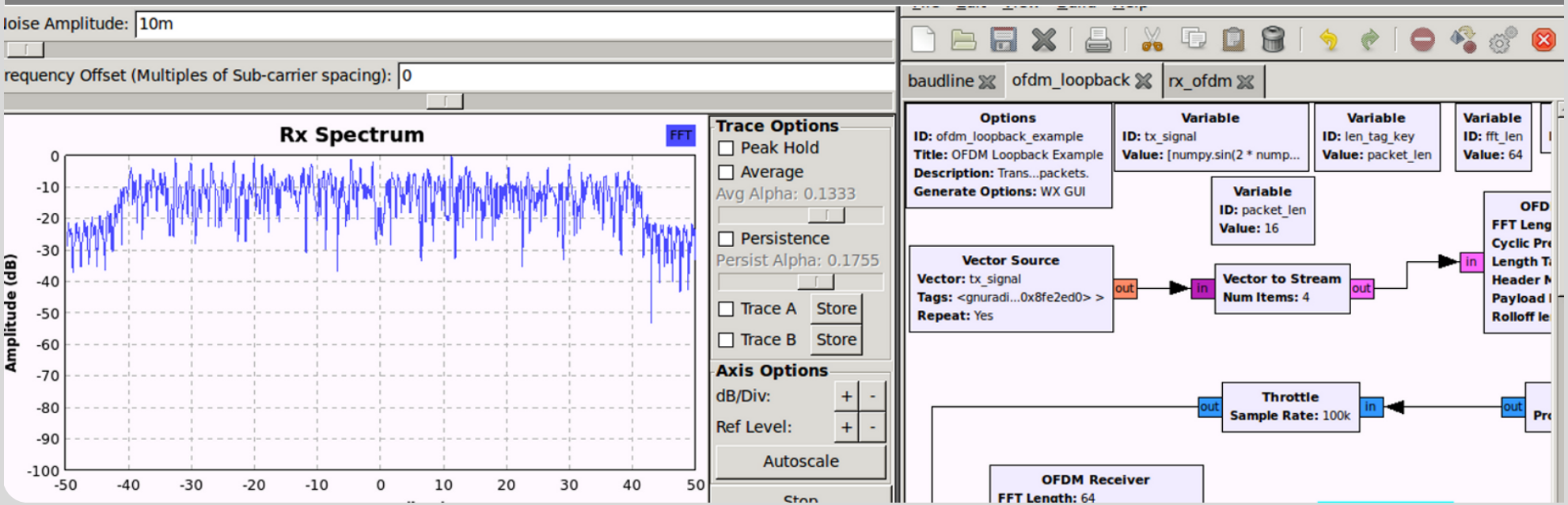
Karlsruher Institut für Technologie

Say, OFDM – You're looking fantastic these days

GNU Radio Conference, 2013, Boston

Martin Braun | 2.10.2013

Communications Engineering Lab

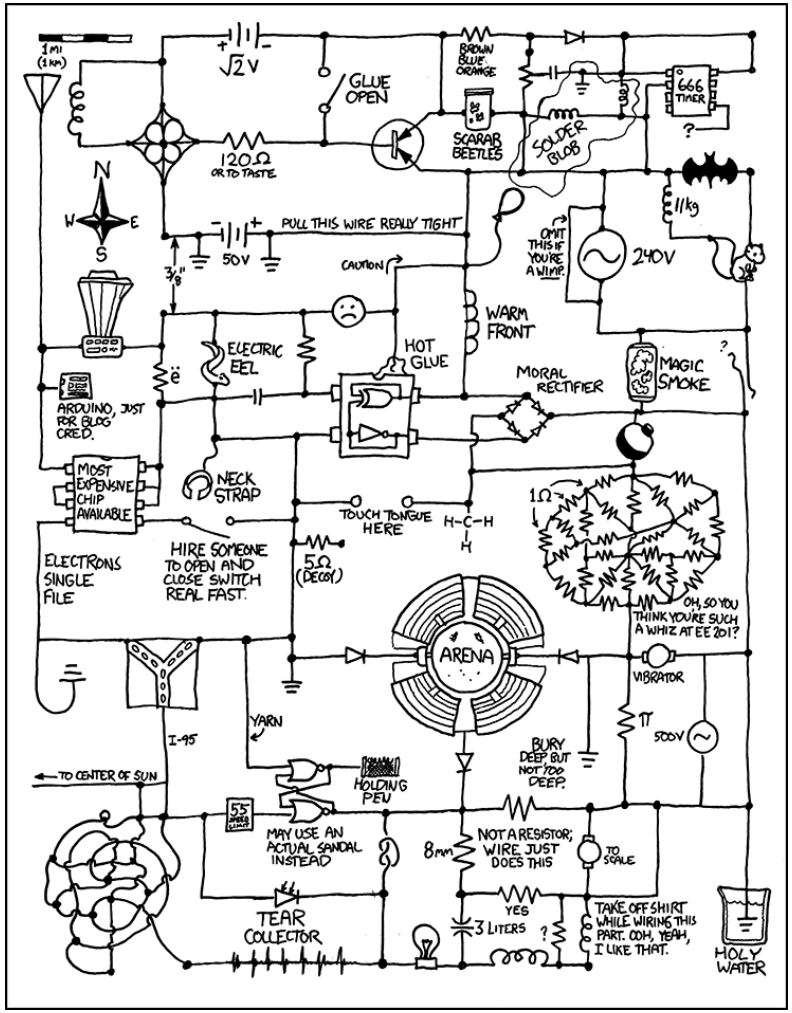


Before I start...

