**NDN-RTC搭建与调试**

1. 文档说明

本文档描述基于NDN的RTC搭建与调试过程，用于工程师及研究员研究与学习。

本次搭建视频直播环境使用ubuntu20.04 LTS虚拟机环境.

系统架构及原理，可以参见论文《NDN-RTC Real-Time Videoconferencing over Named Data Networking》

1. 搭建过程：

在home目录下生成一个NDNRTC目录，用于统一存放所有项目所需的包和组件。

1. 安装depot\_tools：

git clone https://chromium.googlesource.com/chromium/tools/depot\_tools.git

export PATH=$PATH:`pwd`/depot\_tools

2. 编译安装webrtc：

mkdir webrtc-checkout && cd webrtc-checkout/

fetch --nohooks webrtc

cd src

git checkout -b branch-heads-59 refs/remotes/branch-heads/59

gclient sync

修改/build/install-build-deps.sh的第114行，增加对ubuntu20.04 LTS focal的支持：

supported\_releases="(trusty|xenial|yakkety|jessie|focal)"

./build/install-build-deps.sh

gn gen out/Default --args='is\_debug=false'

ninja -C out/Default

3. 编译安装NDN-CPP：

git clone <https://github.com/named-data/ndn-cpp>

cd ndn-cpp && mkdir -p build/share

./configure --with-std-shared-ptr=no --with-std-function=no --prefix=$(pwd)/build

make && make install

4. 编译安装OpenFEC：

wget http://openfec.org/files/openfec\_v1\_4\_2.tgz

tar -xvf openfec\_v1\_4\_2.tgz && rm openfec\_v1\_4\_2.tgz

mkdir -p openfec\_v1.4.2/build && cd openfec\_v1.4.2/

针对src/CMakeLists.txt，作如下修改：

修改add\_library(openfec SHARED ${openfec\_sources}) 为add\_library(openfec STATIC ${openfec\_sources})

修改target\_link\_libraries(openfec pthread IL) 为target\_link\_libraries(openfec pthread)

增加set(CMAKE\_C\_FLAGS "-fPIC")

或者

wget https://raw.githubusercontent.com/remap/ndnrtc/master/cpp/resources/ndnrtc-openfec.patch && patch src/CMakeLists.txt ndnrtc-openfec.patch

cd build/

cmake .. -DDEBUG:STRING=OFF

make

5. 编译安装NDN-RTC：

git clone --recursive https://github.com/remap/ndnrtc

cd ndnrtc/cpp && mkdir -p build/share

echo 'CPPFLAGS="-g -O2 -DWEBRTC\_POSIX" CXXFLAGS="-g -O2 -DWEBRTC\_POSIX"' > build/share/config.site

echo NDNCPPDIR=`pwd`/../../ndn-cpp/build/include >> build/share/config.site

echo NDNCPPLIB=`pwd`/../../ndn-cpp/build/lib >> build/share/config.site

echo OPENFECDIR=`pwd`/../../openfec\_v1.4.2 >> build/share/config.site

echo WEBRTCDIR=`pwd`/../../webrtc-checkout/src >> build/share/config.site

./configure --prefix=$(pwd)/build

\*)修改Makefile，在2997行增加 LIBS = -lX11

\*)修改tools/stream-scrubber/renderer.cpp，在顶部增加#include <string.h>

\*)修改cpp/src/network-data.hpp，由于异步操作导致操作同一个packet，注释掉setHeader()函数中如下代码：

// throw std::runtime\_error("Sample header has been already "

// "added to the packet");

同时修改cpp/src/audio-stream-impl.cpp，在onSampleBundle()函数中增加如下代码：

…

boost::lock\_guard<boost::mutex> scopedLock(me->internalMutex\_);

bundle->clear();

me->bundlePool\_.push\_back(bundle);

…

\*)修改cpp/src/remote-stream-impl.cpp，在construct ()中增加如下代码：主要在于语音没有估算值，原代码中只针对视频做了估算，否则会导致Consumer端语音interest发不出来，保证至少发出1个interest（该值由cpp/tests/sample-producer.cfg配置文件中定义的segment\_size /NDN\_packet\_size决定）。

