Telescope Interface Software

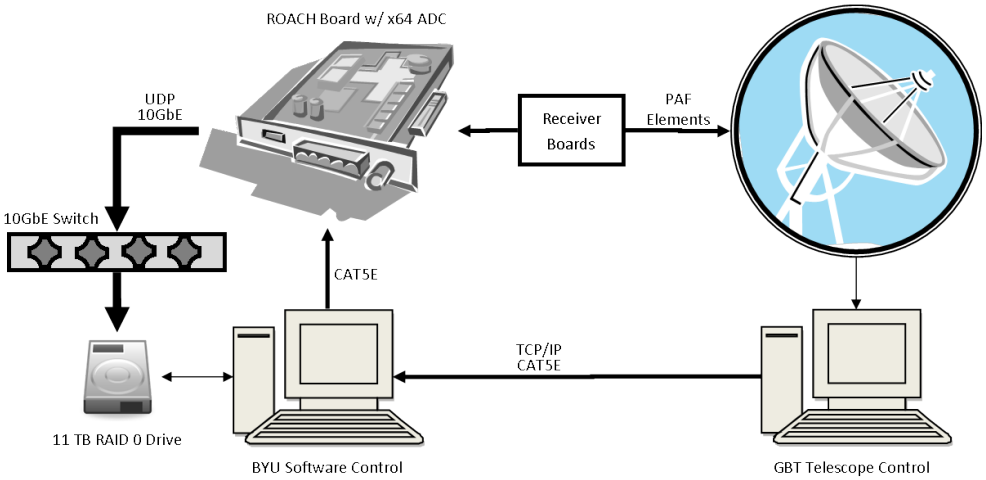
Brigham Young University

June 27, 2013

Contents

1 Introduction

The purpose of this document is to describe a customizable digital back-end for use with a Radio Telescope controller. The overall functionality is depicted in the below figure:



2 Hardware Requirements

2.1 Summary

This section outlines more detailed descriptions of the hardware as well as how the entire system is interconnected.

In order to make the system fully-functional, it must use the following hardware at minimum.

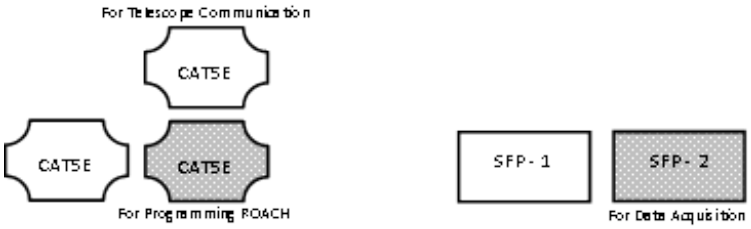
- Dell PowerEdge C2100
- ROACH-1 Development Board
- Fujitsu XG2000C 10GbE Switch
- BYU Custom Receiver Boards

2.2 Dell PowerEdge C2100

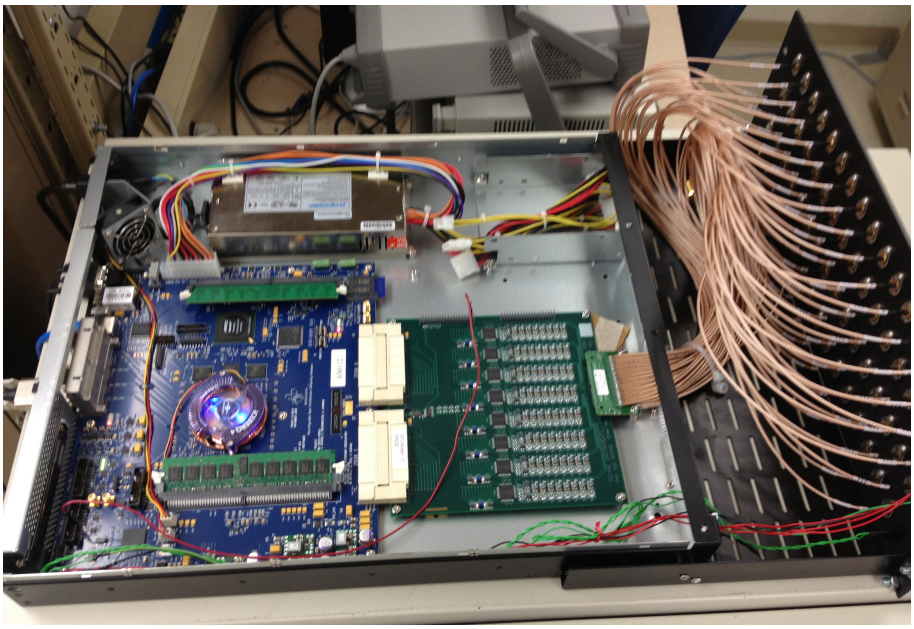
The Telescope Interface Software is hosted on a Dell PowerEdge C2100 server with the following specifications:

- 16 3.07 GHz Intel Xeon CPUs
- Intel 10GbE SFI/SFP+ Network Card
- 12 2TB hot-swappable SATA drives configured into 2 11TB RAID0 virtual drives
- FusionIO ioDrive II 785 GB

The back of the PC has several connectors that have an approximate configuration as shown:



2.3 ROACH-1 Development Board



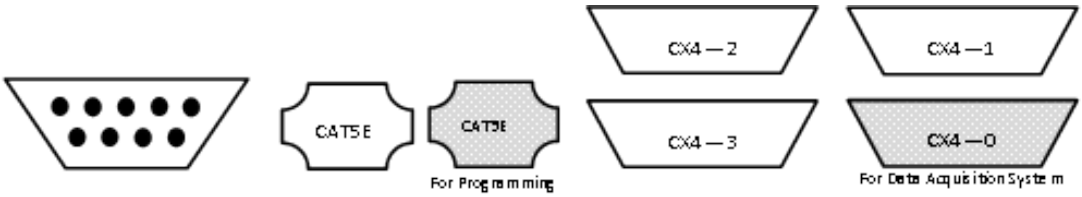
The ROACH Development Board requires an FPGA and an ADC card. The system is designed to use the following two components:

- x64 ADC Card
- Virtex 5 FPGA

There are three types of connectors on the back of each ROACH board:

1. Male DE-9 Serial
2. CAT5 Ethernet
3. CX4 10Gb Ethernet

These connectors are approximately configured as shown in the below figure.



The ROACH sends status messages through the Serial port. The left-most CAT5 Ethernet port is used for maintenance, while the right-most (the shaded port in the above figure) is used for programming. The four CX4 ports are used for high-speed data transfers. For example, the current Data Acquisition System uses CX4-0 (the shaded port).

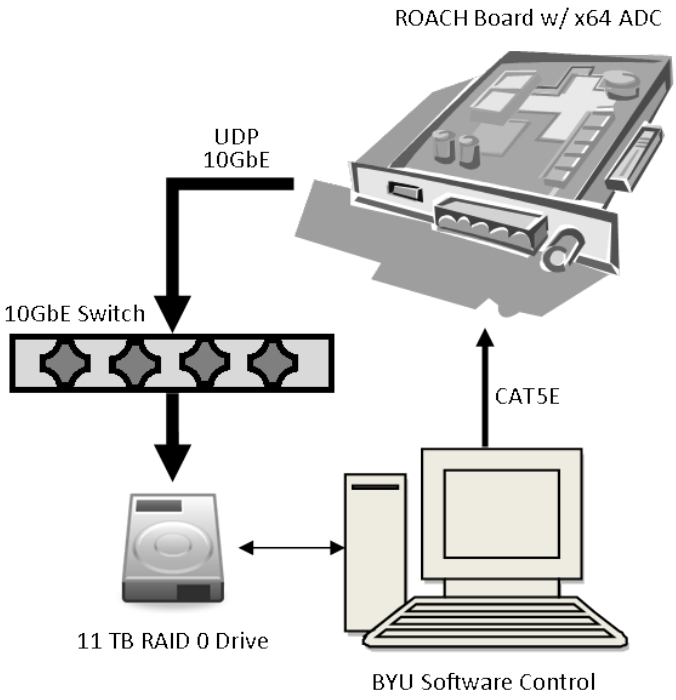
2.4 Custom Receiver Boards

16 4-port receiver cards bandpass filter and mix the PAF signals from L-Band down to 25 MHz-wide bandpass signals centered at 37.5 MHz (i.e. the second Nyquist zone). When the signals are sampled at 50 Mega-samples per second, they are aliased into the first Nyquist zone.

