

Communicate with different nodes

Description

In this module, we will learn on how to communicate with different nodes in the network.

Unlike, IP based networks, which uses special protocols based on the type of data send and other factors but in NDN, it is very simple.

We send the interest packets that containing the info that we need to send the neighbor node. The neighbor node will receive the interest packet, and will send the data packet as a response to the interest packet. The data packet will contain the info that we need to receive from the neighbor node.

We don't need to worry about the type of data that we are sending or receiving interest / data packets.

But one thing to remember is the authenticity of the interest packet. We need to make sure that the interest packet is authentic depending on the criticality of the data that we are sending.

Task

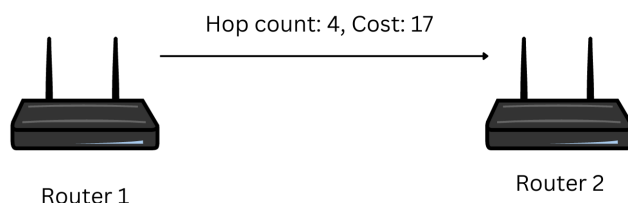
To demonstrate the communication between different nodes, we will create a consumer class, that will send interest packets to the forwarder, and will receive the data packets from the forwarder.

Here i will transfer a information that is shared between the routers in MPP based strategy. The information is the number of hops that the interest packet has traveled so far with cost to acquire it.

Prerequisites

1. Interest Packet
2. Data Packet
3. Scopes of Interest and Data Packets in NDN
4. Interest-In Pipeline

Procedure



1. Let's take we have two router (**R1** and **R2**).
2. **R1** wants to send the MPP info to **R2**. So **R1** will send the interest packet to **R2** with the MPP info inside it.

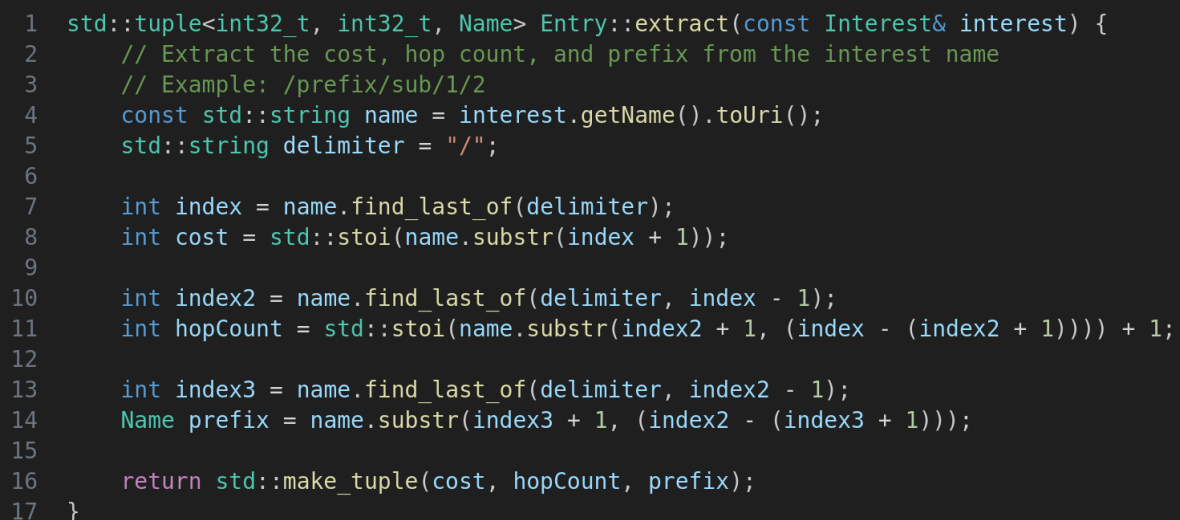
It will be in a format like this:

```
/ndn/edu/wustl//mpp/share/<number of hops>/<cost to acquire>

/ndn/edu/wustl/mpp/share/4/17
```

The format can vary depending on the characters allowed to operate inside the interest packet name.

3. **R2** will receive the interest packet, and analyze it. It will extract the MPP info from the interest packet, and will send the data packet to **R1** as a response to the interest packet.



```
1  std::tuple<int32_t, int32_t, Name> Entry::extract(const Interest& interest) {
2      // Extract the cost, hop count, and prefix from the interest name
3      // Example: /prefix/sub/1/2
4      const std::string name = interest.getName().toUri();
5      std::string delimiter = "/";
6
7      int index = name.find_last_of(delimiter);
8      int cost = std::stoi(name.substr(index + 1));
9
10     int index2 = name.find_last_of(delimiter, index - 1);
11     int hopCount = std::stoi(name.substr(index2 + 1, (index - (index2 + 1)))) + 1;
12
13     int index3 = name.find_last_of(delimiter, index2 - 1);
14     Name prefix = name.substr(index3 + 1, (index2 - (index3 + 1)));
15
16     return std::make_tuple(cost, hopCount, prefix);
17 }
```

See in the image, how **R2** processes the interest packet.

4. Sometimes, the forwarder will ignore any scopes which it couldn't understand.

So we need to make sure that the forwarder understands the scope of the interest packet.

```
// Understand new mpp share scope
bool isMppShare = scope_prefix::MPPSHARE.isPrefixOf(interest.getName());

if(isMppShare) {
    NFD_LOG_INFO("onIncomingInterest in=" << ingress
        << " interest=" << interest.getName() << " isMppShare"); }
}
```

Here is a simple modification in `OnInterest` function of `R2`'s forwarder that will make sure that the forwarder understands the scope of the interest packet.