CurateND

UNIVERSITY OF NOTRE DAME | HESBURGH LIBRARIES

A Framework for Implementation and Evaluation of Cooperative Diversity in Software-Defined Radio

Glenn Joseph Bradford

Publication Date

19-12-2008

License

This work is made available under a All Rights Reserved license and should only be used in accordance with that license.

Citation for this work (American Psychological Association 7th edition)

Bradford, G. J. (2008). *A Framework for Implementation and Evaluation of Cooperative Diversity in Software-Defined Radio* (Version 1). University of Notre Dame. https://doi.org/10.7274/vh53ws87v0p

This work was downloaded from CurateND, the University of Notre Dame's institutional repository.

For more information about this work, to report or an issue, or to preserve and share your original work, please contact the CurateND team for assistance at curate@nd.edu.

Appendix A

PYTHON HIGH LEVEL CODE FOR DF RELAY NETWORK

A.1 Copyright Issues

GNU Radio is copyrighted by the Free Software Foundation (FSF), and is covered by the GNU General Public License version 3 (GPLv3).

Because this code is a derivative work of GNU Radio, it is also covered by the same license even if not explicitly stated in the listings.

A.2 experiment.py

```
#! /usr/bin/env python

# Simple DF relay network experiment code
from gnuradio import gr, packet_utils
from dbpsk_tb import exp_top
import time, struct, random
from math import log10

# Function to determine number of bits two packets differ in
def compare_str(string1,string2):

if(len(string1) != len(string2)):
print len(string1), len(string2)
raise ValueError
nwrong = 0
i = 10

while i < 500:
wrong = ord(string1[i]) ^ ord(string2[i])
for j in range(0,8):
nwrong += ( (wrong >> j) & 0x01)
i += 1

return nwrong

# Returns amplifier gains to be used
def tx_rng():
a = [31.6]
i = 0
while i < 10:
a.append(100)
i += 1

i = 0
while i < 100:
a.append(316)
i += 1

return a

# Main function that implements experiment
def main():
tb = exp_top() # create PHY
tb.start() # start PHY
f = open('final.dat','a')
access_code = '\xac\xdd\xa4\xe2\xf2\x8c\x20\xfc'
train = 50 * chr(0xaa)
pkt_length = 500

tx_range = tx_rng()

time.sleep(5)
print '\a'
cc = 0
while(cc < 100): # Number of times to loop thru tx powers
print cc
print '\a'

for i in tx_range: # tx power loop

# set TX amp for Soure and Relay
tb.s_tx.set_amp(i)
tb.r_tx.set_amp(i)

# reset bit accumulators
d0  = 0
```

```
d1  = 0
d2   = 0
r  = 0
rd = 0
bits = 0

j=0
pktno = 0
while j < 1: # packet loop
#------------------ Generate --------------------
k=0
payload = ''
while k < pkt_length - 10:
payload += chr(random.randint(0,255))
k += 1

pkt = access_code + struct.pack('!H', pktno) + payload
#-------------------- Source ---------------------
tb.s_tx.send_pkt(train + pkt + train)

#-------------------- Relay ----------------------
msg_r = None
t = time.time()
while time.time() - t < 0.5:
if(tb.r_de.check_q() != 0):
msg_r = tb.r_de.get_msg()
break
if msg_r == None:
print 'Pkt missed at relay'
pktno += 1
continue

r_data = msg_r.to_string()
pkt_r = access_code + r_data[8:]

# send msg from relay
tb.r_tx.send_pkt(train + pkt_r + train)

#----------------- Destination -------------------
msg_0 = None
msg_1 = None
msg_2 = None
t = time.time()
while (time.time() - t) < 0.5:
if(tb.div.check_q() != 0):
msg_0 = tb.div.get_msg()
break
if msg_0 == None:
tb.comb.msgq().flush()
pktno += 1
print 'Pkt(s) missed at destination'
continue

msg_1 = tb.dir.get_msg()
msg_2 = tb.rel.get_msg()

#------------------ Bit Errors --------------------
d0_data = msg_0.to_string()
d1_data = msg_1.to_string()
d2_data = msg_2.to_string()

if pkt[8:10] != d0_data[8:10]:
tb.comb.msgq().flush()
pktno += 1
print 'Pkt Numbers do not match'
continue

# compare rcvd packets to determine bit errors
```

```
d0 += compare_str(pkt, d0_data)
d1 += compare_str(pkt, d1_data)
d2 += compare_str(pkt, d2_data)
r  += compare_str(pkt, r_data)
rd += compare_str(r_data, d2_data)
bits += pkt_length * 8 - 10

j += 1
pktno +=1

#write data to file
f.write(str(i) + ' ' + str(d0) + ' ' + str(d1) + ' ' + str(d2) + ' ' + \
  str(r) + ' ' + str(rd) + ' ' + str(bits) + '\n')
f.flush()

cc += 1

tb.stop()
tb = None
print '\a'


if __name__ == '__main__':
try:
main()
except KeyboardInterrupt:
pass
```

