Paris Saclay University and TELECOM ParisTech

Internship report
Master of Multimedia Netwokring

Routing Algorithms in NDN Networks

Author:
Shahab SHARIAT BAGHERI

Responsbile of Internship:
Luca MUSCARIELLO

September 2016

Learn from yesterday, live for today, hope for tomorrow. The important thing is not to stop Questioning.

Albert Einstein

PARIS SACLAY

Abstract

PIRL Lab

Cisco Systems France

Routing Algorithms in NDN Networks

by Shahab SHARIAT BAGHERI

Information **C**entric **N**etworking (ICN) in one sentence is rethinking of internet in the sense that it will be no more calling (IP-architecutre) for communication rather we search the content on the network using caching datas with different on each node. Notably this architecure is a good suite for embedded systems in which you have good control on chaching level , memory allocations, network functionalities. In this case like all networks in each node your packets need routing table to send. In this work we propose 4 different strategies called *TreeOnConsumer*, *TreeOnProducer*, *MinCostMultipath*, *MaxFlow* which decide the best path to create faces through proper interfaces then forward packets. Finally we design an algorithm to automatize the algorithm to be chosen in function of network conditions. We developed *Lurch*, the emulator to interact with linux containers dynamically on cisco's server. We use *ndn-icp-download* an application for downloading ndn content. We use *ifstat* linux utility to read network interface statistics. Results are real data rate packet in Kbps using these algorithms which choose different paths according to proper algorithm and network condition.

Contents

List of Figures

To My Parents and PIRL ...

Chapter 1

Introduction

1.1 Named Data Networking

Today's Internet's hourglass architecture centers on a universal network layer (i.e., IP) which implements the minimal functionality necessary for global interconnectivity. This thin waist enabled the Internet's explosive growth by allowing both lower and upper layer technologies to innovate independently. However, IP was designed to create a communication network , where packets named only communication endpoints. Sustained growth in e-commerce, digital media, social networking, and smartphone applications has led to dominant use of the Internet as a distribution network.

Distribution networks are more general than communication networks, and solving distribution problems via a point to point communication protocol is complex and error-prone. Named Data Networking (NDN) project proposed an evolution of the IP architecture that generalizes the role of this thin waist, such that packets can name objects rather than communication endpoint.

1.2 PIRL

My internship has been held in CICSO Company and i had a great pleasure to be under direction of Cisco Principle Engineer Luca MUSCARIELLO for this period of 6 months. PIRL is acrynom for **P**aris **I**nnovation and **R**esearch **L**ab which is located in Issy les moulineaux region, near to Paris (1.1). Leader of ICN in PIRL is Giovanna CAROFIGLIO. Alain FIOCCO is the Director CTO of PIRL.

FIGURE 1.1: Cisco Systems France - 9h:40, 22 July 2016

1.3 Tools of Internship

I used a machine with an operating system of the latest distribution of Linux Ubuntu 16.04, lenovo ThinkPad Laptop serie T450 of CORE i5 with 2 cores. Our expermimental tests are totally done on powerful cisco's server (pirl-ndn-2.com) with 48 cores and 252 GB memory RAM. We have used *ifstat* of linux implementation to see the rates output of result and *ndn-icp-download* application to downloading in MultiThreading processing on different containers at the same time. Latest version of Python (3.5) was used to compile the program.

Lurch is an orchestrator for large scale and highly reconfigurable NDN experimental test-beds written in python for any platform, i.e., large Grids, local lab, providing connectivity among servers involved in the test-bed, ...

Basically Python language is perfectly designed to develop new algorithms becasue of it's very usefull functionalities and handfull algorithm sense, I mean you can write a 10 lines of C code in just one line in python! Albeit according to [2] we note that Python implementations may be not as fast as C/C++ counterparts but they scale with the

input size according to the theory which claims a good reason to choose *Python* anyway as the core part of Lurch.

Actually this emulator emulates linux containers which are like virtual machines in function of your setup. I said like, becasue there is difference between VMs and containers and the difference is the overhead that comes with running a separate kernel and simulating all the hardware when we have VMs. These containers are produced by ubuntu image server installed on the server then *Lurch* clone this image to different machines to have virtual network machines.

This tool can be used for researchers to test their customized network on the real virtual machines and run some usefull experiments. This means you can have your own OpenFlow who enables network controller to determine the path of network packets across a network of switches or virtual ethernet interfaces (MACVLAN interfaces).

Lurch produces some Modules in which you can setup your network: *NDNmanager*, *MobilityManager*, *Topology*, *Clustermanager*. We added a new Module Called *RoutingNDN* in which you will find different algorithms to choose and some functionalities for modifying the Graph of network. We also added some new functionalities on *CommandLineInterface*, *NetworkManager*, *ConfigReader* and *NDNmanager* module to have the ability to change the network parameters like capacity, delay, forwarding plane instantly which was very important in our testing experiments.

Figure 1.2 shows configuration, setup step of Lurch. '../Network' is the address of input configuration files like Topology, position of Clients and Producers and the content that they search, mobility model of network a setting file about URL server, layer2 protocols, information about mobility parameteres, ... By this way you can make your clusters on server.

$m942$ is ExperimentID of the user who runs his expermiment on this server. We use SSH remote login service to enter to each containers.

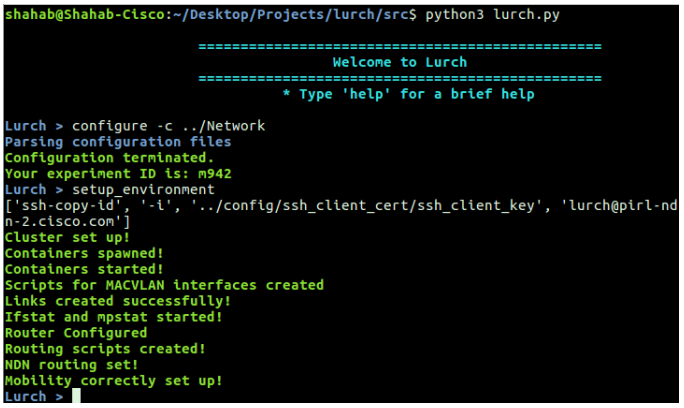

FIGURE 1.2: Command Line Interface

Figure 1.3 is the table of linux containers running on the server *pirl-ndn-2.com*. You can see the IPv4 virtual interfaces on each container and the name NodeID for each interfaces which have the same as NodeID of end link. On Container *m942michel* you see a *wlan0* interface becasue this node is choosen Access Point node in setup level. This interface is now eumulated by ndnSIM using lxc-wifi-tap script. In figure 1.4 you can see an example of how we can get our output in Kbps for each link.

NAME	STATE	IPV4	IPV6	TYPE	SNAPSHOTS
m942jordan	RUNNING	10.2.0.7 (m942luca) 10.2.0.11 (m942mauro) 10.2.0.9 (m942michel) 10.2.0.4 (m942shahab) 10.3.0.4 (eth0)		EPHEMERAL	0
m942luca	RUNNING	10.3.0.3 (eth0) 10.2.0.8 (m942jordan) 10.2.0.6 (m942shahab)		EPHEMERAL	0
m942mauro	RUNNING	10.2.0.2 (m942shahab) 10.3.0.2 (eth0) 10.2.0.12 (m942jordan)		EPHEMERAL	0
m942michel	RUNNING	10.2.0.10 (m942jordan) 10.1.0.1 (wlan0) 10.3.0.5 (eth0)		EPHEMERAL	0
m942shahab	RUNNING	10.2.0.1 (m942mauro) 10.3.0.1 (eth0) 10.2.0.3 (m942jordan) 10.2.0.5 (m942luca)		EPHEMERAL	0

FIGURE 1.3: Table of Emulated Linux Containers on pirl-ndn-2.com

setup environment will setup completely your containers on the server with all of information needed to have a virtual network. As figure 1.2 shows step by step setups which are successful which is not quiet short! *Routing scripts created* is where the routing scritps are creating in each different containers then you have *NDN routing set* step when you push your scripts on containers. normally this stage takes more time than the previous step. Each routing scripts includes bash commands to run which are coming from NFD command line interfaces (Picture 1.5). By *create* and command you can create a new face and *register* you can add the routes in the RIB.

```
09:42:33      0.00      0.00
09:42:34  11712.59    468.15
09:42:35  54105.64   2166.01
09:42:36  52293.70   2089.66
09:42:37  49956.70   2000.38
09:42:38  54431.79   2170.36
09:42:39  50790.21   2034.02
09:42:40  54644.65   2179.14
09:42:41  53018.27   2124.87
09:42:42  52428.17   2088.31
09:42:43  53834.90   2161.84
09:42:44  50228.21   2000.75
09:42:45  48227.50   1923.28
09:42:46  54835.30   2195.19
09:42:47  55025.01   2198.84
09:42:48  50525.77   2018.22
  Time        m942shahab
HH:MM:SS   Kbps in  Kbps out
09:42:49  50620.87   2022.41
09:42:50  46882.78   1864.56
09:42:51  46961.30   1872.60
09:42:52  48090.80   1921.70
09:42:53  47202.18   1886.52
09:42:54  45114.15   1800.44
```

FIGURE 1.4: Rates Trace using ifstat

Next step to continue is doing *start repo* command line which creates a dameon of *repo-ng* application on all producer nodes NFD using a timer which waits for state of process intelligently. Figure 1.6 shows this step in Lurch. As you can see it declares you on which node the repository is created and how many repositories exist on the network.

```
#!/bin/bash

nfdc create ether://[00:16:3e:00:00:0a]/m33jordan
nfdc create ether://[00:16:3e:00:00:0c]/m33luca
nfdc create ether://[00:16:3e:00:00:08]/m33mauro
nfdc register ndn:/n ether://[00:16:3e:00:00:0a]/m33jordan
nfdc register ndn:/n ether://[00:16:3e:00:00:0c]/m33luca
nfdc register ndn:/n ether://[00:16:3e:00:00:08]/m33mauro

exit 0
~
~
~
~
~
~
```

FIGURE 1.5: NDN Routing Script

```
Lurch > setup_environment
['ssh-copy-id', '-i', '../config/ssh_client_cert/ssh_client_key', 'lurch@pirl-nd
n-2.cisco.com']
Cluster set up!
Containers spawned!
Containers started!
Scripts for MACVLAN interfaces created
Links created successfully!
Ifstat and mpstat started!
Router Configured
Routing scripts created!
NDN routing set!
Mobility correctly set up!
Lurch > start_repo
Repositories started!
Please wait ...
Repos on m956shahab:
        Repo ID:"repo-0", Name:"n"
Lurch >
```

FIGURE 1.6: Starting Repository Daemon

In figure 1.7 you can see by *list client* and *list repo* command line which lists the clients and producers on your network. Then as we have started repositories engine, now it's time to choose the routing algorithm. It's easy, you just write route set, then TAB to

see the different algorithms which will appear on your terminal, then you can select one of them and click ENTER.

Now it's time to run your experiment, this is doing just by *start* command line you can see 2 downloading thread is started at the same time becasue you had 2 clients searching same content.

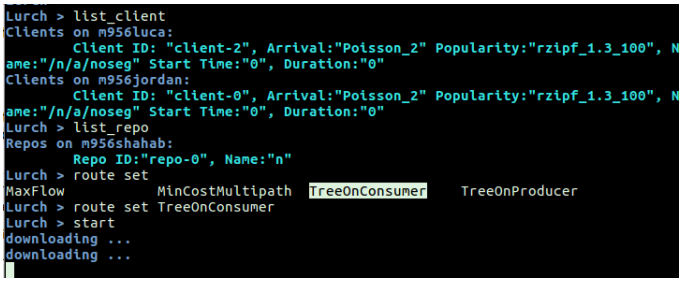

FIGURE 1.7: Downloading Step

In figure 1.8 by *link show* command you can see all of link capacities. by *link edit shahab jordan 20000* you can change link capacity between 'shahab' and 'luca' container.