There are two mixing stages for each port, each of which require a local oscillator (LO). The first LO (i.e. LO1) is variable based on which 50MHz band should be observed. The second LO (i.e. LO2) should be fixed at 442.5 MHz.

The two LOs are split in order to accommodate the four channels on each card. Therefore, the LO ports at each receiver card should be driven with a 13 dBm signal. In order to accommodate all 16 boards, a 16-splitter must be driven with a 25 dBm signal.

2.5 Hardware Integration



The Dell PowerEdge C2100 is connected to the right-most CAT5E port on the ROACH board (or a switch with multiple ROACH boards connected to it) with a CAT5 Ethernet cable.

The ROACH boards that will run the Data Acquisition should be connected to the Fujitsu switch using a CX4/CX4 cable. The CX4-0 port should be used on the ROACH board. Make sure that the CX4/CX4 is no longer than 2 meters.

The Fujitsu switch connects to the PC with a XFP/SFP+ 10GbE cable. Make sure that the XFP/SFP+ cable is no longer than 2 meters.

3 Software Requirements

3.1 Summary

In order to interface with the system, a set of Python scripts are used. These (1) facilitate communication with the Telescope Control (byu_slave.py), (2) parse TCP/IP messages and error checks them (msg_parse.py), and (3) manage ROACH and process resources (res_manage.py). These are described in sections 3.2, 3.3, and 3.4 respectively.

In order to use the Data Acquisition System, an open-source piece of software called "Gulp" is used for packet sniffing. However, to minimize packet loss, modifications must be made,

and these are described in Appendix A. Information about the unmodified version are described in Section 3.5.

The ROACH boards boot up a Linux kernel over the network. In order to do this, the PC must be configured as a DHCP server. This is easily accomplished using dnsmasq, which is described in Section 3.6.

The Python scripts used for controlling the system have several dependencies, which are outlined in Section 3.7. How to install each one is described as well.

The SFP+ 10GbE Network Interface Controller (NIC), also needs to be configured for jumbo packets and a larger buffer. How to accomplish this is described in Section 3.8.

3.2 byu_slave.py

This module continuously listens for TCP/IP socket connections and services them one at a time. It has multiple methods in its API that facilitate communication between the Telescope Control and this system. Instructions on how to make an example socket connection to the system follow. Then a description of the various methods are outlined.

3.2.1 Establishing Socket Connections

Using Python, it is possible to establish a TCP/IP socket connection to the system by using the following code:

```
import socket
import time

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(("10.0.0.206",6000))
time.sleep(0.1)
```

"`10.0.0.206`", for example, would be the IP address of the Ethernet port on the BYU machine to communicate with. Once the socket connection is made, a message can be sent in the following manner:

```
sock.send("Hello There!")
```

There is a closed set of messages that will be recognized. These commands are outlined in the remaining sections.

Once the message has been sent, the socket must be discarded. This is accomplished in the following manner:

```
sock.shutdown(socket.SHUT_RDWR)
sock.close()
time.sleep(1.0)
```

3.2.2 Configuration File

In order for the BYU system to properly function, many settings must be set. Any important values can be set in a configuration .xml file. The format of the .xml file is as follows:

```
<Telescope>
    <header> ... </header>
    <socket> ... </socket>
    <msg_parse> ... </msg_parse>
    <res_manage> ... </res_manage>
</Telescope>
```

The `<header>` tag outlines basic information such as the name of the telescope, date, experiment name, etc. The tags can be whatever the user requires, and they will be printed at the start of a log file. An example of a header is as follows:

```
<header>
    <telescope>AO40 Simulation</telescope>
    <date>June 11, 2013</date>
    <author>Richard Black</author>
</header>
```

The `<socket>` tag specifies network parameters for socket communication. There are only four tags that should be specified here: `my_ip`, `my_port`, `their_ip`, and `their_port`. The `my_ip` and `my_port` specify the IP address and fabric port to be used by the BYU machine. The `their_ip` and `their_port` specify the IP address and fabric port to be used by the Telescope PC. An example of a `socket` tag is as follows:

```
<socket>
    <my_ip>10.0.0.1</my_ip>
    <my_port>6000</my_port>
    <their_ip>10.0.0.2</their_ip>
    <their_port>6000</their_port>
</socket>
```

The remaining xml tags (`<msg_parse>` and `<res_manage>`) correspond to the remaining Python scripts `msg_parse.py` and `res_manage.py` and, thus, will be fleshed out in more detail in their respective sections.

3.3 msg_parse.py

This script is called by sock_server.py to parse the TCP/IP messages that are received from the Telescope Control. In order for sock_server.py to function properly, this file must be in the same directory.

This file verifies that any received socket messages match user-defined regular expressions. These regular expressions are defined in the configuration file under the <msg_parse> tag.

The sub-tags inside of <msg_parse> are all <msg> tags. These tags give meaning to the socket messages that will arrive. Each <msg> tag has the following attributes:

- name
- ack
- err
- cmd

The name attribute gives the socket message a name. The ack attribute allows the user to specify if an acknowledge message should be sent ("1") or not ("0"). The err attribute allows the user to specify if error messages should be sent ("1") or not ("0"). The cmd attribute describes which DAQ command should be run. This links the socket messages with our system commands. The available DAQ commands are as follows:

- daq_start
- daq_setup
- daq_scan
- daq_spec
- daq_end

These commands are fleshed out in Section 4.3. If the ack or err flag is set, a formatting tag is needed. For ack, the ack_format tag is needed to specify what should be returned to acknowledge. Additionally, an <ack_timing> tag must be there to specify when the acknowledge should be sent ("before" or "after"); "before" indicates that the acknowledge will be sent before res_manage.py begins the process, and "after" indicates that the acknowledge will be sent after the process has completed.
If the user specified err to be "1," then an <err_format> tag is needed. This is effectively a printf statement that can contain a single %s to indicate where the error message should print (if at all).

Lastly, if a socket message has any data acquisition parameters (such as acquisition/integration time), the user can specify a <param> tag that houses a regular expression with (?#) markers around the parameter's regular expression. Since this allows for dynamic parameterization, any parameters set in the res_manage tag will be superseded.

An example of a msg_parse tag is as follows:

```
<msg_parse>
    <msg name="INIT" ack="1" err="1" cmd="daq_start">
        INIT
        <ack_format>init ok</ack_format>
        <err_format>init err %s</err_format>
        <ack_timing>after</ack_timing>
    </msg>
    <msg name="SETUP" ack="1" err="1" cmd="daq_setup">
        SETUP ([A-Z_]*=[A-Za-z.+0-9 ]*)*
        <ack_format>setup ok</ack_format>
        <err_format>setup err %s</err_format>
        <ack_timing>after</ack_timing>
        <param name="num_secs">TM_SECS=(?#)[0-9]+(.[0-9]*)?(?#)</param>
    </msg>
    <msg name="START" ack="1" err="1" cmd="daq_spec">
        START( [0-9.])*
        <ack_format>start ok</ack_format>
        <err_format>start err %s</err_format>
        <ack_timing>before</ack_timing>
    </msg>
    <msg name="EXIT" ack="0" err="0" cmd="daq_end">
        EXIT
    </msg>
</msg_parse>
```

The parser will verify that the received socket message matches one of the user and then return the socket message in list format (delimited by spaces). The `byu_slave.py` script will then forward this information to the `res_manage.py` script to call the appropriate commands.