A.3 dbpsk_tb.py

```
# Top block for PHY of simple DF implementation
from gnuradio import gr, packet_utils
from gnuradio import coop

from dbpsk import *
from options_50k import *
import u

class exp_top(gr.top_block):
def __init__(self):
gr.top_block.__init__(self)

# Parameters
rx_opts = rx_options()
tx_opts = tx_options()

# USRPs
self.u_s_tx = u.u_tx(0,tx_opts)
self.u_r_rx = u.u_rx(1,rx_opts)
self.u_r_tx = u.u_tx(1,tx_opts)
self.u_d_rx = u.u_rx(2,rx_opts)

# Receive Channel fileters
sw_decim = 1
self.chan_taps = gr.firdes.low_pass (1.0,
 sw_decim * rx_opts.samps_per_sym,
       1.0,
    0.5,
    gr.firdes.WIN_HANN)

self.filtr = gr.fir_filter_ccc(sw_decim,self.chan_taps)
self.filtd = gr.fir_filter_ccc(sw_decim,self.chan_taps)

# Network Node Components
self.s_tx = dbpsk_tx(tx_opts)

self.r_fe = dbpsk_fe(rx_opts)
self.r_de = dbpsk_de(rx_opts)
self.r_tx = dbpsk_tx(tx_opts)

self.d_fe = dbpsk_fe(rx_opts)
self.div = dbpsk_de(rx_opts)
self.dir = dbpsk_de(rx_opts)
self.rel = dbpsk_de(rx_opts)

arity = pow(2, rx_opts.bits_per_sym)
aa = packet_utils.default_access_code
access = int(packet_utils.default_access_code, 2)
access2 = []
for i in range(0,64):
access2.append(int(aa[i]))

self.comb = coop.comb_c(2, 1, 400) #diversit order, combo method:eq, queue limit
self.msg_sink = coop.msg_sink_c(blks2.psk.constellation[arity],
(0,1),
access2,
rx_opts.pkt_length * 8,
2 * 8,
     'snr_dest.dat',
     False,
     self.comb.msgq())

# Connect blocks
self.connect(self.s_tx, self.u_s_tx) #source

self.connect(self.u_r_rx, self.filtr, self.r_fe, self.r_de) #relay
```

```
self.connect(self.r_tx, self.u_r_tx)

self.connect(self.u_d_rx, self.filtd, self.d_fe, self.msg_sink) #destination
self.connect((self.comb,0), self.div)
self.connect((self.comb,1), self.dir)
self.connect((self.comb,2), self.rel)
```

