MASTER II IFI (Informatique : Fondements et Ingénierie)
Parcours UBINET – Ubiquitous Networking and Computing

Rapport de Stage de Fin d'études 2012

SOFTWARE DEFINED NETWORKING
IN WIRELESS MESH NETWORK

Stagiaire : Xuan Nam NGUYEN

Maître de stage : Thierry TURLETTI

Tuteur Enseignant : Giovanni NEGLIA

Equipe-Projet : PLANÈTE
Laboratoire de Recherche : INRIA Sophia Antipolis
1er Mars 2012- 31 Août 2012

Résumé

The explosion of mobile devices and contents in recent years leads to high mobility demand. Wireless Mesh Network is considered as next generation of wireless networking, which supports high mobility and network access. Among user services, content delivery is an important service of the future Internet and Content Centric Networking is the most promising architecture for this service. To deploy new services over Wireless Mesh Network, we need the programmability in network and it comes from Software Defined Networking (SDN), a new approach to networking in which network devices and software controlling them is decoupled. However, mostly SDN implementations have targeted to wired, fixed network. Motivated by this vision, the objective of this work is to explore how to use OpenFlow- the leading SDN implementation - in Wireless Mesh Network and then improving content delivery service over Wireless Mesh Network with the support of OpenFlow.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor Mr. Thierry Turletti, Mr. Damien Saucez and Mr. Walid Dabbous for their comments, enthusiasm, and immense knowledge.

Contents

List of Figures

Chapter 1

INTRODUCTION

The explosion of mobile devices and contents in recent years leads to high mobility demand. Wireless Mesh Network (WMN [2]) is considered as next generation of wireless networking, which supports high mobility and network access. Among user services, content delivery is an important service of the future Internet and Content Centric Networking (CCN [3]) is the most promising architecture for this service. To deploy new services over WMN, we need the programmability in network and it comes from Software Defined Networking (SDN [1]), a new approach to networking in which network devices and software controlling them is decoupled. However, mostly SDN implementations have targeted to wired, fixed network. Motivated by this vision, the objective of this work is to explore how to use OpenFlow [4]- the leading SDN implementation - in Wireless Mesh Network and then improving content delivery service over Wireless Mesh Network with the support of OpenFlow.

Chapter 2 provides the necessary background on SDN, WMN, CCN[1] and the problem statement. Chapter 3 presents ideas for supporting OpenFlow in WMN. In chapter 4 and 5, we discuss how to improve content delivery. We finally draw the conclusion in Chapter 6 and show the new perspectives that are now open.

[1]The reader who is familiar with SDN, WMN, CCN can skip the section 2.1,2.2,2.3

Chapter 2

STATE OF THE ART

2.1 Wireless Mesh Network

Infrastructure-based and infrastructure-less are two architectures used to interconnect computer systems. Comparing to the infrastructure-based networks, infrastructure-less networks have some advantages, they do not need access points to communicate each other. Nodes are self-organized and self-configured. This provides an inexpensive and quick way to connect devices. In the future, infrastructure-less networks might become popular because the explosion of mobile devices.

Wireless Mesh Network (WMN)[2] is a type of infrastructure-less networks and have emerged as a key technology for next generation wireless networking because of its advantages comparing with other wireless networks. WMN consists of mesh routers and mesh clients, and gateways. Mesh clients are laptops, smart phones, they can be stationary or mobile. Mesh routers form a mesh topology, have minimal mobility and wirelessly relay traffic of other mesh routers or clients. Moreover, mesh routers can perform intensive functions because they are often not limited by resources compared to mesh clients. WMN delivers wireless services including Internet access for clients. Even though there are recent advances in the research and development of WMN, many research challenges still remain: protocol proposals in all layers; new schemes for network management and the security problem [2].

2.2 Content Centric Networking

The host to host is a communication model between exactly two machines: one has the resources and one wants to access them. In a long time, they realize that this model has some limitations: lack of availability, mobility and security support[3]. Nowadays, contents are ubiquitous and everywhere, user does not need to care where they come from. Content-based model communication has been proposed to fit today's requirement. There are different architectures for this model and the most popular one is Content Centric Networking [3] (CCN, 2.400.000 search results on ieee.org).

With CCN, contents are retrieved directly from their names, independently of their location. CCN nodes have multiple faces (A face is a network interface (3G, Wifi, Bluetooth, Ethernet) or an application process within a machine). A CCN node maintains three types of data structures: Firstly, list of faces waiting for contents, **Pending Interest Table** (PIT). Secondly, list of potential content sources, **Forwarding Information Base** (FIB). Finally, list of contents can serve by this node, **Content Store** (CS). If a user wants to get content (a video, document, a web page...) he sends a request (Interest packet) to a face of a CCN node. Intermediate CCN nodes use their FIB to forward Interest, while updating theirs PITs to deliver the content (Data packet) to the user later. CCN is a promising architecture to provide content delivery, mobility support. Some open research issues in CCN: How to name contents to satisfy user-friendly, structured, effective; routing protocols proposals to improve deliver performance and reduce traffic overhead; what is effective caching policy.

CCNx [11] is an implementation of CCN, which follows the overlay approach, CCN over IP. CCN message header (name content, chunk number, security information...) is encapsulated into payload of UDP packet, that allows middlebox traversal. CCNx is still on developing (current version is 0.62 and supports Linux and Android platform).

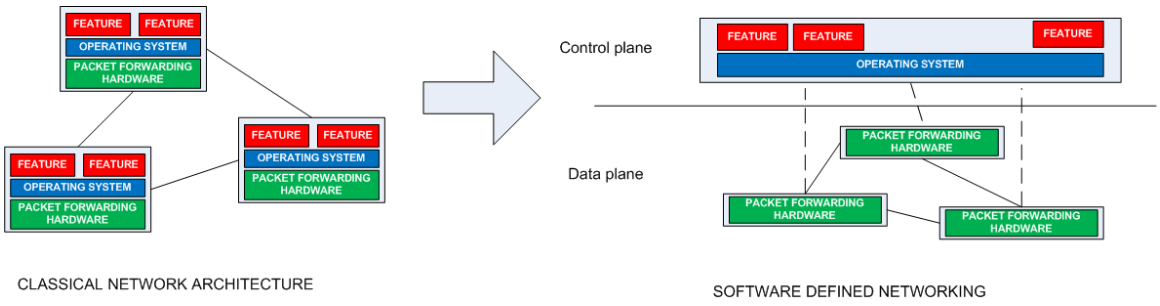

Figure 2.1: Classical Network Architecture vs. Software Defined Networking

2.3 Software Defined Network

Today, network devices (switches, routers) are directly implemented by vendors such as Cisco [5], Juniper [6]. The control software is the asset of vendors and is located on devices (the left of Fig. 2.1). Because of this, it is hard for operators, researchers and even vendors to innovate their network. Operators can not create and deploy quickly new applications and services. On the other hand, researchers can not experiment new ideas, new protocol in real network setting. Even vendors can not adapt fast enough to meet their customer requirements.

Software Defined Networking (SDN) is a new approach to networking. In SDN, network devices and software controlling them are separated (The right side of Fig. 2.1), a standard interface to program with network devices is provided and the underlying network is abstracted. With SDN, operators and researchers have the flexibility to configure, to manage, and to optimize network resources via dynamic, automated SDN programs.

Most promising SDN implementation is OpenFlow (OF [4]). It has been receiving attention from academia and industry, and currently supported by OpenNetwork Foundation [1](Google, Microsoft, Facebook . . .) Moreover, it has been implemented by many vendors (e.g. Cisco, HP, Juniper).

