There will be 5 nodes (node1, node2...) , spread out between our two Rpis. These 5 nodes will have simulated data from 8 sources (sensor1, sensor2,....).

——————————————————————————————————————————

**COMPONENTS**

**Tables:**

1. **FIB:**

The FIB table will map the node ID to server port and server IP. Whenever a hello message will come from a node, the FIB table will be updated.

FIB table will be a **dictionary of dictionary** and will look like:

**{**

**node1:{ip:<>, port:<>, certificate:<>},**

**node2:{ip:<>, port:<>, certificate:<>},...**

**}**

As a table it will look like -

|  |  |  |
| --- | --- | --- |
| Node ID  (eg; node 2) | IP | Port |

**2. PIT (Pending Interest Table):**

This table will have data requested by particular node. Whenever a node needs to forward data, the destination node ID will be fetched from PIT.

It is a **dictionary** and will look like:

**{**

**(<data id>, <neighbor node id>) : (num),**

**(<data id>, <neighbor node id>) : (num),...)**

**}**

As a table it will look like -

|  |  |  |
| --- | --- | --- |
| Data ID  (eg; /wristband/2/sensor/heart) | Neighbor node ID  (eg; node 3) | Num  (eg; 2) |

———————

Packets:

There will be 3 packet types as per NDN layer:

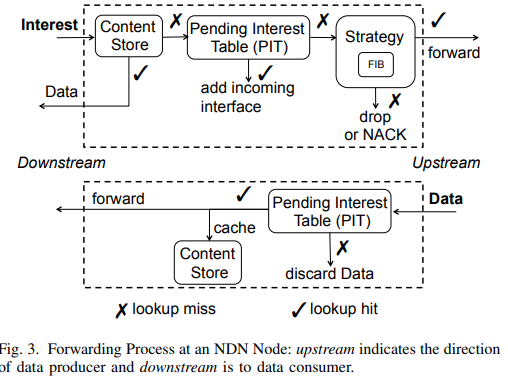
1. **Hello packet**: This will be a simple *HELLO* message sent as keepalive messages every <> seconds. These packets will also be used to update FIB (Forwarding Information Base) tables.

Hello packet is a **string** and will look like:

|  |  |  |  |
| --- | --- | --- | --- |
| HELLO | Node ID  (eg; node 1) | Server Port  (eg; 34001 where the server will listen for incoming messages | Server IP |

2. **Interest packet**: In NDN, a node sends interest packets to ask for specific data. In our case, 3 scenarios take place -

(diagram from NDN paper)



* Cache hit: if the node has particular data stored in cache.
* Owner: If the particular node is the owner of data (i.e; sensor generates reading in the node)
* Cache Miss: When the particular reading is not found in cache.

Interest packet is a **string** and will look like:

|  |  |  |  |
| --- | --- | --- | --- |
| **INTEREST** | **Node ID**  (where data is requested) | **Data ID**  (eg: /wristband/2/sensor/heart) | **Number**  (a request number added in the end to keep track of requests in order to avoid loops) |

3. **Data Packet:** In NDN, data packet is the actual data requested by nodes. There are 2 scenarois which take place when a data packet comes to a node:

* **Requester**: When the data reaches the node which asked for it.

Each node will have a PRT(Pending Request Table) which will have data ID

(node<>, sensor<>) as a **list**. When a node receives its data, the entry will be removed from the list.

* **Data Forward:** When the receiving node is not the recipient of data and needs to forward it.

The node will put a for loop to forward to all the neighbors having the same data ID. (This will be checked from PIT and entry will be removed once packet is forwarded).

Data packet will be a string and look like:

|  |  |  |
| --- | --- | --- |
| DATA | <actual sensor data> | Data\_ID of sensor node  (/wristband/2/sensor/heart) |

————————————————————————————————————————-

**CODE STRUCTURE**

There are two main classes here – Communication class (responsible to listen and send messages) and NDNLayer class (responsible for routing).

Function Call

Comm Class

NDNLayer Class

Callback

**COMMUNICATION CLASS**

|  |  |
| --- | --- |
| **Server (will listen to incoming messages)**  The server will check the string. It will callback the following functions from NDNLayer class, if the first word matches with it. Eg; “Hello” —-> Hello\_callback | **Client (will send outgoing messages)**  The basic work of client is to send data. The NDNLayer will make a function call to this class and hand over the *send* packet. |
| 3 kinds of callbacks:  Hello\_callback | Send(payload, DST IP, DST port) |
| Interest\_callback | Payload could be data/interest/hello. |
| Data\_callback |  |
|  |  |

**NDNLAYER CLASS**

The following explains what will happen when callbacks are executed:

1. **Hello\_callback**

The FIB will be updated, ie; node ID will be mapped to port and IP.

**2. Interest\_callback**

1. Cache Hit:

In this case, NDNLayer class will make a function call to ‘send\_data’ of Communication class.

Data packet will be constructed as (<data>, DST IP, DST Port) and passed to send\_data function.

**B)** Data Owner:

Same as above

**C)** Cache Miss:

If the packet is not found in cache, the next thing node will do is to check in PIT (refer the diagram above). 3 cases arise while checking in PIT:

* Duplicate request:

If num is same as earlier request, inorder to avoid loops the request will be dropped.

* Retry:

In this case, the node has sent another request for interest packet, but with a different number.

The number will be updated by 1 and forwarded (mechanism explained below).

* New Request:

If it’s a completely new request then it will be added to PIT tablewith the source neighbor from where the request came from.

For last two cases, the forwarding will happen as follows:

A for loop will be used with all the neighbors except where it came from.

The interest packet will be created to function call to ‘send\_data’ in communication class.

The interest packet will look like : (interest, neighbor port, neighbor IP)

interest = (INTEREST, self node ID, data ID, num)

**3. Data\_callback**

If the data packet is sent to a node, 2 cases will arise:

* Requester:

In this case, the node received the data it requested for. Hence, it will remove the entry from PRT table.

* Data Forward:

If the node needs to forward the data packet:

A for loop will be created to forward to all neighbors in PIT with same data ID.

Check the node ID of all those neighbors in FIB

Make a data packet : (data, DST port, DST IP)

data = (DATA, <data>, data-ID)

Remove entry from PIT table

——————————————————————————————————————————