…

sampleEstimator\_->bootstrapSegmentNumber(1, SampleClass::Delta, SegmentClass::Data);

sampleEstimator\_->bootstrapSegmentNumber(0, SampleClass::Delta, SegmentClass::Parity);

sampleEstimator\_->bootstrapSegmentNumber(0, SampleClass::Key, SegmentClass::Data);

sampleEstimator\_->bootstrapSegmentNumber(0, SampleClass::Key, SegmentClass::Parity);

…

\*)修改cpp/src/pipeliner.cpp，将getBatch()函数的const修饰符删除，同时修改fillUpPipeline ()函数如下，语音没有Parity类型的interest：

…

while (interestControl\_->room() > 0)

{

Name n = nameScheme\_->samplePrefix(threadPrefix, nextSamplePriority\_); n.appendSequenceNumber((nextSamplePriority\_ == SampleClass::Delta ?

seqCounter\_.delta\_ : seqCounter\_.key\_));

Name m("audio");

int result = n.compare(4, 1, m, 0);

std::vector<boost::shared\_ptr<const Interest>> batch；

if (!result)

batch = getBatch(n, nextSamplePriority, true);

else

batch = getBatch(n, nextSamplePriority\_);

int64\_t deadline = playbackQueue\_->size()+playbackQueue\_->pendingSize();

…

bool full = buffer\_->requested(batch);

if (!full)

{

interestControl\_->decrement();

break;

}

…

nextSamplePriority\_ = SampleClass::Delta;

}

\*)修改cpp/src/interest-queue.cpp，将enqueueInterest()函数修改如下：

if (queue\_.size() > 0) //直接发出，不需要等待10个。

\*)修改cpp/src/audio-stream-impl.cpp，，将AudioStreamImpl()函数修改如下：

ps.segmentWireLength\_ = settings\_.params\_.producerParams\_.segmentSize\_;

\*)修改cpp/src/pipeline-control-state-machine.cpp，将bool processMetadata(boost:: shared\_ptr<AudioThreadMeta>metadata,boost::shared\_ptr <PipelineControlStateMachi ne::Struct> ctrl)函数修改如下：

…

unsigned int pipelineInitial = ctrl->interestControl\_->getCurrentStrategy()-> calculateDemand(metadata->getRate(),initialDrd, initialDrd \* 0.05);

ctrl->sampleEstimator\_->bootstrapSegmentNumber(1, SampleClass::Delta, SegmentClass::Data); //保证发出最少1个interest，这里每个语音sample/bundle发送的interest个数要等于packet-publisher.cpp中publish()函数打印出来的sliced into segments的个数，目前为1（该值由cpp/tests/sample-producer.cfg配置文件中定义的segment\_size/NDN\_packet\_size决定）。

ctrl->interestControl\_->initialize(metadata->getRate(), pipelineInitial<100?100: pipelineInitial);

…

\*)修改cpp/src/segment-controller.cpp，将onData()、onTimeout()、onNetworkNack()函数中的定时开关在数据处理前关掉，处理后打开，修改比较多，可参照github版本。

\*)修改cpp/src/pipeline-control-state-machine.cpp，在BootstrappingT、Fetching、Adjusting类中增加onStart()成员函数，用于超时处理，修改比较多，可参照github版本。

\*)修改cpp/src/audio-playout-impl.cpp，将stop()函数中下列语句注释掉：

//renderer\_->stopRendering();

\*)修改cpp/src/media-stream-base.cpp，在函数MediaStreamBase()将Cache中的Data包保留时间延长到10000ms。

cache\_->setMinimumCacheLifetime(10000);

\*)修改cpp/src/interest-control.cpp、interest-control.hpp、remote-stream-impl.cpp、remote-stream-impl.hpp、remote-audio-stream.cpp、remote-video-stream.cpp，增加对象InterestControl的MediaStreamParams::MediaStreamType类型控制，用于语音interest发送控制，修改比较多，可以参考githug版本。