- ...a word on `rx_ofdm.grc`.
- Is it the new `tunnel.py`?
- It looks nice
- Was never intended for productive use

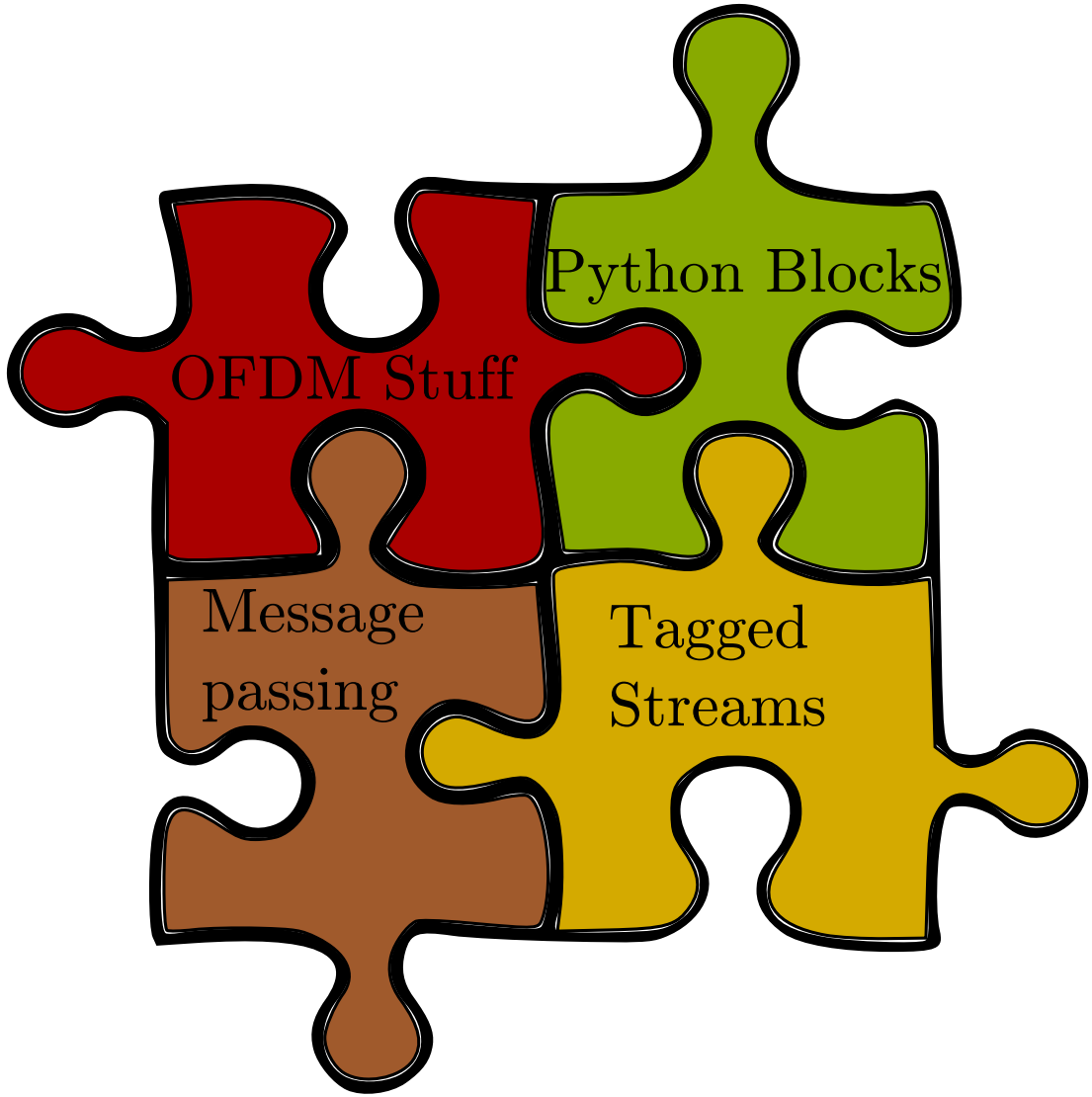
New OFDM codes

- Last 3.6 and 3.7 releases came with new OFDM codes
- What was wrong with the old ones?
  - Hard to reconfigure
  - Convoluted flow graphs
  - Didn't match up with intuition on how GNU Radio apps should work
- . . . but they were like that for a reason!



More than OFDM blocks was needed...

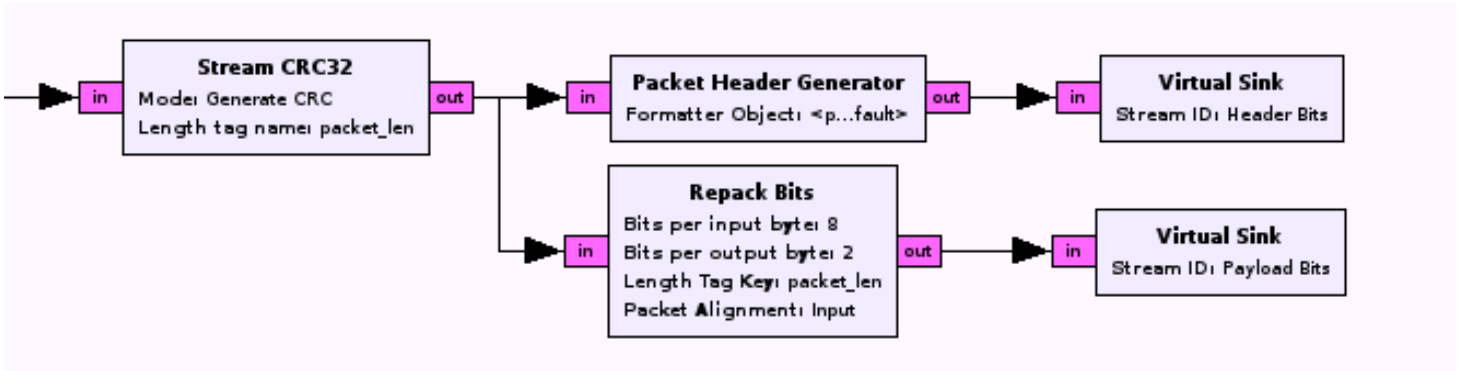
- Message passing
- Python Blocks
- Tagged stream blocks



