Scalable Computing Project 3 Report Group 4

Harry Doyle, Yuze Wang, Pallavit Aggarwal, Jiaming Deng

I. Brainstorming and Going through existing implementations

- General

The first thing we did to implement our network was meet and discuss implementations and general ideas. We looked at current implementation of ICN's, with a specified general architecture of an NDN. Upon doing some research of NDN we came to a unanimous decision of our basic architecture, whereby our 2 Pis would act as different networks. In each "network" we would have various peers/nodes that communicate via peer-to-peer protocols within the network. For extra-network communication (from one Pi to another) a node will communicate via a server that is connected to another server running on the other Pi. This way we could do simplified between network communication and the servers could do all the necessary routing.

- Actual Implementation

Once we had agreed on a common architecture within the group we set out and began the implementation. However, after implementing code, for our server with extra-network functionality we decided that the included server functionality was not sufficient in a peer-to-peer architecture. As such, we decided to alter our network to get rid of these servers and allow all communication and routing to be done via only peers. To do so we needed to ensure that every node was capable of sending and receiving nodes from within the network and outside the network (I.e., we have a common architecture regardless of if the node connects to only nodes within its own network or outside it). We would then update the architecture so that every node would receive the requests, but file returning would be done via the shortest possible path to avoid nodes receiving the traffic that did not need to.

II. Sending and Receiving with multicasting

- Implemented

As previously mentioned, messages were received by the entire network. This was done via a recursive multicast function (which essentially led to the messages being broadcasted, whereby all nodes received the message). This functionality was implemented by assigning an outbound multicast IP group to each node to send messages to. Only neighboring nodes would join this group. Therefore, the node would be able to simultaneously transmit messages to numerous nodes utilizing this group. Upon receiving a message, a node checks its previous messages "cache" and will forward the message to its neighboring nodes if it has not received the message recently, otherwise the message would be discarded to avoid recursively sending the same

message to its neighbors. Upon receiving a request for a sensor within the specified node, that node will call the weather API for the required data from our “sensor” and return it through the network.

- Future improvements

As a result of the multicast functionality, we introduced a heavy IP dependency in the network. For an NDN, routing should be done via unique names instead of IP and port. However, as the current network set-up works following IP protocols, desired functionality could not be implemented unless we defined a whole new network. While messages were sent to nodes containing the name of the node, the routing was done majorly via IPs and these multicast IP groups. To improve on the “named” networking, these IP groups could have been masked to a name, however this would not alter the functionality at all as the nodes send messages to their outbound group (I.E., if a node named A wanted to send a message to B, A would send the node to the group IP masked as A_group. As such there is no mention of B in this “name” only that B belongs in the group, but other nodes may also). To improve the “named” functionality in our implementation we could change the network to only define group IPs for messages to be sent to (whereby, A-group would only be for messages coming to A), however by doing so we are getting rid of the multicasting functionality. An appropriate improvement to make would be to change all the connections from multicast groups and treat them as connections to individual sockets (where a socket has an IP and a port). This socket address could then be masked as a unique name the backend, routing using this name to establish paths. However, this implementation would have a negative effect on scalability (on single nodes, resource wise) on low powered devices as they would need to assign an internal IP and port to every connection. However, this trade-off of lesser resource scalability is outweighed by the positive scalability aspects of the network, as nodes would not be receiving every message sent in the network. (This change would also make the implementation of the PIT/FIB table more effective, as we will discuss later).

III. Creating "inner Group" and "out Group"

- Implemented:

The inner group and the out group are the two categories into which we split the several IP addresses that a node stores in the software we have finished. The nodes in the network that are already connected when a node is established are referred to as the initial group. When the file is given to the node that made the request, the listening and receiving procedures of this group are invoked. The information-transfer-flow for the same request is always one-way, preventing misunderstanding, as each node only needs to transmit the file to the node that submitted the request to it. When a request for a file is made, the out group is notified and instructed to send the request around itself until it locates the node that owns the desired file. The inner group won't change until new connections are made at a node, but the

outgroup will keep changing as more nodes that join the network are continuously added. The out group of each node may be continuously updated since new nodes will continue to broadcast their own "still alive" information to other nodes in the network.

- Future improvements

Our recently finished work is still reliant on network communication using the previous IP architecture. Even while we have implemented the 'interest filename' feature request in the NDN network, we have not yet completely realized the network ecology based only on data flow. As a result of this implementation, while our network is ICN and data is named, our nodes and communication protocols are lacking this named-networking functionality. The primary issue is that because our team's node communication is performed using multicast sockets, it is challenging for us to build the routing table function in NDN due to reasons we will discuss in the PIT&FIB implementation. The IP overlay network, commonly known as P2P, is where most of the research on address aggregation in current data-oriented networks is conducted. This is mostly because the IP-based network has been highly effective, but individual devices' storage and processing capacity are still constrained, and access to all forms of data information in the network is a major issue. To break free from the network restricted by IP, our team will continue to use data flow as the foundation for network building and switch the criterion for identifying distinct nodes from the IP to the type of data that this node contains.

IV. Routing , PIT & FIB and Demonstration of a NDN scenario

- Implemented:

As per the feedback received after demo, an example class have been setup to demonstrate the proper functioning of the below defined methods, their interaction through a use case where data is transferred from India to China through Ireland. There is an Israel node, but that is bypassed. The current PIT, FIB and Cache Store are implemented to accommodate for the following functions. Buildnode builds a node and connects it to another node at inception point. If there’s not another node then it just connects to a null node. The build node internally calls another function advertise, which passes the routing table map to another node that is at cost 1. getAllDestForName, addName, getNameAddress are all implemented to work with the routing table and support FIB functions. CacheStore is used to save data in a map and the supporting functions are addToCacheStore, getFromCacheStore and getData. The getData method currently implements the PIT functionality by storing the interest.

- Future Improvements

The PIT could store frequency as well, and based on frequency caching could be implemented at specific nodes. The possibilities for improvement are endless, but something that definetly would be the way to go forward would be to enable encryption and decryption in the data being passed. Also, the routingTable doesn’t store the cost effectively right now, as the multicast IP functionality creates a group and broadcasts the messages to every Ip in that group, making them at cost 1 distance from it, thereby changing the hierarchical order. We could use more APIs and

maven to utilize more libraries, if we had more liberal access on the Pi. Currently all setup is done, using vanilla Java.

V. Weather Api for real-world data

- Implemented:

Each node requires 10 sensors, each of which needs to store some information. In our implementation a node is considered as a country and the individual “sensors” in the node as a city in that country, which stores the weather information for the specified city. First, we searched the web for a suitable Api to query the weather, and we found one that could query the city for air quality index, pm25, temperature, weather type and wind currently by entering the city name. We have implemented a function that returns the above weather-related data by entering the city name.

- Future improvements

The work we have done is only able to call a limited number of cities with a limited amount of weather-related information and due to the use of Api's implemented by others, sometimes there are problems such as errors being reported. Therefore, we thought we could get the weather information we needed by writing a crawler ourselves, at each country's official weather information website.