The **OF Switch** is responsible for packet forwarding and the network control is located on the external device, namely the **controller** (Fig. 2.2). An OF switch contains a flow table, which is a set of flow entries. A flow entry has 3 components: **matching rule** – fields of the packet with value, **actions** – list of actions to execute

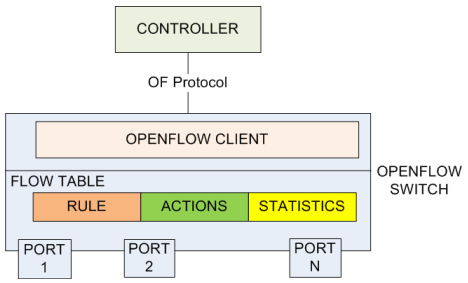

Figure 2.2: OF Architecture

when seeing a packet matching with this rule and **statistics** of this flow entry. The controller communicates with OpenFlow Client – a software on OF switches by OF Protocol which defines a set of primitives, messages via a SSH connection [7], over network itself (in-band control) or a private network (out-of-band control). Upon packet arrival, the OF switch checks whether there is a match between field contains in packet header (Switching covers the headers at layer 2, 3 and 4 of IP stack) and executes appropriate actions: forwarding, modification, drop... If no match is found, the OF switch queries the controller how to process the packets. Afterward, the controller inserts a flow entry to the OF switch and next packets of this flow are processed using this flow entry. There are several possible choices for the controllers which support OF protocol. Beacon [8] is stable, easy to extend and it is the fastest controller as showed in the performance evaluation done in [4].

There are some demonstrations of SDN/OF: in campus network to test new protocol, new applications, and new services; in data center for saving energy, load balancer; in cloud for resource allocation and virtualization [4]. Google is using OpenFlow to connect its data centers [9]. However, there are many research challenges in SDN/OF [10]: how to design switches, software platform that offer more flexibility and high performance; how to create, to test new applications that leverages the programmability; how to adapt an existing network to SDN.

2.4 Problem Statement

The first issue, several protocols for routing in WMN have been proposed, such as AODV [12], OLSR [13]. The routing table must be deployed and updated quickly. For this reason, SDN/OF is a good candidate. However OF is designed to work with infrastructure-based network, for example, Data Center Networks. If we use OF in WMN, the following points must be considered [14]: Firstly, because of variations in wireless link (e.g. fading, interference) and nodes joining and leaving the network, the network topology changes at a higher frequency than in wired networks. This requires the ability to react on changes in the network. Secondly, the link capacity of WMN is smaller than the one of wired Ethernets. Therefore, the communication overhead between OF switches and the controller must be taken into account. Motivated by this challenge, our first target is to adapt OF to WMN. The first result is the basic brick to find an efficient way to transfer contents in WMN.

The second issue, the "*host to host*" communication model does not fit in such mobile and dynamic environment. CCN is a good approach to this issue. In CCN, nodes can perform content caching, make contents available close to the users (Section 2.3). Therefore, the load overall is reduced and the QoS is improved. In other way, CCN exploits all available faces of devices: 3G, Bluetooth, Wifi, Ethernet... and enables mobility. These features are perfectly adapted for WMN environment. Two problems of CCNx - the current implementation of CCN: Firstly, route by name is executed at the application layer. This implementation can not exploit the processing ability hardware devices at layer 2 and 3. Secondly, CCNx advocates to cache content on shortest path from content provider to user. In [15], they showed such on-path caching policy is not the best approach because it causes duplication, wasting cache space and reducing overall hit rate. They also propose and validate a off-path caching policy which we explain later. Our second target is to improve content delivery in WMN, this can be done in two steps: providing CCN functionality over WMN and then implementing off-path caching policy in [15] using OF capacity. In next chapter, we present the idea to adapt OF for WMN.

Chapter 3

OPENFLOW ADAPTATIONS FOR WIRELESS MESH NETWORK

As we introduced in section 2.4, there are two problems when using OF in WMN, high dynamic changes of topology and the overhead between the controller and OF switches. After studying OF specification [7] and OF-related work such as FlowVisor [16], DevoFlow [17], I propose three extensions should be done to adapt OF to WMN: Dynamic configuration, Backup Controller, Local Action.

3.1 Dynamic Configuration

As the definition, mesh routers and mesh clients in WMN can be mobile. That leads to a high dynamic chance of topology, high frequency of link up and down event. In [15] they propose having an agent on each OF switch and a dedicated server to handle the mobility then to exchange the information to the controller. Their solution has the flexible but it is not easy to manage. For our case, we use a simple way to provide mobility. The controller has a module to handle this problem; we call this Dynamic Configuration. A OF switch will send a message (OF_PORT_STATUS) to the controller when an adjacent link is down. Using that information, the controller can

detect the topology changes, selects the corresponding configuration then populates new flow table to OF switches. To support that purpose, I developed a parser for our XML files, which defines a set of flow entries will be inserted to each OF switch.

3.2 Backup Controller

The link between controller and OF switches may be down in a dynamic and mobile environment. In that case, the OF switch works as a normal switch or a black hole, which drops every packets (depending on the configuration). To ensure the service continuity and robustness, OF switches need one or many back up controllers.

FlowVisor is a special controller, it can be considered as a transparent proxy between OF switches and multiple controllers. FlowVisor can be used for supporting back up controller. We propose that on each OF switch, there is an instance of FlowVisor. Therefore, FlowVisor can manage to change to the backup controller when the main controller is overload the link is broken. In its current version, FlowVisor is implemented as a standalone program, running on a normal PC. In the future, we expect FlowVisor can be integrated on the OF switch. Indeed, if we add the FlowVisor between main controllers and OF switches the cons of this method is decreasing the performance (i.e. the number of messages which controller is able to process per second). I modified the source code of FlowVisor to handle the event "*lost connection with the controller*" and reconnect to one of predefined backup controllers.

3.3 Local Action

This is the extension to reducing the overhead between OF switch and controller. As the OF specification [7], there are several kinds of messages which OF switches and controller usually exchanges:

- ECHO request/reply: controller sends to check status of the OF switch
- READ_STATE request/reply: controller sends to get statistics from OF switches
- PACKET_IN: OF switch sends to the controller when a new flow starts.

- MODIFY_STATE: the controller manages states on the OF switches.
- PORT_STATUS: OF switch informs when the port status changes (down/up)

ECHO and READ_STATE could be reduced by increasing the exchanging interval, for example, normally in every 5s, the controller send STATUS message, if we increase to 10s, the number of READ_STATE messages reduce 2 times. The other messages depend on the network demand. In large scale network, it might be significant. DevoFlow [17] is targeted to reduce the overhead of OpenFlow protocol in Data Center. We see that it can be extended for reducing overhead in WMN. The main idea is to give the intelligence (Local Action) on the OF switch. This local intelligence can be considered as local controller, which helps OF switch to decide in some cases without invoking the main controller. Some actions can be used from [17]

- Rule cloning: The controller can insert a flow entry with IP wildcard mask (for example 10.0.0.1/24) to reduce the number of PACKET_IN/MODIFY_STATE messages. But if we do this, we lose the view of traffic because the statistics (number of bytes, number of packets, duration) will count everything that matches this rule. So the idea is that when new flow that matches the flow entry with wildcard, the local controller makes a new flow entry (no wildcard mask) that matches every field of new flows and then inserts to flow table of OF switch. Thus, the number of PACKET_IN and MODIFY_STATE message is reduced.
- Rapid Re-routing: if one of ports is down, the local controller inserts a new flow entry to redirect traffic to another port. Using this, the number of PORT_STATUS messages is reduced.
- Trigger and Reports: a counter and comparator are put on the local controller. When trigger condition is met, the local controller will generate a message to the main Controller to report the statistics. This can reduce the number of READ_STATE request.

Devoflow is closed-source so we do not implement Local Action. Afterward, we configure and set up a OF test bed (Fig. 3.1) to validate above ideas.