A.4 dbpsk.py

```
# Hierarchical signal processing blocks for DF network
from gnuradio import gr, gru, blks2, packet_utils
from gnuradio import coop

#-----------------------------------------------------------------------------
#---------------------------- DBPSK Transmitter ---------------------------
#-----------------------------------------------------------------------------
class dbpsk_tx(gr.hier_block2):
def __init__(self, options):
gr.hier_block2.__init__(self,"dbpsk_rx",
gr.io_signature(0, 0, 0),
gr.io_signature(1, 1, gr.sizeof_gr_complex))
# passed values
self._bitrate = options.bitrate
self._samps_per_sym = options.samps_per_sym
self._ampl = options.ampl
self._excess_bw = options.excess_bw
self._bits_per_sym = options.bits_per_sym

arity = pow(2, self._bits_per_sym)

self.msg_src = gr.message_source(gr.sizeof_char, 1000)
self.bytes2chunks = gr.packed_to_unpacked_bb(self._bits_per_sym, gr.GR_MSB_FIRST)
self.symbol_mapper = gr.map_bb(blks2.psk.binary_to_gray[arity])
self.diffenc = gr.diff_encoder_bb(arity)
self.chunks2symbols = gr.chunks_to_symbols_bc(blks2.psk.constellation[arity])

ntaps = 11 * self._samps_per_sym
self.rrc_taps = gr.firdes.root_raised_cosine(self._samps_per_sym,
       self._samps_per_sym,
       1.0,
       self._excess_bw,
       ntaps)

self.rrc = gr.interp_fir_filter_ccf(self._samps_per_sym, self.rrc_taps)

self.amp = gr.multiply_const_cc(self._ampl)

self.connect(self.msg_src, self.bytes2chunks, self.symbol_mapper,
 self.diffenc, self.chunks2symbols, self.rrc,self.amp, self)

def send_pkt(self, pkt, eof=False):
if eof:
msg = gr.message(1)
else:
msg = gr.message_from_string(pkt)
self.msg_src.msgq().insert_tail(msg)

def send_msg(self, msg):
self.msg_src.msgq().insert_tail(msg)

def set_amp(self, amp):
self.amp.set_k(amp)

#-----------------------------------------------------------------------------
#---------------------------- DBPSK Front End ----------------------------
#-----------------------------------------------------------------------------
class dbpsk_fe(gr.hier_block2):
def __init__(self, options):
gr.hier_block2.__init__(self,"dbpsk_fe",
gr.io_signature(1, 1, gr.sizeof_gr_complex),
gr.io_signature(1, 1, gr.sizeof_gr_complex))
# passed values
self._bitrate = options.bitrate
self._samps_per_sym = options.samps_per_sym
self._excess_bw = options.excess_bw
```

```
self._bits_per_sym = options.bits_per_sym
self._decim = options.decim
self._pkt_length = options.pkt_length

# carrier and clock recovery constants
self._alpha  = options.alpha
self._beta  = options.beta
self._mu  = options.mu
self._gain_mu = options.gain_mu
self._omega = options.omega
self._gain_omega = options.gain_omega
self._omega_rel = options.omega_rel
self._fmin  = options.fmin
self._fmax  = options.fmax

arity = pow(2, self._bits_per_sym)
scale = (1.0/16384.0)

self.pre_scaler = gr.multiply_const_cc(scale)

ntaps = 11 * self._samps_per_sym
self.rrc_rx_taps = gr.firdes.root_raised_cosine(1.0,
         self._samps_per_sym,
         1.0,
         self._excess_bw,
         ntaps)

self.rrc_rx = gr.interp_fir_filter_ccf(1, self.rrc_rx_taps)

self.receiver = gr.mpsk_receiver_cc(arity, 0,
     self._alpha, self._beta,
     self._fmin, self._fmax,
     self._mu, self._gain_mu,
     self._omega, self._gain_omega,
     self._omega_rel)

self.diffdec = gr.diff_phasor_cc()


aa = packet_utils.default_access_code
self.corr = coop.access_cc(blks2.psk.constellation[arity],#constellation
 (0,1),#mapping
 int(aa,2),#access code
 12,  #threshold
 self._pkt_length * 8)  #pkt len syms


self.connect(self, self.pre_scaler, self.rrc_rx, self.receiver,
 self.diffdec, self.corr, self)

#-----------------------------------------------------------------------
#------------------------------- DBPSK Decode ------------------------------
#-----------------------------------------------------------------------
class dbpsk_de(gr.hier_block2):
def __init__(self, options):
gr.hier_block2.__init__(self,"dbpsk_de",
gr.io_signature(1, 1, gr.sizeof_gr_complex),
gr.io_signature(0, 0, 0))

self._bits_per_sym = options.bits_per_sym
self._pkt_length = options.pkt_length

arity = pow(2, self._bits_per_sym)
constellation = blks2.psk.constellation[arity]

self.slicer  = gr.constellation_decoder_cb(constellation,
range(arity))
self.mapper  = gr.map_bb(blks2.psk.gray_to_binary[arity])
```

```
self.pack  = gr.unpacked_to_packed_bb(self._bits_per_sym, gr.GR_MSB_FIRST)
self.msg_q  = gr.msg_queue(100)
self.pkt_sink = coop.pkt_sink_b(self._pkt_length, self.msg_q)

self.connect(self, self.slicer, self.mapper, self.pack, self.pkt_sink)

def check_q(self):
return self.msg_q.count()

def get_msg(self):
return self.msg_q.delete_head()
```

A.5 u.py

```
# Functions to instantiate USRP transmitter and receiver
from gnuradio import usrp

# USRP Transmitter
def u_tx(which, opts):
u = usrp.sink_c(which) #usrp as a sink
dac_rate = u.dac_rate()   #get dac rate

u.set_interp_rate(opts.interp) # set interpolation rate
spec = usrp.pick_tx_subdevice(u)
u.set_mux(usrp.determine_tx_mux_value(u,spec))
subdev = usrp.selected_subdev(u, spec)

freq_ok = u.tune(subdev._which,subdev, opts.tx_freq) # tune and check
if not freq_ok:
print "Failed to set TX frequency"
raise ValueError

subdev.set_gain(subdev.gain_range()[1]) # set gain
subdev.set_auto_tr(True)

return u

# USRP Receiver
def u_rx(which, opts):

u = usrp.source_c(which) # usrp as a source
adc_rate = u.adc_rate()     # get dac rate

u.set_decim_rate(opts.decim) # set decimation rate

spec = usrp.pick_rx_subdevice(u)
subdev = usrp.selected_subdev(u, spec)
u.set_mux(usrp.determine_rx_mux_value(u, spec))

r = subdev.gain_range() # setting gain
gain = (r[0] + r[1]) / 2
subdev.set_gain(gain)
u.tune(subdev._which, subdev, opts.rx_freq) # tune

subdev.set_auto_tr(True)

return u
```