OFDM – Wishlist

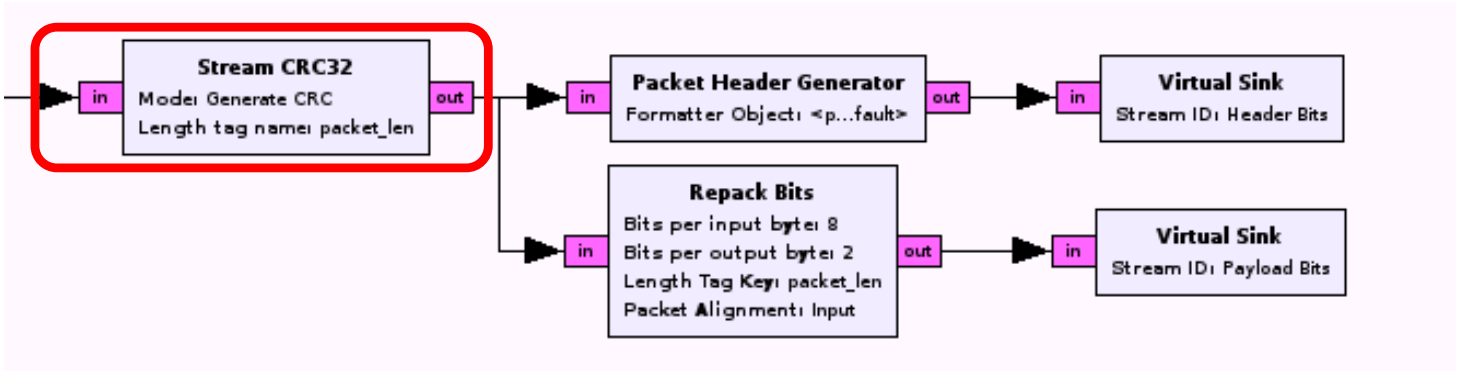
- Fully configurable frame configuration (pilot tones, occupied carriers. . .)
  - Can we reconfigure the whole thing to do 802.11a *and* DAB?
- Any part of the flow graph should be exchangeable
- . . . and individually useful
- Scopes, file sinks etc. can be inserted anywhere
- Headers are created dynamically, different modulation than payload

OFDM Tx Implementation Walkthrough



- The very first block is already causing problems!
- It's kind of a sync block. . . but needs to be basic block
- Perhaps we need a new block type?

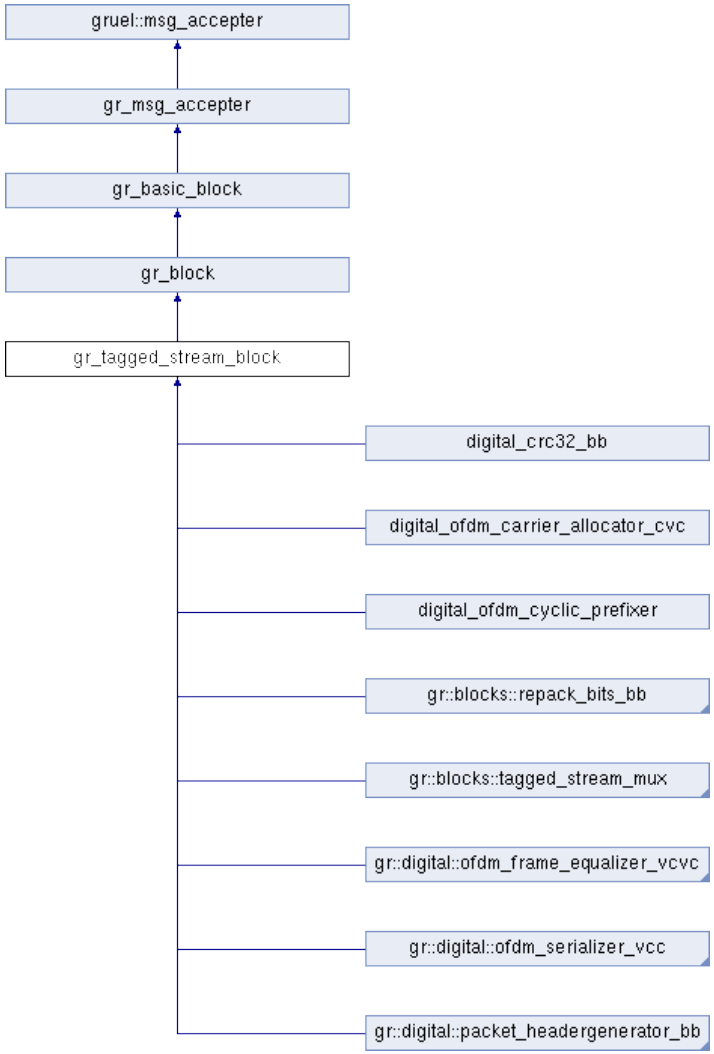
OFDM Tx Implementation Walkthrough



- The very first block is already causing problems!
- It's kind of a sync block. . . but needs to be basic block
- Perhaps we need a new block type?

gr_tagged_stream_blocks

- Handle stream boundaries (“Dynamic Vector Lengths”)
- Input-driven
- Uses tags
- Not really the same category as sync, decimator, interpolator
- Tag on the first item defines packet length
- Examples:
  - CRC32
  - Most OFDM stuff
  - FEC

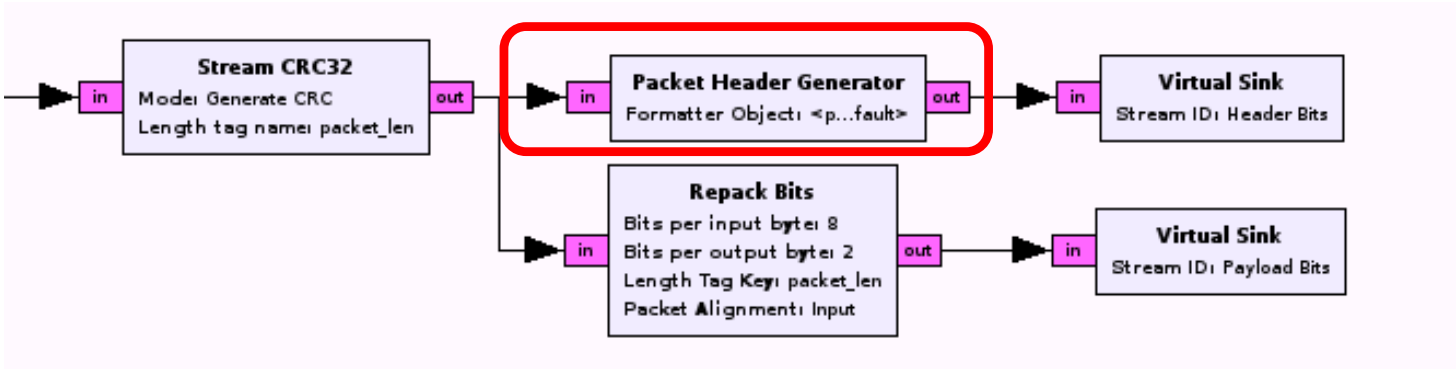


Example: Packet Header Generator

```
76    int
77    packet_headergenerator_bb_impl::work (int noutput_items,
78                        gr_vector_int &ninput_items,
79                        gr_vector_const_void_star &input_items,
80                        gr_vector_void_star &output_items)
81    {
82      unsigned char *out = (unsigned char *) output_items[0];
83      if (!d_formatter->header_formatter(ninput_items[0], out)) {
84        GR_LOG_FATAL(d_logger, boost::format("header_formatter() returned false (this shouldn't happen). Offending header started
85        throw std::runtime_error("header formatter returned false.");
86      }
87
88      return d_formatter->header_len();
89    }
```