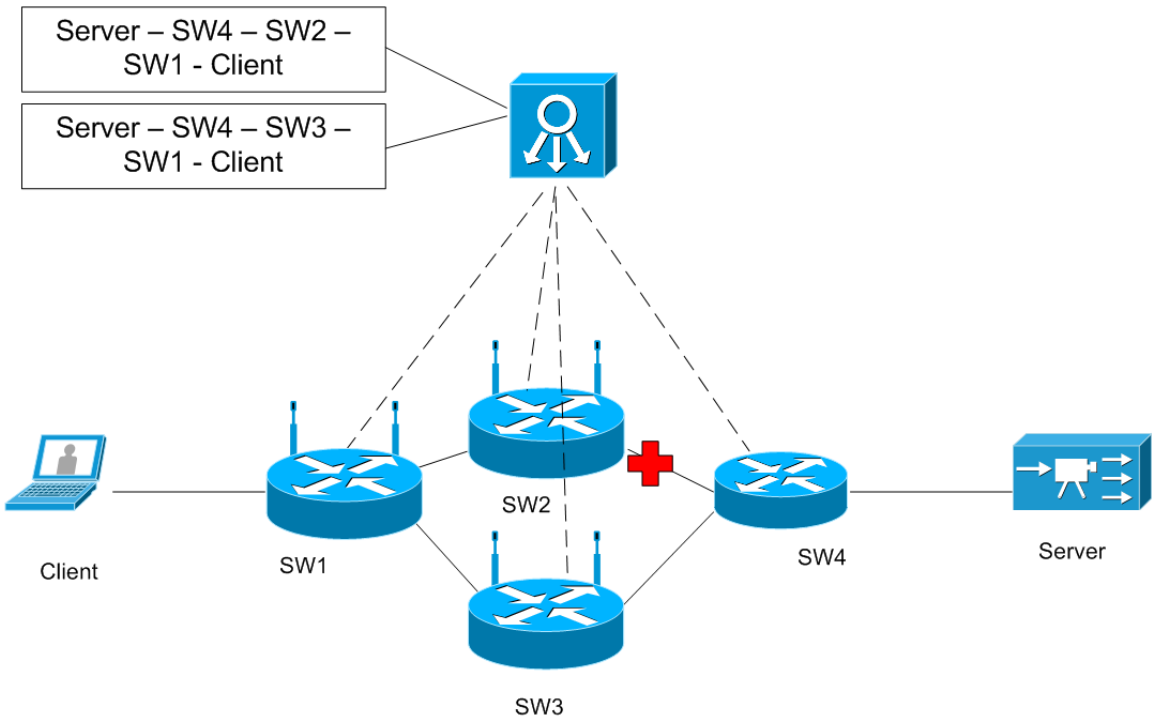

Figure 3.1: OF Test bed

3.4 Evaluation

Scenario

Assume that the Server streams a video to the Client. Normally, the controller sets the path for every packet: Server – SW4 – SW2 – SW1- Client. For some reasons, the link between SW4 and SW2 is down at t=5s. Actually, SW2 and SW4 notice this event and informs to the Controller. Afterward, the controller control OF switches to change to new path: Server – SW4 – SW3 – SW1 – Client (XML file 1). When link SW4 and SW2 is on at t=15s, the original path is recovered: Server – SW4 – SW2 – SW1 – Client (XML file 2)

Methodology

The link quality is measured by Iperf [18]

Result

Fig. 3.2 shows the characteristics of the link from client to server, the top graph is the bandwidth measurement in Mb/s from t=0 to t=30s. At 5s, the link 2-4 is down

Element	Description	Note
SW 1,2,3	TP Link WR1043ND 4 Ethernet 1Gbps, 1 wifi (54Mbps)	OpenWRT Firmware, OF 1.0 Compatible
SW4	Pronto 3290, 48 Ethernet (1Gbps)	Indigo Firmware, OF 1.0 Compatible
Controller	Dell Latitude E6400 Core 2 Duo P8400 2.4Ghz, Ram 3.4 GB, Fedora 13	Beacon Controller 1.0
Client	Dell Latitude E6400 Core 2 Duo P8400 2.4Ghz, Ram 3.4 GB, Fedora 13	VLC Client
Server	Dell Latitude E6500 Core 2 P8400 2.4GHz x 2 RAM 4GB, Ubuntu 12.04	VLC Server

Table 3.1: OF Testbed description

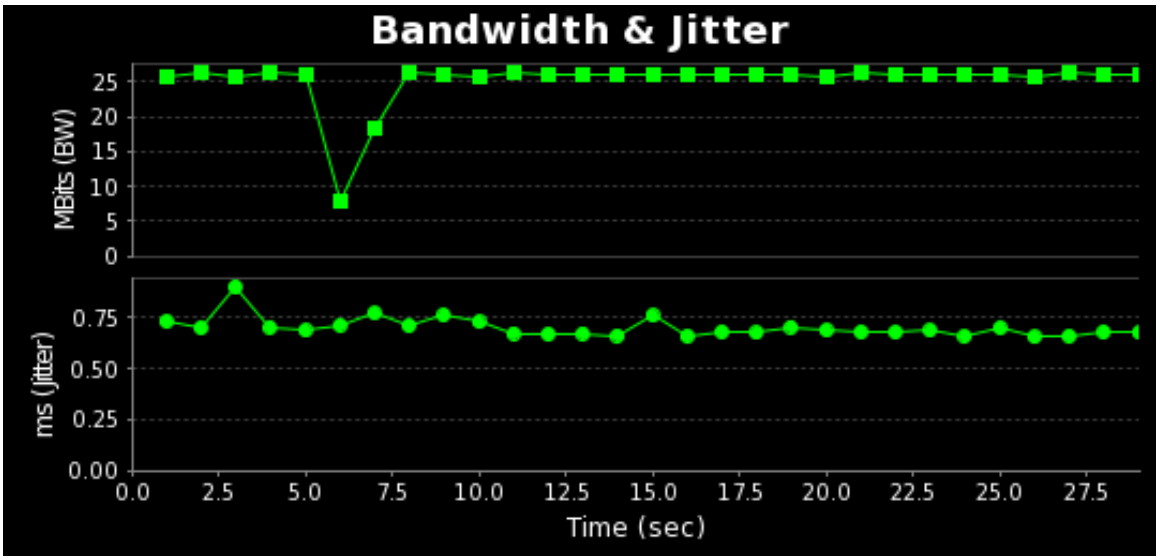

Figure 3.2: Dynamic Configuration Validation

and the client stop receiving packets and therefore, the bandwidth changed. Actually, the bandwidth dropped to zero but because of the granularity 1s, so we do not see it. The controller detects the failure, removes old configuration and inserts a new one. At t=8s, the client starts receiving packets. At t=15s, link 2-4 recovers. The controller switches to the old path and it is transparent with client, we do not see bandwidth changes. The bottom graph is jitter measurement. Jitter is latency variance, an important parameter for network support VoIP. The jitter is stable around 0.713 ms. The total packet loss is 3.3%, more than 2000 datagrams. Indeed, with such low packet loss and jitter, we still watch the video at full speed and good quality.

About backup controller, we start two controllers, one main and one backup. Normally, OF switches connect to main controller through FlowVisor. Then we stop the main controller. We observe on log screen that FlowVisor detects the failure and switches to the backup controller, transparency with OF switches as we expected. We also test the impact of FlowVisor using Cbench [4], a tool to measure the performance of controller. Without FlowVisor, the controller is able to process 48686 new flows/s (on average). With FlowVisor, the performance is 39776 new flows/s, or 81.7% of performance without FlowVisor.

Now we know OF can be used in WMN but with some adaptations, as the first target we mentioned in section 2.4. In next chapters, we discuss about second target: improving content delivery in WMN. To achieve this target, we need to solve two problems: how to provide CCN functionality over WMN (Chapter 4) and then how to implement off-path caching using OF capacity (Chapter 5)

Chapter 4

CONTENT CENTRIC NETWORKING FUNCTIONALITY OVER WIRELESS MESH NETWORK

In [19], to support CCN functionality, they propose to extend the OF switch with new flow tables and new matching rules. This idea has not been implemented. There is also an ongoing work using current OF switch to provide content-based communication, but it follows a different approach, CONET [20]. This work is not complete and need to be evaluated. We follow a different approach, integration of CCNx [11] (the implementation of CCN) with current generation of OF switches.

