment parameters
9  num_samples = 50000; % This must match the YML number of samples
10 num_chans = 1; % Number of channels to use for transmit and receive
11 sample_rate = 1e6; % This must match the YML sample rate
12 num_cycles = 5; % Number of cycles to run
13 proc_time = 3; % How long to wait between each cycle
14
15 % Initializes usrpRadio object
16 usrpRadio = local_usrp; % Sets up USRP object
17 usrpRadio = usrpRadio.set_usrp(type, num_samples);
18
19 % Get edge node logic ID
20 logicId = usrpRadio.logicalId();
21
22 % Generate sine wave
23 t=0:1/sample_rate:(1/sample_rate*(num_samples*0.5 - 1));
24 fc = 0.1e6; % 0.1 MHz or 100 kHz
25 sine = exp(1i*2*pi*fc*t);
26
27 % Transmit or Receive every `proc_time` seconds after start_time
28 rxdat = [];
29 for i = 1:num_cycles
30     if logicID == 1
31         % Transmit sine at start_time and save
32         usrpRadio.tx_usrp(start_time,sine,numChans); % sine must be
           ↪  num_samples x numChans, complex double matrix
33         save('tx_buff.mat','sine');
34
35     else
36         % Receive num_samples samples from usrp at start_time and save
37         rx_buff = usrpRadio.rx_usrp(start_time,numChans); % rx_buff will be
           ↪  num_samples x numChans, complex double matrix
38         save('rx_buff.mat','rx_buff');
39     end
40
41     % Set start time for next cycle by adding on the processing time
42     start_time = start_time + proc_time;
43 end
44
45 % Clean up usrpRadio objects and connections
46 usrpRadio.terminate_usrp();
47
48 end
```

Example Baseband (Python):

```
1  import numpy as np
2  import sys
3  sys.path.append('../lib/')
4  from local_usrp import *
5
6  # First start time for experiment
7  start_time = float(sys.argv[1])
8
9  # Experiment parameters
10 num_samples = 50000 # This must match the YML number of samples
11 num_channels = 1 # Number of channels to use for transmit and receive
12 sample_rate = 1e6 # (Hz) This must match the YML sample rate
13 num_cycles = 5 # Number of cycles to run
14 proc_time = 3 # (seconds) How long to wait between each cycle
15
16 # Initialize LocalUSRP object
17 usrp_radio = LocalUSRP(num_samples) # Specify Number of samples
18
```

```
# Get edge node logic ID
logic_id = usrp_radio.logical_id()

# Generate sine wave
t = np.arange(0, (1/sample_rate)*(num_samples),1/sample_rate)
fc = 0.1e6 #0.1 MHz or 100 kHz
test_data = np.exp(1j*2*np.pi*fc*t)
test_data = np.reshape(test_data,(num_samples,num_channels))
test_data = test_data.astype(np.cdouble)

# Transmit or Receive every `proc_time` seconds after start_time
for i in range(0, num_cycles):
    if logic_id == 1:
        # Transmit sine at start_time
        usrp_radio.tx_usrp(start_time, test_data, num_channels)
        np.savetxt('tx_buffer.dat',test_data)

    else:
        # Receive num_samples samples from usrp at start_time
        rx_buffer = usrp_radio.rx_usrp(start_time, num_channels)
        np.savetxt('rx_buffer.dat', rx_buffer)

    # Set start time for next cycle by adding on the processing time
    start_time = start_time + proc_time

# Clean up usrpRadio objects and connections
usrp_radio.terminate_usrp()
```

4.3 Running your experiment

: First, ensure that all of your desired edge nodes are connected (Can be done with the `1. Print enode status` options).

: Next, run the `2. Download user MATLAB code` option. It will print out the configuration for each edge node and ensure that configuration and the MATLAB baseband is installed on it.

: Next, Execute across all the edge nodes with the `3. Execute MATLAB code` option. This will begin the process. First it will launch uControl (USRP Control), and once it has started, it will launch the specified MATLAB processing loop.

: When the baseband has finished running, or has crashed, or you are satisfied with its behavior. Kill the execution on the edge nodes by running the `4. Stop MATLAB code` options.

: Collect the logs (saved as .mat files in the specified `matDir`) and bring them back to the cnode. They are placed in the folder: `~/wdemo/run/cnode/log/` folder, and when a new experiment started, placed in the `log_history` folder.

: Choosing option `6. Log Analysis` will launch a copy of MATLAB on the Control Node running the specific logging script.

4.4 Debugging your experiment

! → : If the edge node console prints `[enode] end of USRP Control`, anytime before the experiment has ended, this means that the radio controller has crashed. The log for this can be found at

`~/wdemo/run/enode/conlog/usrpLog`. Common errors include incorrect subdevice specifications, inaccessible radios, or a lack of GPS synchronization

5 Experiment Design

A wiscanet experiment runs in the following loop and is passed a single start time. This start time is always encoded as a unix timestamp, and should be an integer value, but is cast to a floating point.

Algorithm 1: WISCANET Application Flow