\*)修改cpp/src/playout-impl.cpp，在extractSample()函数中增加如下代码：

…

actualDelay = actualDelay<0 ? 0 : actualDelay;

jitterTiming\_.updatePlayoutTime(actualDelay);

…

\*)修改cpp/src/audio-capturer.cpp、audio-renderer.cpp和webrtc-audio-channel.cpp，主要是设置录制和播入语音设备，以及增加解码FEC功能，修改比较多，可以参考githug版本。

修改完后编译安装：

make && make install

6. 安装PyNDN2

sudo apt-get install build-essential libssl-dev libffi-dev python3-dev python3-pip

sudo apt-get install python-setuptools python-dev python3-setuptools

pip install -U pytest

pip3 install -U pytest

pip install -U pytest-html

pip3 install -U pytest-html

pip install -U pytest-rerunfailures

pip3 install -U pytest-rerunfailures

下载PyNDN2-master到home目录下并解压。

cd /home/PyNDN2-master

sudo python3 setup.py install

export PYTHONPATH=$PYTHONPATH:$(pwd)/python

make -C doc html

./configure

make && make install

cd examples

python3 test\_encode\_decode\_data.py

python3 test\_get\_async.py

7. 编译无头APP：Headless client ，用于测试NDN-RTC功能：



mkdir /usr/local/include/ndnrtc

cd home/NDNRTC/ndnrtc/cpp/include

cp \* /usr/local/include/ndnrtc/ -r

将cpp\client\src目录下precise-generator.cpp文件中的如下代码

timer\_.async\_wait(lib\_bind::bind(&PreciseGeneratorImpl::onFire, shared\_from\_this(), boost::arg<1>()));

修改为：

timer\_.async\_wait(boost::bind(&PreciseGeneratorImpl::onFire, shared\_from\_this(), std::placeholders::\_1));

修改Makefile，在170行去掉-lXdamage和- lXcomposite，修改成am\_\_append\_15 = -ldl -lX11 -lXrender -lXext -lnss3 -lssl3 -lXfixes

或安装libxcomposite-dev和libxdamage-dev包

修改cpp/client/src目录下config.cpp文件中的loadFreshnessSettings()函数如下：

int loadFreshnessSettings(XXX)

{

const Setting &freshnessSettings = s[PRODUCER\_FRESHNESS\_KEY];

freshnessSettings.lookupValue("metadata", params.metadataMs\_);

freshnessSettings.lookupValue("sample", params.sampleMs\_);

freshnessSettings.lookupValue("sampleKey", params.sampleKeyMs\_);

return EXIT\_SUCCESS;

}

其它地方类似如下修改，主要在于某些配置不存在时，不能直接取值，否则会抛出异常，可以直接使用github上的config.cpp版本即可。

if (s.exists("sink"))

{

const Setting &sinkSettings = s["sink"];

执行如下操作，重新编译：

make ndnrtc-client

8. 运行无头APP：Headless client ，测试NDN-RTC功能：

使用上节生成的 ndnrtc-client进行快速Loopback test测试NDN-RTC，按下述步骤生成两个实例（生产者producer和消费者consumer），这两个实例运行在同一台机器上的不同终端窗口中，用于模拟双方进行RTC通信。

ndnrtc-client的参数介绍：

-c（配置文件）——描述应用程序配置的配置文件；

-s（签名标识）——NDN签名标识，用于创建应用程序证书和签名数据包；类似于/ndn/edu/ucla/remap/peter或在系统中安装的任何ndn标识（此参数是必需的）；

