OPENFLOW BASED VIDEO-ON-DEMAND CACHING SOLUTION

Openflow based video-on-demand caching solution

By

Adebakin Funmilola Halimat

Department of Computer

University of Alberta

A project submitted to the faculty of the University of Alberta in partial fulfilment of the requirements for the degree of

Master of Science in internetworking

Edmonton, Alberta

2016

Approved by

Academic Advisor

ABSTRACT

The efficiency of video content delivery is and would continually be a pressing issue as regards the growth in modern day internet except better measures are taken. For the past decade, the internet has taking measures in supporting the distribution of video content. Globally, the internet traffic was a little above 50% in the year 2012, and a prediction was made that the traffic would increase up to 69% in the year 2017. Traditionally, the storage of web content s are forwarded to the web or cache servers but in recent times, better measure are taken by caching the video inside the network and using openflow to coordinate caching decisions in the network.

Video on Demand service makes it possible for individual to gain access to video contents at a time after the initial broadcast was made available. On the other hand, when the delivery of huge content is made through unicast flow, ignoring the fact that similar transmission were made earlier, the availability of network is of great challenge. For these reasons, the Software Defined Networking (SDN) and its protocol called the OpenFlow were introduced in other to reduce the impacts made by the constant repeat in the delivery of similar content previously requested over the network and at the same time, reducing the cost of delivery.

To solve the problem of the delivery and the distribution of video content over the network, an architecture that is capable of making optimum path selection at the same time utilizing the global network view was introduced and this is called an OpenFlow Based Video on Demand Caching Solution.

OpenFlow is the only non-proprietary, general purpose protocol for programming the forwarding plane of SDN switches. Openflow defines the communications protocol between the control and data planes. It is based on an Ethernet switch, with an internal flow table, and a standardized interface to add and remove flow entries. In this project we take advantage of openflow and propose an efficient openflow based video on demand caching solution. This proposed openflow based solution dynamically reconstructs the previous methods of caching a video content to the web or cache server by doing an in network caching and bring these content to the closest nodes which enables for efficient delivery to the users.

We implemented a resource efficient Video on demand caching solution on an openflow controller. This solution contains video on demand sever, openflow switch, openflow controller (Floodlight), openflow video on demand caching (opencache) controller, database (MongoDB), a switch and openflow video on demand caching (opencache) node. To authenticate the proposed video on demand caching solution, we constructed a virtual network topology using the mininet network emulator. We also used the Floodlight controller and deployed the proposed solution to it. This experiment shows that using the openflow protocol with the multicast scheme to protect network resource, giving the users maximum quality of experience.

Table of Contents

1. Introduction 1
  1.1 Background 1
  1.2 Description of Problem 4
  1.3 Project Goals and Approach 4
  1.4 Project Outline 5
2. RELATED WORKS 6
  2.1 Peer to Peer Networking 6
  2.2 Multicast 6
  2.3 Cache and Proxy 7
  2.4 Content Delivery Network 8
  2.5 Software Defined Networking and OpenFlow 9
3. OPENFLOW BASED VIDEO ON DEMAND CACHING NETWORK ARCHITECTURE 16
  3.1 Design 16
4. OpenFlow based Video on demand Caching Core Entities 20
  4.1 OpenCache Controller 20
    4.1.1 Handling Request For Content of Interest 20
    4.1.2 Implementing Caching 22
    4.1.3 Managing Caching Resources 22
  4.2 OpenCache Node 23
    4.2.1 Cache-miss Scenario 24
    4.2.2 Cache-hit Scenario 25
    4.2.3 Non-Caching Scenario 26
5. SIMULATIONS AND RESULTS 28
  5.1 CONTROLLER 28
  5.2 NETWORK EMULATOR 29
  5.3 NETWORK SETUP 29
  5.4 NETWORK TOPOLOGY 29
  5.5 EXPERIMENTS AND RESULTS 32
    5.5.1 Scenario 1- screenshot of instance to declare caching 32
    5.5.2 Scenario 2- Screenshot of Cache-hit 32
    5.5.3 Scenario 2- Screenshot of Cache-miss 33

5.5.4 Scenario 2- Screenshot of no-caching 34
6. CONCLUSION AND FUTURE WORK 35
6.1 CONCLUSION 35
6.2 FUTURE WORK 36
REFERENCES 37