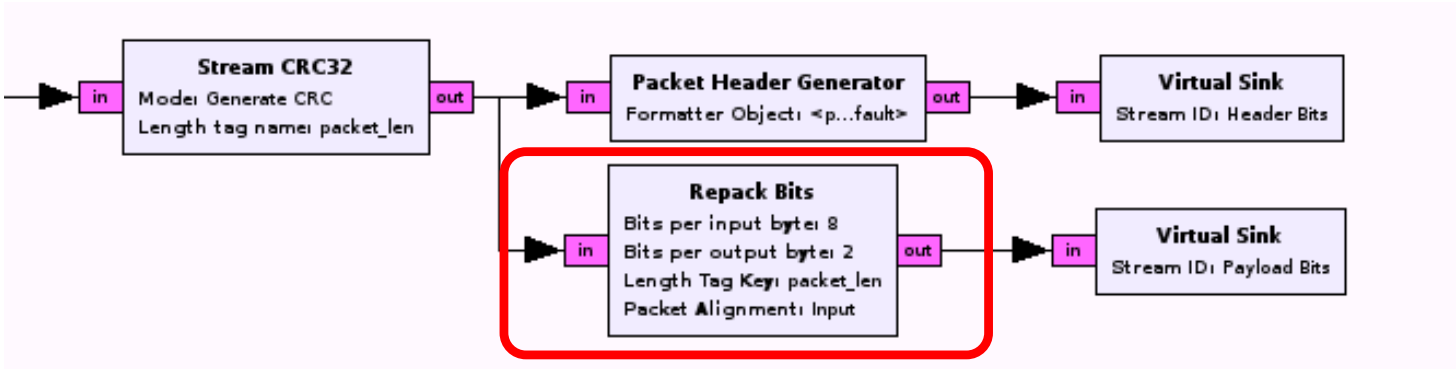
- Extremely simple work function
- Irregular packet size is handled automatically (no `consume()` necessary)

OFDM Tx Implementation Walkthrough



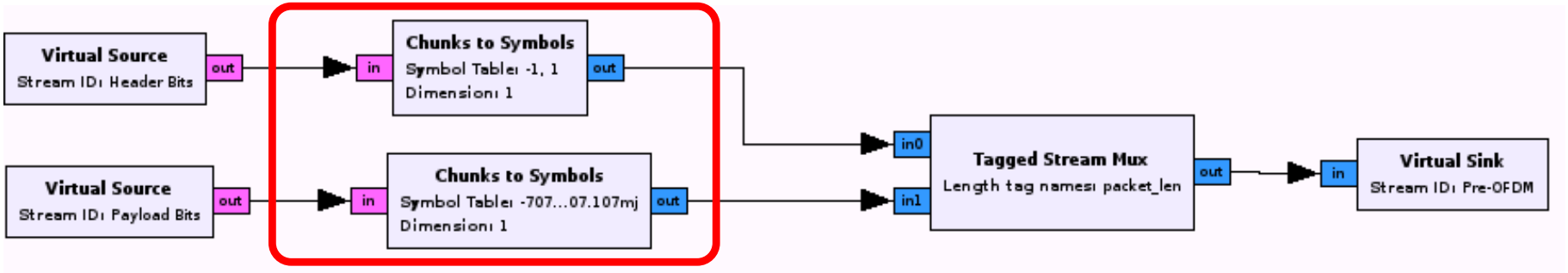
- Packet Header Generator: Packet Header is defined by a packet header generator class
- Bit Repacker: Also tagged stream block

OFDM Tx Implementation Walkthrough



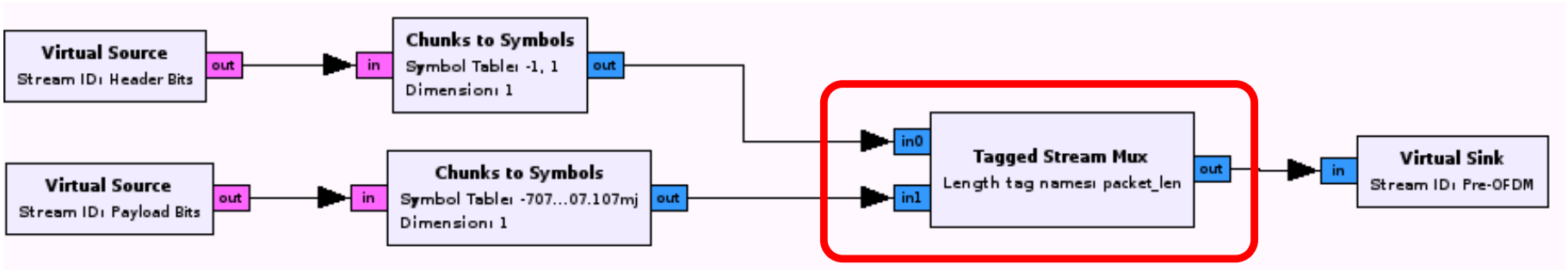
- Packet Header Generator: Packet Header is defined by a packet header generator class
- Bit Repacker: Also tagged stream block