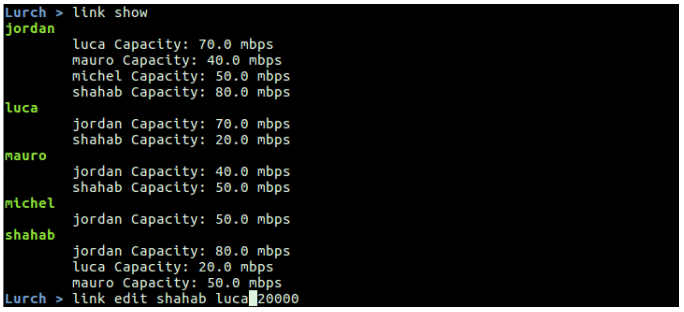

FIGURE 1.8: Link Capacity Modification

We have implemented also implemented an algorithm to full automatization the routing, it means that in function of network situation you can adapt proper algorithm to your routing updates, which depend on workload system can be varried and played. This can be very usefull when whe want to have demonstration on our work.

Picture 1.9 shows well when we do *route auto* TreeOnConsumer algorithm is chosen becasue of the structure of workload. It means that in workload you have one producer and 2 consumers who are searching the content. So this algorithm is just a brain who understands and can adapt algorithms to proper station of system.

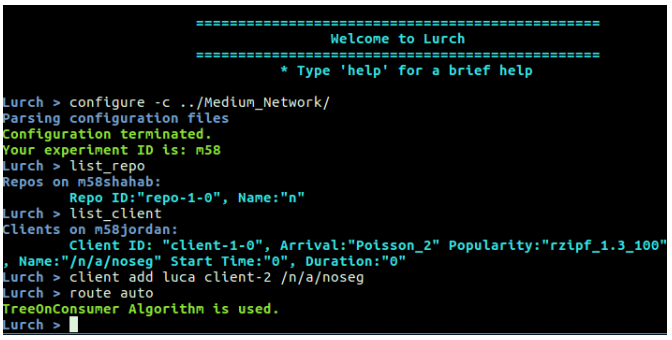

FIGURE 1.9: Link Capacity Modification

In figure 1.10 you can see the difference between linux containers and virtual machines which is very important. Actually linux containers use *cgroups* kernel module to get prior for each machine on the same host server.

This appearance makes linux containers to be very successful becasue of ability to manage and control the guests machines becasue basically they share one kernel on machine.

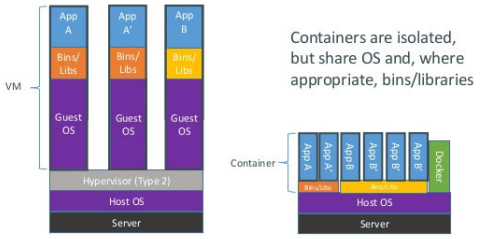

FIGURE 1.10: Link Capacity Modification

1.4 Scope and Objectives

In These days, 2016, there is a lot of research and new ideas on 5G cellular networks on different aspects of Research and Development from advanced chip designing techniques and hardware, antenna desiging to whole different network architectures, algorithms, computations even some ideas to combine it with Quad drones robots and ... for the 2020 plan. It's a bit like the evolution from 1999 to 2000 for IT and computer science domain.

ICN has a fundamental designing philosophy in which you will be able to have like a PeerToPeer (Figure 1.11) or IoT way of thinking in which each node will be a peer who can talk with his neighbours using P2P protocol on second layer of ICN model where he is searching chunk of your data as it's clear on the waist ICN Model (Figure 1.12).

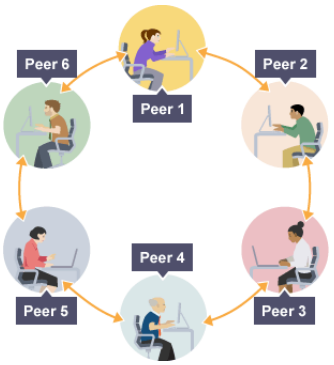

FIGURE 1.11: PeerToPeer Network

Information Centric Networking layer model shows a bit names in ICN can be like a replacement in TCP/IP architecure in this sense you can imagine that all nodes have the same responsibility to pipline packets to the network and you can have Device 2 Device (D2D) communication in which instead of putting clock at a special time alarm in each night, Or at the evening, light and heat of Sunshine (with embedded Sensor installed on window) can capture temperature of heatness then talk to your Table-Lamp installed beside your bedroom and turn it on and your alarm clock to ring and waking you up or immediately turn your microwave on to heat bread inside (1.13)

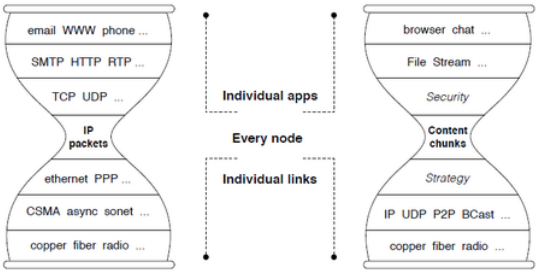

FIGURE 1.12: ICN vs TCP/IP OSI model

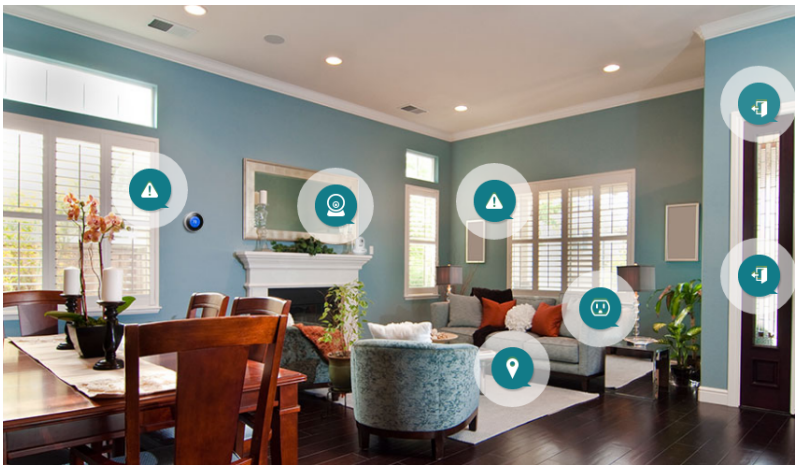

FIGURE 1.13: Smart Room in IoT

Figure 1.14 estimation calculated shows more than 25 Billion things will be connected in 2020! *Smart Cities* is another word that we hear everywhere in Technology domain in these days. So it sounds there exists a strong merging between communicaiton world and computer/artificial intelligence or robotics. This merging needs a better solution for communicating between machines/robots. We believe that ICN is a good architecture to solve this essential problem. It seems also more and more calling communication system philosophy is *expired*! or if it is not today it will be expired in near future. ICN as Jacobson et al explained in [3] has more intelligent and logic architecture.

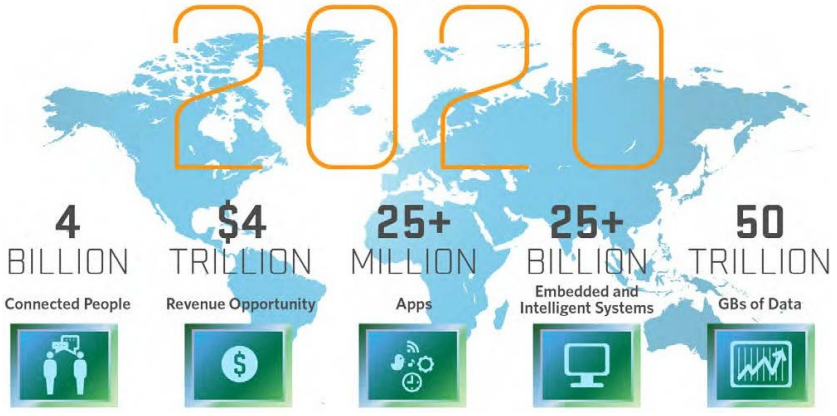

FIGURE 1.14: Plan 2020

We beleive ICN architecture can be a very suitable fit for 5G networks becasue of its advantage against TCP/IP architecture, and important network functionalities like congestion control, storage data, security, and mobility protocols which can be a good idea specially in case of wireless mobile communication in which nodes can be moved

with every speed and channel conditions in the case that all of objects are connected together and they talk together with ICN protocl specification.

So if you are in Cisco Systems Paris and you search *'movie.mp4'* on YouTube maybe on the network one of your neighbours has been downloaded once. So now instead of talking with YouTube's server in USA you can take it from one of your near nodes beside Issy val de seine train station and not making more extra traffic on network.

In this way Cisco has been investigated ICN in context of 5G and notably for popular applications like video delivery, which is one of the most popular applications in these days, people want to see a video on YouTube, Netflix, ... everywhere and everytime in the metro stations, in the building, in the bitches with different Transcoder of video coding and with the highest speed of downloading which network should produce them.

Four Routing algorithms (Strategies) are added to *Lurch* to use in different conditions of Network. *Auto* strategy also is added which is as an algorithm that chooses the proper strategy in function of network condition and in demand of clients. This strategy is added on *NetworkManager* Module.

Chapter 2

Theory of Routing Algorithms

As it is also in IP routing's networks, routing is responsible for building the routing table and maintaining it in face of network changes, including both long-term topology and policy changes as well as short-term churns. When there is a change in the network, routers need to exchange routing updates with each other in order to reach new global consistency. The time period after a change happens and before all routers agree on the new routing state is called the routing convergence period. Routing protocols need to converge fast in order to reduce packet loss and resume packet delivery after network changes. NDN routing does not need to converge fast thanks to its architecture and needs some strategies to pipline the packets to the destinations.

According to [4] as we also work on mobility part of ICN: Some ICN designs are extremely mobile friendly, allowing both clients and servers to move without needing complex rendezvous techniques as in existing IP-based approaches explains well why we have designed our algorithms in this vision.

As figure 2.2 shows normally in literature we devide the whole network in 3 spaces, Mobile edge, Backhaul, Backbone.

In our model we try to model this architecture which is used in cellular networks like 3G, 4G LTE, ... in virtual space we have a real experiment. Our try is to adapt it to 5G network and to create a new network.

```
  udp6://[::]:6363
  unix:///run/nfd.sock
  ws://[::]:9696
Faces:
  faceid=1 remote=internal:// local=internal:// counters={in={0i 55d 28311B} out
={45i 0d 6620B}} local permanent point-to-point
  faceid=254 remote=contentstore:// local=contentstore:// counters={in={0i 0d 0B
} out={0i 0d 0B}} local permanent point-to-point
  faceid=255 remote=null:// local=null:// counters={in={0i 0d 0B} out={0i 0d 0B}
} local permanent point-to-point
  faceid=65687 remote=fd://22 local=unix:///run/nfd.sock counters={in={30i 2d 40
55B} out={2i 29d 15478B}} local on-demand point-to-point
  faceid=67344 remote=ether://[00:16:3e:00:00:0a] local=dev://m33jordan counters
={in={0i 0d 0B} out={0i 0d 0B}} non-local permanent multi-access
  faceid=67876 remote=ether://[00:16:3e:00:00:0c] local=dev://m33luca counters={
in={0i 0d 0B} out={0i 0d 0B}} non-local permanent multi-access
  faceid=68588 remote=ether://[00:16:3e:00:00:08] local=dev://m33mauro counters=
{in={0i 0d 0B} out={0i 0d 0B}} non-local permanent multi-access
  faceid=69299 remote=fd://23 local=unix:///run/nfd.sock counters={in={3i 0d 155
B} out={0i 2d 1001B}} local on-demand point-to-point
FIB:
  /localhost/nfd/rib nexthops={faceid=65687 (cost=0)}
  /n nexthops={faceid=67344 (cost=0), faceid=67876 (cost=0)}
  /localhost/nfd nexthops={faceid=1 (cost=0)}
RIB:
  /localhost/nfd/rib route={faceid=65687 (origin=0 cost=0 ChildInherit)}
  /n route={faceid=67344 (origin=255 cost=0 ChildInherit), faceid=67876 (origin=
255 cost=0 ChildInherit)}
Strategy choices:
  / strategy=/localhost/nfd/strategy/best-route/%FD%04
  /localhost strategy=/localhost/nfd/strategy/multicast/%FD%01
  /ndn/broadcast strategy=/localhost/nfd/strategy/multicast/%FD%01
  /localhost/nfd strategy=/localhost/nfd/strategy/best-route/%FD%04
root@m33shahab:~#
```

FIGURE 2.1: nfd-status on shahab container

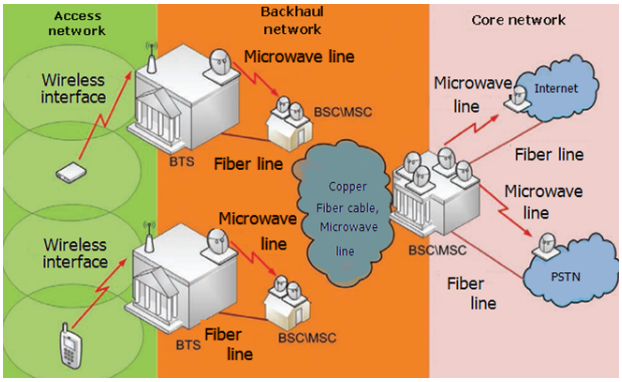

FIGURE 2.2: Real Architecture of Network

2.1 NFD in NDN Networks

NFD is a network forwarder that implements NDN forwarder. After the initial release, NFD became a core component of the NDN Platform and will follow the same release cycle.