```
USRP Radio and local_usrp initialization;
Experiment initialization and configuration;
for cycle_num ∈ {1, 2, ..., N_cycles} do
    if transmitting on cycle_num then
        Execute transmit processing;
        tx_usrp(time_to_run,transmit_buffer, num_channels);
    else if receiving on cycle_num then
        receive_buffer = rx_usrp(time_to_listen, num_channels);
        Execute Receive Processing;
    else
        Do nothing;
    end
    Set the next time_to_run variable;
end
usrpRadio.terminate_usrp();
Clean up any MATLAB objects before shutdown;
```

When transmitting, often processing will need to occur before the block of data is transmitted, so the time_to_run must bake-in the time required to complete that processing, and get the samples preloaded onto the USRP, or at least start the process. This will also block until the signal has been successfully transmitted and the time has passed.

When receiving, the receiving MATLAB function blocks until it has completely read the received samples at the given time into MATLAB, so any proccessing will strictly occur after the signal has been received.

The `time_to_run` variable should match on all nodes, so it is typically done in a common integer addition such as start_time+5, and then in subsequent loops adding another +5 onto the last used time. If this does not match on all nodes, they will be transmitting and receiving at different times and the system won't work the way you expect. This variable is fed to the USRP Controller as a floating point UNIX timestamp, and is accurate to (+/- 50ns) as determined by the GPS accuracy. So with enough tuning it is possible to run WISCANET at a very tight loop time, assuming the processing can be completed, or simply saving off the data for later processing, such as in a radar.

! → The `terminate_usrp()` call closes the connections to the USRP Controller. This must be done, otherwise you will need to kill MATLAB and uControl on the node and restart. If you are testing locally, be sure to do this. MATLAB's class object for local_usrp can sometimes behave oddly, especially if

you lose track of its pointer, or the pointer to a specific instantion of it. It is always important that when doing set_usrp you must always assign the result of `usrpRadio.set_usrp` to the `usrpRadio` object `set_usrp()` is being called from, assuming usrpRadio is the instantiation of local_usrp.

5.1 Processing Time

In the MATLAB baseband, you can select how long you want to wait before simulated time cycles. It is important to ensure you do not miss the window on all nodes, as the USRP Controller will reject any packets that arrive late, so it is advisable to build in some margin in your loop in regards to processing time. Using MATLAB's `tic` and `toc` can help provide an estimate of how long each segment takes to run.

5.2 Scaling

It is important to remember that the `local_usrp.m` library expects that you provide it complex-valued arrays in the range $[0, 1]$. It will also return a complex-valued array in the same range $[0, 1]$. These matrices are natively treated as complex doubles and should be in the range 0.0-1.0 to properly full-scale the DAC. Anything above 1.0 will clip at the DAC.

The USRP radios natively input and output signed 16-bit complex integers, and the WISCANET drivers handle conversion back and forth between the scaled complex-double MATLAB arrays that are exposed by the `local_usrp.m` library.

WISCANET works with the tradition of number of samples x number of channels matrices and that is what it expects to be sent to the `tx_usrp` function, and what will be returned from the `rx_usrp` function.