The main challenge of providing CCN functionality over OF network is the fact that current OF specification (version 1.1) can process only IP packets, and that it is not possible to analyze and process all the fields of the packet headers. Indeed, the OF switch can not process the header of CCN packet because it is stored in UDP payload. For the long term, OF could be extended to support processing CCNx header and caching capacity as the proposal of [19]. For the short term, which means implementing CCN with current OF switch, a solution mentioned in [21] consists to map CCN header into one or more IP header fields. When a field is used, it will lose

the original meaning but OF switch can process the CCN header information. In next section, we present our mapping technique extending from this idea.

4.1 Mapping technique

All fields which the OF switch performs switching can be used to map the content names: MAC source, MAC destination, IP source, IP dest, transportation port... if we assume that all devices in network supports OF functionality. A common mapping technique is using hash function. When mapping a large space to a small one, there is a risk of collisions: i.e. 2 or more different names have a same mapping value. For example, if there are 32 bits for hashed values and 10^{10}content names, it means that on average 2.33 content names have the same hashed value ($10^{10}/2^{32} = 2.33$). A collision is likely to cause wrong incorrect switching because OF switches can not distinguish 2 content names and switches it to wrong destination. We propose a mapping technique based on the hash function and the collisions counter (CC). The content names are projected on smaller space representing by fields which OF performs switching. The space for mapping will split into two parts: one for the hashed value and one for the CC. The value of CC increases by one when having a collision of this hash value.

Without CC, when there is a collision, we do not have any information about it. Using CC does not reduce the probability of having collision, but a method to control the collision. If we use more bits for CC, we can handle more collisions but it causes the higher probability of having collisions because the space for hashed value is smaller. We look for the expected number of times before having an unhandled collision $E[x]$ in order to find optimal number of bits of the CC field with these notations: a is number of bits for hashed value; b is number of bits for collision counter (CC); x is the time (in terms of packets) to have an unhandled collision and an uniform hash function. For simplicity, we name two new variables, number of hashed value $n = 2^a$ and maximum size of a collision counter $m = 2^b$

$x > k$ if no hashed value is duplicated more than m times (the CC is not full) in first k value. For example, $m = 0$, no hashed value can be duplicated in first k value. The first value has n choices; the second value has $(n - 1)$ choices because it must

differ with first value and so on. The total number of cases is n^{k+1} and the number of cases results $x > k$: $n(n-1)(n-2)...(n-k)$. Then we have probability of having $x > k$

$p(x > k) = \frac{n(n-1)...(n-k)}{n^{k+1}}$

$m = 1$, each hashed value can be duplicated one time in first k value. The first value has n choices; the second value also has n choices, the third value has $(n-1)$ choices (not n because there is a constraint one hashed value is duplicated maximum one time) and so on. The total number of cases is $n^{k+1}$and the number of cases results $x > k$: $n^2(n-1)^2(n-2)^2...(n-t)^r$, with t is the quotient of $(k+1)$ for 2 and r is the remainder. Then we have probability of having $x > k$

$p(x > k) = \frac{n^2(n-1)^2...(n-t)^r}{n^{k+1}}; k+1 = 2t+r; 0 \leq r \leq m-1$

Generally, $m > 1$, each hashed value can be duplicated m times in first k value. The first value has n choices; the second value also has n choices ... the $(m+1)^{th}$ value has $(n-1)$choices (not n because there is a constraint one hashed value is duplicated maximum m times) and so on. The total number of cases is n^{k+1} and the number of cases results $x > k$: $n^{m+1}(n-1)^{m+1}...(n-t)^r$, with t is the quotient of $(k+1)$ for $m+1$ and r is the remainder. Then we have probability of having $x > k$

$p(x > k) = \frac{n^{m+1}(n-1)^{m+1}...(n-t)^r}{n^{k+1}}; k+1 = (m+1)t+r; 0 \leq r \leq m-1$

$p(x = k) = p(x > k-1) - p(x > k)$

Assume that $k = (1+m)t + r \Rightarrow k+1 = (1+m)t + r; t, r \epsilon \mathbb{N}$

$p(x = k) = \frac{n^{m+1}(n-1)^{m+1}...(n-t)^r}{n^k} - \frac{n^{m+1}(n-1)^{m+1}...(n-t)^{r+1}}{n^{k+1}} = \frac{n^{m+1}(n-1)^{m+1}...(n-t)^r}{n^{k+1}}(n - (n-t)) = \frac{n^{m+1}(n-1)^{m+1}...(n-t)^r t}{n^{k+1}}$

Maximum value of x is $mn+1$ when all counters are full. Minimum value of x is $m+1$ when one counter for a hashed value is full at beginning. Thus, we deduce a formula of $E[x]$ by sum.

$E[x] = \sum_{m+1}^{mn+1} k \times p(x = k) = \sum_{m+1}^{mn+1} \frac{n^{m+1}(n-1)^{m+1}...(n-t)^r \times t \times k}{n^{k+1}}$

In our problem$a + b$ is a constant and we determine the gain of collision counter in term of $E[x]$. Fig. 4.1 shows the evolution of $E[x]$ for $a + b = 24$. $E[x]$ increases with b faster than with a. The result showed that the bigger size of CC, the higher value of $E[x]$. But it does not mean that we should set b to maximum value because the optimal size for CC also depends on the traffic distribution. For example, if 90%

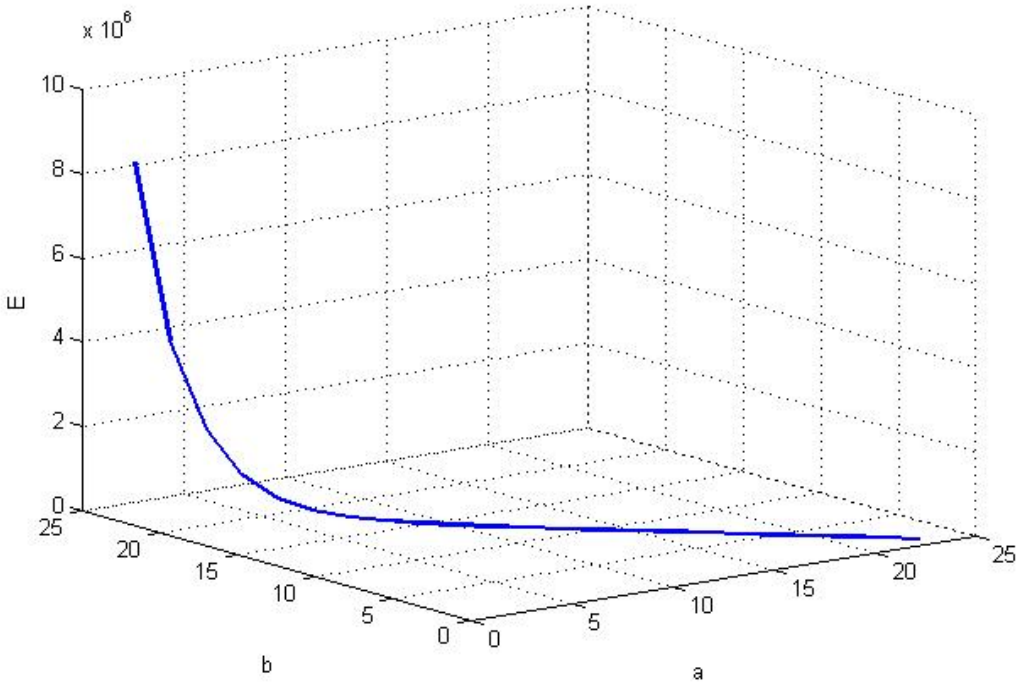

Figure 4.1: Expected time before having collision

of the flows last less than 1 second, it is not necessary to avoid critical collisions for 10 second, 1 or 2 seconds is enough.