TABLE OF FIGURES

Figure 1 Global consumer internet traffic in Petabytes per month. SOURCE: [8] 2
Figure 2 Mobile consumer internet traffic in Exabyte per month. SOURCE: [8] 2
Figure 3 - The SDN System Architecture SOURCE: ONF 10
Figure 4 - OpenFlow switch components. Source: [20] 12
Figure 5 - Simplified Simulation Topology 18
Figure 6 - Declaring Content of Interest as Cacheable 21
Figure 7 - Removal of Cached Content 23
Figure 8 - A Cache-miss Scenario 25
Figure 9 - A Cache-hit Scenario 26
Figure 10 - Non-Caching Scenario 27
Figure 11 - Detailed Simulation Topology 30
Figure 12 - Activity Diagram For Network Administrator 31
Figure 13 - Activity Diagram 31
Figure 14 - screenshot of instance to declare caching 32
Figure 15 - Screenshot of Cache-hit 32
Figure 16 - Screenshot of Cache-miss (i) 33
Figure 17 - Screenshot of Cache-miss (ii) 33
Figure 18 - Screenshot of no-caching (i) 34
Figure 19 - Screenshot of no-caching (ii) 34

LIST OF TABLE

Table 1 - Group Table Entries for OpenFlow 1.1 and later versions 14

1. Introduction

This chapter provides a brief introduction to Video on Demand, Software defined network (SDN), and the OpenFlow protocol. It also presents the motivation and problem statement. Then it presents research goals and its proposed solution.

1.1 Background

The popularity of video streaming in recent have experienced very rapid growth whether live streaming or on-demand via a wired or mobile end device. Figure 1 and 2 shows the global and mobile consumer internet traffic respectively [4]. There has been a persistent rate in the increase of the demand for network bandwidth. This is as a result of constant arrival of “killer applications” e.g. peer-to-peer file sharing application, video on demand services, as well as the increase in the popularity of network-hungry end devices [5]. According to Cisco Visual Networking Index (VNI) Global Mobile Data Traffic Forecast, the global internet video traffic accounted for 57% of all consumer internet traffic and predictions have it that in 2017, internet traffic would increase up to 69% [4, 6, 7]. For mobile devices, similar predictions have been made that video in the year 2018, would have an overall mobile video traffic of approximately 69% [5, 8]. Video-on-demand has gained so much popularity that there is continual increase in traffic, consumer video-on-demand (VoD) traffic has been predicted to nearly double by 2020. The amount of VoD traffic in 2020 will be equivalent to 7.2 billion DVDs per month [6]. High Definition (HD) traffic has by far surpassed the Standard Definition (SD) traffic, and this makes HD the standard video quality level for video consumers. It is undeniable that high quality online streaming has become an essential to the everyday life of a consumer [4].

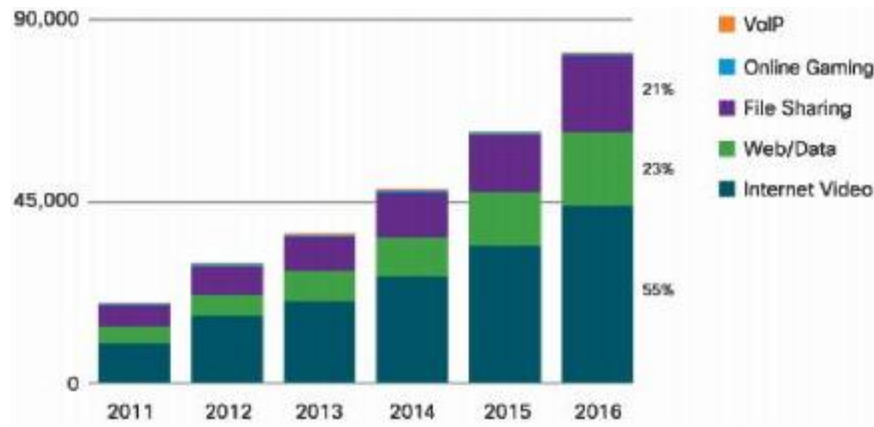

Figure 1 Global consumer internet traffic in Petabytes per month. SOURCE: [8]