! $\rightarrow$ For WISCANET Developers: It is possible to change the way USRP Control (uControl) interacts with `local_usrp_mex` such that the exchange format between MATLAB and uControl is complex float 64-bit ('fc64'). The UHD driver supports using 'fc64' as the CPU format and will automatically convert between 'fc64' and its on the wire format of 'sc16'. This would take some reworking of `uControl.cpp`, `local_usrp.m`, and `local_usrp_mex.c`.

5.3 Radio Configuration Options

This section provides a guide to the `usrconfig_ipaddr.xml` file options that configure the radio and channels.

For X310's operating below $1GHz$ `"dboard_clock_rate=20e6,mode_n=integer"` must be added to the devaddr field for phase stable operation. For improved spurious emission performance it is recommend to add `"mode_n=integer"` to the devaddr field.

In depth and weird question answering can be found in the Ettus Docs here: Identifying USRP Devices and Device Configuration through Address String

5.4 Genie Channel

There is a genie channel setup in WISCALite/WISCANet that shares an NFS folder called `~/wdemo/data` between all of the nodes. Placing a file in

USRP Radio	<devaddr>
B210	serial=30F419C
X310 + UBX160	addr=192.168.30.2
X310 + UBX160	addr=192.168.30,2,second_addr=192.168.40.2
2x X310's + UBX160's	addr0=192.168.10.4,addr1=192.168.10.7
USRP Radio	<subdev>
B210	"A:A" or "A:B"
X310 + UBX160	"A:0" or "B:0" or "A:0 B:0"
X310 + UBX160	"A:0" or "B:0" or "A:0 B:0"
2x X310's + UBX160's	"A:0 B:0"
USRP Radio	<antennas>
B210	"TX/RX" or "RX2"
X310 + UBX160	"TX/RX" or "RX2"
X310 + UBX160	"TX/RX" or "RX2"
2x X310's + UBX160's	"TX/RX" or "RX2"
USRP Radio	<channels>
B210	"0"
X310 + UBX160	"0" or "0,1"
X310 + UBX160	"0" or "0,1"
2x X310's + UBX160's	"0,1,2,3"

Table 1

there will make it available in the same place to all other nodes, enabling applications to save out a '.mat' file with some information and then be retrieved by another part of the experiment on another node. This is shared from the control node, which is running the NFS server, which all the edge nodes (running the NFS client) then mount at start time.

This also provides a convenient way (for small files) to be shared back to the control node, without having to go through the log collection process. NFS struggles with a multitude of small writes, so it is best to write your '.mat' or other files as one, rather than a separate '.mat' for every shared variable. The performance of this genie channel is limited by the network connection between the control and edge nodes, so it is also occasionally required to program in a watch for the file, or a delay

6 Errata

6.1 USRP X310s

The X310's are very heat sensitive and the warmer they get, the worse performance they exhibit and will begin to miss windows, and some of their FPGA behavior becomes illogical. They require very nice SFP+ adapters to work well. The best situation is to run SFP+ to SFP+ rather than going to an RJ45 adapter, but it can be done. Running them at 1Gb/s ethernet is possible, but it is often slow, and switching between 1Gb/s and 10Gb/s will require a firmware flash which is somewhat frustrating. The SFP+ ports are programmed in the FPGA bitstream, and as such can be a bit finicky to deal with.

6.2 X310 + UBX-160

When using the X310 in combination with the UBX-160 daughterboard, there is some amount of TX ramp-up time required for each frame. To avoid this affecting your transmitted signal, prepend your waveform with 0's in the MATLAB baseband handler. For $20MHz$, 12500 samples have been used successfully. This translates to $625us$ on air. It is not clear whether the ramp-up time is sample rate dependent, such that it is more tied to the number of samples, rather than the wall-clock time. To this end, you may desire to test your particular configuration with the `tx_sine` and `rx_sine` demonstration to ensure the X310 transmits a stable waveform. This issue has not been observed on any other combination of system as of June 25th, 2020.

A small snippet showing how to fix this in the transmitted frame is provided below:

```
num_samps = 50000; % Number of complex samples to transmit over the air
% Generate a test sine wave
fc = 0.1e6; % 0.1 MHz or 100 kHz
% Time vector at sample rate, for half the number of samples
t=0:1/sample_rate:(1/sample_rate*(num_samps*0.5 - 1));
sine = exp(1i*2*pi*fc*t);

numChans = 1;
rampSampOffset = 12500; % Number of samples to be zero
outSize = size(sine);
outWav = zeros(commsParams.usrpSamples,numChans);
outWav(1+rampOffset:outSize(1)+rampOffset,:) = sine;
```

6.3 local_usrp object in MATLAB

When setting up the `local_usrp` object in MATLAB, it is imperative to ensure that when you call `set_usrp`, you set your local object value to its return value (as follows), otherwise MATLAB will lose the UDP connections, and it will fail in bizarre ways.

```
test_usrp = local_usrp;
% This is the critical assignment
test_usrp = test_usrp.set_usrp(0, 0, 0, 0, 0, num_samples,0,0, 0, 0, 0, 0);

% Do a simple recieve 10 seconds from the current time for 1 channel
numChans = 1;
rx_buff =
↪  test_usrp.rx_usrp(double(uint64(posixtime(datetime('now','Timezone','UTC'))))+10,numChans);
test_usrp.terminate_usrp();
```

6.4 MIMO

WISCANET supports arbitrary NxN MIMO via a number of channels parameter fed to the `rx_usrp` and `tx_usrp` MATLAB functions.

To do MIMO, you also need the correct configuration in your usrconfig.xml in the channels field as well as specifying enough subdevices in the subdev field to provide all of the desired channels.

For reference, this is the magic line to feed wiscanet, to combine two X310s to perform 4x4 MIMO: addr0=192.168.10.7,addr1=192.168.10.2 These X310's will have the addr0 X310 set as the master, and any other X310's will be

slaves to the masters reference clock, and PPS. The master needs to have a synced GPSDO, however, the others do not. The slave X310's will need to have their 10 MHz REF IN and PPS IN ports fed from the masters 10 MHz REF OUT and PPS OUT ports. To do this without an Octoclock, they will need to be daisy chained, from master, to first slave, and then first slave to second, and so forth.

6.5 Antenna Selection

In the usrconfig.xml file, you can also set which antenna port to use, in the antennas field. Typically the options are only "TX/RX" or "RX2".

6.6 USRP B210s

B210's are great little radios and will stay on forever. However, despite their claims of doing multi-channel operation, they crash with a 0x02 error, which manifests in the error log as LATE_COMMAND. This is documented in some e-mails on the ettus users mailing list and its not really clear why it happens. The new version of the UHD driver (v4.0.0) may fix this and improve the situation, but as of now WISCANET still uses v3.15.0. The B210 also lacks some of the phase stability guarantees that are available with the correct daughterboard on an X310.

6.7 USRP Dynamic Range

The dynamic range of the USRPs is often very strange, and knowing absolute power is very tricky. In UHD v4.0.0 there are provisions to help you determine the absolute power received, but there is still no way to really determine the amount of power you are transmitting. The B210 expects you give it about 30-40dB of analog gain on both transmit and receive to get it in a happy place for dynamic range.

The X310 it is fully dependent on daughterboard, and there are often no documents for this. Experimentation is your friend.

6.8 WISCANET Oddities and Information

WISCANET currently has a hard coded configuration of 50000 samples at whatever sample rate you configure. This exists in a bunch of different places, but is removable with some fixing of the code that handles the samples being copied from MATLAB to uControl and also the other direction.

WISCANET's timing is solely dependent upon the accuracy of the GPSDO (+/- 50ns). There are two (common, but not limited to just these two ways) that the edge node system can miss the timing window. The timing window is most commonly missed by commanding a time before the current time that the GPSDO has decided it is. The other way to miss the timing window is for the radio not to have fully loaded the samples in before the time has occured, or the computer cannot keep up transferring the samples during the transmission/receive window. If this occurs, it will hang in the middle of the local_usrp_mex script, and uControl will hang as well. This will require a hard stop and reset of the system (Typically done from option 4 in the control node (Kill uControl and MATLAB)). It is not as bad as it sounds and will only bite you in the middle of a demo.

It is important to have GPS lock, because the GPSDO defaults to a very early timestamp (somewhere in the 2000s? but multiple years before the current time) and so it will accept the request for the current time. However, it will then wait for potentially years, and therefore not do what you want. So it is important that the GPS is correctly synced to the current time.

Acronyms

GIT : the stupid content tracker.

GITBliss : Bliss Lab's GIT server.

GPS : Global Positioning System. 3, 6

GPSDO : GPS Disciplined Oscillator. 6

MATLAB : Matrix Laboratory. 1, 3–6

MIMO : Multiple Input Multiple Output. 2, 16

NI : National Instruments. 6

RF : Radio Frequency. 3

SDR : Software Defined Radio. 3

SDRN : Software Defined Radio Network. 1

TCP : Transmisison Control Protocol.

UDP : User Datagram Protocol. 6

UHD : USRP Hardware Driver. 6

USRP : Universal Software Radio Peripheral. 1, 3–7

WISCA : The Center for Wireless Information Systems and Computational Architectures.

WISCANET : WISCA Software Defined Radio Network. 3