3.4 res_manage.py

This script is called by byu_slave.py to manage the various ROACH and process resources. In order for byu_slave.py to function properly, this file must be in the same directory.

This script controls the data acquisition system by interpreting the parsed socket messages and calling the appropriate functions. To customize the script to know what resources should be used, it has its own configuration file xml tag `<res_manage>`.

Inside of the `res_manage` tag, there are three sub-tags:

- `resources`
- `configuration`
- `gulp`

The `resources` sub-tag lists the ROACH resources by IP address. Each ROACH receives its own tag `<roach>` and is declared in the following manner:

```
<resources>
    <roach name="roach1">10.0.1.1</roach>
    <roach name="roach2">10.0.1.2</roach>
</resources>
```

The user can then specify which ROACH boards the processes can use. These processes are defined in the `configuration` sub-tag. The process tag should have at minimum a `process` tag that describes how the data acquisition system should be configured. Currently the real-time beamformer and correlator modes are not implemented since a practical automation has not been defined yet. To configure the data acquisition system, the `process` tag must list the bitstreams, roach name, and acquisition parameters. This can be accomplished in the following manner:

```
<configuration>
    <process name="daq">
        <bitstream fft="256">x64daq256.bof</bitstream>
        <bitstream fft="512">x64daq512.bof</bitstream>
        <bitstream fft="1024">x64daq1024.bof</bitstream>
        <roach>roach1</roach>
        <params>
            <param name="bin_start">103</param>
            <param name="bin_end">153</param>
            <param name="row_start">0</param>
            <param name="row_end">7</param>
            <param name="col_start">0</param>
            <param name="col_end">4</param>
            <param name="fft_length">512</param>
            <param name="lsb_select">10</param>
            <param name="num_secs">5.0</param>
            <param name="num_specs">1000</param>
        </params>
    </process>
</configuration>
```

The various parameters are describe in more detail in Section 4.3.

Lastly, the `gulp` sub-tag sets parameters for the Gulp packet sniffer. The options that can be customized are the ring buffer size (`<buffer>`), listening interface name (`<if>`), verbose flag (`<verbose>`) and the output directory (`<out_dir>`). To better understand these parameters, see Appendix A. At the moment, the verbose flag **must** be set to 1. If it is not, the system will hang during the `daq_start` command. This is due to the fact that the system calls Gulp as a subprocess and uses pipelined stdin and stdout to communicate.

An example of the Gulp configuration tag is as follows:

```
<gulp>
    <buffer>10000</buffer>
    <out_dir>/media/Disk0</out_dir>
    <if>eth3</if>
    <verbose>1</verbose>
</gulp>
```

3.5 Gulp

Gulp is an open-source optimized packet sniffer written in C. It uses multiple cores and a ring buffer to divide the workload and buffer the data. The Data Acquisition System uses Gulp to capture each sample spectrum onto the PC hard drives. The software can be found at,

`http://staff.washington.edu/corey/gulp/`

To further enhance Gulp and make it compatible with this system, several modifications need to be made. The modifications made are documented in Appendix A.

Additionally, the schedtool program is needed to set core affinities. This can be installed at the command line with the following code:

```
apt-get install schedtool
```

3.6 dnsmasq

dnsmasq is a small DHCP server for a local network. It is used to assign IP addresses to new clients on the network as well as to provide a TFTP server for clients needing to network-boot. It is therefore used to allow ROACH boards on the network to boot and use a filesystem located on the PC.

In order to install dnsmasq, the following command is used

```
sudo apt-get install dnsmasq
```

To start up the server, type

```
dnsmasq
```

To properly configure the server, the following files need to be modified:

1. `/etc/hosts`
2. `/etc/dnsmasq.conf`
3. `/etc/ethers`

The following sections outline how these files should be configured for this system. Once these files have been modified, the server must be restarted by using the following command-line code:

```
/etc/init.d/dnsmasq restart
```

Lastly, in order to make the filesystem available for use on the ROACH boards, the Network File System (NFS) must be installed and configured as well. This is installed by using the following code:

```
apt-get install nfs-kernel-server nfs-common
```

Then the `/etc/exports` file is appended with

```
/srv/roach_boot 169.254.145.0/255.255.255.0
    (rw,subtree_check,no_root_squash,insecure)
```

Lastly, the filesystem is started with

```
exportfs -a
```

You can verify that dnsmasq is configured properly when the ROACH is restarted, booted-up and you can open a secure shell on the ROACH using

```
ssh root@roach3
```

3.6.1 /etc/hosts

This file lists mappings of IP addresses to hostnames. An example of this file would be:

```
169.254.145.11  roach1  roach1.ee.byu.edu
169.254.145.12  roach2  roach2.ee.byu.edu
169.254.145.13  roach3  roach3.ee.byu.edu
169.254.145.14  roach4  roach4.ee.byu.edu
```

This, in short, connects the IP address `169.254.145.11` with hostnames `roach1` and `roach1.ee.byu.edu`, and so forth.

3.6.2 /etc/dnsmasq.conf

This file lists commands that would normally be used at the command line when running dnsmasq. The following code outlines a configuration file that sets dnsmasq up to listen on interface "eth1 (IP `169.254.145.1`)" and designate a filesystem on the host PC for use on the ROACH board:

```
interface = eth1
dhcp-range = 169.254.145.100, 169.254.145.200, 12h
read-ethers
dhcp-option = 42, 0.0.0.0
dhcp-option = 17, 169.254.145.1:/srv/roach_boot/etch_devel
bind-interfaces
dhcp-boot = uImage
enable-tftp
tftp-root = /tftpboot
```

3.6.3 /etc/ethers

This file is used to assign IP addresses to clients with particular MAC addresses. An example of this file follows:

```
02:00:00:03:01:14  169.254.145.14
02:00:00:03:01:13  169.254.145.13
02:00:00:03:01:12  169.254.145.12
02:00:00:03:01:11  169.254.145.11
```

3.7 Python Libraries

In order for the Python scripts to function, we need the following libraries:

- setuptools
- dev
- numpy
- katcp
- corr
- aipy
- pyephem
- iniparse
- construct
- spead
- h5py
- matplotlib
- bitstring

Many of these libraries can be installed using:

```
apt-get install python-<library_name>
```

These libraries are:

- setuptools
- dev
- numpy
- h5py
- matplotlib

The remainder of the libraries can be found in a .tar.gz format at

```
pypi.python.org/pypi/<library_name>
```

Once the downloads are complete, you can extract their contents using:

```
tar -xvf <library.tar.gz file>
```

The libraries can be installed by navigating to the directories that are extracted from the .tar.gz archive and typing

```
python setup.py install
```

The only library that does not fit this mold is *bitstring*. To extract the contents of the archive, you must use the *unzip* program, installed using the following code:

```
apt-get install unzip
```

The archive can then be unzipped by using the following code:

```
unzip bitstring_3.0.2.zip
```

3.8 NIC Configuration Tools

In order for the onboard SFP+ 10GbE Network Interface Controller (NIC) to be optimized for 10GbE transfer, the two following settings need to be changed:

1. Maximum Transmission Unit (MTU)
2. Receive Buffer Size

3.8.1 Maximum Transmission Unit

The MTU can be changed quickly on the command-line by typing:

```
ifconfig ethX mtu 9000
```

where `ethX` is the name of the SFP+ 10GbE interface. However, this is not a permanent change. To make the change permanent, the MTU parameter should be changed in "Network Settings."