-p（验证策略文件）——用于验证传入数据包的验证策略文件（此参数是必需的）；

-t（应用程序运行时间）——应用程序运行时间（秒）；

-i（应用程序实例名称）——应用程序实例名称，该名称将附加到提供的签名标识中，以生成应用程序证书；

-n（统计采样间隔）——统计采样周期（毫秒）（可选，默认为100ms）；

-v（详细模式）——std:：out的详细输出（不适用于配置文件中指定的日志文件）。

8.1执行下面的命令生成并安装证书，两个命令执行一个即可，取决于NDN-cpp版本，最新版运行第一个即可。

ndnsec key-gen /ndnrtc-loopback | ndnsec cert-install -

或

ndnsec-keygen /ndnrtc-loopback | ndnsec-install-cert -

8.2执行下面的命令生成并安装证书，两个命令执行一个即可，取决于NDN-cpp版本，最新版运行第一个即可。

ndnsec cert-dump -i /ndnrtc-loopback > tests/policy\_config/signing.cert

或

ndnsec-dump-certificate -i /ndnrtc-loopback > tests/policy\_config/signing.cert

8.3 mkdir loopback

8.4 nfd-start &> /tmp/nfd.log

8.5使用producer配置（tests/loopback-producer.cfg）启动ndnrtc-client模拟数据生产者，并运行35秒。

./ndnrtc-client -c tests/loopback-producer.cfg -s /ndnrtc-loopback -p tests/policy\_config/rule.conf -i producer -t 35

8.6在另一个终端中，使用consumer配置（tests/loopback-consumer.cfg）启动ndnrtc-client模拟数据消费者，用于获取producer的数据，并运行30秒。

./ndnrtc-client -c tests/loopback-consumer.cfg -s /ndnrtc-loopback -p tests/policy\_config/rule.conf -i consumer -t 30

8.7 cd loopback

8.8 ffmpeg -f rawvideo -vcodec rawvideo -s 320x240 -r 25 -pix\_fmt argb -i producer-camera.320x240 -c:v libx264 -preset ultrafast -qp 0 producer-camera.mp4

8.9 ffplay producer-camera.mp4此时可以播放视频了。

**以下步骤用于真实摄像头集成测试：**

下载https://github.com/umlaeute/v4l2loopback.zip。解压后安装：

make && sudo make install

sudo depmod -a

modprobe v4l2loopback



9. 下载libyuv库，修改linux.mk，在编译选项中增加对宏HAVE\_JPEG的支持

CFLAGS+=-Iinclude/ -DHAVE\_JPEG -ljpeg

CXXFLAGS+=-Iinclude/ -DHAVE\_JPEG -ljpeg

修改下面语句去掉对psnr的支持。

all: libyuv.a i444tonv12\_eg yuvconvert yuvconstants cupid

make -f linux.mk

10. 编写服务器端和客户端代码。

10.1编写server.c服务器端代码，用于producer从摄像头读取图像数据并存入管道PIPE，参见github上的代码。将之放入libyuv-master/util中。

修改linux.mk，增加对server.c的编译支持

server: util/server.c libyuv.a

$(CXX) $(CXXFLAGS) -Iutil/ -o $@ util/server.c libyuv.a -ljpeg

10.2编写client.c服务器端代码，用于consumer读取图像数据并存入播放器设备，参见github上的代码。将之放入libyuv-master/util中。

修改linux.mk，增加对client.c的编译支持

client: util/client.c libyuv.a

$(CXX) $(CXXFLAGS) -Iutil/ -o $@ util/client.c libyuv.a -ljpeg

修改linux.mk，增加编译目标

all: libyuv.a i444tonv12\_eg yuvconvert yuvconstants cpuid server client

执行下面操作重新编译

make -f linux.mk

11.将cpp/client/src/config.hpp中的第82行，修改默认为pipe读取数据，见下：

…ProducerStreamParams() : source\_({"", "pipe"}) {}…

将344行修改为默认的pipe：

…params.sink\_.type\_ = "pipe";…

修改生产者配置文件的source type为"pipe"，并填上source为PIPE管道文件（样例可参见cpp/tests/loopback-producer.cfg，其中source = "/home/producerFIFO";），这里source需要与上面第9步server.c中的PIPE为同一个管道文件，主要用于一个写，一个读。