There are different modules on NFD for different stuffs like *core* which produce different varities between different modules of NFD. *Faces* implements the face concept in NDN. *Tables* which contains some data structures like Content Store (CS) and tables like FIB and PIB for forwarding and pending information for NDN Data/Interest packets and *Forwarding*, which interacts with forwarding strategies to choose and manage faces and *RIB Management* which implements routing table information about routes and each

prefix according to propoer faces. this is not a seperate module from NFD Management. Basically we create some faces for each prefix by a given routing strategy which are proposed in this work and it will fill up FIB properly against these information by which you find proper routes.

For example figure 2.3 shows 2 faceIDs which are created by our proposed routing strategies in RIB and FIB: 67344, 67876. As you see these faces have routes toward 'jordan' and 'luca' thanks to strategies of TreeOnProducer.

As you can see also in this figure there are different Forwarding strategies for different namespaces. We also added an option on Lurch to change it on run time.

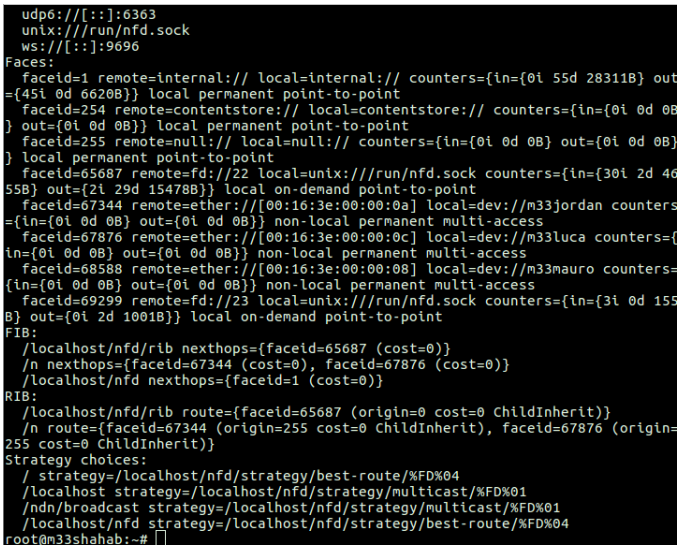

FIGURE 2.3: nfd-status on shahab container

Picture 2.3 shows well that there is no route for 68588 in RIB which means as [3] also says forwarder engine needs a Software to do this kinds of selection which explains very well the reason why we work on these strategies. Each forwarder must decides this selection using proper strategy.

2.2 Routing in NDN

In NDN, the forwarding plane is the actual control plane since the forwarding strategy module makes forwarding decisions on its own. This fundamental change prompts us to rethink the role of routing in NDN. Routing protocols are responsible for disseminating topology and policy information, computing routes and handling short-term network changes.

Shortest path, *Distance Vector*, *Link state* and their variants IS-IS and OSPF are the routing algorithms that are most widely used inside large networks and the Internet of today from ARPANET (Bellman, 1957; and Ford and Fulkerson, 1962). ([1])

2.3 Routing Strategies

ShortestPath and *Djikstra* algorithms are designed to obtain the paths which minimizes the cost metric of each link.

In this subsection we introduce 4 different algorithms which are defined in function of network's need for NDN networks.

We will also mention the ideas and reseaons for these strategies. Next section we will show some results and figures about algorithms for one medium size and large size of network.

In this work we will work on Shortest path using different metrics. Basically number of hops for links in order to calculate our routes properly in function of network postion.

ShortestPath In *Networkx* Python library is called *Unweighted* method for Graph becasue it assumes number 1 for each link which is simply number of hops. In ndnSim which is an ndn simulator, this weight has been put 1 because of NDN. In this library *Djikstra* is called *weighted* methods becasue it cares about the number of weight to which the link is mapped and it's the way how it obtains proper paths.

For Discovering more about algorithms we refere you to A to see the details of implementation.

2.3.1 Consumer & Producer Trees Strategy

In ICN the cost metric is *number of hops* becasue of naturity of ICN architecture in which you *search* content so basically in Graph sense nearer nodes are preferred always. This is the idea that enables us to think about *TreeOnConsumer* and *TreeOnProducer* strategies in which you want to find the tree which finds the shortest paths either from one producer to consumers or one consumer to producers properly.

The idea of *TreeOnConsumer* is based on this concept that imagine once in cellular networks at the same time maybe you can have N clients or consumers that search exactly the same *content*, like a special video.

Figure 2.4 shows this concept properly.

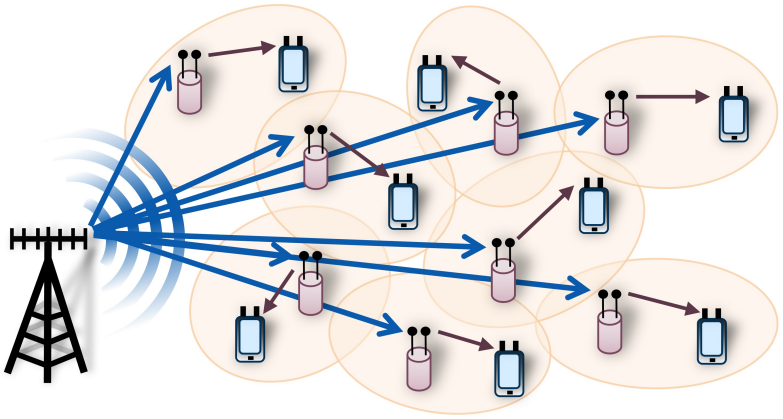

FIGURE 2.4: TreeOnConsumer

TreeOnProducer is a bit inverse thinking of previous algorithm, in which imagine you, as a client to speed up your video downloading you can retrive your data using different repositories packets to catch up (Figure 2.5).

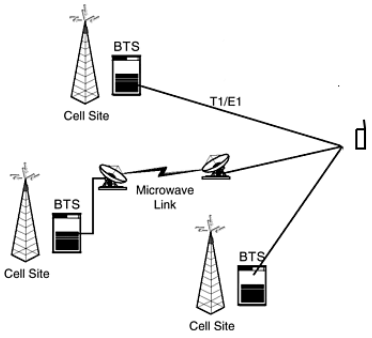

FIGURE 2.5: TreeOnProducer

So the idea of *TreeOnProducer* is to use all of network available repositories on the core part of the network by this way imagine a case of very dense clients on one cell network will be able to understand the amount of demands and to respond them at the same time using this strategy to have maximum possible usage of network.

2.3.2 Minimum Cost MultiPath Strategy

Imagine a case you have multiDestination, like a live videostreaming football match that has a lot of viewers in the network at the same time, in this case nodes should do some kind of multicast sending packets to the networks so network should choose the routes which minimizes the cost of links from source to destinations. The cost is always number of hops in ICN context because of naturity of architecture.

Figure 2.6 shows well the idea of this strategy which is designed for multi-destination case with minimizing cost.

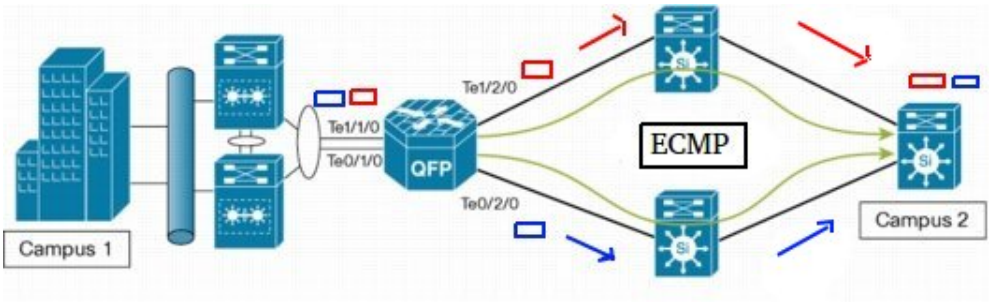

FIGURE 2.6: Eqaul Cost Multipath Strategy on CISCO Router

As you can see in 2.6 is a simple topology example of real cisco routers implemented between 2 campus in this case you can deliver packets, one by one wich means packets can travers through one upper path and the other with belower path.

Multi-Producer, Multi-Consumer is when you have multiple consumers who are searching the same content and multiple producers for this content. This algorithm is the core of the strategies becasue of its ability. Actually this algorithm looks at the position of clients and producers then for each client finds the closest producer (as we said number of hops is always our metric) then routing is done.

2.3.3 Maximum Flow Strategy

This strategy is done by calculation of paths and links which produce maximum throughput from source node to sink node. *MaxFlow* is very important strategy which allows the links to participate in subgraph which maximizes the capacity and basically it creates an one-directed graph and all ratio traffic through these links should be calculated by this strategy. Figure 2.7 shows well this explanation. Algorithm has chosen paths which maximizes flow to the sink node.

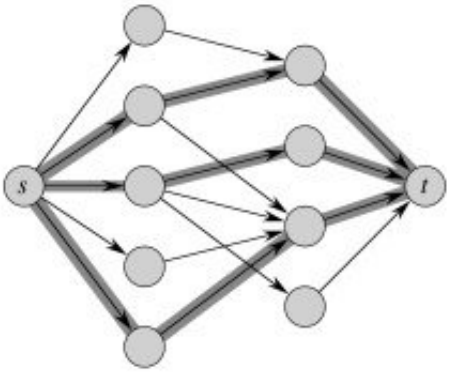

FIGURE 2.7: Maximum Flow Strategy

Specially this strategy can be used when mobile phones want download video packets with very high quality video coding rates (i.e Ultra HD ,4K, ...) or to speed up downloading content and the network allows to get information through different paths from sources to clients. This strategy works with a weight (*capacity*) on each link.

As it's clear there is some paths which are not selected becasue it wouldn't produce total maximum flow on network. The idea of this strategy is to use throughput capability of network to achieve to have the best quality at the edge user experience.

This can be usefull whenever imagine a case the base station as a one cellular have to decide a lot of connected mobile stations at the same time. This needs a heavy packet transmission to all clients. Imagine also if all clients want to see a 4K video, so how much data should network be able to transfer to all clients. This decision can be taken by Lurch in which you are able to say to this base station to send the packets.

Chapter 3

Experimental Results on Routing Strategies

3.1 Experiment Test Setup

We tested our experiments on clusters of cisco's server with linux containers with different configuration set ups on Lurch. In the case of needs, dynamic modifications also have been added on Lurch to be more interactive for different test scnearios.

In this chapter we will show, test and validate our 4 routing strategies on 2 different setup networks with 5 nodes (medium size) and another one with 17 nodes (large size) each one with costumized capacity. We plot rates in Kbps vs time for each link involved in strategy. $\{ai\}$, i =1:8 are Access Points and nodes with *NodeID* are Autonomous nodes in the network.