3.8.2 Receive Buffer Size

In order to modify the Receive Buffer Size, the *ethtool* program is needed. This is installed using the following code:

```
apt-get install ethtool
```

The Receive Buffer Size is then modified by using the following code:

```
ethtool -G ethX rx 4096
```

4 Data Acquisition System

4.1 Summary

The Data Acquisition System captures frequency windows of data captured from 64 elements sampled at 50 MHz. The data is transmitted via 10GbE to a packet sniffer named Gulp. To use the system, the user must (1) start every process, (2) perform one or more scans, and (3) shut down every process.

4.2 Hardware Requirements

In addition to the universal necessities of every system, the Data Acquisition System requires additional connections.

1. a 10GbE CX4/CX4 connection between the programmed ROACH and a Fujitsu XG2000C 10GbE switch.
2. a 10GbE XFP/SFP+ connection between the Fujitsu switch and the Dell PowerEdge's on-board SFP+ 10GbE port.

4.3 Command Definitions

As described in Section 3.3, socket messages are translated to DAQ commands. This section describes all of the possible commands that these socket messages can translate to.

daq_start	Initializes the system
daq_setup	Non-essential Specifies additional parameters and metadata
daq_scan	Performs a single acquisition (scan)
daq_spec	Starts a pseudo-real-time spectrometer
daq_bfscan	Performs three acquisitions across the entire bandwidth (512 FFT only)
daq_end	Shuts the system down

These commands are described in greater detail in the following sections. In order to understand the row and column parameters, a section about the output matrix will also follow.

4.3.1 Output Matrix

The x64 ADC card acquires 64 simultaneous samples that staggered across eight (8) ADC chips. Each chip then outputs its 8 samples serially. The FPGA that processes the samples runs on a 200 MHz clock and thus provides 2 output lines for each ADC chip that update on each cycle. Thus, there are 16 samples available on every clock cycle. The FFT then serially outputs 2 frequency bins every cycle even though it calculates the frequency bins for 4 input lines. Because of this, the outputs appear in a peculiar order, as displayed below:

	0	1	2	3	4	5	6	7
0	0	8	2	10	4	12	6	14
1	1	9	3	11	5	13	7	15
2	16	24	18	26	20	28	22	30
3	17	25	19	27	21	29	23	31
4	32	40	34	42	36	44	38	46
5	33	41	35	43	37	45	39	47
6	48	56	50	58	52	60	54	62
7	49	57	51	59	53	61	55	63

This illustrates what data is available for each clock cycle. The columns represent time and the rows represent data lines. Each column's frequency bins are outputted first and then the next column is outputted.

If the user wanted to capture from elements 2, 3, 4, 5, 10, 11, 18, 19, 20, 21, 26, and 27, he or she would request rows 0-3 and columns 2-4.

	0	1	2	3	4	5	6	7
0	0	8	2	10	4	12	6	14
1	1	9	3	11	5	13	7	15
2	16	24	18	26	20	28	22	30
3	17	25	19	27	21	29	23	31
4	32	40	34	42	36	44	38	46
5	33	41	35	43	37	45	39	47
6	48	56	50	58	52	60	54	62
7	49	57	51	59	53	61	55	63

There is a restriction on the number of rows that can be requested. The number of rows must be 0 modulo 4, or, in other words, a multiple of 4. Therefore, the user can request either 4 contiguous or all 8 rows during capture. There is no restriction on columns.

4.3.2 daq_start

This command initializes the entire data acquisition system with a particular FFT length. If the system has been started previously, then this command restarts the system. It does not start an acquisition.

The length of the FFT should either be specified in the configuration file as described in Section 3.4 or through the socket message, which requires the user to define how the parameter is set in the message by modifying the configuration file as described in Section 3.3.

The FFT length parameter name is `fft_len` and must be either 256, 512, or 1024.

4.3.3 daq_setup

This command allows the telescope control to dynamically specify some parameters and indicate any additional metadata not supported by the x64 system. The remainder of these sections outline which parameters can be specified using this command.

4.3.4 daq_scan

This starts the Data Acquisition System for a single scan. Once the desired number of seconds have been captured, the system returns to a standby mode and waits for additional commands.

Each scan, naturally, needs a filename. The user can specify this filename through a `daq_setup` command that was called previously or in the current socket message (see Section 3.3 for details). If a filename is not specified, then a timestamp (`<yyyymmdd-hhmmss.bin>`) is used by default.

The acquisition parameters can be set through the socket message, through a `daq_setup` command, or in the configuration file (as described in Section 3.4). These parameters are the following:

bin_start	The frequency window starting bin
bin_end	The frequency window ending bin
row_start	The starting row in the output matrix
row_end	The ending row in the output matrix
col_start	The starting column in the output matrix
col_end	The ending column in the output matrix
num_secs	The number of seconds to capture

bin_start should be between 0 and $\frac{\text{len}}{2} - 1$ and less than **b_end**.

bin_end should be between 0 and $\frac{\text{len}}{2} - 1$ and greater than or equal to **bin_start**. The number of bins captured should also be greater than 2.

row_start should be between 0 and 7 and less than **row_end**.

row_end should be between 0 and 7 and greater than **row_start**. Additionally, **row_end** - **row_start** + 1 should be divisible by 4.

col_start should be between 0 and 7 and less than **col_end**.

col_end should be between 0 and 7 and greater than **col_start**.

num_secs should be between 0 and 9,999. This can also be represented as a float (e.g. 240.5).

4.3.5 daq_bfscan

This command, like `daq_scan`, puts the system into capture mode, but it performs three separate captures in order to acquire across the entire bandwidth. Once the three captures are completed, the system returns to standby mode.
The acquisition parameters are the same for this command as for `daq_scan` except for the `bin_start` and `bin_end` parameters (since we are capturing all frequency bins).

4.3.6 daq_spec

This command puts the system into pseduo-real-time spectrometer mode. The user can then open the MATLAB script `plot_specs-06252013.m` to plot constantly updating spectra for all specified elements. To use this properly, the output directory specified in the configuration file (see Section 3.4) must be used in the MATLAB script. The script will then examine the output directory and plot the latest file. In order for this script to work, the `parse_and_plot_x64.m` MATLAB script must be in the same working directory as `plot_specs-06252013.m`.

The parameters used for this command are identical to the ones needed for `daq_scan`, with exception to the `filename`. The filename that is created is a timestamp (yyyymmdd-hhmmss) following by a spectrum number. An example of a set of 100 spectrometer files is as follows:

```
20130615-153237_spec_1.bin
20130615-153237_spec_2.bin
20130615-153237_spec_3.bin
...
20130615-153237_spec_99.bin
20130615-153237_spec_100.bin
```

The update speed of the MATLAB plots is directly correlated to the capture time (i.e. `num_secs`), therefore (for better real-time updates) a shorter acquisition time is preferred. Additionally, if the number of packets parsed is smaller, the faster the update will be, so modifying this in the `plot_specs-06252013.m` script is recommended.

4.3.7 daq_end

This command performs a clean shutdown of the data acquisition system. Once this command has been issued, the data acquisition system can only be started up again with the `daq_start` command.

