Using Kodo (Network Coding) to provide Enhanced Delivered Video QoS over SDNs

[topology]

H1----H2----H3----H4
Set the packet loss rate between H2 and H3 to 10%
H1:video sender
H4:video receiver
H2:video encoder
H3:video decoder

Note. Please install ffmpeg first.
You can download the related codes:
http://csie.nqu.edu.tw/smallko/sdn/test_kodo_ffmpeg.zip

[mininet-script]

```
#!/usr/bin/env python
from mininet.net import Mininet
from mininet.cli import CLI
from mininet.link import Link,TCLink,Intf

if '__main__' == __name__:
    net = Mininet(link=TCLink)
    h1 = net.addHost('h1', ip="10.0.0.2/24")
    h2 = net.addHost('h2')
    h3 = net.addHost('h3')
    h4 = net.addHost('h4', ip="10.0.2.2/24")
    linkopts0={'bw':10, 'loss':0}
    linkopts1={'bw':10, 'loss':5}
    net.addLink(h1, h2, cls=TCLink, **linkopts0)
    net.addLink(h2, h3, cls=TCLink, **linkopts1)
    net.addLink(h3, h4, cls=TCLink, **linkopts0)
    net.build()
    h2.cmd("echo 1 > /proc/sys/net/ipv4/ip_forward")
    h2.cmd("ifconfig h2-eth0 0")
    h2.cmd("ifconfig h2-eth1 0")
    h3.cmd("echo 1 > /proc/sys/net/ipv4/ip_forward")
    h3.cmd("ifconfig h3-eth0 0")
```