Now that we have the mapping technique, we focus how to provide CCN functionality over OF Enabled WMN. We see that there are three approaches to combine CCNx – the implementation of CCN and OF switches. The first approach is CCNx modification to map CCN header into outgoing packets. The second one is OF modification to support processing of CCNx header. The third approach is building a wrapper between OF and CCNx. We follow the wrapper approach as it does not require to modify neither the CCN implementation (i.e., CCNx), nor the OF implementation. The wrapper is adapted easily to new releases of CCN and OF implementation.

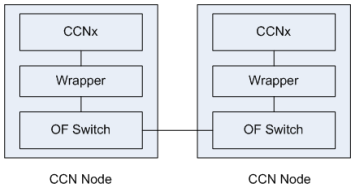

Figure 4.2: CCN Node

4.2 Wrapper

Our architecture which provides CCN functionality is showed in Fig. 4.2. CCNx and the Wrapper runs on a normal PC, connect to a dedicated port of OF switch. CCNx daemon is responsible for content caching, and tracking the content requests to return the content after (e.g. maintains Content Store, Pending Interest Table). Because the content is already hashed into IP fields (using mapping technique), OF switch can forward the interest/data packets. (e.g. maintains Forwarding Interest Base). Firstly, the face(e.g port) of OF Switch is mapped to the face of CCNx deamon, so we use the ToS (Type Of Service field) which can be modified by OF Switches. On Interest arrival, OF Switch sets ToS value to indicate the incoming face 1,2,3... After that, Wrapper extracts this information and forward to right face of CCNx. In our testbed, we use 32 bits of IP source address for hashed value of content names; which means 2^{32} contents can be tracked at the same time; is reasonable for a testbed.

In our design, face W is a special face between Wrapper and CCNx. Data packets from OF Switch only come in face W and the interest packet from CCNx only come out from face W. The behavior of the Wrapper is showed in Fig. 4.3 and 4.4.

The wrapper waits for CCN packets from OF switch and from CCNx. If it is Interest packet from OF switch, the wrapper forwards it to corresponding face of CCNx. If it is Data from OF switch, the wrapper forwards it to face W (Fig. 4.3. If it is Data packet fro CCNx, the Wrapper set corresponding ToS field. Then both Interest and Data from CCNx is decoded and the name is hashed to IP source field (32 bit) of packets. Finally, the wrapper sends the packets to OF switches (Fig.4.4) Wrapper is implemented in C. We use libccn for encoding and decoding CCN message, raw socket to receive and to send the packet and the hash function djb2 [22]. In section

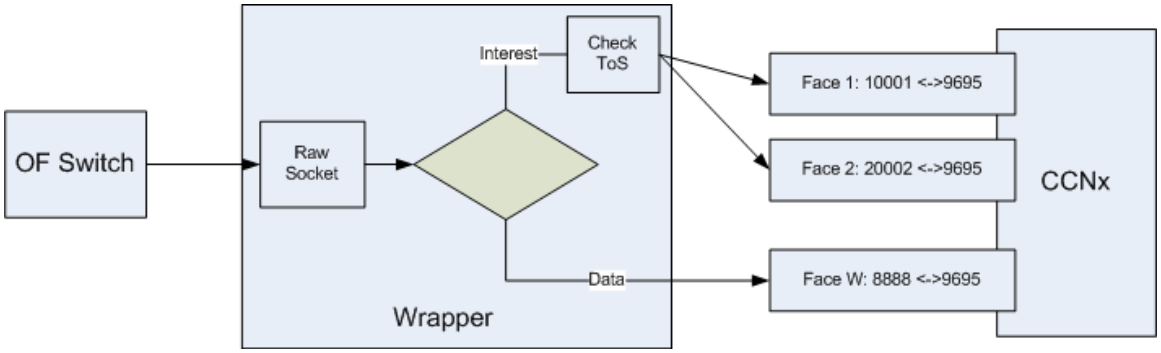

Figure 4.3: Behavior of the Wrapper 1

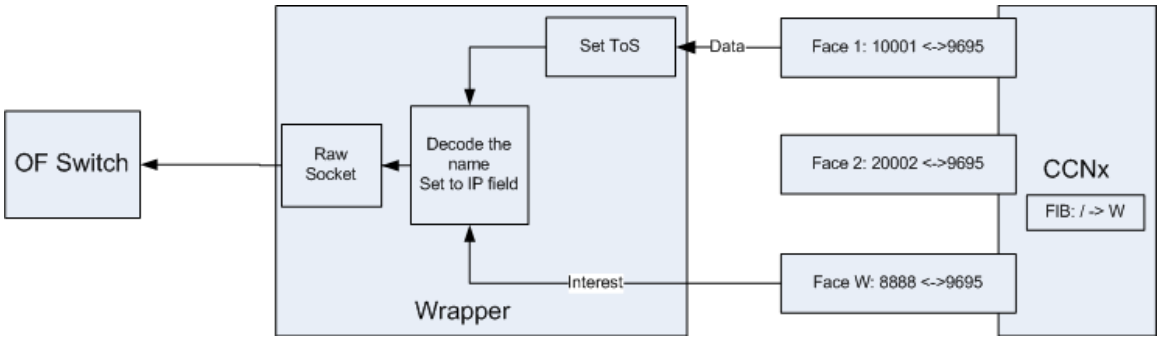

Figure 4.4: Behavior of the Wrapper 2

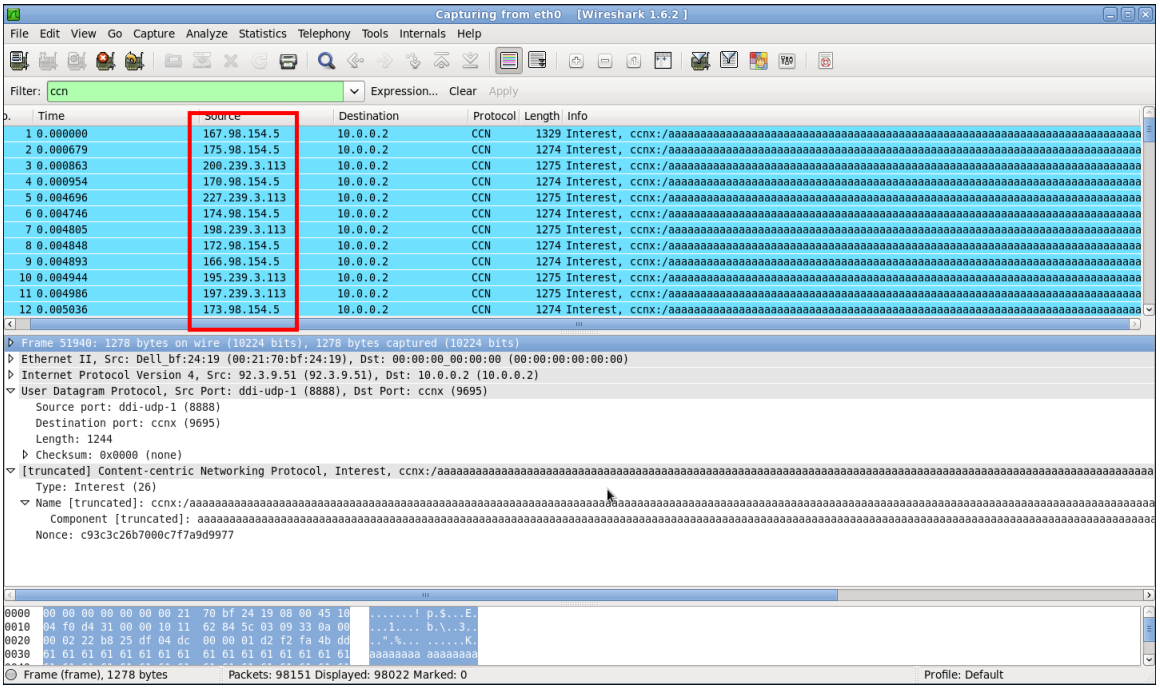

Figure 4.5: Captured packets from Wrapper

4.3, we validate the wrapper.