4.4 Data Format

4.4.1 Packet Header

The data is stored into packets. Each packet contains a 64-bit header word that describes the parameters provided by the user. Bits 63-54 (where 64 is the MSB) represent the starting frequency bin. Bits 53-44 represent the ending frequency bin. Bits 43-41 represent the starting row number. Bit 40 represents whether 4 or 8 rows were captured. Bits 39-37 represent the starting column. Bits 36-34 represent the ending column. Bits 33-32 represent the length of the FFT by using a encoding as outlined:

Bits 33-32	**FFT Length**
00	256
01	512
10	1024
11	Not supported

Bits 31-0 represent the packet number. The 0th packet is the first packet sent after requesting a capture. The following table outlines the first 32-bits of the header:

63→54	53→44	43→41	40	39→37	36→34	33→32
bin_start	bin_end	row_start	row_flag	col_start	col_end	fft_len

4.4.2 Packet Data

The data is stored into packets. Each 64-bit word consists of 4 8-bit real and 8-bit imaginary values that represent a single frequency bin of a single input element as such:

15→8	7→0
$\Re e$(Bin)	$\Im m$(Bin)

Each 64-word represents a frequency bin for four rows of a single column. The next word is the next frequency bin for the same four elements until the bins are exhausted. The next four rows are then stored in the same manner. This continues until every column has been treated. The packet is then closed.

If a capture contained rows 0-3 and columns 2-4 and bins 0-9, then the packet will be structured in the following manner:

63 downto 48	**47 downto 32**	**31 downto 16**	**15 downto 0**
Element 2, Bin 0	Element 3, Bin 0	Element 18, Bin 0	Element 19, Bin 0
Element 2, Bin 1	Element 3, Bin 1	Element 18, Bin 1	Element 19, Bin 1
⋮	⋮	⋮	⋮
Element 2, Bin 9	Element 3, Bin 9	Element 18, Bin 9	Element 19, Bin 9
Element 10, Bin 0	Element 11, Bin 0	Element 26, Bin 0	Element 27, Bin 0
⋮	⋮	⋮	⋮
Element 10, Bin 9	Element 11, Bin 9	Element 26, Bin 9	Element 27, Bin 9
Element 4, Bin 0	Element 5, Bin 0	Element 20, Bin 0	Element 21, Bin 0
⋮	⋮	⋮	⋮
Element 4, Bin 9	Element 5, Bin 9	Element 20, Bin 9	Element 21, Bin 9

If eight rows are requested, then the top four rows of a column are stored, followed by the bottom four rows. For example, a capture containing rows 0-8, columns 5-6, and bins 20-45, each packet would be structured as follows:

63 downto 48	**47 downto 32**	**31 downto 16**	**15 downto 0**
Element 12, Bin 20	Element 13, Bin 20	Element 28, Bin 20	Element 29, Bin 20
Element 12, Bin 21	Element 13, Bin 21	Element 18, Bin 21	Element 29, Bin 21
⋮	⋮	⋮	⋮
Element 12, Bin 45	Element 13, Bin 45	Element 18, Bin 45	Element 29, Bin 45
Element 44, Bin 20	Element 45, Bin 20	Element 60, Bin 20	Element 61, Bin 20
⋮	⋮	⋮	⋮
Element 44, Bin 45	Element 45, Bin 45	Element 60, Bin 45	Element 61, Bin 45
Element 6, Bin 20	Element 7, Bin 20	Element 22, Bin 20	Element 23, Bin 20
⋮	⋮	⋮	⋮
Element 6, Bin 45	Element 7, Bin 45	Element 22, Bin 45	Element 23, Bin 45
Element 38, Bin 20	Element 39, Bin 20	Element 54, Bin 20	Element 55, Bin 20
⋮	⋮	⋮	⋮
Element 38, Bin 45	Element 39, Bin 45	Element 54, Bin 45	Element 55, Bin 45

4.5 Limitations

4.5.1 Packet Size

The Data Acquisition transmits jumbo packets, that is to say 1024 64-bit words, to maximize data transfer. This means that no single packet can exceed 1024 64-bit words.

Due to the packet structure described in 4.4, the number of words in a single packet is calculated as such,

$$\text{num\_words} = \frac{\text{num\_bins} \times \text{num\_rows} \times \text{num\_cols}}{4} + 1$$

where **num_bins** is the number of frequency bins to capture
(i.e. bin_end−bin_start + 1),

num_rows is the number of rows in the output matrix (described in Section 4.3.1) being captured
(i.e. row_end−row_start + 1), and

num_cols is the number of columns in the output matrix being captured
(i.e. col_end−col_start + 1).

If the number of words exceeds 1024, then the Data Acquisition System will not proceed and will send an error message to the Telescope Control.

4.5.2 Bitrate

The Data Acquisition supports bitrates up to 6 Gigabits per second (Gbps). Due to the packet structure described in Sections 4.4.1 and 4.4.2, the bitrate can be calculated as follows

$$\text{bitrate} = \frac{\text{num\_words} \times \text{sample\_rate} \times \text{bits\_per\_word}}{\text{fft\_length}}$$

Where **num_words** is defined as in Section 4.5.1.

sample_rate $= 50 \times 10^6$ (50 MHz).

bits_per_word $= 64$.

fft_length is the length of the FFT being used.

If the bitrate exceeds 6 Gbps, the Data Acquisition System will not proceed and will send an error message to the Telescope Control.

4.5.3 Frequency Bins

Due to the buffering scheme used in the system, it is not possible to acquire more than half of the available frequency bins. The F-Engine (the module that performs the Discrete Fourier Transform) outputs only $\frac{1}{2}$fft_length frequency bins. Of these available bins, only $\frac{1}{2}$ can be acquired in a single acquisition. Thus the maximum number of acquirable bins are

$$\frac{1}{4} \text{ fft\_length}$$

Thus, for example, an fft_length of 1024 allows the user to acquire up to 256 frequency bins.

4.6 Interpreting Data

In order to parse and plot the captured data, the user must run the `parse_daq_data.m` MATLAB script. The format of this command is as follows:

```
parse_daq_data(file_name, num_packets)
```

file_name	The name of the file that was captured to.
num_packets	The number of packets to be captured.

The **num_packets** parameter specifies how many packets (starting with the first one captured) should be parsed and plotted. If the user wants to parse all of the captured packets, -1 should be used for this argument.

The script will then parse the data and make a MATLAB plot for every captured element's frequency content. Each captured spectrum is plotted as a different color.

5 Beamformer

5.1 Summary

The Beamformer is really 7 independent broadband (25MHz) beamformers. Using the x64 ADC, the Beamformer has the same number and organization of inputs as does the DAQ. The input for each of the 7 beamformers is generated by a 512 point FFT, and the output of accumulated and stored in a BRAM on the ROACH board. The outputs can be plotted and viewed on the host computer.

