```
import nengo
model = nengo.Model("Communication Channel")

model.make_ensemble("A", nengo.LIF(30), 1)

model.make_ensemble("B", nengo.LIF(30), 1)

model.make_node("Input", 0.5)
model.connect("Input", "A")

model.connect("A", "B")

model.probe("B", filter=0.01)
sim = model.simulator()
sim.run(1)
Bdata = sim.data("B")
```

```
import numpy as np
import pynn.nest as pyNN
pyNN.setup(timestep=1)
lif_params = {'tau_refrac': 2.0, 'tau_syn_E':100, 'tau_syn_I':100}
def encoders(n_neurons, dimensions):
    samples = np.random.randn(n_neurons, dimensions)
    norm = np.sum(samples * samples, axis=1)
    return np.sqrt(norm)[:, np.newaxis]
def gain_bias(max_rates, intercepts):
    x = 1.0 / (1 - np.exp((lif_params['tau_refrac'] - (1.0 / max_rates))
                          / pyNN.IF_cond_exp.default_parameters['tau_m']))
    gain = (1 - x) / (intercepts - 1.0)
    bias = 1 - self.gain * intercepts
    return gain, bias

A = pyNN.Population(30, pyNN.IF_cond_exp, lif_params)
Again, Abias = gain_bias(np.random.uniform(80, 100, 30), np.random.uniform(-1, 1, 30))
biasinputA = [pyNN.DCSource(amplitude=val) for val in Abias]
for i, pulse in enumerate(biasinputA):
    pulse.inject_into(A[i:i+1])
Aencoders = encoders(30, 1) * Again[:, np.newaxis]

B = pyNN.Population(30, pyNN.IF_cond_exp, lif_params)
Bgain, Bbias = gain_bias(np.random.uniform(80, 100, 30), np.random.uniform(-1, 1, 30))
biasinputB = [pyNN.DCSource(amplitude=val) for val in Bbias]
for i, pulse in enumerate(biasinputB):
    pulse.inject_into(B[i:i+1])
Bencoders = encoders(30, 1) * Bgain[:, np.newaxis]

inputnode = [pyNN.DCSource(amplitude=val) for val in 0.5 * Aencoders]

# Decoder solving too long to include; assume we have Adecoder and Bdecoder
weights = []
for i in xrange(30):
    for j in xrange(30):
        weights.append((i, j, np.dot(Adecoder[i], Bencoder[j]), 1.0))
connection = pyNN.Projection(A, B, pyNN.FromListConnector(weights))

B.record('spikes')

pyNN.run(1000)

Bdata = numpy.zeros(1000)
for i in xrange(1000):
    Bspikes = B[i:i+1].getSpikes()[:,1].astype('int')
    Bdata[B_spikes] += Bdecoder[i]
decay = np.exp(-1.0 / 100)
Bdata[0, :] *= (1 - decay)
for i in xrange(1, 1000):
    Bdata[i,:] = decay * Bdata[i-1,:] + (1-decay) * Bdata[i,:]
```