A.6 options_50k.py

```
# System parameters to use in DF relay network
class rx_options:
def __init__(self):
self.rx_freq = 450e6
self.bitrate = 50e3
self.samps_per_sym = 5
self.excess_bw = 0.35
self.bits_per_sym = 1
self.decim  = 256
self.pkt_length = 500

self.alpha = 0.15
self.beta = 0.00562
self.mu = 0.5
self.gain_mu = 0.1
self.omega = self.samps_per_sym
self.gain_omega = 0.0025
self.omega_rel = 0.01
self.fmin =-0.025
self.fmax = 0.025


class tx_options:
def __init__(self):
self.tx_freq = 450e6
self.bitrate = 50e3
self.samps_per_sym = 5
self.excess_bw = 0.35
self.bits_per_sym = 1
self.interp = 512
self.ampl = 50
```

Appendix B

C++ SIGNAL PROCESSING BLOCKS FOR DF RELAY NETWORK

B.1 Copyright Issues

GNU Radio is copyrighted by the Free Software Foundation (FSF), and is covered by the GNU General Public License version 3 (GPLv3).

Because this code is a derivative work of GNU Radio, it is also covered by the same license even if not explicitly stated in the listings.

B.2 coop_access_cc.h

```
#ifndef INCLUDE_COOP_ACCESS_CC_H
#define INCLUDE_COOP_ACCESS_CC_H

#include <gr_block.h>
#include <gr_complex.h>


class coop_access_cc;
typedef boost::shared_ptr<coop_access_cc> coop_access_cc_sptr;

coop_access_cc_sptr coop_make_access_cc
(std::vector<gr_complex> constellation, std::vector<int> mapping,
unsigned long long access, int threshold, int pcklength);


class coop_access_cc : public gr_block
{
private:
std::vector<gr_complex> d_constellation;
std::vector<int> d_mapping;
int d_pcklength;
int d_threshold;
int d_count;

unsigned long long d_register;
unsigned long long d_access;
unsigned long long d_mask;
int d_buf_len;
int d_buf_pos;
gr_complex* d_buffer;

void init_mask();
void slice(gr_complex in);

friend coop_access_cc_sptr coop_make_access_cc
(std::vector<gr_complex> constellation, std::vector<int> mapping,
 unsigned long long access, int threshold, int pcklength);


protected:
coop_access_cc(std::vector<gr_complex> constellation, std::vector<int> mapping,
       unsigned long long access, int threshold, int d_pcklength);


public:
~coop_access_cc();

int general_work(int noutput_items,
 gr_vector_int &ninput_items,
 gr_vector_const_void_star &input_items,
 gr_vector_void_star &output_items);
};

#endif /*INCLUDE_COOP_ACCESS_CC_H*/
```