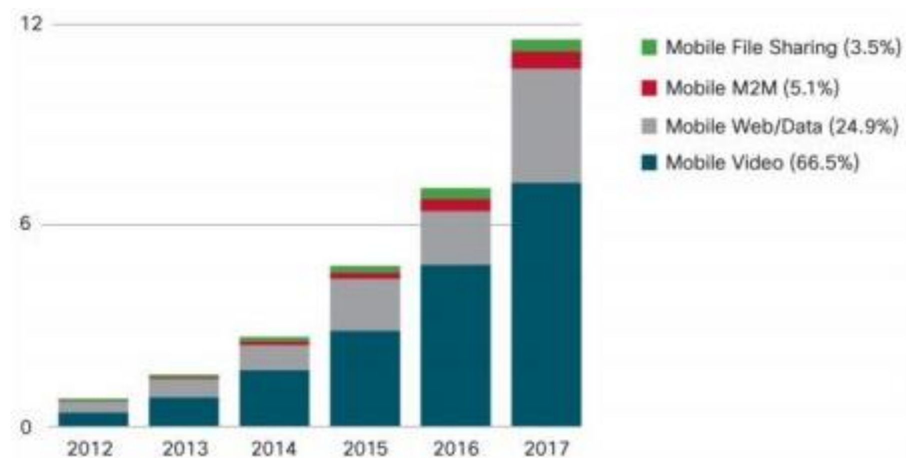

Figure 2 Mobile consumer internet traffic in Exabyte per month. SOURCE: [8]

With VoD services, users are allowed to select and watch video content when they choose rather than having to watch it at the time of its initial broadcast [9]. The VoD service is becoming a dominant service in the telecommunication market and with the explosive increase in demand and popularity in the quality of video content, a disturbing challenge to the underlying network infrastructure is becoming evident. Network has to transfer an enormous amount of data to the end user and this has to be done in a little time frame. The quality of video content will continue to increase and this would continually be the trend as

we move from HD to Ultra HD and 3D video as the thirst of more quality video can never at least for now be quenched.

The distribution of live content which has to do with the streaming of live data to users has to be done simultaneously, which enables it to exploit multicast mechanism that provides such simultaneous delivery [4]. For VoD requests, the reverse is the case. VoD requests must be handled individually and this leads to an independent media flow in the distribution network for each user request. For such unicast content delivery paradigm, huge amount of identical media objects (in the order of gigabytes for each HD film) on the same network segment is likely to be delivered frequently [4]. For efficient support of video streaming, the end-to-end capacity of the network must constantly equal the rising number of the internet video users and the growing recognition of HD content. For these reasons, Software Defined Networking (SDN) is one of the best solutions for the improvement of VoD distribution [4].

According to an IEEE international symposium on multimedia paper [1], Software defined network (SDN) has emerged to be one of the promising solutions. It is an emerging paradigm in network research and industry areas to deal with the appearing demands for flexible and agile network control while implementing through open standards [2, 3]. SDN is defined as an approach to networking in which control plane is decoupled from data plane and tasks related to control plane are offloaded to separate piece of software called controller. SDN is claimed to have a more flexibility than the legacy networks and provides abilities to speed up innovation [2].

SDN has better advantages as compared to the other traditional methods. SDN, as programmable approach, promises to bring new level of flexibility, control and management to networking. Also, it has made programmable the network logic at the controller level which implies that innovations can be introduced and network can be customized to satisfy organisational needs.

OpenFlow is a subset of technologies under the SDN umbrella. It is defined as the communication protocol between the control plane and the data planes. It is the only non-proprietary, general purpose protocol for programming the forwarding plane of SDN switches. OpenFlow allows direct access to the manipulation of the forwarding plane and defines some of the behavior of the data plane of the network devices. OpenFlow is considered the as the most successful approach for the realization of SDN. [10, 11]

1.2 Description of Problem

Video on Demand has been gaining popularity over recent years with proliferation of high-speed networks. Distributed continuous media applications, are expected to provide service to a large number of clients often geographically dispersed over a metropolitan, country-wide or even global area. However, when huge content is delivered through independent unicast flows, naively ignoring that much of it is identical to transmissions hours or days earlier, network provision is greatly challenged. End-to-end capacity of network infrastructure must grow continuously to match the increasing number of Internet video users. The increasing popularity of VoD and especially of HD content worsens this.

A related problem to on-demand video streaming is that of live streaming. Live streaming accomplishes a similar goal (distributing video or other content to large number of people) but does not have the requirement that the user be able to skip to arbitrary points in the stream (for example, in the future). However, this is what this project tackles: the ability of clients to exchange pieces from anywhere in the file with high throughput, great quality of experience and good video quality.

1.3 Project Goals and Approach

In view of the problem encountered with VoD, for this project, we propose an efficient OpenFlow based in network caching. We enumerate our project goals to solve the current

VoD issues and some brief explanation on the proposed approach to achieve our goals. The project goals of our proposed solution are as follows:

- Propose an OpenFlow based Video On Demand Caching Solution
- reduces the distribution load from VoD content provider and all the transient networks along the path of the VoD server to the end-user
- implementation of the proposed solution as an openflow application
- transparently caching the content closer to the user, OpenFlow based VoD caching solution, aims at minimize the distance between the VoD streaming and the user which in return, provides great improvements to the quality of experience of the end user, as the streaming application observes higher throughput, less latency and smaller start up and buffering times.
- Validate our proposed solution by deploying the application into an emulated network.