5.2 TCP/IP Command Definitions

The following TCP/IP messages are used to control the Beamformer:

bf(...)	Starts the beamformer
bf_updateBFCoeffs(...)	Updates weights of a single beamformer
bf_multi_updateBFCoeffs(...)	Updates weights of all beamformers
bf_get_data(...)	Plots output of a single beamformer
bf_get_all_data(...)	Plots output of all beamformers
bf_set_slice(...)	Sets slicing of FFT output
bf_set_acc_len(...)	Sets accumulation time

5.2.1 bf

This starts the beamformer. The FFT output is packaged into a 4-bit real/4-bit imaginary byte before being sent to the beamformers. The user can specify which 4 bits should be used as inputs for the beamformers. The output of each beamformer is accumulated for a user specified length of time.

```
bf(roach,acc_len,win_slice,num_beams,coeff_file1 ... coeff_file7)
```

roach	The name of the ROACH board to program
acc_len	Accumulation length in seconds
win_slice	The starting bit of the 4-bit window
num_beams	The number of beamformers to create
coeff_fileX	The name of the coefficient file to assign to beam X

roach should be a string of the format "roachX" where "X" is a digit between 0 and 9.

acc_len must be greater than .00002. A value bigger than 10 can cause overflow issues in the output BRAM. This ceiling is dependent on input levels, and varies across different applications. A single time sample can be accumulated by using a value of -1.

win_slice should be between 0 and 7. 0 corresponds to least significant bits an 7 corresponds to most significant bits.

num_beams should be between 1 and 7.

coeff_fileX is the location on the desired coefficient file. For correct coefficient file formatting, see Section 5.4.2

5.2.2 bf_updateBFCoeffs

Updates a beamformer's coefficients. In order to do this, the `bf(...)` command should have been previously called.

```
bf_coeff(roach,beam_num,coeff_file)
```

roach	Name of the ROACH board that has a beamformer
beam_num	Beamformer to modify
coeff_file	Location of the new coefficient file

beam_num should be a digit between 1 and 7.

5.2.3 bf_multi_updateBFCoeffs

Updates multiple beamformers with the same coefficient file.

```
bf_multi_updateBFCoeffs(roach, num_beams, coeff_file)
```

roach	Name of the ROACH board that has a beamformer
num_beams	Number of beamformers to update
coeff_file	Location of the new coefficient file

num_beams is the number of beams that will be updated. This always starts with beamformer 1 and goes up to the number specified.

5.2.4 bf_get_data

The output of each beamformer is being continually accumulated and stored in a BRAM on the roach. This stored value can be plotted and viewed on the host computer by calling this function.

```
bf_get_data(roach, beam_num, log)
```

roach	Name of the ROACH board that has a beamformer
beam_num	Beamformer number
log	Plots on a log scale. (optional)

log is an optional string which is used to plot the output data on a log scale (similar to MATLAB's semilogy(...))

5.2.5 bf_get_all_data

The outputs of multiple beamformers is plotted in the same figure.

```
bf_get_all_data(roach, num_beams, log)
```

roach	Name of the ROACH board that has a beamformer
num_beams	Number of beamformers to plot
log	Plots on a log scale. (optional)

num_beams is the number of beams to plot, starting with beamformer 1.

5.2.6 bf_set_slice

The 18_18 (real_imaginary) bit value output by the FFT must be sliced down to 4_4 before it is sent to the beamformers. This function sets which set of 4_4 bits are chosen. Idealy, this should be set so that it is as low as possible while allowing the the next bit after the lsb to toggle with no input.

```
bf_set_slice(roach, slice_num)
```

roach	Name of the ROACH board that has a beamformer
slice_num	Index of desired slice window

slice_num is the index of the desired slice. This can be any digit between 0 and 7. This index corresponds to the bit used for rounding which is the desired LSB - 1. For example: slice 0 contains bits 4 through 1 with bit 0 being used for rounding.

5.2.7 bf_set_acc_len

The accumulation length can be changed after initialization. This may be done at any time.

```
bf_set_acc_len(roach, acc_len)
```

roach	Name of the ROACH board that has a beamformer
acc_len	Accumulation length in seconds

acc_len must be greater than .00002. A value bigger than 10 can cause overflow issues in the output BRAM. This ceiling is dependent on input levels, and varies across different applications. A single time sample can be accumulated by using a value of -1.

5.3 Data Output

Not sure what the output will really be. just a plot maybe?

5.4 Coefficient Files

5.4.1 Coefficient File Organization

Like the ADC input ordering, the coefficient file has a confusing organization. (For more on ADC input ordering, see Section 4.3.1.) The organization comes from the need to present coefficients in the order that the ADC samples data. Because not all inputs are sampled simultaneously, they are also not presented to the beamformers simultaneously. As shown in Section 4.3.1, the beamformers see samples from inputs 0, 1, 16, 17, 32, 33, 48, 49 at once, followed by 8, 9, 24, 25, 40, 41, 56, 57 and so on. The coeffecient file must be organized so that all frequency bins for each set of eight inputs are presented before moving on to the next set of eight inputs.

In addition to the input ordering, the beamformers have been designed to use a single weight for four consecutive frequency bins. (This was done to meet timing constraints.) So, for the 64 input, 512 point FFT design, the coefficients can be organized into a 64 x 64 matrix.

For example, consider the matrix W

$$W_{m,n} = \begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{1,n-1} \\ w_{1,0} & w_{1,1} & \cdots & w_{2,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m-1,1} & w_{m-1,2} & \cdots & w_{m-1,n-1} \end{bmatrix}$$

where the m corresponds to each set of four frequency bins, and n corresponds to each input port.

The coefficients should be reorganized into a single vector before being written to the coefficient file. The correct organization of the coefficient vector $\overline{w}$ for the example matrix W is as follows:

$$\overline{w} = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,16} & w_{1,17} & w_{0,32} & w_{0,33} & w_{0,48} & w_{0,49} \\ w_{0,8} & w_{0,9} & w_{0,24} & w_{0,25} & w_{0,40} & w_{0,41} & w_{0,56} & w_{0,57} \\ w_{0,2} & w_{0,3} & w_{0,18} & w_{1,19} & w_{0,34} & w_{0,35} & w_{0,50} & w_{0,51} \\ w_{0,11} & w_{0,12} & w_{0,26} & w_{0,27} & w_{0,42} & w_{0,43} & w_{0,58} & w_{0,59} \\ w_{0,4} & w_{0,5} & w_{0,20} & w_{1,21} & w_{0,36} & w_{0,37} & w_{0,52} & w_{0,53} \\ w_{0,12} & w_{0,13} & w_{0,28} & w_{1,29} & w_{0,44} & w_{0,45} & w_{0,60} & w_{0,61} \\ w_{0,6} & w_{0,7} & w_{0,22} & w_{1,23} & w_{0,38} & w_{0,39} & w_{0,54} & w_{1,55} \\ w_{0,14} & w_{0,15} & w_{0,30} & w_{1,31} & w_{0,46} & w_{1,47} & w_{0,62} & w_{1,63} \\ w_{1,0} & w_{1,1} & w_{1,16} & w_{2,17} & w_{2,32} & w_{2,33} & w_{2,48} & w_{2,49} \cdots \\ w_{63,14} & w_{63,15} & w_{63,30} & w_{63,31} & w_{63,46} & w_{63,47} & w_{63,62} & w_{63,63} \end{bmatrix}$$

This vector follows the input port organization explained in Section 4.3.1.

5.4.2 Coefficient File Format

Each line of the coefficient file contains two complex, 16 bit (8 real, 8 imaginary), coefficients written as a two's compliment hexadecimal number. Any hex designation (\x, #, 0h, etc.) is left off. As stated, each line contains two coefficients. Referring back to the coefficient vector $\overline{w}$ these will be consecutive values, i.e. $w_{0,0}$ $w_{0,1}$. The file is a simple text file with a *.coe* extension.

As an example, consider the coefficients $c_1 = 0$, $c_2 = 1$, $c_3 = j$ and $c_4 = 1 + j$.

Using these coefficients, the file would be as follows:

Coefficient	Value	Hex Value
c_1	0	0000
c_2	1	0100
c_3	j	0001
c_4	1 + j	0101

```
00000100
00010101
<empty line>
```

Be sure to remember the final newline character.