B.3 coop_access_cc.cc

```
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <coop_access_cc.h>
#include <gr_io_signature.h>
#include <iostream>
#include <gr_count_bits.h>

coop_access_cc_sptr coop_make_access_cc
(std::vector<gr_complex> constellation, std::vector<int> mapping,
unsigned long long access, int threshold, int pcklength)
{
return coop_access_cc_sptr (new coop_access_cc
(constellation, mapping, access, threshold, pcklength));
}

coop_access_cc::coop_access_cc
(std::vector<gr_complex> constellation, std::vector<int> mapping,
unsigned long long access, int threshold, int pcklength)
: gr_block ("access_cc",
gr_make_io_signature (1, 1, sizeof(gr_complex)),
gr_make_io_signature (1, 1, sizeof(gr_complex)))
{
d_constellation = constellation;
d_threshold = threshold;
d_pcklength = pcklength;
d_mapping = mapping;
d_count = 0;

d_access = access;
d_register = 0;
d_buf_pos = 0;
d_buf_len = 0;

init_mask();

d_buffer = new gr_complex[d_buf_len];
}

coop_access_cc::~coop_access_cc()
{
delete d_buffer;
}

void coop_access_cc::init_mask()
{
d_mask = 0;
while(d_mask < d_access) {
d_mask = (d_mask << 1) | 0x1;
d_buf_len++;
}
}

void coop_access_cc::slice(gr_complex in)
{
unsigned int alphabet = d_constellation.size();
unsigned int min_index;
float min_dist = 0;
float dist = 0;

// find closest constellation point
min_index = 0;
min_dist = norm(in - d_constellation[0]);
```

```
for(unsigned int i = 1; i < alphabet; i++) {
dist = norm(in - d_constellation[i]);
if(dist < min_dist) {
min_dist = dist;
min_index = i;
}
}

// shift in corresponding bits to register
//****fix for more than 1 bit/sym later******
d_register = (d_register << 1) | (d_mapping[min_index] & 0x1);
}

int coop_access_cc::general_work
(int noutput_items,
 gr_vector_int &ninput_items,
 gr_vector_const_void_star &input_items,
 gr_vector_void_star &output_items)
{
const gr_complex *in = (const gr_complex*) input_items[0];
gr_complex *out = (gr_complex *) output_items[0];
unsigned long long wrong_bits;
int outcount = 0;
int nwrong;

for(int i=0; i < ninput_items[0]; i++) {

// output if in the middle of a packet
if(d_count > 0) {
out[outcount] = d_buffer[d_buf_pos];
outcount++;
d_count--;
}

// insert new symbol
slice(in[i]);
d_buffer[d_buf_pos] = in[i];
d_buf_pos = (d_buf_pos + 1) % d_buf_len;

if(d_count == 0) {
// set to output if this input puts us under the threshold
wrong_bits = (d_register ^ d_access) & d_mask;
nwrong = gr_count_bits64(wrong_bits);

if(nwrong <= d_threshold) {
d_count = d_pcklength;
//printf("Pkt Threshold met. Wrong bits: %d\n", nwrong);
}
}
}

consume_each(ninput_items[0]);

return (outcount);
}
```

B.4 coop_access_cc.i

```
GR_SWIG_BLOCK_MAGIC(coop, access_cc);

coop_access_cc_sptr coop_make_access_cc
(std::vector<gr_complex> constellation, std::vector<int> mapping,
unsigned long long access, int threshold, int pcklength);

class coop_access_cc : public gr_block
{
protected:
coop_access_cc(std::vector<gr_complex> constellation, std::vector<int> mapping,
       unsigned long long access, int threshold, int pcklength);
public:
~coop_access_cc();
};
```

B.5 coop_comb_c.h

```
#ifndef INCLUDE_COOP_COMB_C_H
#define INCLUDE_COOP_COMB_C_H

#include <gr_block.h>
#include <gr_msg_queue.h>
#include <gr_message.h>

class coop_comb_c;
typedef boost::shared_ptr<coop_comb_c> coop_comb_c_sptr;


coop_comb_c_sptr coop_make_comb_c
(int diversity, int method, int limit);

class coop_comb_c : public gr_block
{
private:
int d_diversity;
int d_method;

int d_open;
unsigned d_msg_offset;
float* d_factors;
gr_msg_queue_sptr d_msgq;
gr_message_sptr* d_msg;

void set_factors();

friend coop_comb_c_sptr coop_make_comb_c
(int diversity, int method, int limit);

protected:
coop_comb_c(int diversity, int method, int limit);

public:
~coop_comb_c();
gr_msg_queue_sptr msgq() { return d_msgq; };

int general_work(int noutput_items,
  gr_vector_int &ninput_items,
  gr_vector_const_void_star &input_items,
  gr_vector_void_star &output_items);

};

#endif /*INCLUDE_COOP_COMB_C_H*/
```