In all scenarios we plot the figures of each link by nodeID of ends connected. In, Out means the direction of packets that come to node. So In always is Data packets and Out are Interest packets. As it's clear they are larger than We consider number of hops as cost of link as it is meaningful in ICN architecture. *TreeOnConsumer*, *TreeOnProducer*, *MinCostMultipath*, *MaxFlow* are 4 different parameters of RoutingNDN module to choose which are done for *Medium* and *Large* size. We have put the result of each case inside the subsection.

In all cases Clients are searching /n/a/noseg chunk and repositories are containing /n folder which has alphabetical letters to like a,b,c,d, ... then chunks of *BigBuckBunny.mp4* are appended to them. Engine of repository is an application called *repo-ng*.

3.2 Strategies on Medium Size NDN Network

3.2.1 TreeOnConsumer

As figure 3.1 shows, 'luca' and 'jordan' nodes are clients searching '/n/a/noseg' chunk and 'shahab' node is producer of content. The first prefix of content is /n folder contains.

Figure 3.2 shows 4 traced rates in which yellow and green one are data packets. Same packets are routed through *TreeOnConsumer* tree to the clients with different rates. In this figure you can see the delay between 'sj' and 'sl' which is becasue of capacity difference of between containers. So you have 2 threads on 2 different machines occuring and recorded at the same time.

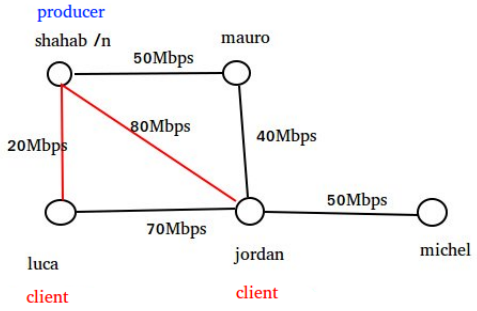

FIGURE 3.1: TreeOnConsumer Tree Medium

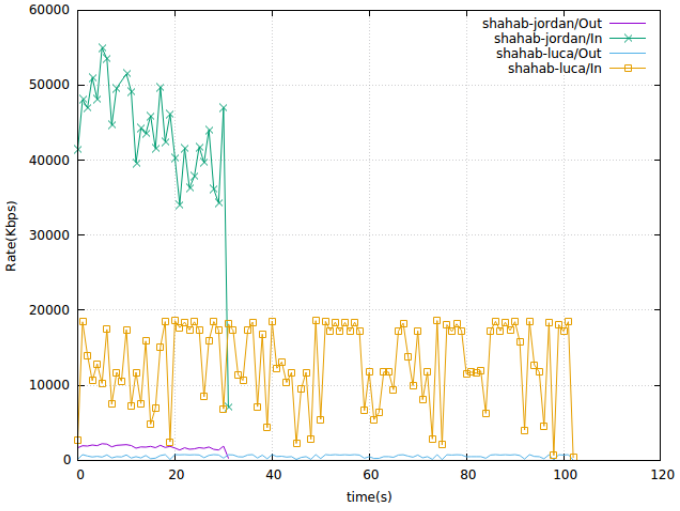

FIGURE 3.2: TreeOnConsumer (Rate vs Time) Medium

3.2.2 TreeOnProducer

Figure 3.3 shows 'jordan', a client who searches '/n/a/noseg' and 2 producers ('shahab','mauro') who send packets to client by this strategy using tree of *TreeOnProducer*.

Figure 3.4 shows as well this downloading traced during time which can be used whenever client wants a high quality content.

Imagine a case in which the clients are searching a very huge amount of data, specially let's say 10 clients that they want to watch a video at the same timethrough connected base station.

This strategy can be very usefull in the case that by understanding multihoming of the core system or backhaul which base stations are connected.

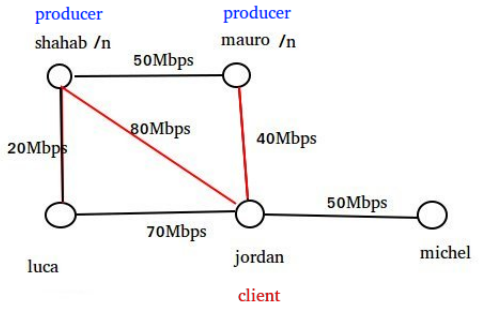

FIGURE 3.3: TreeOnProducer Tree Medium

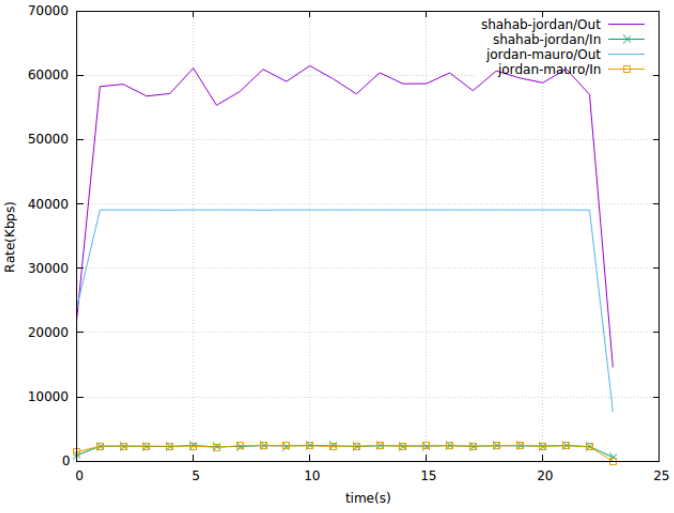

FIGURE 3.4: TreeOnProducer (Rate vs Time) Medium

3.2.3 MinCostMultipath

Figure 3.5 shows *MinCostMultipath* tree of minimum of network in which we use number of hops as cost of links. This is the case in which we want to broadcast video livestreaming as we discussed. Same chunk of datas are searching at the same time for a live movie like a football match content to all clients.

This figure will be usefull when you have a multihoming system in which you are able to download your content from different nodes. This strategy will be a good idea in the case that you have a lot clients on one base station are attached.

This is a co-algorithm of TreeOnConsumer or a way of rethinking of that. In TreeOnConsumer you assume that multi nodes are searching the same content while in TreeOnProducer you have just one client that searches content but with using multihoming advantage of network.

Figure 3.6 shows traced data as well.

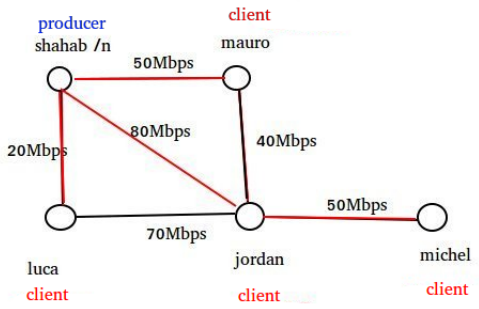

FIGURE 3.5: MinCostMultipath Tree Medium

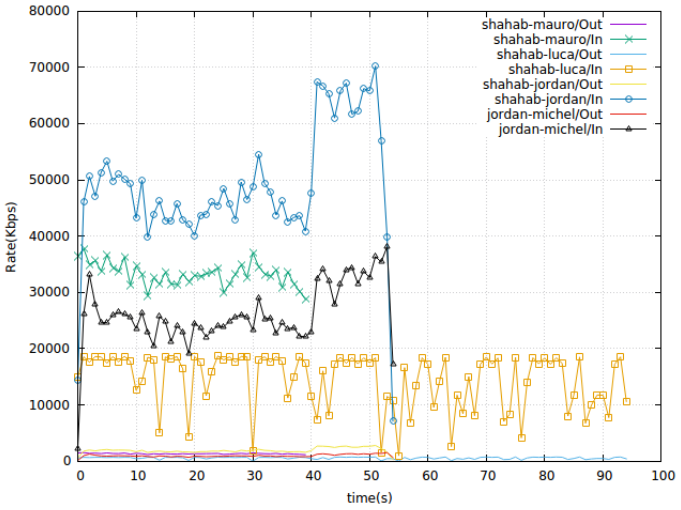

FIGURE 3.6: MinCostMultipath (Rate vs Time) Medium

3.2.4 MaxFlow

In figure 3.7 'jordan' node gets information from all of path possible to maximize data throughput.

Figure 3.8 shows data rates which are maximum for each link. It can be used whenever clients need higher quality data.

You can see for marginal paths you have on the stable rates.

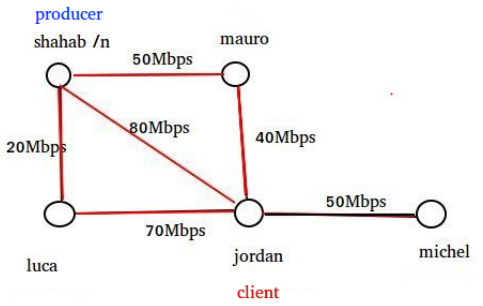

FIGURE 3.7: MaxFlow Medium

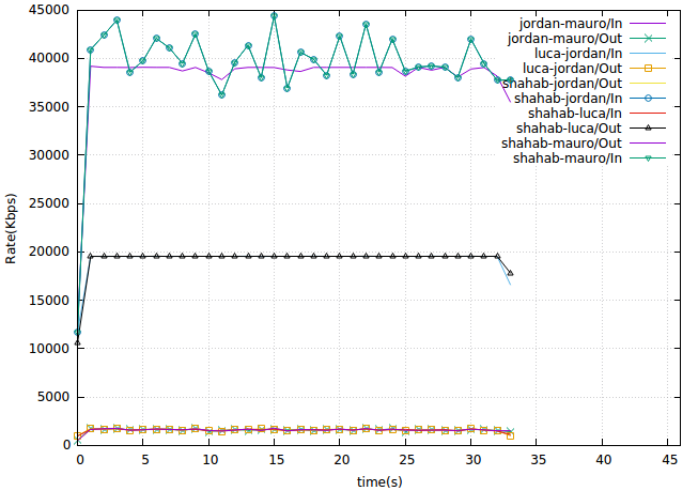

FIGURE 3.8: MaxFlow (Rate vs Time(s)) Medium

3.3 Strategies on Large Size NDN Networks

3.3.1 TreeOnConsumer

Figure 3.9 shows a red subgraph, $a1, a2, a3, a4$ are Acces Points on which the mobile clients are connected through *TreeOnConsumer* tree on the network.
In 3.10 rates are traced against time as well.

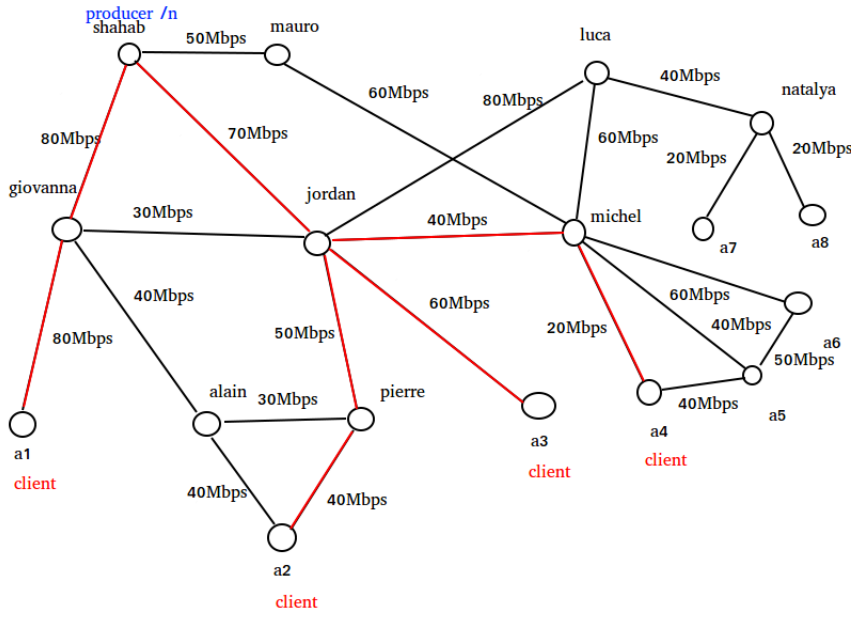

FIGURE 3.9: TreeOnConsumer Tree Large

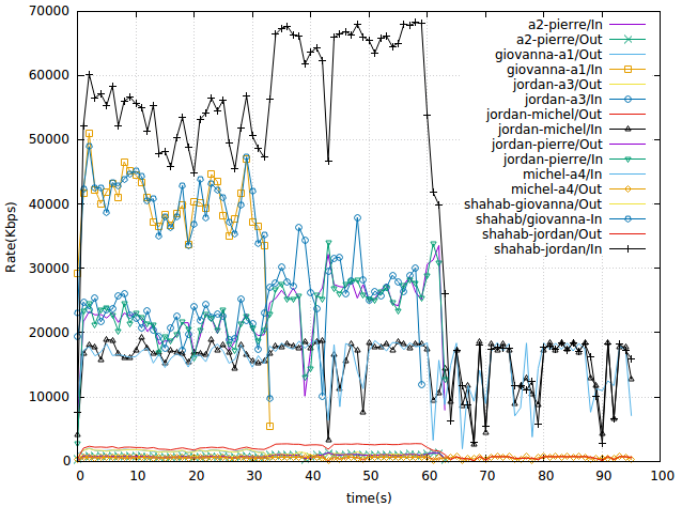

FIGURE 3.10: TreeOnConsumer (Rate vs Time) Large

Figure 3.11 shows the first step of client mobility in which the mobile station is attached to node $n10$. Figure 3.12 shows also the Interest/Data packets which are going through shortest paths.