OFDM Tx Implementation Walkthrough



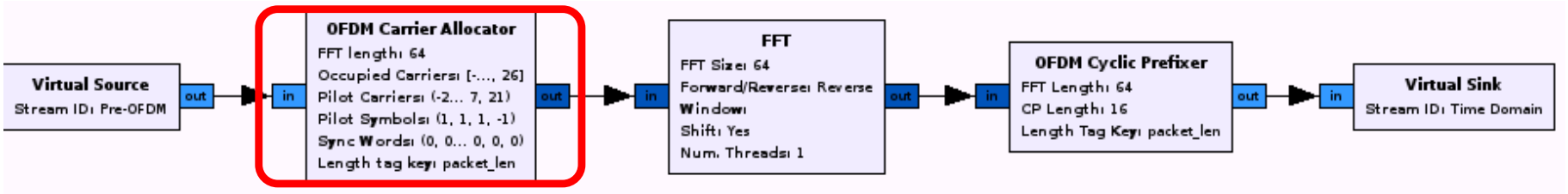
- BPSK / QPSK Modulators: Plain old blocks
- Tagged Stream Mux: Does what it promises

OFDM Tx Implementation Walkthrough



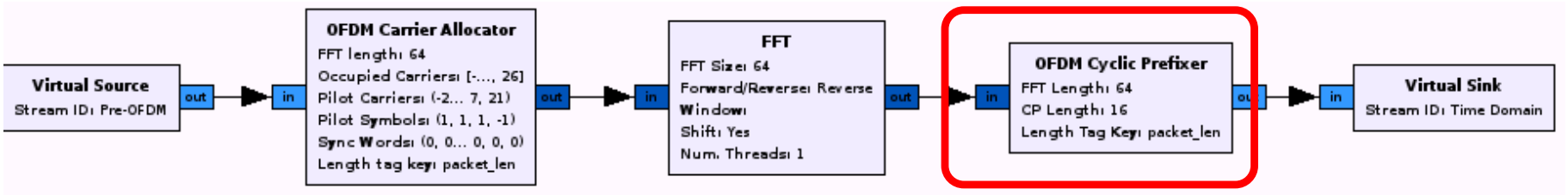
- BPSK / QPSK Modulators: Plain old blocks
- Tagged Stream Mux: Does what it promises

OFDM Tx Implementation Walkthrough



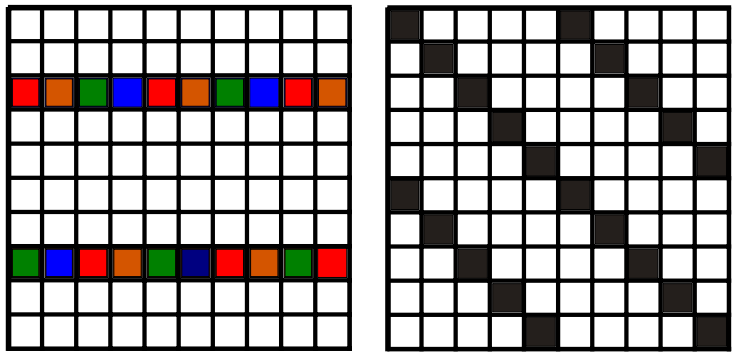
- Carrier Allocator: Distributes data and pilot symbols
- Cyclic Prefixer: Prefixes cyclically, also performs pulse shaping

OFDM Tx Implementation Walkthrough



- Carrier Allocator: Distributes data and pilot symbols
- Cyclic Prefixer: Prefixes cyclically, also performs pulse shaping

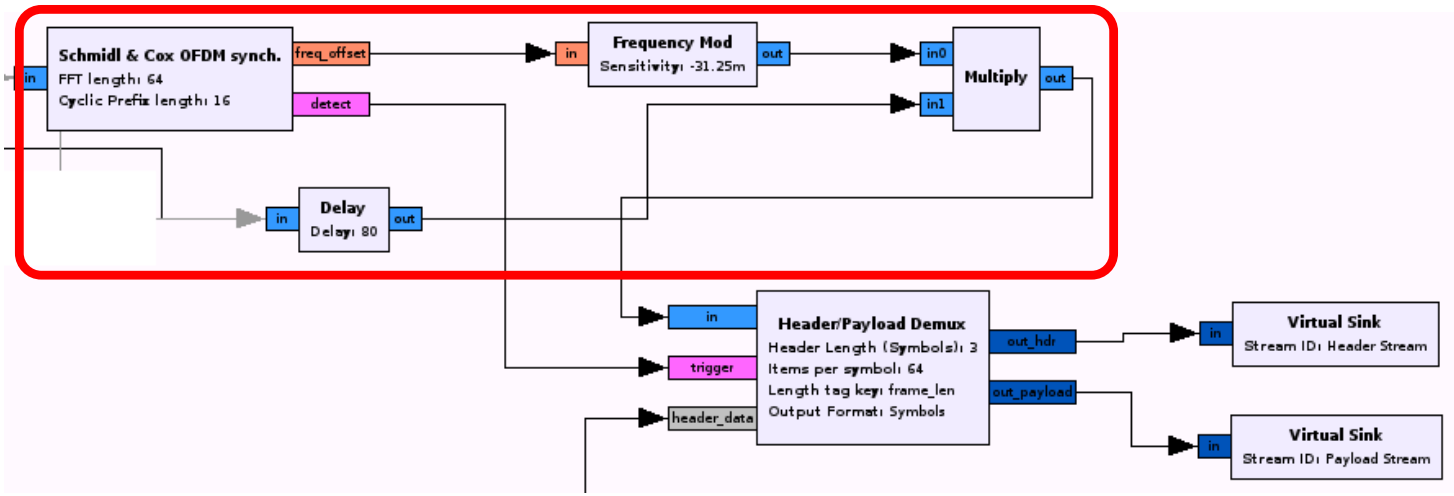
Carrier Mapping



- Every OFDM symbol of a frame can be individually allocated
- Pilot symbols can be mapped anywhere inside the frame
- From the docs:

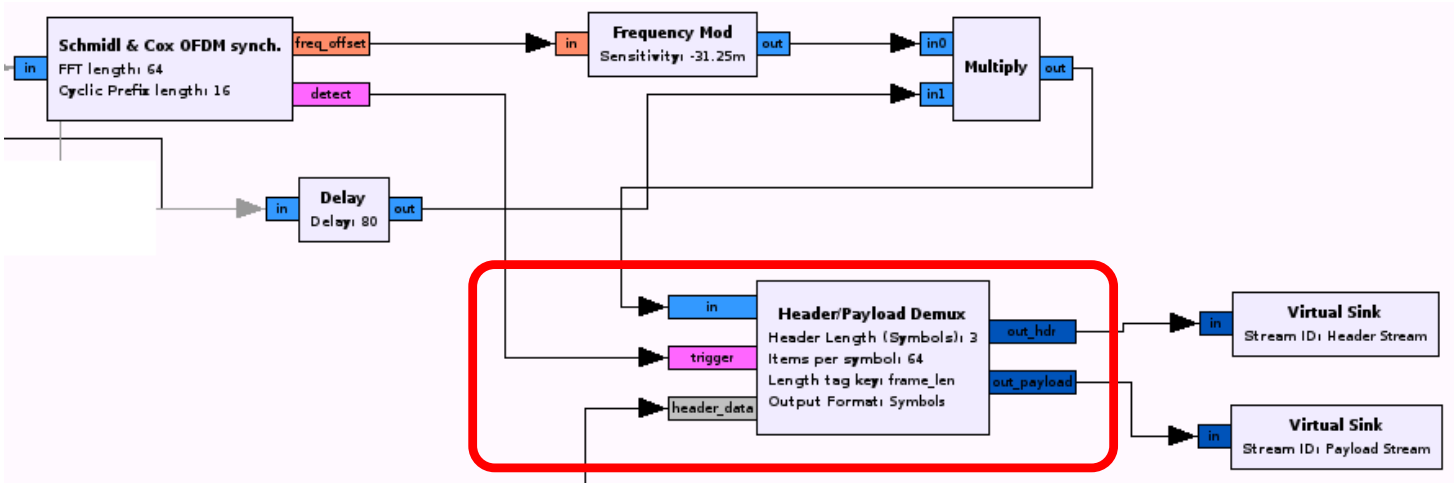
```
occupied_carriers = ((-2, -1, 1, 3), (-3, -1, 1, 2))
pilot_carriers = ((-3, 2), (-2, 3))
pilot_symbols = ((-1, 1j), (1, -1j), (-1, 1j), (-1j, 1))
```

OFDM Rx Implementation Walkthrough



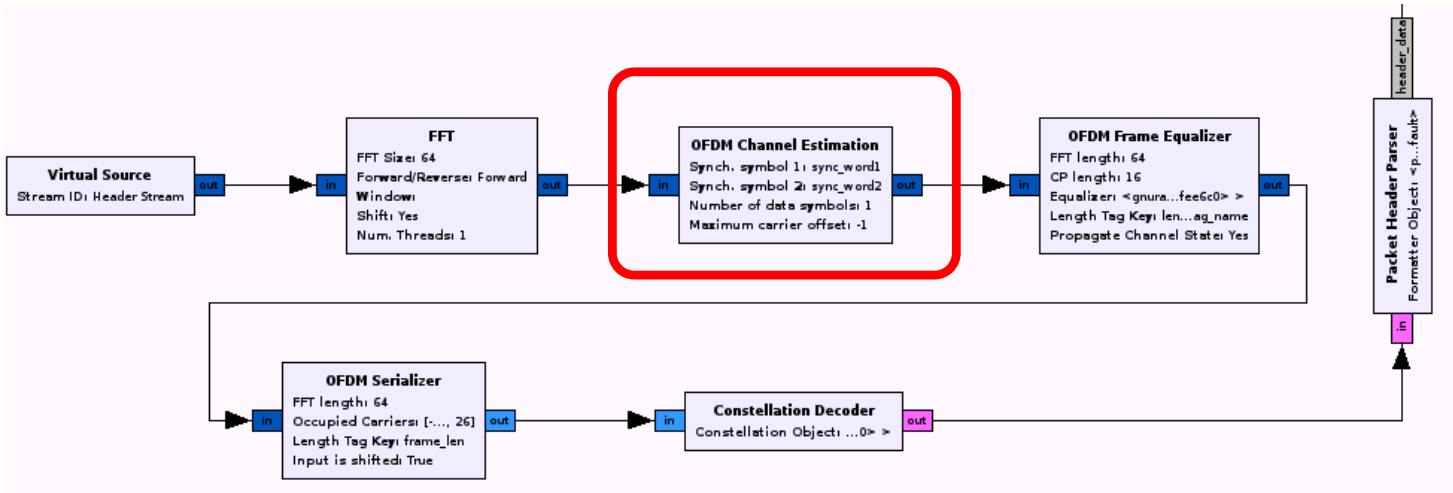
- Synchronization & Detection: Just regular blocks
- Header/Payload Demuxer: Converts infinite streams into tagged streams of correct length

OFDM Rx Implementation Walkthrough



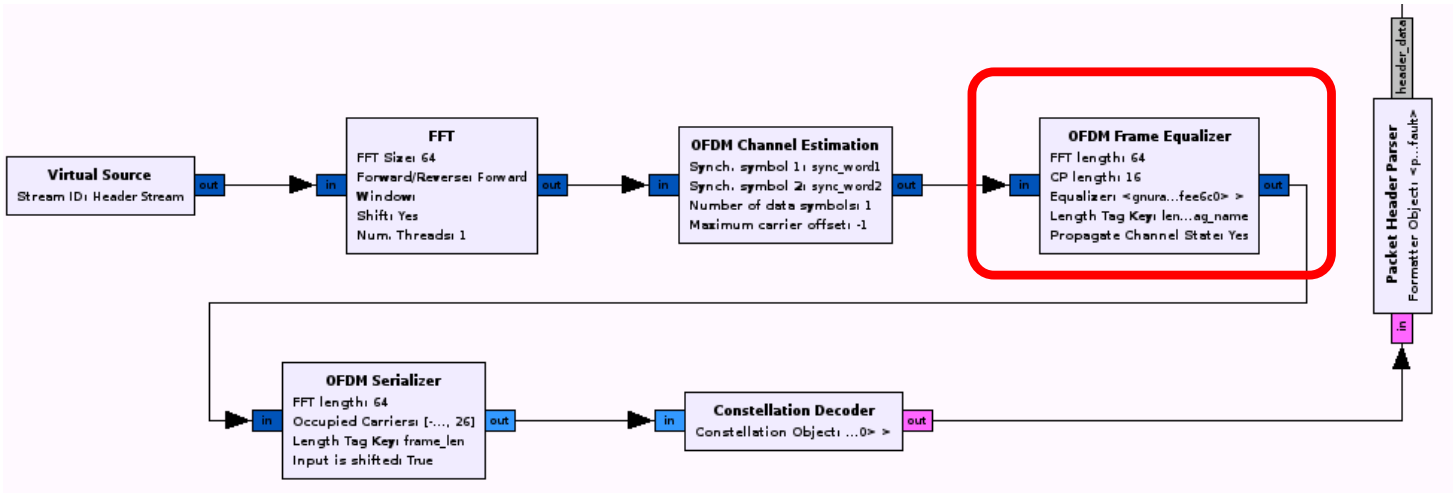
- Synchronization & Detection: Just regular blocks
- Header/Payload Demuxer: Converts infinite streams into tagged streams of correct length