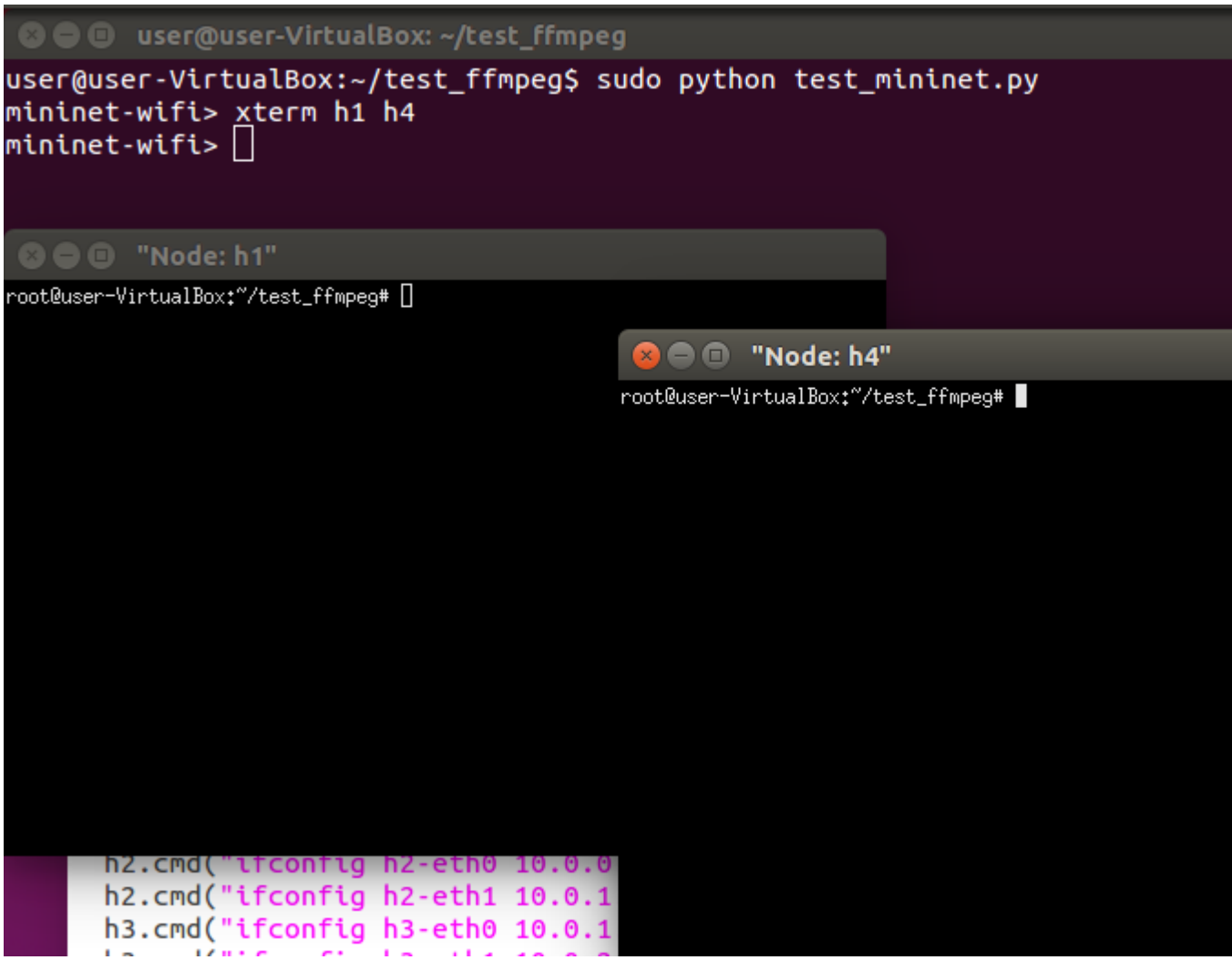
```
    h3.cmd("ifconfig h3-eth1 0")
    h2.cmd("ifconfig h2-eth0 10.0.0.1 netmask 255.255.255.0")
    h2.cmd("ifconfig h2-eth1 10.0.1.1 netmask 255.255.255.0")
    h3.cmd("ifconfig h3-eth0 10.0.1.2 netmask 255.255.255.0")
    h3.cmd("ifconfig h3-eth1 10.0.2.1 netmask 255.255.255.0")
    h1.cmd("ip route add default via 10.0.0.1")
    h4.cmd("ip route add default via 10.0.2.1")
    h2.cmd("ip route add 10.0.2.0/24 via 10.0.1.2")
    h3.cmd("ip route add 10.0.0.0/24 via 10.0.1.1")
    CLI(net)
    net.stop()
```

[Experiment 1 –no network coding]

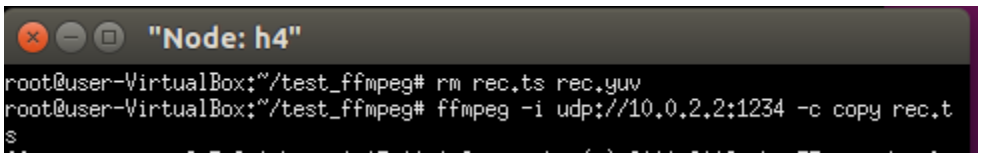
Encode the video

```
user@user-VirtualBox:~/test_ffmpeg$ ffmpeg -f rawvideo -s:v 352x288 -i foreman_c
if.yuv -c:v libx264 -qscale 10 -g 10 -bf 2 -r 30 -f mpegts output.ts
```

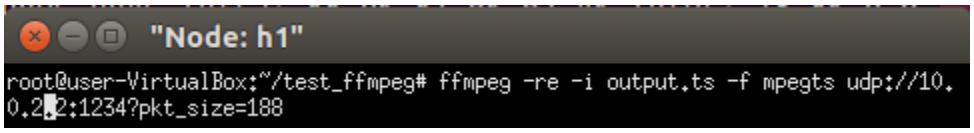
Execute the evaluation.