You can see in figure 3.13 that as soon as mobile client is attached to $n11$ routing update is happened and new routing is traced.

Same case for 3.15 which is for showing that update can occured. This update can be designed periodically or to have an algorithm. After we need to know more practical constraints like how much time takes that an routing update happens? The answer depends on what we use for engine forwarder and its implementations.

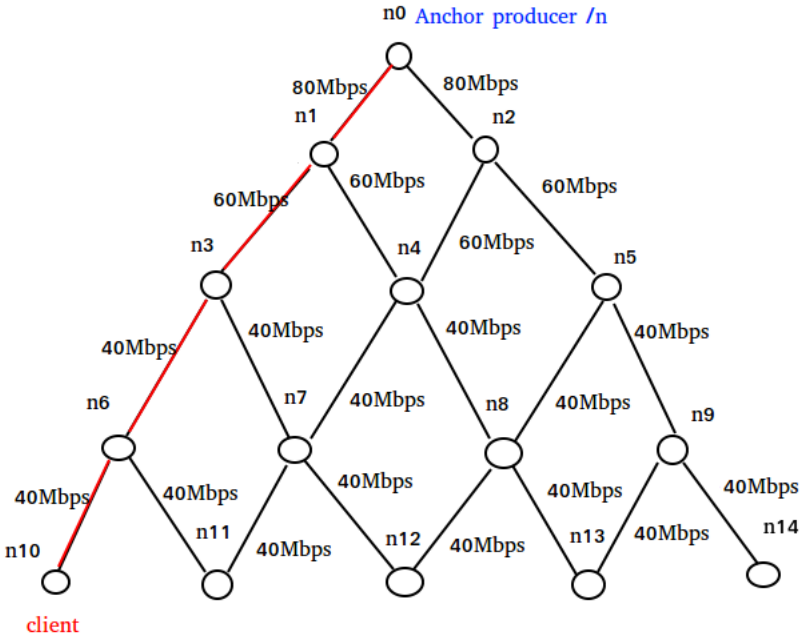

FIGURE 3.11: TreeOnConsumer Mobility Large

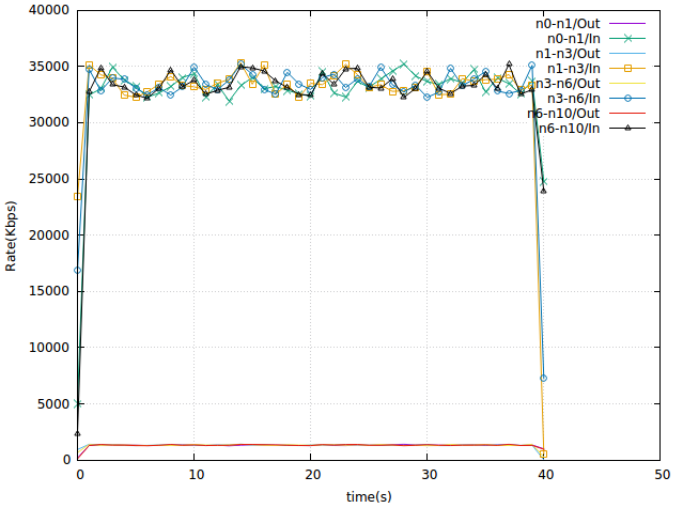

FIGURE 3.12: TreeOnConsumer Mobility (Rate vs Time) Large

Now in figure 3.12 and 3.14 you see the rates which are changed whenever producer is changed. This changed is done mannually in this case.

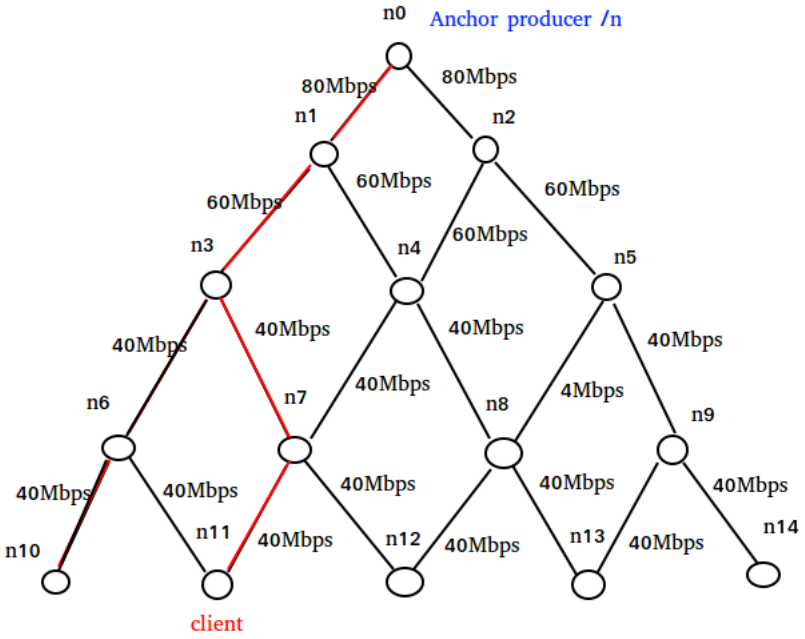

FIGURE 3.13: TreeOnConsumer Mobility Large

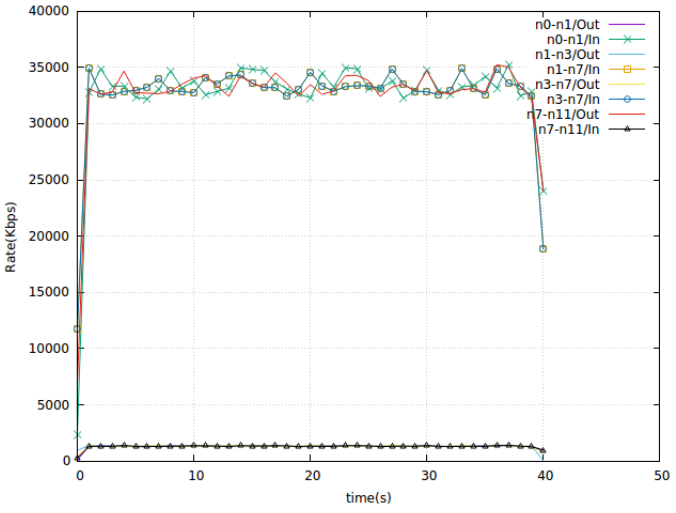

FIGURE 3.14: TreeOnConsumer Mobility (Rate vs Time) Large

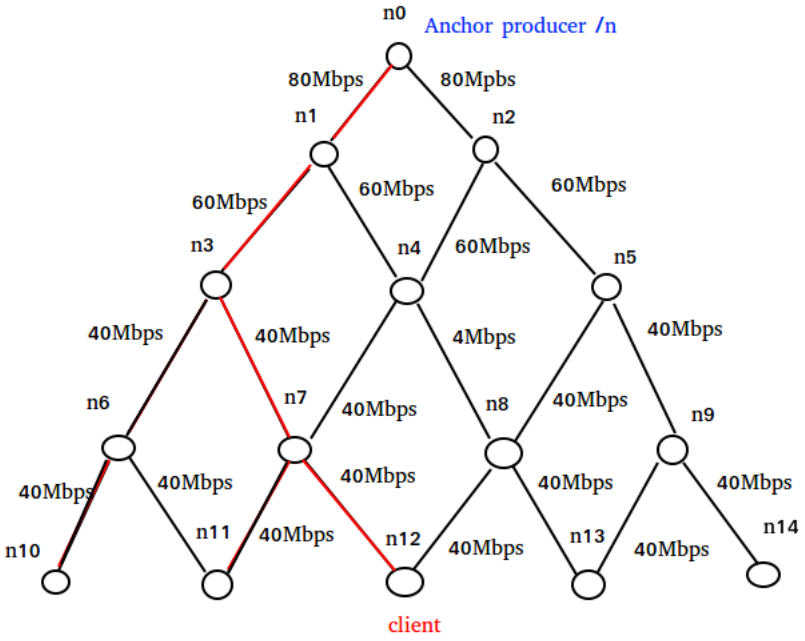

FIGURE 3.15: TreeOnConsumer Mobility Large

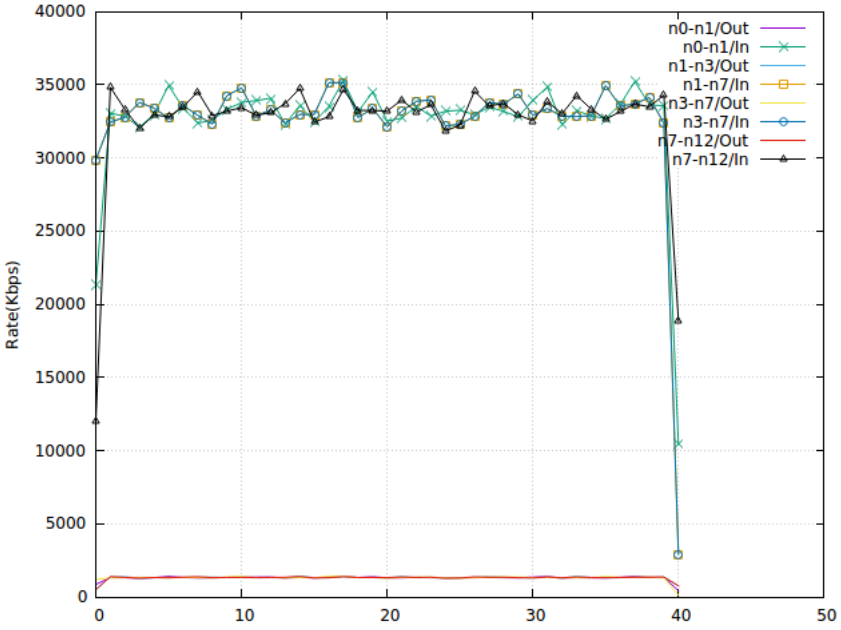

FIGURE 3.16: TreeOnConsumer Mobility (Rate vs Time) Large

3.3.2 TreeOnProducer

Figure 3.17 shows $a6$ as Access Point client which recieves data from different producer nodes at the same time.

This strategy can be used whenever client can achieve to multiple repositories and it can download his chunk at the same time.

This is very usefull specially in case of load balancing strategy. Client can get his packets with different paths using different producers in the sense that maybe on the network mobile node wants download a higher data rates so he can use the benefit of multihoming of the content which is searching.