B.6 coop_comb_c.cc

```
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <coop_comb_c.h>
#include <gr_io_signature.h>
#include <stdexcept>


coop_comb_c_sptr coop_make_comb_c
(int diversity, int method, int limit)
{
return coop_comb_c_sptr (new coop_comb_c(diversity, method, limit));
}


coop_comb_c::coop_comb_c
(int diversity, int method, int limit)
 : gr_block("comb_c",
 gr_make_io_signature(0, 0, 0),
 gr_make_io_signature(1, gr_io_signature::IO_INFINITE, sizeof(gr_complex)))
{
d_diversity = diversity;
d_method = method;

d_open = 0; //~~
d_msg_offset = 0;
d_msgq = gr_make_msg_queue(limit);
d_msg = new gr_message_sptr[diversity];
d_factors = new float[diversity];
}

coop_comb_c::~coop_comb_c()
{
delete [] d_msg;
delete [] d_factors;
}

void coop_comb_c::set_factors()
{
if(d_method == 2) { //selections

int max = 0;
float max_val = d_msg[0]->arg2();
float temp_val = 0;
for(int i=1; i < d_diversity; i++) {
temp_val = d_msg[i]->arg2();
if(temp_val > max_val) {
max = i;
max_val = temp_val;
}
}
for(int i=0; i < d_diversity; i++) {
d_factors[i] = 0;
}
d_factors[max] = 1;

} else if(d_method == 1) { //max ratio

float sum = 0;
for(int i=0; i < d_diversity; i++) {
d_factors[i] += float( d_msg[i]->arg2() );
sum += d_factors[i];
}
for(int i=0; i < d_diversity; i++) {
```

```
d_factors[i] = d_factors[i] / sum;
}

} else { //equal gain

float eq_gain = (float) 1 / d_diversity;
for(int i=0; i < d_diversity; i++) {
d_factors[i] = eq_gain;
}
}
}

int coop_comb_c::general_work
(int noutput_items,
 gr_vector_int &ninput_items,
 gr_vector_const_void_star &input_items,
 gr_vector_void_star &output_items)
{
int nout = output_items.size();
gr_complex* out[nout];
int nn = 0;
size_t itemsize = sizeof(gr_complex);

for(int i=0; i < nout; i++) {
out[i] = (gr_complex*) output_items[i];
}

while(nn < noutput_items) {

if(d_open == d_diversity) { //messages open, consume what we have ~~

      int mm = std::min(noutput_items - nn, (int)((d_msg[0]->length() - d_msg_offset) / itemsize));
      gr_complex out_temp[mm];

for(int i=0; i < d_diversity; i++) {
memcpy(&out_temp, &(d_msg[i]->msg()[d_msg_offset]), mm * itemsize);

for(int j=0; j < mm; j++) {
if(i == 0) { //assign if first time through
out[0][nn + j] = d_factors[0] * out_temp[j];
} else { //add in additional paths
out[0][nn + j] += d_factors[i] * out_temp[j];
}
if(i+1 < nout) { //output without diversity if there is a connection
out[i+1][nn + j] = out_temp[j];
}
}

}

nn += mm;
d_msg_offset += mm * itemsize;
      assert(d_msg_offset <= d_msg[0]->length());

if(d_msg_offset == d_msg[0]->length()) { //end of messages, reset
d_open = 0; //~~
}

} else { //not enough messages open

/*if(d_msgq->count() < d_diversity && nn > 0) // not enough msgs in queue
break;

for(int i=0; i < d_diversity; i++) { // open new msgs
d_msg[i] = d_msgq->delete_head();
if ((d_msg[i]->length() % itemsize) != 0)
throw std::runtime_error("msg length is not a multiple of d_itemsize");
}
```

```
d_open = true;
printf("SD: %f RD: %f\n", d_msg[0]->arg1(), d_msg[1]->arg1() );
set_factors();
d_msg_offset = 0;*/

if(d_msgq->count() == 0)// && nn > 0)
break;
while((d_msgq->count() > 0) && (d_open < d_diversity)) {

d_msg[d_open] = d_msgq->delete_head();
d_open++;

if(d_open > 1) {
if( d_msg[d_open-2]->arg1() != d_msg[d_open-1]->arg1() ) {
d_msg[0] = d_msg[d_open-1];
printf("Packet dropped at combiner\n");
d_open = 1;
}
}

}
if(d_open > 1) {
printf("Msg0: %f Msg1: %f\n",d_msg[0]->arg1(), d_msg[1]->arg1());
set_factors();
d_msg_offset = 0;
}

}
}
return nn;
}
```

B.7 coop_comb_c.i

```
GR_SWIG_BLOCK_MAGIC(coop, comb_c);

coop_comb_c_sptr coop_make_comb_c
(int diversity, int method, int limit);

class coop_comb_c : public gr_block
{
protected:
friend coop_comb_c_sptr coop_make_comb_c
(int diversity, int method, int limit);

public:
~coop_msg_sink();
gr_msg_queue_sptr msgq() { return d_msgq; };

};
```