6 Correlator

6.1 Summary

Stuff

6.2 Hardware Requirements

Stuff

6.3 TCP/IP Command Definitions

The following TCP/IP messages are used to control the Correlator:

x(...)	Starts the correlator

6.3.1 x

6.4 Data Output

Stuff

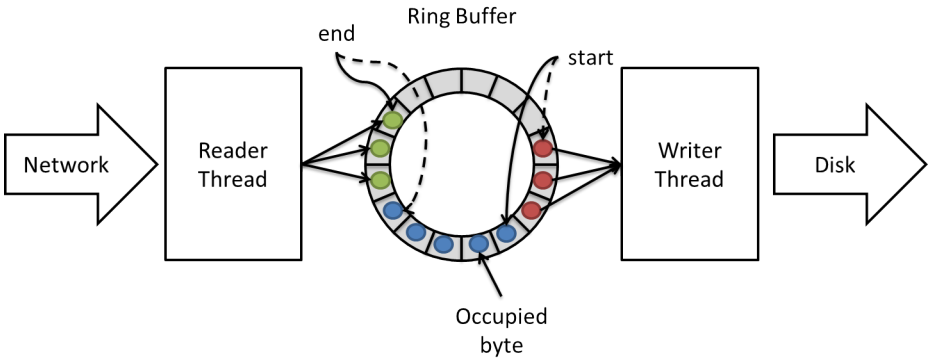
A Gulp Code

Gulp is a network packet sniffer developed by Corey Satten from the University of Washington (2007). The goal of the software was to provide a means of capturing data with speeds up to 1 Gigabits-per-second (Gbps) without loss. Satten's software was able to achieve this data rate with a 2.6 GHz Intel core2duo CPU.

For application in the x64 system, the 1 Gbps benchmark is insufficient. Therefore, several additions to both software and hardware were needed. Changes to hardware have been well documented in the main body of this text, but the changes made to the Gulp software have yet to be outlined. This appendix serves to address these changes. However, before describing these changes, it is important to understand the basic structure and function of Gulp.

A.1 Overview

The basic structure of Gulp is characterized by two threads on individual CPU cores: the first thread serves as a "Reader," which constantly monitors the activity at the Network Interface Controller (NIC), also known as the Ethernet port. It then transfers the data from the NIC to a buffer in RAM, hereafter referred to as the "ring buffer." The second thread serves as a "Writer," which extracts data from the ring buffer and saves it to a user-defined file on disk. Below is a diagram of this process.



A.2 Application Programming Interface (API)

append
Takes an array of data and appends it to the end of the ring buffer. If the appending marks the end of a file, the file boundary logic is updated appropriately.

char * ptr	Pointer to the next set of data to append to the buffer
int len	Length in bytes of the data to append
int bdry	Flag to specify if multiple files are being written

got_packet
The callback function to be used by pcap_loop. It is triggered by the reception of a new packet at the NIC. It also performs some error checking based on system parameters. Once, all pre-processing has been completed, the packet is written to the ring buffer through a call to `append`.

u_char * args	The last argument that was passed to the `pcap_loop` function
const struct pcap_pkthdr * header	The pcap header, which contains info such as time stamp, size, etc
const u_char * packet	The actual packet data

cleanup
The callback function used for the SIGINT signal (Ctrl+C). It breaks the pcap_loop call and begins a clean shutdown of the system.

int signo	The number associated with the signal SIGINT

Reader
A thread (CPU 1) that constantly monitors the activity on the NIC and captures packets that it sees. It then writes these packets to the ring buffer. It then writes a file header of `0xa1b2c3d4` and calls `pcap_loop`.

void * arg	The arguments passed to the thread. There are none for this application

newoutfile
Redirects the output of the Writer thread to the specified file.

char * dir	The name of the new file
int num	Deprecated

Writer
A thread (CPU 0) that copies a chunk of the ring buffer to the current file. If a write to disk will overwrite a file boundary, the `newoutfile` function is called.

void * arg	The arguments passed to the thread. There are none for this application

usage
Prints the usage statement for Gulp along with the various flags and switches that can be used. They are described here for convenience:

–help	Prints usage statement
-d	Decapsulates Cisco ERSPAN GRE packets (sets -f value)
-f ”..”	Specify the pcap filter
-i eth#	Specifies the ethernet interface to be used for capture
-s #	Specifies packet capture ”snapshot” length limit
-r #	Specifies the ring buffer size in megabytes
-c	Forces packets to output to stdout instead of a file
-x	Request an exclusive lock (to be the only instance running).
-X	Run even when locking would forbid it
-v	Print program version and exit
-V xx	Display verbose output (buffer use and packet loss)
-p	Specify full/empty polling interval in microseconds
-q	Suppress full buffer warnings
-z	Specify write block size (must be even power of 2, default 65536)
-o *dir*	Redirect output into a collection of files found in *dir*
-C #	Limits each file in the directory specified by -o to # times the ring buffer size
-W #	Overwrite files in the directory specified by -o instead of starting over

main
The main function. It starts the Reader and Writer threads, initializes the file boundary logic, and parses the command-line arguments. Once the two threads are independently running, this function begins printing status messages every 0.5 seconds (if ”-V xx” is set).

A.3 Changes

There are multiple things that needed to be changed in order to accommodate the x64 system. These changes will be addressed one at a time.

A.3.1 64-Bit Support

In order to utilize ring buffers larger than 4 GB, we need to have 64-bit support. Therefore, all variables that represent buffer indices or byte counts must be updated from `int` to `long`. These include the following (in a global context):

1. int maxbuffered
2. int ringsize
3. int volatile start, end
4. int volatile boundary (will soon change to array)

There are also some local variables that need to be changed from `int` to `long`. These are enumerated in the following table:

Function	Variables		
`append`	avail	used	
`Writer`	used	writesize	n
`main`	used		

A.3.2 Ring Buffer

The x64 system has approximately 192 GB of RAM. However, Gulp, by default, only supports ring buffers up to 1 GB in size. Therefore, the -r argument parsing code in the `main` function needs to be slightly changed.

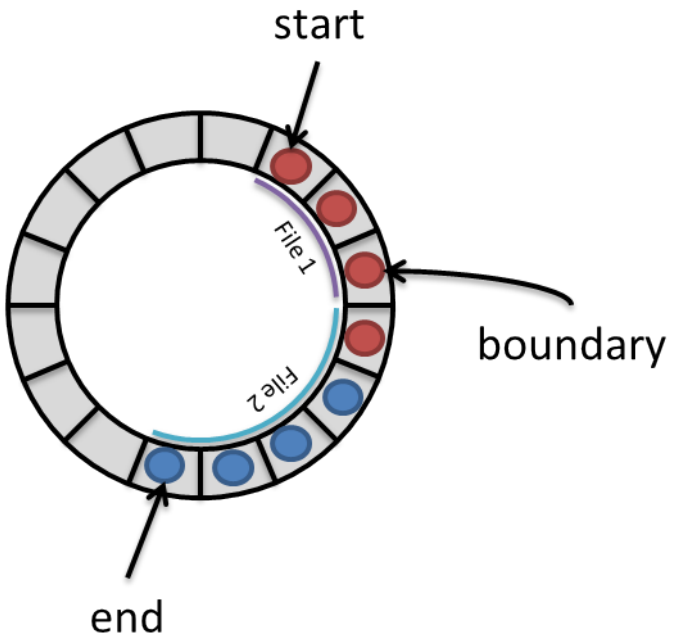
```
case 'r':
    t = atoi(optarg);
    if (t < 1 || t > 1024*180)
        fprintf(stderr, "%s: -r number must be 1-184320\n", progname);
        ++errflag;

    else
        ringsize = t * 1024*1024L;

    break;
```

This makes it possible to create ring buffers up to 180 GB.