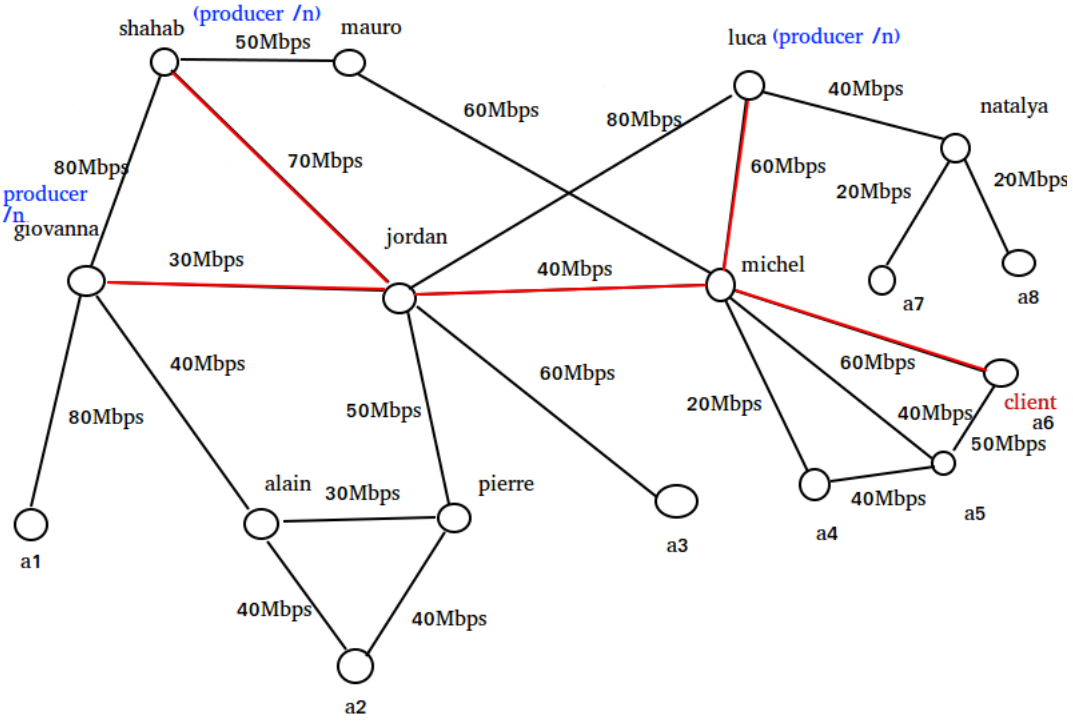

FIGURE 3.17: TreeOnProducer Tree Medium

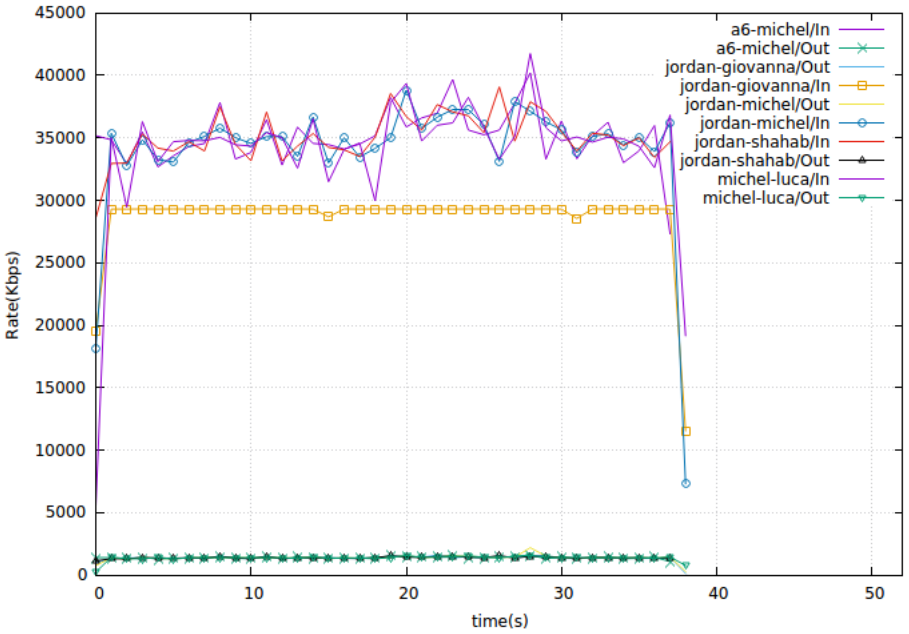

FIGURE 3.18: TreeOnProducer (Rate vs Time) Large

3.3.3 MinCostMultipath

Figure 3.21 shows how the *MinCostmultiPath* tree is chosen to deliver content to all clients which are connected to access points.

Notably this strategy can be used when you want to do a *Equal Cost Multi Path* (ECMP) routing in which your minimized paths have equal cost to destination so we should use load-balance or multipath forwading strategy to allow traffic to pass along the network.

Figure 3.20 shows well a simple case usage of this strategy using *load balancing* strategy to use multipath routing.

There is a very important issue in this case of strategy, and the important concept is that when the routing strategies has decided or multi faces for each node the forwarder strategy can not be more best-route! Actually in this case engine of forwarder should allow the kernel to pass packets through diferent faces. Kernel lets operates like a switch to forward the packets into th network.

By this way we see clearly that designing a routing strategy also depends on forwarding strategies.

In NFD, there is different forwarding strategies *best-route* which uses the best route in terms of cost link function. *multicast* which send the packets in multicast way, *broadcast* which pipline all packets toward all, faces in PIT table. These namespaces has been implemented and used in our studied works and experiments.

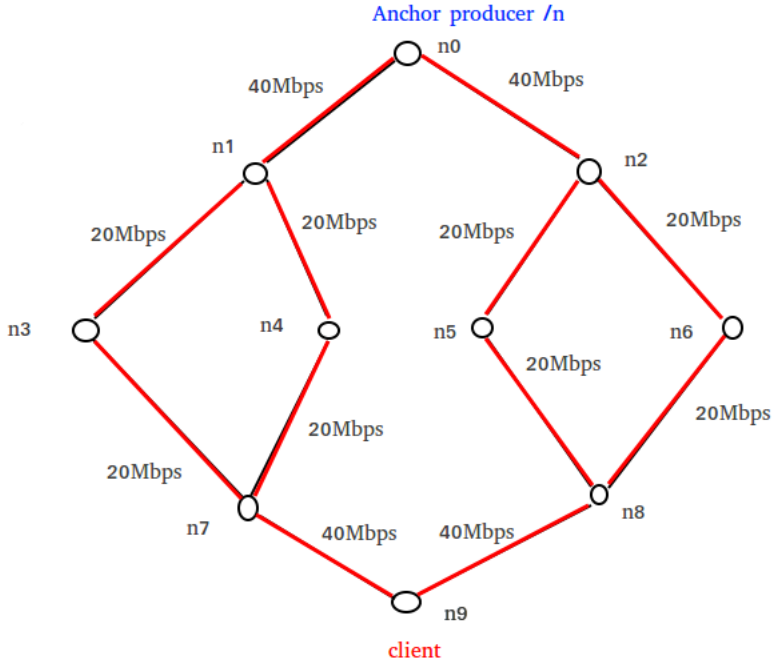

FIGURE 3.19: Multipath load balancing strategy Graph

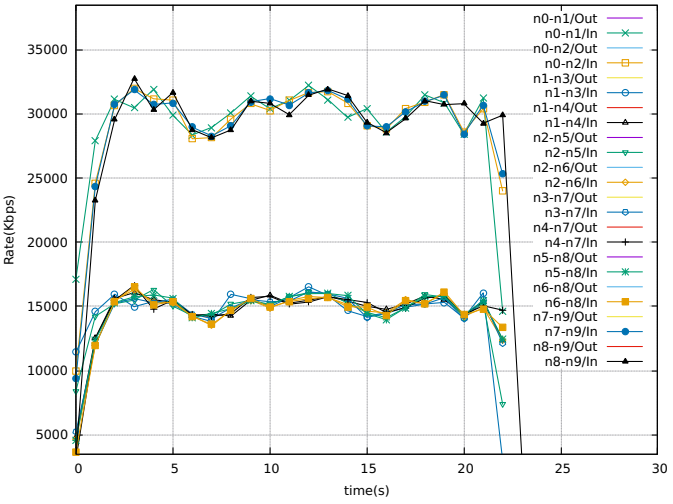

FIGURE 3.20: Multipath load balancing strategy (Rate vs Time)

The algorithm of this strategy is written by the idea of searching minimum multipath from producer to client.

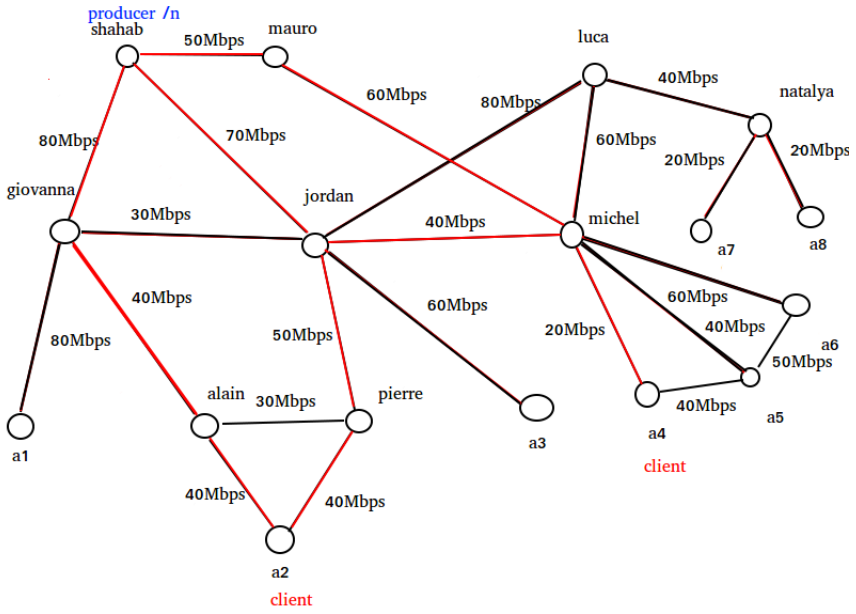

FIGURE 3.21: MinCostMultiPath Tree Large

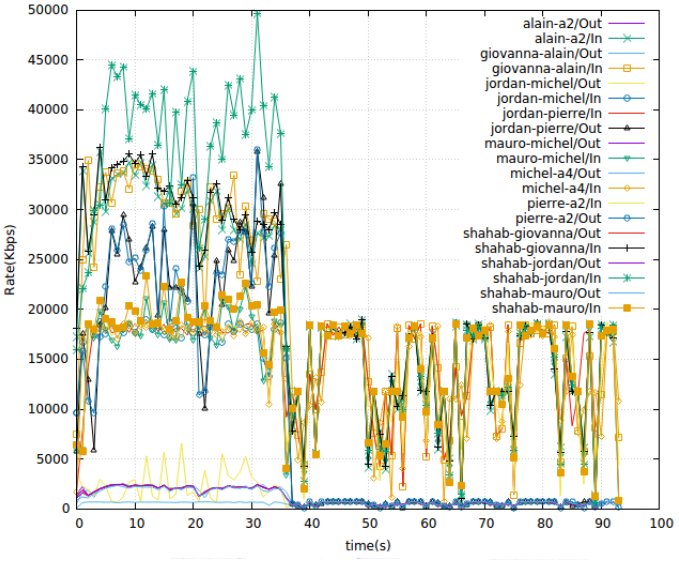

FIGURE 3.22: MinCostMultiPath (Rate vs Time) Large

Multi Consumers and Producers. Figure 3.23 as you can see we can have n producers and m consumers want to download at the same time. This strategy chooses the shortest paths between clients and consumers using to give routes update. Actually for each consumer, algorithm looks which producer is closer using *shortest path* then set routes properly. This strategy can be very usefull in ICN networks becasue of its architecture.

Routing is the core issues of any network even Network of streets and broadways! It's the idea of Routing in general is to have knowledge about network and its proper graph

then have a table at each router to calculate best routes for each packet. This table can be static or dynamic, normally in practice routers are dynamic in function of network needs.