4.3 Evaluation

We check the compliance of the Wrapper with CCN protocol by inspecting packets on Wireshark [23]. Fig. 4.5 shows CCN packets which the Wrapper generates. The content name of CCN packet is hashed into source IP field of IP packets (in the rectangle).

After that, we evaluate the impact of Wrapper to performance of the system in term of packets per second (pps) with 3 cases:

- OF Case: Using only the OF switch to forward Interests (Fig. 4.6)
- CCN Case: Using OF switch + CCNx daemon to forward Interests (Fig. 4.7)
- Wrapper Case: Using OF switch + Wrapper + CCNx daemon to forward Interests (Fig. 4.8)

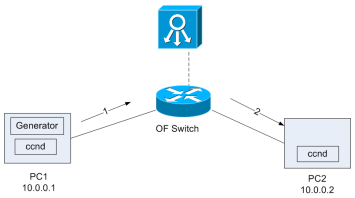

Figure 4.6: OF Case

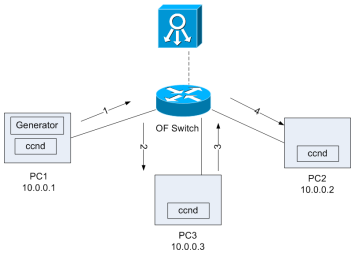

Figure 4.7: CCN Case

Methodology

- PC1 runs a program to generate difference name (prefix 1000 characters + increasing number) with difference rates using our active loop. We set the long name (to increase the size of payload of CCN packets) in order to reducing loss rate.
- PC1 uses command **netstat –u –s; sleep 1; netstat –u –s;** and computes the sending rate from number of sending packet.
- PC2 uses Wireshark to capture and filter CCN packet and to get the receiving rate (Menu Statistics, Summary, Avg. packets/s)

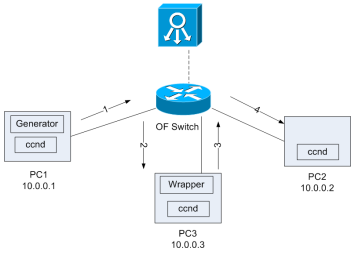

Figure 4.8: Wrapper Case

Element	Description	Note
OF Switch	Pronto 3290, 48 Ethernet (1Gbps)	Indigo Firmware, OF 1.0 Compatible
Controller	Dell Latitude E6400 Core 2 Duo P8400 2.4Ghz, Ram 3.4 GB, Fedora 13	Beacon Controller 1.0
PC1, PC2, PC3	Dell Latitude E6500 Core 2 P8400 2.4GHz x 2 RAM 4GB Ubuntu 12.04	

Table 4.1: Wrapper testbed description

Result

The first result (Fig. 4.9) show the poor performance of Wrapper, around 1000 pps. We finally found the problem, not at the wrapper but on the controller. The reason is that the OF switch considers the packets from the wrapper as new flows and send it to the controller. This is the limitation of the OF switch, our OF switch is a cheap one and it supports only 2000 flow entries at the same time. The impact of the controller and OF switch need to study more.

We inserts a flow entries with wildcard for IP source field, i.e. every packets with different IP source is take into account as a flow, it also means the statistics is disabled and we got a better result (Fig. 4.10) At the low value of PPS in, there is no difference in PPS out for all cases. At high PPS in, OF case obtains the best PPS out because the CCN packet is forwarded at switch level. In the other hand, the wrapper case's PPS out is smaller than CCN case. This behavior was expected because the Wrapper has to process the packet coming from both OF and the Wrapper. During the experiments, we have noticed that an important packet loss (20%) while handing Interest packet with short length name content at high sending speed. The short name causes high packet overhead that leads to high percent of packet loss. In our experiment, the size of content name is bigger than 1000 (1000 characters + numbering). The lost drops to 5%. Now we have a solution to provider CCN functionality over OF, we focus on improving content delivery.