B.8 coop_msg_sink_c.h

```
#ifndef INCLUDE_COOP_MSG_SINK_C_H
#define INCLUDE_COOP_MSG_SINK_C_H

#include <gr_sync_block.h>
#include <gr_message.h>
#include <gr_msg_queue.h>
#include <fstream>

class coop_msg_sink_c;
typedef boost::shared_ptr<coop_msg_sink_c> coop_msg_sink_c_sptr;


coop_msg_sink_c_sptr coop_make_msg_sink_c
(std::vector<gr_complex> constellation, std::vector<int> mapping,
std::vector<int> header, int pkt_length, int pktno_length, char* file_name,
bool std_out, gr_msg_queue_sptr msgq);


class coop_msg_sink_c : public gr_sync_block
{
private:
// inputs
std::vector<gr_complex> d_constellation;
std::vector<int> d_mapping;
std::vector<int> d_header;
unsigned int d_pkt_length;
unsigned int d_pktno_length;
bool d_std_out;
gr_msg_queue_sptr d_msgq;

std::ofstream d_file;
unsigned int d_count;
gr_message_sptr d_msg;
unsigned char* d_data_ptr;
unsigned int d_pktno;

//SNR variables
gr_complex d_C;
gr_complex d_E;
gr_complex d_a;

int slice(gr_complex in);
void error_add(gr_complex in);

friend coop_msg_sink_c_sptr coop_make_msg_sink_c
(std::vector<gr_complex> constellation, std::vector<int> mapping,
std::vector<int> header, int pkt_length, int pktno_length, char* file_name,
bool std_out, gr_msg_queue_sptr msgq);

protected:
coop_msg_sink_c
(std::vector<gr_complex> constellation, std::vector<int> mapping,
std::vector<int> header, int pkt_length, int pktno_length, char* file_name,
bool std_out, gr_msg_queue_sptr msgq);

public:
int work(int noutput_items,
 gr_vector_const_void_star &input_items,
 gr_vector_void_star &output_items);

};

#endif /*INCLUDE_COOP_MSG_SINK_C_H*/
```

B.9 coop_msg_sink_c.cc

```
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <coop_msg_sink_c.h>
#include <gr_io_signature.h>

coop_msg_sink_c_sptr coop_make_msg_sink_c
(std::vector<gr_complex> constellation, std::vector<int> mapping,
std::vector<int> header, int pkt_length, int pktno_length, char* file_name,
bool std_out, gr_msg_queue_sptr msgq)
{
return coop_msg_sink_c_sptr (new coop_msg_sink_c(constellation, mapping,
header, pkt_length, pktno_length, file_name, std_out, msgq) );
}


coop_msg_sink_c::coop_msg_sink_c
(std::vector<gr_complex> constellation, std::vector<int> mapping,
std::vector<int> header, int pkt_length, int pktno_length, char* file_name,
bool std_out, gr_msg_queue_sptr msgq)
: gr_sync_block ("msg_sink_c",
    gr_make_io_signature(1, 1, sizeof(gr_complex)),
    gr_make_io_signature(0, 0, 0))
{
gr_complex header_temp;

d_constellation = constellation;
d_mapping = mapping;
d_header = header;
d_pkt_length = pkt_length;
d_pktno_length = pktno_length;
d_std_out = std_out;
d_msgq = msgq;

d_count = 0;
d_pktno = 0;

d_msg = gr_make_message(0, 0, 0, d_pkt_length * sizeof(gr_complex) );
d_data_ptr = d_msg->msg();

d_file.open(file_name);

for(unsigned int i = 0; i < d_header.size(); i++) {
header_temp = d_constellation[d_header[i]];
d_a += header_temp * conj(header_temp);
}
}

int coop_msg_sink_c::slice(gr_complex in)
{
int constellation_size = d_constellation.size();
int min_index;
double min_dist;
double dist;

//slice input to find min index
min_index = 0;
dist = norm(in - d_constellation[0]);
min_dist = dist;
for(int i = 1; i < constellation_size; i++) {
dist = norm(in - d_constellation[i]);
if(dist < min_dist) {
min_dist = dist;
min_index = i;
```

```
}
}

return min_index;
}

void coop_msg_sink_c::error_add(gr_complex in)
{
d_C += in * conj(d_constellation[d_header[d_count]]);
d_E += in * conj(in);
}

int coop_msg_sink_c::work(int noutput_items,
     gr_vector_const_void_star &input_items,
     gr_vector_void_star &output_items)
{
const gr_complex *in = (const gr_complex*) input_items[0];
int ind;

for(int i = 0; i < noutput_items; i++) {

if(d_count < d_header.size()) { // accumulate for SNR estimate

error_add(in[i]);

} else if(d_count < d_header.size() + d_pktno_length) { //determine pktno

ind = slice(in[i]);
d_pktno = (d_pktno << 1) | (d_mapping[ind] & 0x1);

} else if(d_count == d_header.size() + d_pktno_length) { // output pktno and SNR to file

gr_complex snr = ( d_C * conj(d_C) ) / ( d_a * d_E - d_C * conj(d_C) );

if(d_file.is_open()) {
d_file << "Pktno: " << d_pktno << " SNR: " << snr.real() << "\n";
d_file.flush();
}
if(d_std_out == true) {
printf("Pktno: %d  SNR: %f\n", d_pktno, snr.real());
}

d_msg->set_arg1(d_pktno);
d_msg->set_arg2(snr.real());

//reset for next time around
d_pktno = 0;
d_C = 0;
d_E = 0;
}

//copy input symbol to current message
memcpy(d_data_ptr + sizeof(gr_complex) * d_count, &in[i], sizeof(gr_complex) );

d_count++;
if(d_count == d_pkt_length) {
d_count = 0;
d_msgq->insert_tail(d_msg);
d_msg = gr_make_message(0, 0, 0, d_pkt_length * sizeof(gr_complex) );
d_data_ptr = d_msg->msg();
}
}

return noutput_items;
}
```