One beautiful idea which is very *à la mode* in research and development in these days becasue of its advantages is to have a **S**oftware **D**efine **N**etwork. The SDN can be a central controller who allows users to modify the network in terms of capacities, network topology, routing, and whatever you can think about the network which can be changes dynamically.

You can also imagine a scenario to occur, you can have algorithms inside algorithms to run full automate evidences. This strategy can be very usefull in these scnearios becasue of NDN architecutre.

By running this strategy on Lurch, it can understand whenever you want to do a ECMP or Multi repository client. It's important becasue network when you have your controller (Lurch) who understands the scenario in which you want to apply your strategies.

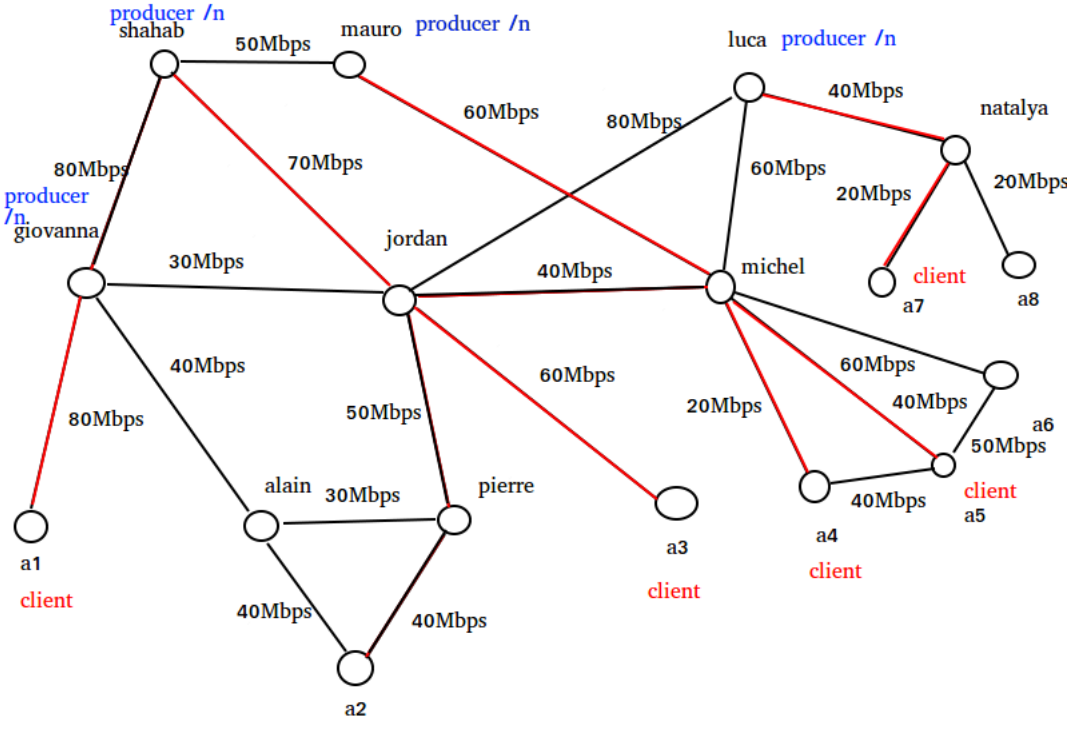

FIGURE 3.23: Multi Consumers and Producers Large

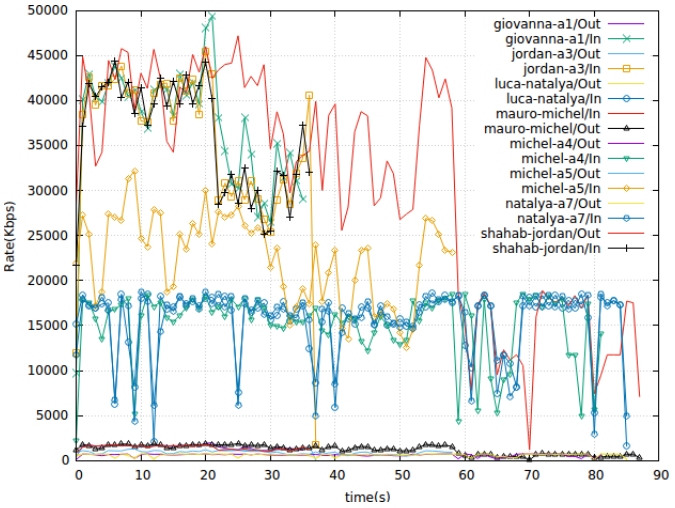

FIGURE 3.24: Multi Consumers and Producers (Rate vs Time) Large

Now let's consider an interesting point of this strategy in Mobility of ICN. Figure 3.25 shows the first step of a mobility scenario in which producer is attached to n18. In this case producer can be a Robot carrying a webcam to capture a film from environment send it to network as the content producer. The clients are mobile stations and are attached to base stations as well.

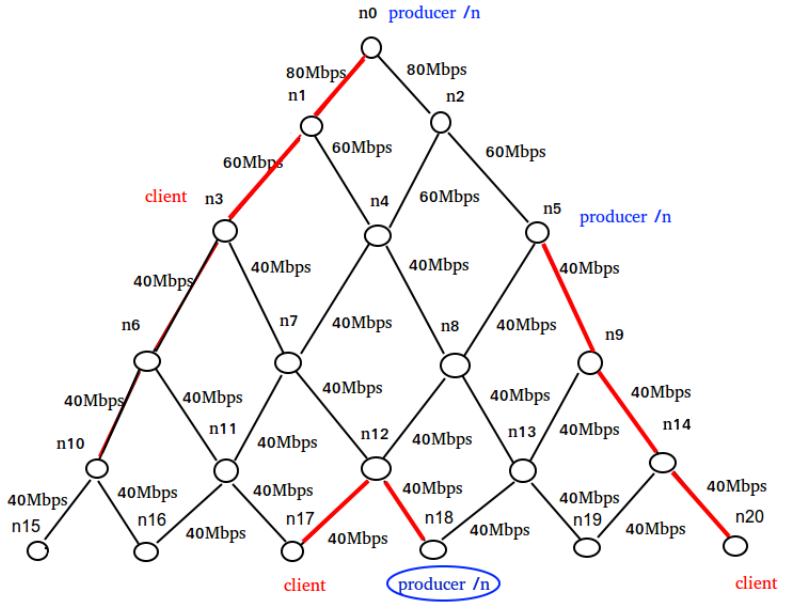

FIGURE 3.25: Producer Mobility step 1 & Routing update

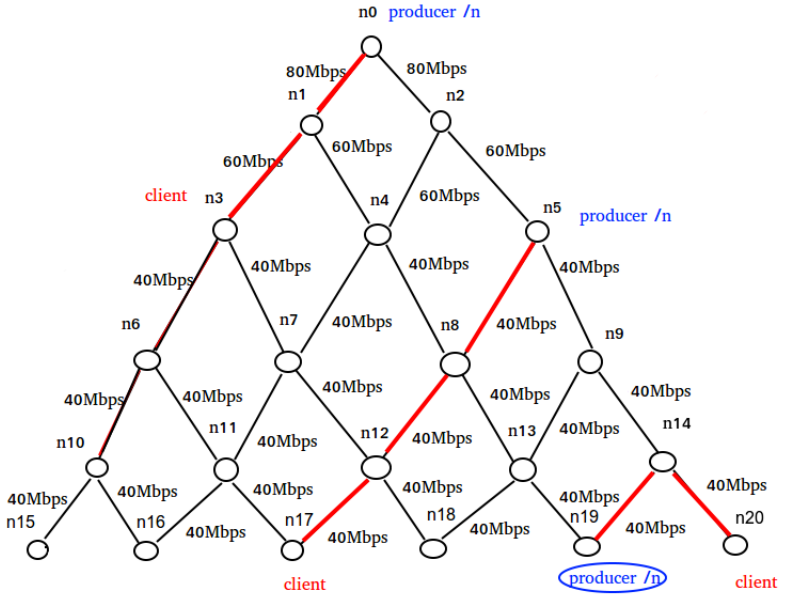

FIGURE 3.26: Producer Mobility step 2 & Routing update

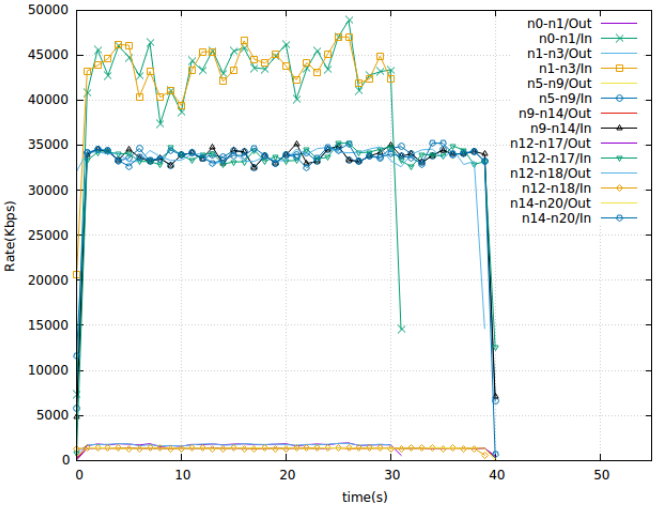

FIGURE 3.27: Producer Mobility step 1 (Rate vs Time) Large

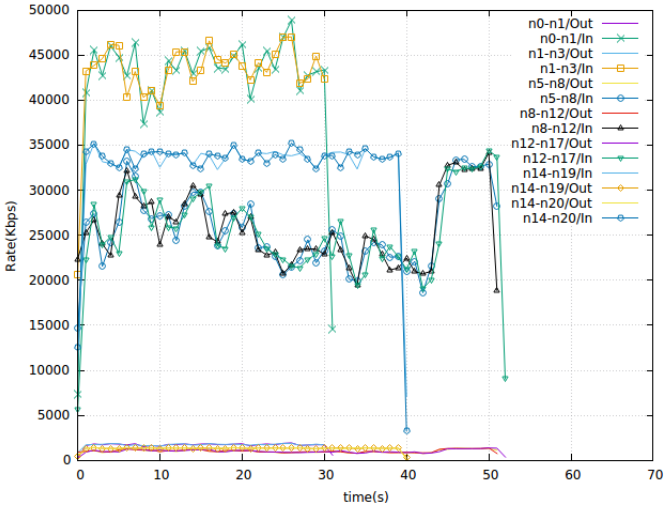

FIGURE 3.28: Producer Mobility step 2 (Rate vs Time) Large

Now let's consider a case which i call it *Dense* network becasue the number of clients and repositories are more than rests.

Figure 3.29 shows well automatic routing best shortest path for each consumer to achieve to producers using this strategy.

Figure 3.30 shows also the stairs of rates falling down in the network becasue of waterfall sense of Network. Basically more you go deeper to network the capacity rate will be increased which is real assumption in networks becasue (backbone / backhaul / Mobile edge, ...)