H4: receiver



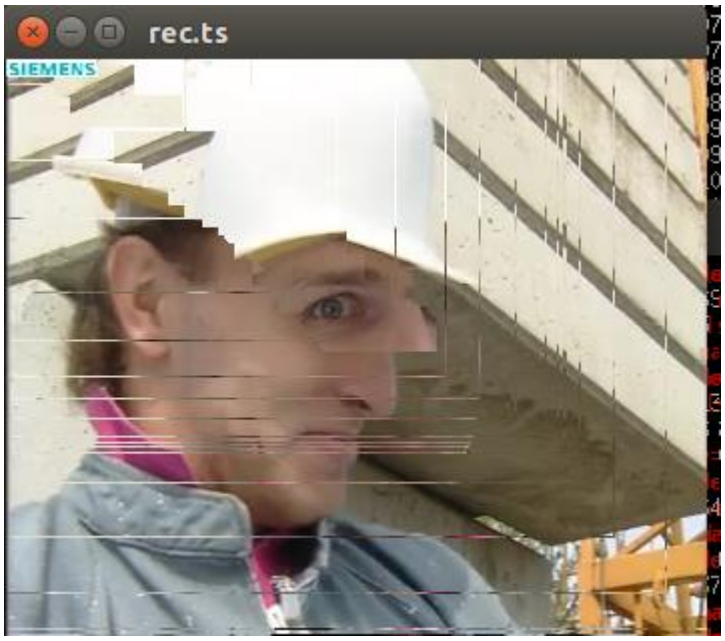
H1: sender



After transmission.

At H4:

See the visual effects.



Compare the PSNR.

```
root@ubuntu14:~/test_ffmpeg# ffmpeg -i rec.ts rec.yuv
```

```
root@user-VirtualBox:~/test_ffmpeg# ./avgpsnr 352 288 420 foreman_cif.yuv rec.y
uv
avgerage psnr:15.166970
root@user-VirtualBox:~/test_ffmpeg#
```

[Experiment 2: with network coding]

[replay_encode.py]

```
import kodo
import os
import socket
import sys
import time
import struct

UDP_IP="10.0.0.1"
UDP_PORT=1234

sock = socket.socket(socket.AF_INET, # Internet
                     socket.SOCK_DGRAM) # UDP
sock.bind((UDP_IP, UDP_PORT))
```

```
sock_replay = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_address = ('10.0.1.2', 1234)

symbols = 10
symbol_size = 188

encoder_factory = kodo.OnTheFlyEncoderFactoryBinary(
            max_symbols=symbols,
            max_symbol_size=symbol_size)
encoder = encoder_factory.build()
#encoder.set_systematic_off()
print "encoder.block_size()=", encoder.block_size()

j=0 #generation number
while True:
        databuf=''
        for i in range(0,10):
           data, addr = sock.recvfrom(188)
           #print "received message:", data
           print "addr:", addr, "size:", len(data)
           databuf+=data
           print "len(databuf)=", len(databuf)

        for i in range(0,15):
           rank=encoder.rank()
           encoder.set_const_symbols(databuf)
           packet = encoder.write_payload()
           myhdr=struct.pack("!HHH", len(databuf), j, len(packet))
           nwpkt=myhdr+packet
           #data=databuf[i*188:(i+1)*188]
           print "len(nwpkt)=", len(nwpkt)
           sent = sock_replay.sendto(nwpkt, server_address)
           #print "sent:", sent

        encoder_factory =
kodo.FullVectorEncoderFactoryBinary(max_symbols=symbols,max_symbol_size=sy
mbol_size)
        encoder = encoder_factory.build()
```

```
        #encoder.set_systematic_off()
        j+=1
        print "new encoder is built for generation:", j
```