OFDM Rx Implementation Walkthrough



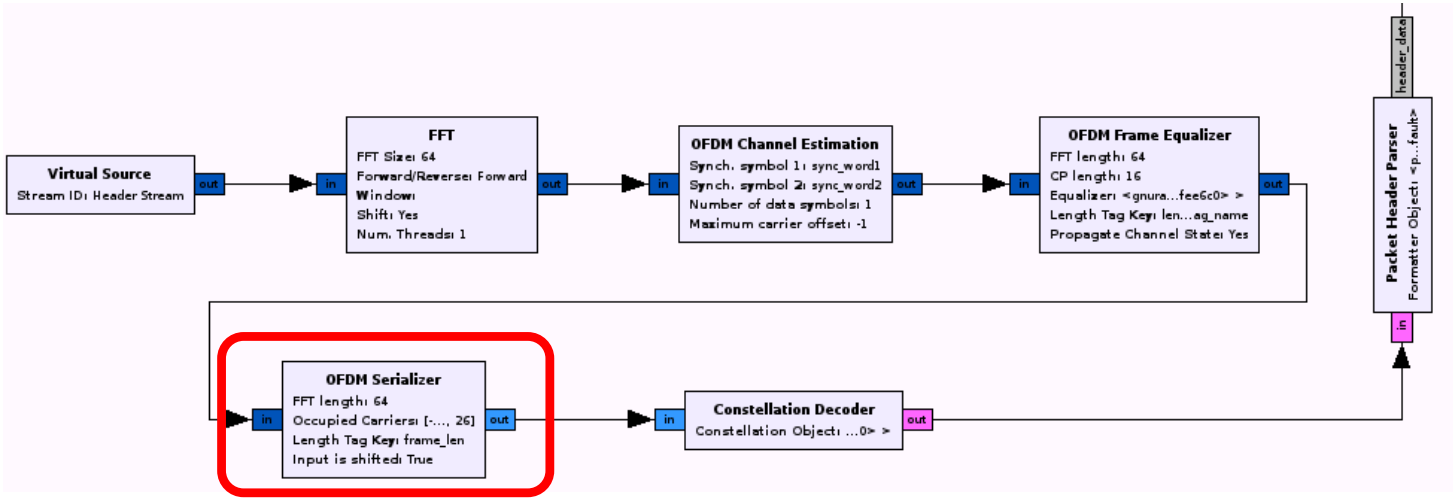
- Channel Estimator: Outputs channel coefficients and coarse frequency offset
- Frame Equalizer: Uses `ofdm_frame_equalizer` objects for actual equalizing
- Serializer: Inverse operation to carrier allocator
- Packet Header Parser: Sends a message to the HPD with details on the packet (feedback loop!)

OFDM Rx Implementation Walkthrough



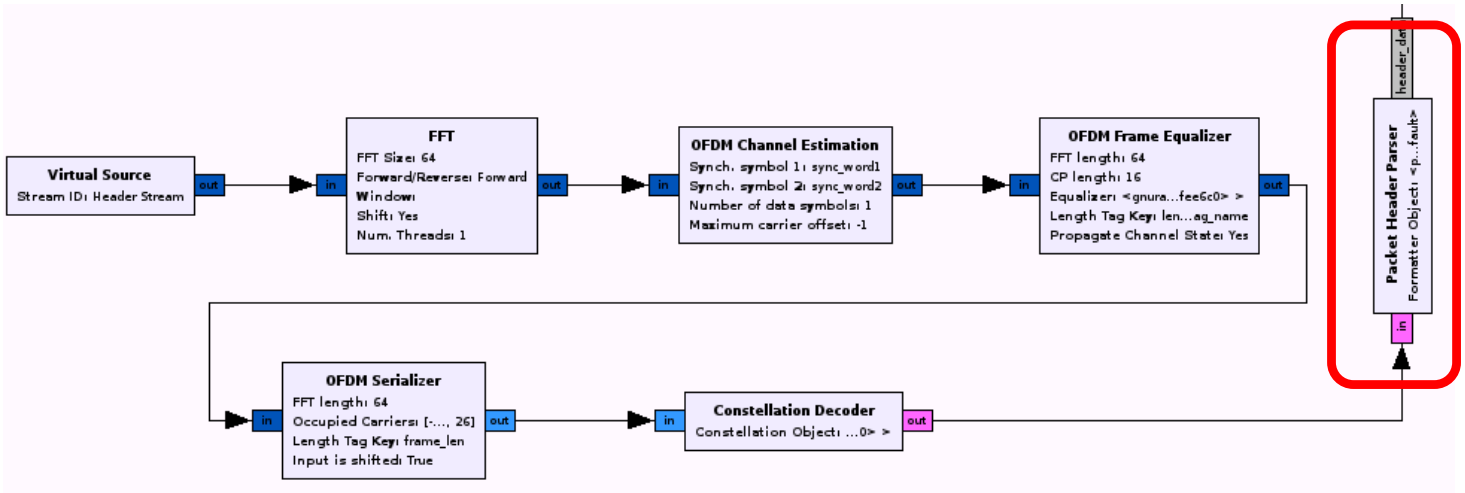
- Channel Estimator: Outputs channel coefficients and coarse frequency offset
- Frame Equalizer: Uses `ofdm_frame_equalizer` objects for actual equalizing
- Serializer: Inverse operation to carrier allocator
- Packet Header Parser: Sends a message to the HPD with details on the packet (feedback loop!)