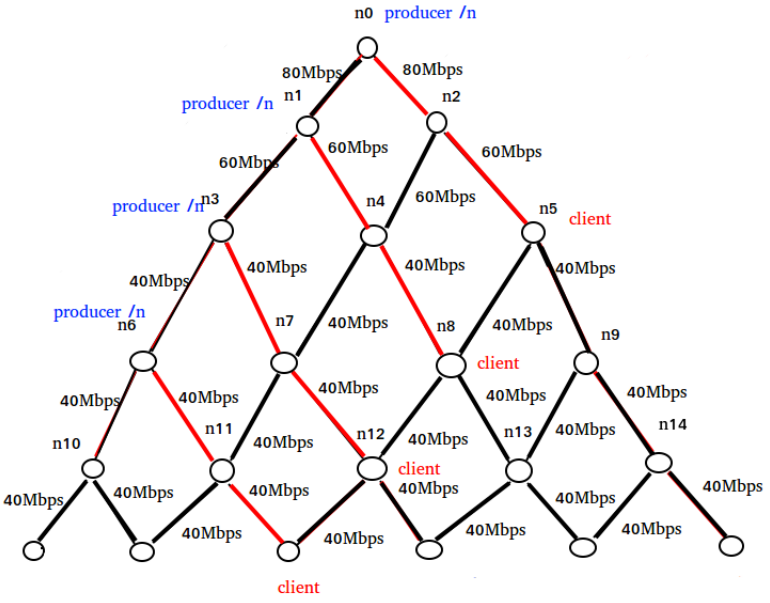

FIGURE 3.29: Dense Network

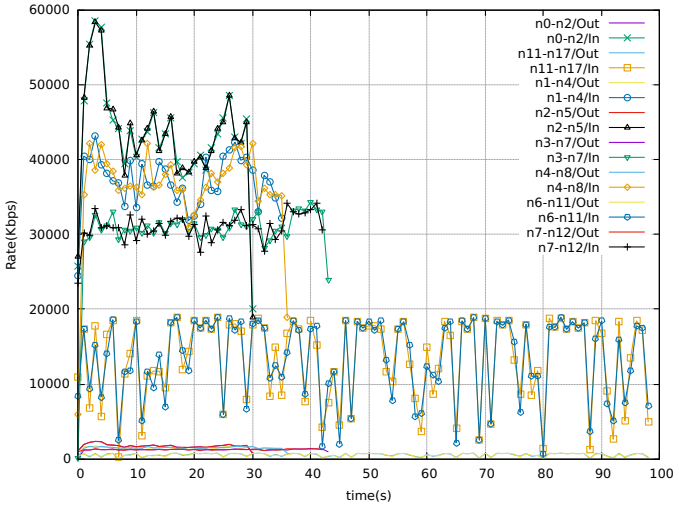

FIGURE 3.30: Dense Network (Rate vs Time) Large

3.3.4 MaxFlow

Figure 3.31 shows the paths chosen by strategy to maximize the throughput. This strategy works with capacity of each link and according to these value it returns the paths which can maximum the throughput toward clients.

Figure 3.32 shows how the rates are changing during time with this strategy. As you can see data packets have maximum value to achieve the output.

The mobile stations are connected to $a2, a3$ Access Point can download their contents through taps interfaces.

The algorithm of this strategy is written by finding the paths which maximize throughput with preventing out of range extra loops.

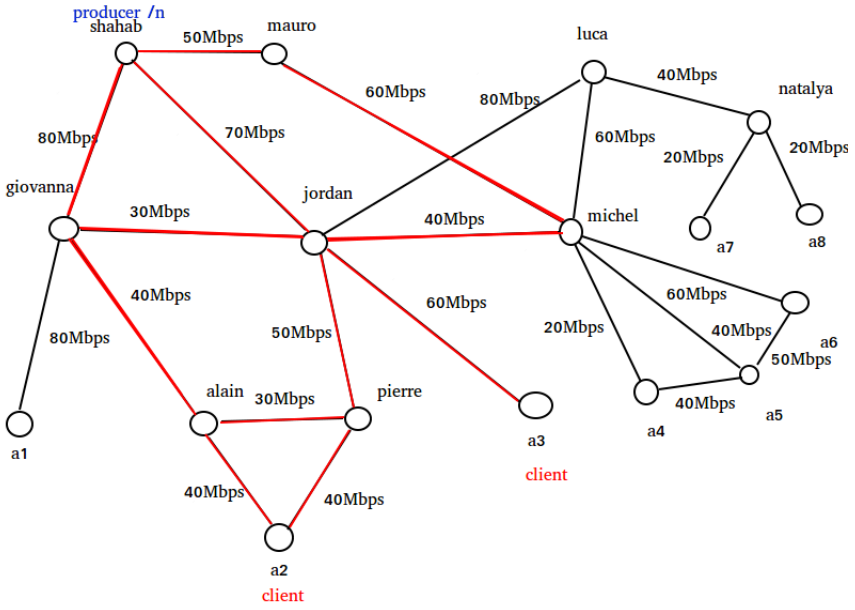

FIGURE 3.31: MaxFlow Tree Large

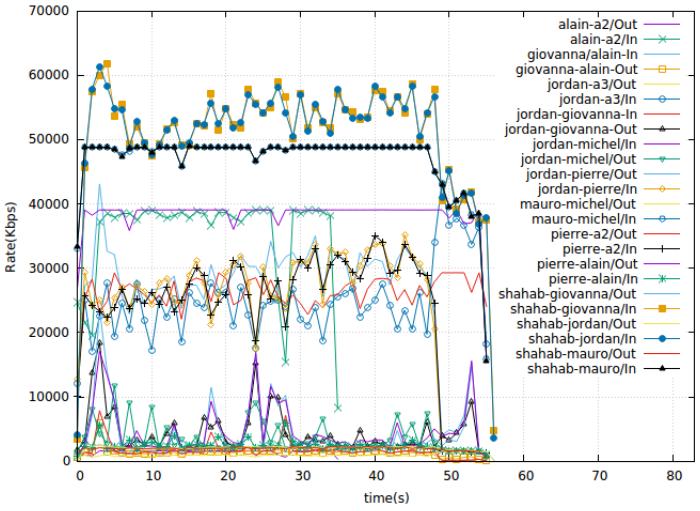

FIGURE 3.32: MaxFlow (Rate vs Time) Large

Chapter 4

Conclusions

Always in every Engineering systems there are trade off which Communication Systems also is not an exception of this history, each day we are going to develop our ideas about something maybe a great technological idea like IoT, SDN, ... Actually when you are an engineer that you understood when it should use a clever way to solve your problem. 'genius' is to apply a science in the real world is just a combination of art and science and it's beautiful.

To be a good engineer it should learn all of tools that are going to be useful and they define your power to progress your work without problems. Some days you don't get a good result, it's a *good news* because you got a great experience to not repeat it and it must always try to find the answers and suitable results. So it seems that you should have a project and you must define a goal for your project.

Network protocols are completely different by several reasons and visions so depends on your system your designing must be varied. In Wireless and Wired Networks we are limited in some constraints who are mainly definitive for our objects, control of objects from go away is now our ideas because the thing that connect our real world to Computer, Internet or generally virtual worlds are *Sensors*

Computer Science is the science that i can implement and see my results and that's it's beautiful. when you understand how it works.

Routing algorithms should exist in the routers and how this is perfect. The very basic idea is that each router should know what should it does with packets. This logic which should be implemented is the core of one network. Becasue otherwise you can not flow your packets to the good destinations. Our strategies are proposed to solve this fundamental question and need.

Appendix A

Code Implementation

RoutingNDN Module added to Lurch.

```
"""
  This class Implements virtual Routing algorithms in NDN
  Algo_Name to be chosen : 'TreeOnProducer', 'TreeOnConsumer',
  MinCostMultipath', 'MaxFlow'
"""
__author__ = 'shahab SHARIAT BAGHERI'

import random
import networkx as nx
import configparser
from collections import defaultdict
import itertools
import Lurch.TopologyStructs as TopologyStructs
import Lurch.Globals as Globals


class RoutingNdn:
    def __init__(self, node_list=None):

        """
        This function initializes node list and builds graph

        :param node_list: list of nodes.
        """

        self.node_list = node_list

        # Creating Graph
        # Topology dictionaries
        self.G = nx.Graph()
        self.dict_repo = {}
        self.dict_client = {}
        self.network_index = 0

    def create_graph(self):
```

```
        self.G.add_nodes_from(list(self.node_list.keys()))

        network_index = 0
        for i in self.node_list.values():

            # Wired Part of Network

            # Create Graph
            for j in i.links.values():
                self.G.add_edge(i.get_node_id(), j.node_to.get_node_id(),
                                capacity=j.capacity if type(j) is TopologyStructs.WiredLink else
                                cost=1 / j.capacity if type(j) is TopologyStructs.WiredLink else
                network_index += 1

            # Repository and Client Dictionary

            repositories = i.get_repositories()

            if repositories:

                self.dict_repo[i.get_node_id()] = []
                for repo in repositories:
                    content = repo.get_folder()
                    self.dict_repo[i.get_node_id()].append(content)

            clients = i.get_client_apps()

            if clients:

                self.dict_client[i.get_node_id()] = []
                for client in clients:
                    content = client.get_name()
                    self.dict_client[i.get_node_id()].append(content)

    def get_index(self):

        """
        Get maximum network index
        """

        return self.network_index

    def get_graph(self):

        """
        Get graph
        """

        return self.G

    def algo_ndn(self, Algo_Name):

        """
        Find best path between consumer and producer.
```

```
and list.

:param Algo_Name: name of chosen algorithm
:return lis:  list of calculated routes.
"""

self.create_graph()

#  Init Routing
for i in self.node_list.values():
    i.routes = {}

# TreeOnConsumer Algorithm

if Algo_Name == 'TreeOnConsumer':

    # TreeOnConsumer Algorithm

    for repo, prefix in self.dict_repo.items():
        for p in prefix:
            for k, v in nx.single_source_shortest_path(self.G, repo).items():

                if len(v) > 1 and v[-1] in self.dict_client :

                    for i in range(0, len(v) - 1):

                        self.node_list[v[i + 1]].add_route(self.node_list[v[i]], p)
# TreeOnProducer Algorithm

elif Algo_Name == 'TreeOnProducer':

    for client, prefix in self.dict_client.items():
        name =  self.dict_repo.values()
        for p in name:

            for k, v in nx.single_source_shortest_path(self.G, client).items():
                if len(v) > 1 and v[-1] in self.dict_repo:
                    for i in range(0, len(v) - 1):

                        self.node_list[v[i]].add_route(self.node_list[v[i + 1]], p[0])

# MinCostMultipathConsumer Algorithm

elif Algo_Name == 'MinCostMultipath':

    lc = len(self.dict_client.keys())
    lr = len(self.dict_repo.keys())

    if (lc > 1 and lr == 1) or (lc == 1 and lr == 1):

        for repo, prefix in self.dict_repo.items():
            for client, prefix in self.dict_client.items():
               name =  self.dict_repo.values()
               for p in name:
                  l = min(map(len, nx.all_simple_paths(self.G, repo, client)))
```

```
                        for member in nx.all_simple_paths(self.G, repo, client):

                            if len(member) == l:
                                for i in range(0, l-1):

                                    self.node_list[member[i + 1]].add_route(self.node_list[memb


            if lc > 1 and lr > 1:

                for client, prefix in self.dict_client.items():

                    l = []

                    for repo , p in self.dict_repo.items():

                        l.append(nx.shortest_path(self.G, client, repo))


                    v = min(l,key=len)
                    for i in range(0,len(v)-1):

                         self.node_list[v[i]].add_route(self.node_list[v[i+1]], p[0])


        elif Algo_Name == 'MaxFlow':

            for client, prefix in self.dict_client.items():

                for repo, p in self.dict_repo.items():

                    flow_value, flow_dict = nx.maximum_flow(self.G, client, repo, capacity='capa
                    for k, v in flow_dict.items():

                        for ki, vi in v.items():
                            if vi != 0:
                                self.node_list[k].add_route(self.node_list[ki], p[0])



        else:
            print('-----------------------------------------------------------------')
            print('You should choose the name of algorithm.')
            print('-----------------------------------------------------------------')

    def add_edge(self, n1, n2):
        """
        Add a link to topology

        :param n1: node1 of graph.
        :param n2: node2 of graph.
        :param b: bandwidth of link.
        :param c: cost of link.
        """
```

```
        self.G.add_edge(n1, n2)

    def delete_edge(self, n1, n2):

        """
        remove a linke from topology
        and list.

        :param n1: node1 of graph.
        :param n2: node2 of graph.
        """
        self.G.remove_edge(n1, n2)
```

Bibliography

[1] Computer Networks *Andrew Tanenbaum*

[2] Weighted Graph Algorithms with Python *Marian Smoluchowski Institute of Physics, Jagiellonian Un iversity* April, 2014

[3] Networking Named Content *Van Jacobson Diana K. Smetters James D. Thornton Michael F. Plass*

[4] Information Centric Networking for Media Distribution: Will it Blend? *David Oran, Research & Advanced Development Cisco Systems, Cambridge, MA, USA*