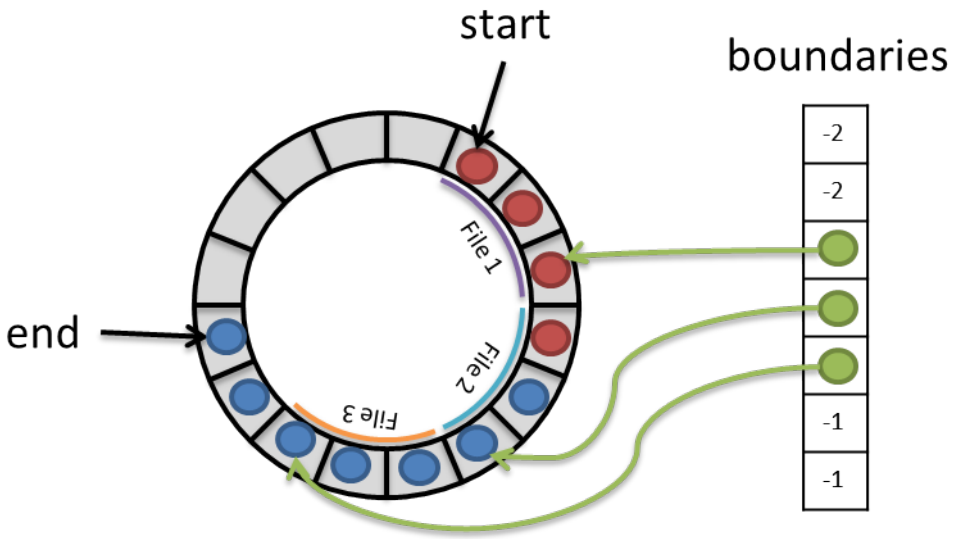
A.3.3 Multiple File Capture

The -o command-line argument already provides a mechanism for creating multiple files. However, this creates a file that are X times the ring buffer size large. However, for the vast majority of the x64 functions, there can be multiple files contained in a single ring buffer. Therefore, a single file boundary pointer is insufficient; an array of pointers is needed.

Gulp, by default, uses a single pointer to mark the byte location in the ring buffer where the current file should be finished. This is depicted in the below figure.



Since Gulp's default behavior is to only create new files when a certain number of ring buffers have been saved, a single pointer is sufficient. Therefore, to save multiple files in a single file, we need to use an array of pointers, as depicted in the below figure.



We therefore create the array globally, using a macro to specify the length.

```
#define MAX_BURSTS 1200
long volatile boundaries[MAX_BURSTS];
int volatile cur_bound = -1;
int volatile next_bound = 0;
```

In the append function, the boundary pointer logic needs to be updated.

```
if (just_wrapped && bdry) {
   just_wrapped = 0;
   if (odir && ++wrap_cnt >= split_after && split_after != -1) {
      if (cur_bound != -1) {
          while (boundaries[cur_bound] >= 0) {
             if (warned < push) {
                warned = push;
                if (warn_buf_full) {
                   fprintf(stderr,"%s: ring bufer full\n", progname);
                }
             }
              usleep(poll_usecs);
          }
      }
      boundaries[next_bound] = end;
      next_bound++;
      wrap_cnt = 0;
      if (!just_copy) {
         append((char *)&fh, sizeof(fh), 0);
      }
   }
}
```

Inside the Writer thread, the file boundary logic should also be updated.

```
if (writesize >= 0) {
  if (cur_bound != -1) {
    if (start<=boundaries[cur_bound] && start+writesize>=boundaries[cur_bound]){
      writesize = boundaries[cur_bound] - start;
    }
  }
  writesize = write(1, buf+start, writesize);
}

⋮

start += (start+writesize >= ringsize) ? writesize-ringsize : writesize;
if (cur_bound != -1) {
    if (start == boundaries[cur_bound]) {
        if (max_files && filec >= max_files) {
            filec = 0;
        }
        newoutfile(odir, filec++);
        boundaries[cur_bound++] = -2;
```

```
    }
    pushed = push;
}
```

Then a small initialization segment needs to be added to `main`.

```
int bcnt;
for (bcnt = 0; bcnt < 1100; bcnt++) {
    boundaries[bcnt] = -1;
}
```

Lastly, we need a command-line argument to put Gulp into a multiple-file capture mode. The below code will be added to the `main` function. The "-g" flag will be used.

```
case 'g':
   if (odir != 0) {
      fprintf(stderr, "%s: -g can't be used with -o\n", progname, odir);
      errflag++;
   }
   g_flag = 1;
   get_next_file();
   fprintf(stderr, "file_name = %s\n", file_name);
   break;
```

This implies that we need a global flag that indicates that the "-g" argument is being used.

```
int g_flag = 0;
```

As you may have guessed, the `get_next_file()` function needs to be defined as well. This waits for the user to input a new file name.

```
void get_next_file() {
    char temp_name[512];
    fprintf(stderr, "Waiting for new file...\n");
    fgets(temp_name, 511, stdin);

    int i;
    for (i = strlen(temp_name)-1; i > 0; i--) {
        if (temp_name[i] == '\n') {
            temp_name[i] = '\0';
        }
    }
    strcpy(file_name, temp_name);
}
```

A.3.4 SIGUSR1 Functionality

In conjunction with the multiple-file capture mode, we need a mechanism to trigger a new file. To this end, the SIGUSR1 system signal will be used.

First a callback function is needed for when the SIGUSR1 signal is asserted. This will update the file boundary pointers and, potentially, the file number.

```
void callback_handler(int signum) {
    boundaries[next_bound++] = end;
    if (cur_bound == -1) {
        cur_bound = 0;
    }
    get_next_file();
}
```

Next we link the SIGUSR1 signal with the callback function. This should be done in the `Reader` thread.

```
if (g_flag == 1) {
    struct sigaction res;
    res.sa_handler = callback_handler;
    sigemptyset(&res.sa_mask);
    res.sa_flags = 0;

    sigaction(SIGUSR1, &res, NULL);
}
```

A.3.5 Additional Output

Gulp, by default, will output dropped packets and ring buffer usage only. However, it would be nice to log how long a capture takes before a packet is dropped. Additionally, it would be nice to get real-time updates about packet drops in both the kernel **and** the NIC. Also, we would like to know how many packets have been captured by the NIC to account for any potential losses that are not detectable by the system (e.g. cable or switch).

Sevearl additional variables need to be declared in order to monitor these parameters. The following variables should be declared globally.

```
int packet_start = 0;
int start_time = 0;
int end_time = 0;
int end_time2 = 0;
int packet_drop = 0;
int packet_drop2 = 0;
```

Inside the `main` function's final `while` loop, modify the `fprintf` statements.

```
fprintf(stderr,
    "Time: %d.%d, Kernel Time: %d.%d, If Time: %d.%d, RX: %d, "
    + "Kernel drp: %d, If drp: %d, ring buf: %.1lf%%, max%.1lf%%\n",
    push/2, (push%2)*5,
    (end_time-start_time)/2, ((end_time-start_time)%2)*5,
    (end_time2-start_time)/2, ((end_time2-start_time)%2)*5,
    pcs.ps_recv,
    (drop_symb > 0 ? pcs.ps_drop : 0),
    pcs.ps_ifdrop,
    100.0*(double)used/(double)(ringsize),
    100.0*(double)maxbuffered/(double)(ringsize));
```

Lastly, a single status message is needed when Gulp is ready for packets. A simple "READY" message should be printed right before the `pcap_loop` call in the `Reader` thread.

```
fprintf(stderr, "READY!\n");
pcap_loop(handle, -1, got_packet, NULL);
```

This will make it possible for interface software to call Gulp as a sub-process and wait until it is fully initialized before commanding packets to arrive.