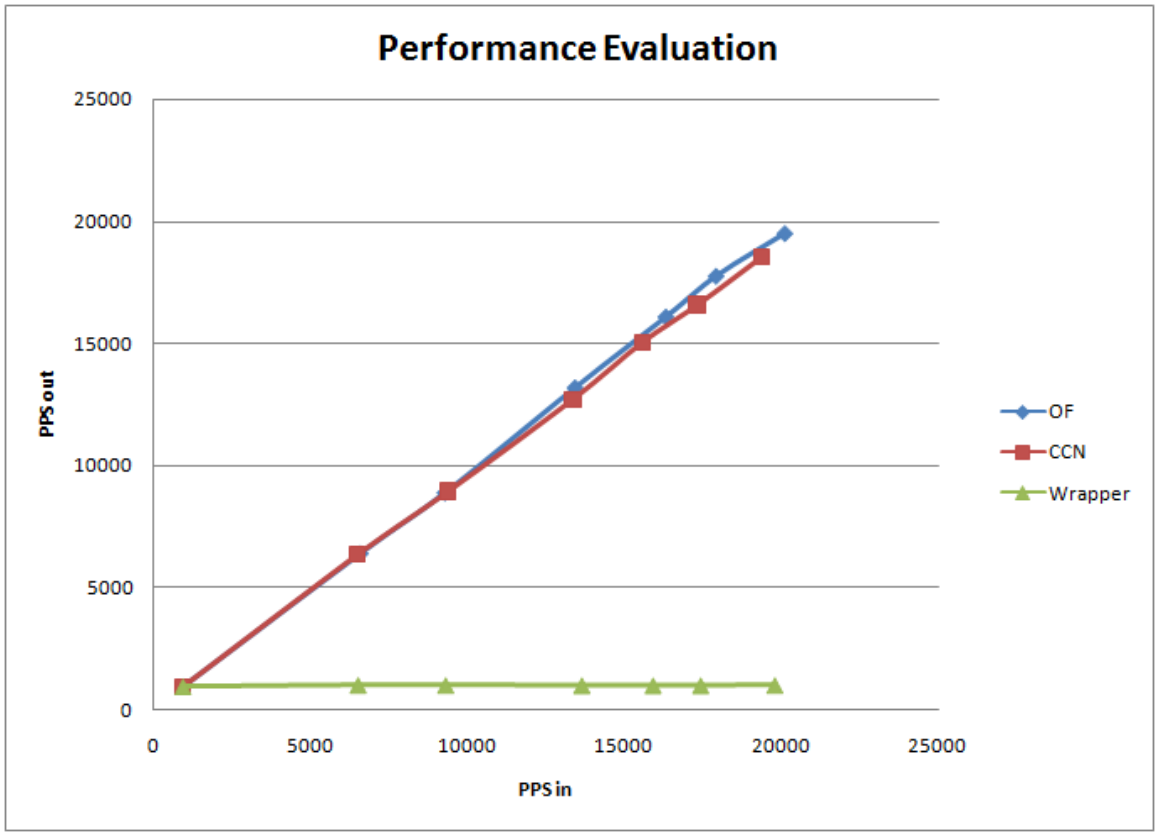

Figure 4.9: Performance Evaluation 1

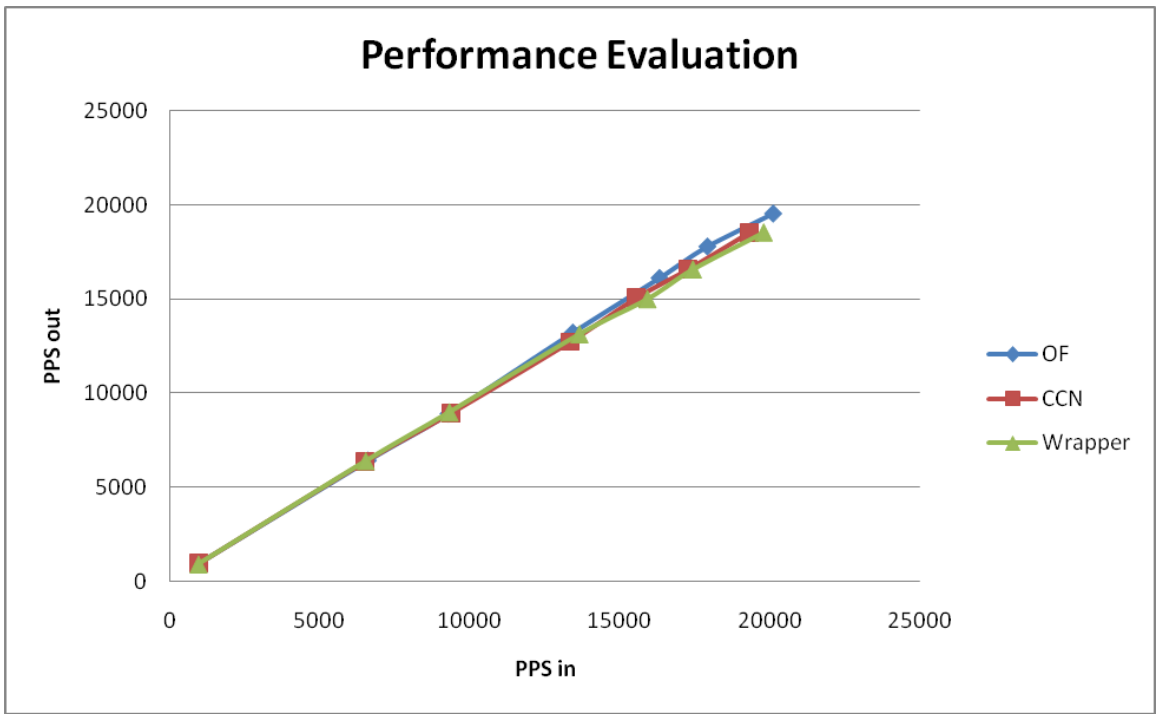

Figure 4.10: Performance Evaluation 2

Chapter 5

OFF-PATH CACHING WITH DEFLECTION

Caching is one of area research of CCN. CCN advocates to cache contents on shortest path from content provider to user (i.e. on-path caching). This on-path caching policy makes contents closing to users and the request delay is minimized. However in [15], they showed that on-path caching is not optimal in term of bandwidth on peer links and wastes cache spaces because of the duplication of contents. To tackle this, an efficient off-path caching policy is proposed with 3 main ideas: Caching the most popular contents, deciding where to cache by solving an optimization problem to minimize the sum of delays, and deflecting Interests to the most convenient provider (i.e. deflection technique). They observed that delay of the off-path caching with deflection is compensated with the gain in term of bandwidth peer link and space.

We extend this work, implementing and validating this cache policy on real device. To implement this policy, we need a method to estimate the popularity, to track the most popular contents which might not be easily to obtain in traditional network and a mechanism to deflect the content requests to the right cache; this mechanism must be dynamic and must adapt to network traffic. There are two good reasons why choosing OF for that purpose. Firstly, we benefit from a centralized controller; the controller has the global view of network status. It can compute and decide which content should be cache and where to cache. Secondly, the controller can populate

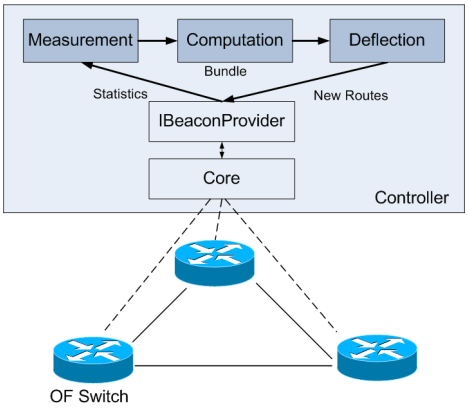

Figure 5.1: Off-path caching architecture

quickly new routing tables to OF Switches, and content requests can be routed to the content provider node. Therefore, deflection can be done. In this section, we present the architecture to implement off-path caching policy mentioned in [15] on the beacon controller.

5.1 Architecture

We need to develop 3 bundles on the controller (Fig. 5.1), the Measurement bundle to measure the popularity of contents, the Computation bundle uses the input of the Measurement bundle to solve the optimization problem in [15], the Deflection bundle uses the result of Computation bundle to insert new routing tables for OF switches in order to deflect the interest to right cache.

Measurement bundle gets statistics of OF switches periodically. Unlike regular switches, an OF switch maintains a set of counter for each table, flow, queue. It keeps track of active flows in the network and update per flow counters. Counters show number of packets, number of bytes and flow duration. This mechanism enables direct and precise flow measurements without packet sampling [24]. Because we already hashed the name in the header field of CCN packet (mapping technique), OF switches can monitor the statistics for each Interest with content name. Using that

information, we know how many requests have been sent for this content, in this manner; we are able to detect the most popular contents.

Computation bundle used the input data from Measurement bundle keeps a linear object function to minimize the sum of the delays and constraints mentioning in [15] $\sum_{c \in C} \sum_{e \in E} \lambda_{c,e} \sum_{r \in R} A_{r,c} \times d_{e,r}$

$C$is the set of deflect contents, E is the set of border OF switches, R is the set of OF switches with caching capacity. N is network caching capacity. For the simplicity, we assume the contents have the same size, thus up to N content can be cached.

$\lambda_{c,e}$ is the request rate for content c made by border OF switch e

$d_{e,r}$ is the round trip delay between two OF switch e and r

$A$is a matrix with value of $A_{r,c}$is 1 if the content c is assigned to OF switch r, otherwise 0.

Each popular content is cached at exactly one OF switch $\sum_{r \in R} A_{r,c} = 1; A_{r,c} \in \{0, 1\}; \forall c \in C$

At each OF switch with caching capacity, contents are not exceed the memory of OF switch r $\sum_{r \in R} A_{r,c} \leq memory_r; \forall r \in R$

Matrix A can be solved using Linear Programming toolboxes. After solving matrix A, we know where the content should be cached.

Deflection means redirecting Interests to the node which has the required content. The Deflection bundle computes the flow tables for each OF switch using the result of the Computation bundle. After that, it populates flow tables for each OF switches. For example, in a network there are three OF switches. Assume that content A will be cached at OF switch 1, B at OF switch 2, C at OF switch 3. Then Deflection bundle built the flow table and populates it to OF switch 1 (Fig. 5.2) and others.

At the moment, we have implemented and validated Measurement module.

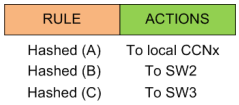

Figure 5.2: Flow table of the OF switch 1

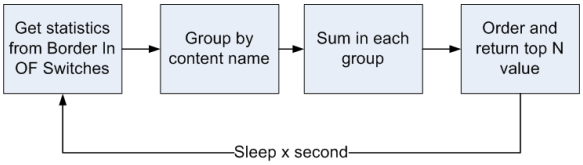

Figure 5.3: Algorithm to detect the most popular contents

5.2 Evaluation

We implemented the Measurement and we ran a test to verify the correctness on real OF devices. The OF switch we use in testbed Pronto 3290 supports up to 2000 flow entries. It is higher bound of number of content names can be tracked. Actually, it strongly depends on the network topology and traffic demand. In the current OF switch, we use pull method (passive), but it is better if the OF switches support pushing statistics periodically to controller (active) because the overhead between the controller and OF switches is reduced.

We implemented an algorithm showing in Fig. 5.3 on the Measurement bundle. Periodically, the statistics will be pulled from border-in OF Switches (e.g. the OF switch which receives the requests directly from user). Then, the controller groups the statistics (number of packets) by content name and sum up in each group. Finally, the controller sorts and returns the top N most request contents, with N is the network caching capacity (i.e. up to N contents can be cached). The complexity of this algorithm is $\Theta(n)$ with n corresponding to the number of all active contents in the network. x is a configurable parameter. if x is small; we have more granularity but more overhead between OF switches and controller. So x should be adapted to the characteristics of network.

