# API of ndn-cxx

Xinyu Ma

April 17, 2019

## 1 Introduction

There is always a gap between the designers and the users, because we are on different sides. In today's tutorial, I will explain how the gap leads to API that leads to today's ndn-cxx API, especially the asynchronized callback, which may be difficult for you.

So let's start with a simple example. Let's design a controller which can turn on and off the air conditioner.

- If the temperature is out of [65, 75], turn on the conditioner.
- If it's in [68, 72], turn off the conditioner.

What we want to write is (one-side only):

**Controller.pseudocpp**

```
while(true){
  // It's not necessary to specify the sensor ID
  temperature = getData("/room/temp");
  aircon_state = getData("/room/aircon/state");
  if(temperature < 65 && aircon_state != "heat"){
    sendCommand("/room/aircon/command/heat");
  }else if(temperature > 68 && aircon_state == "heat"){
    sendCommand("/room/aircon/command/off");
  }
  sleep(10_min);
}
```

**TempSensor.pseudocpp**

```
while(true){
  interest = waitInterest("/room/temp");
  // Notice that we cannot write "/room/temp/${SENSOR_ID}" here
  putData(getTemperature());
}
```

It's obviously that we shouldn't design basic APIs in this way, because

- Not every Interest will return a Data.

- We may want to specify some options like MustBeFresh and Lifetime.
- Those APIs are blocking.
- We should specify how we can talk to the NFD.

## 2 Inconsistent input and output

This is easy, first encapsulate everything into a class, and then add what we can return. So we have value classes like Interest, Data, lp::Nack, Name. The function getData takes an Interest, which is a Name plus all options we want to specify, and returns any of Data, lp::Nack or nothing (Timeout). In C++ a function can only have one return type, but this can be solved by CPS later.

## 3 Connecting to NFD

This is also easy, we have a Face class, which is the abstraction of a Face to NFD. We can specify the way connecting to NFD by passing a Transport when creating the Face.

## 4 Asynchronization

This is a problem. Traditional ways that we learnt from CS118, like select or completion port, is not good in this situation. Because they actually break operations into 2 parts: post a operation and wait for the result, so we need to make a mapping between the working flow and the Interest. We need to do a packet sorting manually by packing everything into a big **switch** statement, which is not good coding style.

Let's do it in another way: delegate to the lib. Tell me what you want to do with the data, I do it. "What to do after get the data" is called *Continuation*, and the delegation way is called *Continuation Passing Style (CPS)*.

## 5 Final Pseudo-code

**Controller.pseudocpp**

```
void after10min(){
  face.expressInterest(Interest("/room/temp", MustBeFresh=true, Can-
BePrefix=true),
                       afterGetInnerTemp,  //What to do with Data?
                       restart,            //NACK?
                       restart);           //Timeout?
}
void afterGetTemperature(Interest, Data data){
  temperature = data.content;
  face.expressInterest(Interest("/room/aircon/state", MustBeFresh=true),
                       bind(afterGetAirconState, inner_temp),
```

```
                        restart,
                        restart);
}
void afterGetAirconState(int temperature, Interest, Data data){
  aircon_state = data.content;
  if(temperature < 65 && aircon_state != "heat"){
    face.expressInterest(Interest("/room/aircon/command/heat"),
                         restart,
                         restart,
                         restart);
  }else if(inner_temp > 68 && aircon_state == "heat"){
    face.expressInterest(Interest("/room/aircon/command/off"),
                         restart,
                         restart,
                         restart);
  }
}
void restart(Interest, Data){
  schedule(10_min, after10min);
}
void main(){
  restart();
  // Give the CPU to the scheduler
  // So the library will wait on events and call our callbacks correspondingly.
  runSchedulerAndIo();
}
```

**TempSensor.pseudocpp**

```
void onInterest(Interest interest){
  face.put(Data(Name("/room/temp").append(SENSOR_ID).appendTimestamp(),
                getTemperature()));
}
void main(){
  // registerPrefix("/room/temp", onInterest);
  setInterestFilter("/room/temp", onInterest);
  runSchedulerAndIo();
}
```

# 6 What's next?

- How to sign or verify packets? (NDN Security)
- How to transmit a large file? (Consumer Producer API)
- How to sync up chat messages between peers? (Sync)
- What if the producer moves? (Mobility like KITE)

- It's hard to remember those commands to set up NFD. (NFD Controller)
- Before that, why we need to configure things manually? (Autoconfig)
- .....

It's not fair to compare ndn-cxx, which is a single library with the full-stack of IP. But people are always doing this comparison.