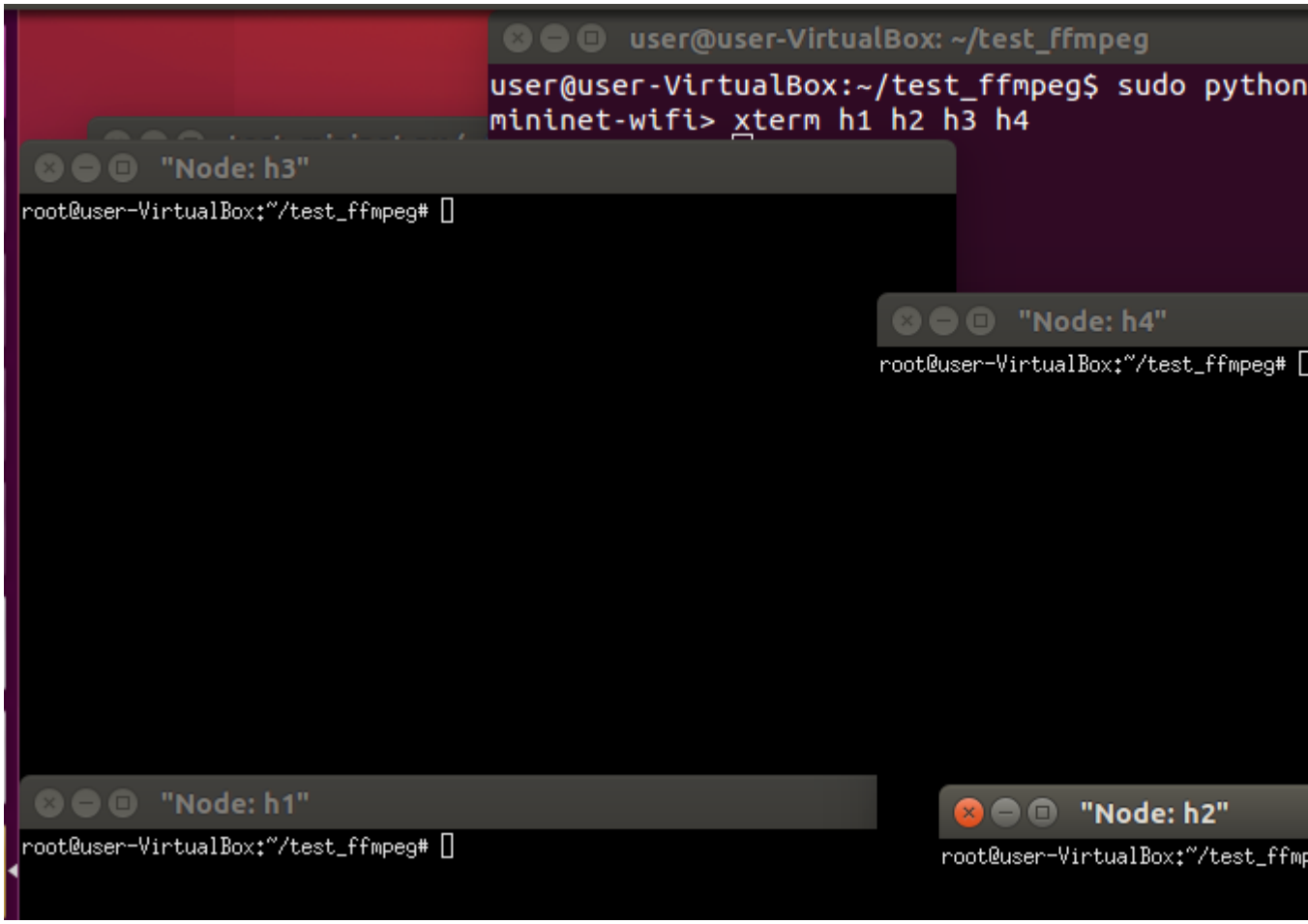
[replay-decode.py]

```
import kodo
import os
import socket
import sys
import time
import struct

UDP_IP="10.0.1.2"
UDP_PORT=1234

sock = socket.socket(socket.AF_INET, # Internet
                                    socket.SOCK_DGRAM) # UDP
sock.bind((UDP_IP, UDP_PORT))

sock_replay = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_address = ('10.0.2.2', 1234)

symbols = 10
symbol_size = 188

decoder_factory = kodo.OnTheFlyDecoderFactoryBinary(
            max_symbols=symbols,
            max_symbol_size=symbol_size)
decoder = decoder_factory.build()

old_gen=0
decoded_gen=-1
decoding_gen=-1
is_decoded=0    #0: not decoded 1:decoded
partially_decoded=''

while True:
```