Configuration

Methodology

Client generates content requests with names hashed in fields following zipf's

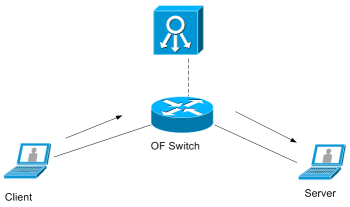

Figure 5.4: Measurement Test bed

Element	Description	Note
OF Switch	Pronto 3290, 48 Ethernet (1Gbps)	Indigo Firmware, OF 1.0 Compatible
Controller	Dell Latitude E6400 Core 2 Duo P8400 2.4Ghz, Ram 3.4 GB, Fedora 13	Beacon Controller 1.0
Client, Server	Dell Latitude E6500 Core 2 P8400 2.4GHz x 2 RAM 4GB Ubuntu 12.04	

Table 5.1: Measurement testbed description

law. We use zipf's law because this law is very common for content requests [25]. Probability of content rank k is $f(k; s, N) = \frac{1/k^s}{\sum_{n=1}^{N} 1/n^s}$ with N is number of contents; k is rank; s is exponent characterizing the distribution

Result

In Fig. 5.5, the x axis is the content rank (in log scale). The y axis is the probability of request of this content (in log scale). For example, content rank 1 has the probability request 0.133123. The blue curve is theoretical (i.e. source generation). The red curve is the measurement's result on the controller. We observe that the red curve is almost perfectly match with the theoretical one. It means that Measurement bundle measures correctly the popularity of contents.

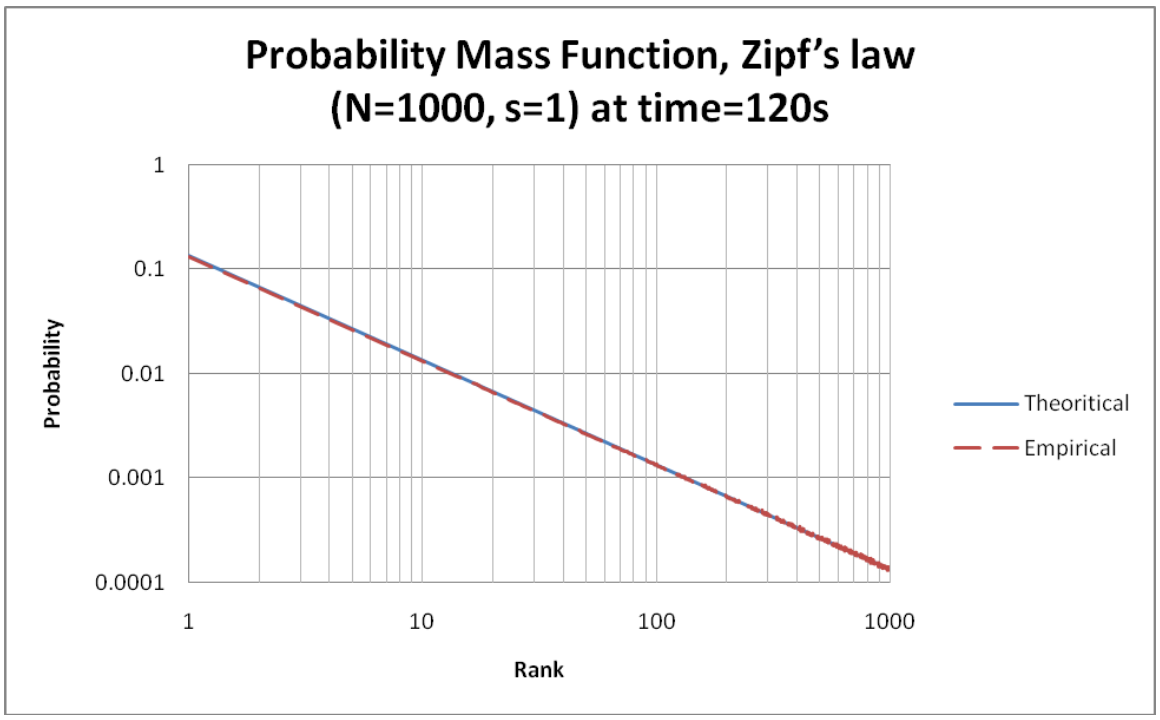

Figure 5.5: Validation of the measurement module

Chapter 6

CONCLUSION

Wireless Mesh Network is a key technology of the future, supports high mobility and allows deploying new services, new applications such as vehicular communication, health care delivery. Nowadays, contents (video, music, webpage ...) are ubiquitous thus content delivery is an important service. Most promising architecture that provides that services is Content Centric Networking. To deploy services over Wireless Mesh Network, Software Defined Networking is a good approach because of the programmability. However, existing Software Defined Networking implementations have targeted to infrastructure-based network. The motivation of work is to explore how to use OpenFlow [4]- the leading Software Defined Networking implementation in Wireless Mesh Network. Afterward, we use OpenFlow to improve content delivery services.

To summarize this work, Software Defined Networking can be used for Wireless Mesh Network, with some adaptations I proposed in chapter 3: **Dynamic Configuration** - a mechanism to react with high topology changes, **Backup Controller** - more than one controller for an OpenFlow Switch and **Local Action** - give more intelligence on OpenFlow Switches. In chapter 4, I use a new approach to provide content delivery service over Wireless Mesh Network, using a mapping technique based on hash function and the **Collision counter**, and the **wrapper**. In chapter 5, I propose an architecture which allows to implement the off-path caching with deflection, in order to improve content delivery service. The cons of this work is the fact

that Computation and Deflection bundle in chapter 5 have not been implemented and validated.

This work can be extended by taking into account new version of OpenFlow specification, interface design between the controller and CCNx in order to support cache management. Further discussion, the impact of the controller and the OpenFlow switch to the result (Section 4.3) need to be studied more.

Bibliography

[1] "Software defined networking," https://www.opennetworking.org/.

[2] I. Akyildiz and al., *A survey on wireless mesh networks.* Communications Magazine, 2009.

[3] V. Jacobson and al., "Networking named content," *ACM CoNEXT*, 2009.

[4] "Openflow," www.openflow.org.

[5] www.cisco.com, "Cisco."

[6] "Juniper," www.juniper.net/.

[7] "Openflow specification version 1.1," 2011.

[8] "Beacon controller - https://openflow.stanford.edu/display/beacon/home."

[9] "Google describes its openflow network," http://www.eetimes.com/electronics-news/4371179/Google-describes-its-OpenFlow-network, 2012.

[10] "Hot sdn topic - http://conferences.sigcomm.org/sigcomm/2012/hotsdn.php."

[11] "Ccnx," www.ccnx.net.

[12] "Aodv," http://www.ietf.org/rfc/rfc3561.txt.

[13] "Olsr," http://www.ietf.org/rfc/rfc3626.txt.

[14] P. Dely and al., "Openflow for wireless mesh networks," *Computer Communications and Networks*, 2011.

[15] D. Saucez, A. Kalla, C. Barakat, and T. Turletti, "Minimizing bandwidth on peering links with deflection in named data networking," *under submission*, 2012.

[16] R. Sherwood and al., "Flowvisor: A network virtualization layer," Tech. Rep., 2009.

[17] J. C. Mogul and al, "Devoflow: cost-effective flow management for high performance enterprise networks," *SIGCOMM*, 2010.

[18] "Iperf," http://sourceforge.net/projects/iperf/.

[19] J. Sub, "Of-ccn: Ccn over openflow," *NDN Workshop*, 2011.

[20] A. Detti and al., "Conet: A content centric inter-networking architecture," *ACM SIGCOMM workshop*, 2011.

[21] N. Blefari-Melazzi and al., "An of based testbed for information centric networking," *Future Network & Mobile Summit*, 2012.

[22] "Djb2," http://www.cse.yorku.ca/ oz/hash.html.

[23] "Wireshark," www.wireshark.org.

[24] A. Tootoonchian and al., "Opentm: Traffic matrix estimator for openflow networks," *PAM*, 2010.

[25] "Zipf law," http://linkage.rockefeller.edu/wli/zipf/.