OFDM Rx Implementation Walkthrough



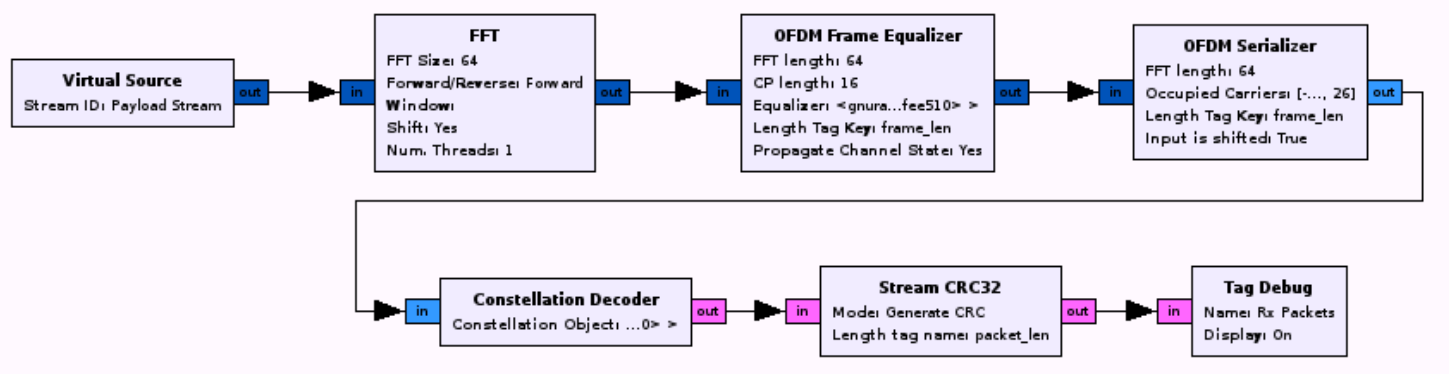
- Channel Estimator: Outputs channel coefficients and coarse frequency offset
- Frame Equalizer: Uses `ofdm_frame_equalizer` objects for actual equalizing
- Serializer: Inverse operation to carrier allocator
- Packet Header Parser: Sends a message to the HPD with details on the packet (feedback loop!)

OFDM Rx Implementation Walkthrough



- Channel Estimator: Outputs channel coefficients and coarse frequency offset
- Frame Equalizer: Uses `ofdm_frame_equalizer` objects for actual equalizing
- Serializer: Inverse operation to carrier allocator
- Packet Header Parser: Sends a message to the HPD with details on the packet (feedback loop!)

OFDM Rx Implementation Walkthrough



- Payload demodulation: Basically same blocks as header demodulation (but different params)
- All info from header demod is passed through tags

benchmark_ofdm.py

- Currently under dev
- Benchmarking / Debugging tool
- Sophisticated debugging blocks (outside the scope of the regular instrumentation blocks)

```
insfl10 examples/ofdm [master●●] % ./benchmark_ofdm.py loopback --noise-voltage=0.11 --payload-modulation=BPSK
linux; GNU C++ version 4.6.3; Boost_104800; UHD_003.005.003-76-ge0cbc2c1

Using Volk machine: avx_64_mmx_orc

OFDM Stats:
===========
Sub-carrier spacing:           3.906 kHz
OFDM symbols per payload:      16
Bits per OFDM symbol duration: 2777.778 kbps
Effective bandwidth:           207.031 kHz

o..ooooo..oo.XoXoo.oo.XoXoo.oo.oXooo.oo.oXoX..oo.oXoooXoo.o..oo.X..ooXooXo..oo.oo.ooXoo.oo.oo.Xo.oo.o..ooXooooooooo.oo.oo.o
o.XoXoo.XoX.o.ooXooXoo.oo.Xo.oo.oo.oooXoXooo.o.oXoXoo.ooXoXoXo.oXooXoooXooXoo.oXooo.XoXoooo..oo.ooXoo.X..oXooXooo.oo.oo.oo
.Xo.oo.oXoo..oo.oo.Xo.o..oo.oXooo.oo.o.oooooo.oXoooXoo.ooooo.oo.oo.Xoooo.XoooXoXo.oXoXo.oo.o.o.oXXoo.Xo.X..oo.X..oo.Xo.o..
o..oo.oXooo.XoXoo.o..oXooo.Xo.o..ooXo..oo.oo.oo.Xo.o..oXoooXooXoo.XoXo..oXooo.X..ooXo..oo.ooXXo.Xo.oo.oXooXooXooo.oo.ooXoo
.ooXoo.oXooX..o.oXooXooo.Xoooo..oXooo.Xo.oXooXo.oXo.oooXoo.oXooX.ooXoo.oXXoo.ooXoo.o..oo.oo.oo.oXooo.oXXoo.Xo.o.ooo.oXooo.
^C
Statistics
================================================
Packets received without error: 507
Packets skipped:                103
Estimated Packet error rate:    16.89%
```

benchmark_ofdm.py

```
74 class PacketCount(gr.basic_block):
75     """ Count incoming packets, assumes packets are numbered in sequence. """
76     def __init__(self, n_packets_total=None, quiet=False, timeout=0):.............................
94     def general_work(self, input_items, output_items):
95         tags_raw = self.get_tags_in_range(
96                 0, # Port 0
97                 self.nitems_read(0),
98                 self.nitems_read(0) + len(input_items[0]),
99                 self.packet_num_tag)
00         tags_crc = self.get_tags_in_range(
01                 1, # Port 1
02                 self.nitems_read(1),
03                 self.nitems_read(1) + len(input_items[1]),
04                 self.packet_num_tag)
05         self.consume(0, len(input_items[0]))
06         self.consume(1, len(input_items[1]))
07         self.headers_rcvd += [pmt.to_python(x.value) for x in tags_raw]
08         self.frames_rcvd += [pmt.to_python(x.value) for x in tags_crc]
09         new_max_frame = self.headers_rcvd[-1]
10         for i in range(self.max_frame_rcvd+1, new_max_frame+1):
11             if i in self.frames_rcvd:
12                 sys.stdout.write('.')
13             elif i in self.headers_rcvd:
```

Closing remarks

- Actually tx'ing OFDM is iffy
- Distortions *will* destroy your signal
- The encoding is not very robust