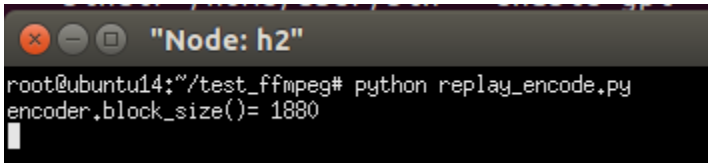
```
        packet = sock.recv(199)
        gensize, mygen, pktsize=struct.unpack("!HHH",packet[0:6])
        print "gensize=", gensize, "mygen=", mygen, "packetsize=", pktsize

        if decoded_gen == mygen:
            continue

        if decoding_gen != mygen:
            print "New Generation Packet is coming", "is_decoded=", is_decoded,
"decoded_gen=", decoded_gen, "decoding_gen=", decoding_gen
            if decoded_gen != decoding_gen and is_decoded==0:
                print "decoding_gen:", decoding_gen, "is not decoded completely"
                #not tested
                i=0
                while(((i+1)*188)<=len(partially_decoded)):
                    data=partially_decoded[i*188:(i+1)*188]
                    sent = sock_replay.sendto(data, server_address)
                    i+=1
                print "================================================="
            is_decoded=0
            partially_decoded=''

        decoding_gen=mygen

        if pktsize==199 and is_decoded==0:
                partially_decoded+=packet[11:]

        if mygen!=old_gen:
            decoder_factory =
kodo.FullVectorDecoderFactoryBinary(max_symbols=symbols,max_symbol_size=sy
mbol_size)
            decoder = decoder_factory.build()
            print "new decoder is built"
            old_gen=mygen

        nwpkt=packet[6:]
        decoder.read_payload(nwpkt)
```

```
        if decoder.is_complete():
            print "decoder is completed:"
            is_decoded=1
            decoded_gen=mygen
            if gensize==1880:
                decoded_data=decoder.copy_from_symbols()
                for i in range(0,10):
                    data=decoded_data[i*188:(i+1)*188]
                    print "len(data)=", len(data)
                    sent = sock_replay.sendto(data, server_address)
            if gensize<1880:
                #this part: not tested
                decoded_data=decoder.copy_from_symbols()[:gensize]
                i=0
                while(((i+1)*188)<=len(decoded_data)):
                    data=decoded_data[i*188:(i+1)*188]
                    sent = sock_replay.sendto(data, server_address)
                    i+=1
                print "================================================="
                break
        else:
            is_decoded_gen=0
```

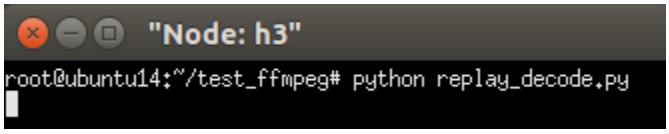
[Execution: with network encoding]



H2: network encoder



H3: network decoder



H4: video receiver

H1: video sender

After transmission.

```
root@user-VirtualBox:~/test_ffmpeg# ffmpeg -i rec.ts rec.yuv
```

```
root@user-VirtualBox:~/test_ffmpeg# ./avgpsnr 352 288 420 foreman_cif.yuv rec.y
uv
avgerage psnr:16.022913
```

Reference

http://docs.steinwurf.com/kodo/index.html

Dr. Chih-Heng Ke (smallko@gmail.com)
Department of Computer Science and Information Engineering,
National Quemoy University, Kinmen, Taiwan.