参见cpp/tests/loopback-consumer.cfg，修改消费者配置文件的sink type为"pipe"，并填上source为PIPE管道文件（样例可参见cpp/tests/loopback- consumer.cfg, 其中source = "/home/ customerFIFO";）。将cpp/client/src/renderer.cpp中的第86行，删除分辨率作为PIPE文件后缀，修改见下：

//sinkPath << sinkName\_ << "." << width << "x" << height;

sinkPath << sinkName\_;

执行如下操作重新编译ndnrtc-client:

make ndnrtc-client

12. 插上罗技C270 摄像头，在linux下可以看到/dev/video1，我们将此设备作为直播摄像头。

开启4个终端。

第1个终端进入/home/NDNRTC/libyuv-master目录下，运行./server  
 第2个终端进入home/NDNRTC/ndnrtc/cpp目录下，运行  
 ./ndnrtc-client -c tests/loopback-producer.cfg -s /ndnrtc-loopback -p tests/policy\_config/rule.conf -i producer -t 35  
 第3个终端进入home/NDNRTC/ndnrtc/cpp目录下，运行  
 ./ndnrtc-client -c tests/loopback-consumer.cfg -s /ndnrtc-loopback -p tests/policy\_config/rule.conf -i consumer -t 30  
 第4个终端进入home/NDNRTC/ndnrtc/cpp目录下，运行./client  
 为了观看实时流，打开VLC播放器 (open vlc -> media -> open capture device -> video device name -> (select correct video device file (/dev/video0)，audio device name: hw:1,0)。

此时可以看到直播视频 (注意选择上面的音频设备audio device name: hw:1,0，若在同一台机器上运行producer和consumer，如果选择不当可能会与producer的音频获取设备冲突)。

**以下步骤用于producer和consumer在不同的机器上运行：**

13. 在两台机器上分别运行producer和consumer，即一台机器做producer，另一台机器做consumer：

1) 两台机器连通后，相互增加face接口和route路由

nfdc face create udp://<other-host>

nfdc route add /ndnrtc-test udp://<other-host>

将两台机器上的home/NDNRTC/ndnrtc/cpp/tests/policy\_config/rule.conf修改如下，去掉包认证validator，否则要保证双方都拥有对方的公钥，且都有正确的公钥私钥对(原作者说法不严谨，容易掉坑)：

….

trust-anchor

{

type any

}

…

2) 在producer机器上插上摄像头，运行如下命令：

apt install pavucontrol

pulseaudio --start --log-target=syslog

sudo depmod -a

modprobe v4l2loopback

ndnsec key-gen /ndnrtc-test | ndnsec-install-cert -

ndnsec cert-dump -i /ndnrtc-test > tests/policy\_config/signing.cert

将tests/policy\_config/signing.cert复制到consumer机器上的tests/policy\_config/signing.cert

修改tests/sample-producer.cfg中的source = "/home/producerFIFO"，type= "pipe";

nfd-start &> /tmp/nfd.log

打开终端，在home/NDNRTC/ndnrtc/cpp目录下运行./ndnrtc-client -c tests/sample-producer.cfg -s /ndnrtc-test -p tests/policy\_config/rule.conf -i instance1 -t 100 -v

打开终端，在/home/NDNRTC/libyuv-master目录下运行./server

请注意打开mix，用于说话声音的传递。

3) 在consumer机器上运行如下命令：

apt install pavucontrol

pulseaudio --start --log-target=syslog

sudo depmod -a

modprobe v4l2loopback

修改tests/sample-consumer.cfg中的source = "/home/customerFIFO "，type= "pipe";

nfd-start &> /tmp/nfd.log

打开终端，在home/NDNRTC/ndnrtc/cpp目录下运行./ndnrtc-client -c tests/sample-consumer.cfg -s /ndnrtc-test -p tests/policy\_config/rule.conf -i instance2 -t 50 -v

打开终端，在/home/NDNRTC/libyuv-master目录下运行./client

为了观看实时流，打开VLC播放器 (open vlc -> media -> open capture device -> video device name -> (select correct video device file (/dev/video0))。