B.10 coop_msg_sink_c.i

```
GR_SWIG_BLOCK_MAGIC(coop, msg_sink_c);

coop_msg_sink_c_sptr coop_make_msg_sink_c
(std::vector<gr_complex> constellation, std::vector<int> mapping,
std::vector<int> header, int pkt_length, int pktno_length, char* file_name,
bool std_out, gr_msg_queue_sptr msgq);

class coop_msg_sink_c : public gr_sync_block
{
protected:
coop_msg_sink_c
(std::vector<gr_complex> constellation, std::vector<int> mapping,
std::vector<int> header, int pkt_length, int pktno_length, char* file_name,
bool std_out, gr_msg_queue_sptr msgq);
};
```

B.11 coop_pkt_sink_b.h

```
#ifndef INCLUDE_COOP_PKT_SINK_B_H
#define INCLUDE_COOP_PKT_SINK_B_H

#include <gr_sync_block.h>
#include <gr_message.h>
#include <gr_msg_queue.h>

class coop_pkt_sink_b;
typedef boost::shared_ptr<coop_pkt_sink_b> coop_pkt_sink_b_sptr;


coop_pkt_sink_b_sptr coop_make_pkt_sink_b
(int pkt_length, gr_msg_queue_sptr msgq);


class coop_pkt_sink_b : public gr_sync_block
{
private:
unsigned int d_pkt_length;
gr_msg_queue_sptr d_msgq;

unsigned int d_count;
gr_message_sptr d_msg;
unsigned char* d_data_ptr;

friend coop_pkt_sink_b_sptr coop_make_pkt_sink_b
(int pkt_length, gr_msg_queue_sptr msgq);

protected:
coop_pkt_sink_b(int pkt_length, gr_msg_queue_sptr msgq);

public:
int work(int noutput_items,
gr_vector_const_void_star &input_items,
gr_vector_void_star &output_items);

};

#endif /*INCLUDE_COOP_PKT_SINK_B_H*/
```

B.12 coop_pkt_sink_b.cc

```
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <coop_pkt_sink_b.h>
#include <gr_io_signature.h>

coop_pkt_sink_b_sptr coop_make_pkt_sink_b
(int pkt_length, gr_msg_queue_sptr msgq)
{
return coop_pkt_sink_b_sptr (new coop_pkt_sink_b
(pkt_length, msgq));
}


coop_pkt_sink_b::coop_pkt_sink_b
(int pkt_length, gr_msg_queue_sptr msgq)
: gr_sync_block ("pkt_sink_b",
    gr_make_io_signature(1, 1, sizeof(char)),
    gr_make_io_signature(0, 0, 0))
{
d_pkt_length = pkt_length;
d_msgq = msgq;

d_count = 0;
d_msg = gr_make_message(0, 0, 0, d_pkt_length * sizeof(char) );
d_data_ptr = d_msg->msg();
}

int coop_pkt_sink_b::work
(int noutput_items,
 gr_vector_const_void_star &input_items,
 gr_vector_void_star &output_items)
{
const char *in = (const char*) input_items[0];

for(int i = 0; i < noutput_items; i++) {

//copy input symbol to current message
memcpy(d_data_ptr + sizeof(char) * d_count, &in[i], sizeof(char) );

d_count++;
if(d_count == d_pkt_length) {
d_count = 0;
d_msgq->insert_tail(d_msg);
d_msg = gr_make_message(0, 0, 0, d_pkt_length * sizeof(char) );
d_data_ptr = d_msg->msg();
}
}

return noutput_items;
}
```

B.13 coop_pkt_sink_b.i

```
GR_SWIG_BLOCK_MAGIC(coop, pkt_sink_b);

coop_pkt_sink_b_sptr coop_make_pkt_sink_b
(int pkt_length, gr_msg_queue_sptr msgq);


class coop_pkt_sink_b : public gr_sync_block
{
protected:
coop_pkt_sink_b(int pkt_length, gr_msg_queue_sptr msgq);
};
```