1.4 Project Outline

The organisation of this project is as follow: Chapter 2 is the literature review of several related works on networking solution for video streaming such as content delivery network, Multicast, Peer-to-Peer, cache and proxy, software defined network, and openflow technologies. Chapter 3 is the openflow based Video on demand Caching Network Architecture. It focuses on discussing the design of the openflow based VoD caching solution, its entities and where they are placed on the network. Chapter 4 is openflow based Video on demand Caching Core Entities. It describes the main entities of the openflow based VoD caching solution and also interfaces designed for intercommunication between the openflow based VoD caching entities. Chapter 5 is for simulations and results and finally chapter 6 for conclusion and future work.

2. RELATED WORKS

The attention of the research community has been managed to be captured by the popularity of video streaming over the network, coupled with the consumer's requirement for high quality video content and high Quality of Experience. Over the years, other solutions have been proposed to solve the problems of video streaming whether live or on-demand and all these proposed solutions comes with pros and cons. In this section, we would discourse how SDN can provide a better solution and improve the problems of video streaming as compared to other solutions.

2.1 Peer to Peer Networking

The participation of an end user and their willingness to share their network resources and limited storage space is highly independent on the efficient performance of a peer to peer based networking solution for on demand video streaming. Peer to Peer sends video contents close to end users and can deliver on demand/live video streaming to end users. This is because end users can share their limited network resources for a short period of time. In the case of peer to peer, VoD streaming timeframe for end users cannot be dictated which could be between the period of days, months or even years which reduces the efficiency of peer to peer [4]. Also, Peer to peer does not guarantee VoD streaming quality because of how it joins and leaves the network at will. The distributed nature of peer to peer brings lack of centralized control especially for authentication, accounting and most importantly, security. For these reasons, content providers and network administrators are discouraged from making informed decisions and service they may provide, using distributed technique, and intelligent caching [4, 13]

2.2 Multicast

Multicast streaming is a one-to-many relationship between a Media server and the clients receiving the stream. Using the multicast stream, the server streams to a multicast IP

address on the network, and all clients receive the same stream by subscribing to the IP address. Content can be delivered as a multicast stream only from a broadcast publishing point. Also, the network routers must be multicast-enabled, meaning that they can transmit class-D IP addresses. If the network routers are not multicast-enabled, contents can still be delivered as a multicast stream over the local segment of your local area network [14]. In the case of VoD, it is something different entirely for reasons that end users could request video contents anytime after the content was made available. Related works that has made efforts to use multicast to improve the efficiency of VoD distribution, encounters different difficulties such as modifying multicast and regular network traffic in other for them to coexist, speeding up or slowing down the client's viewing rate, reserving a portion of the client's bandwidth. These solutions are not good enough and are not efficient for the distribution of VoD contents [4, 14].

2.3 Cache and Proxy

Cache and proxy approach focuses on delivering content to users locally by making provisions for additional network and storage support. M Cha et Al., demonstrates benefits of caching a content on a static cache with long term popular video, focusing on YouTube contents as an example, has determined that there could be an approximate 51% cache-hit ratio [15]. Also, M. Chesire et Al., made comparable demonstration of benefits where a caching policy with an expiration of two hours yields an aggregated request and byte hit rate of 24% using cache storage which is 2% in size of the overall transferred data [16]. Static web contents are the most common example of a cache and proxy server. For this reason, existing solutions are not usually optimized for bandwidth, high storage, and the demanding video delivery requirement such as the ability to skip to a certain part of the video stream. Cache and Proxy approach makes provision for little flexibility and configuration changes as regards the contents that can be cached and their caching policies,

which gives the network administrator little control over their network. This therefore, is a weak approach for VoD content delivery [4].

2.4 Content Delivery Network

A content delivery network (CDN) is a large network of distributed servers that deliver web pages and other Web content to a user based on the geographic locations of the user, the origin of the webpage and a content delivery server. For websites that have global reach, and websites with high traffic, CDN provides an effective service for the speedy delivery of their contents. The speed of CDN content is dependent on the closeness of the CDN server to the user's geographical location. CDNs also provide protection from large surges in traffic [12]. From a content provider's perspective, CDNs are an efficient distribution and cost effective solution. However, from a consumer ISP's perspective, CDNs do not lessen the bandwidth utilisation on middle or last mile connections, as multiple requests for the same video content will create an equal amount of flows serving the same amount of content to end-users. Also, despite the fact that CDNs deploy their servers worldwide, it is unrealistic to expect them to deploy in all ISP networks or last mile environment. The inability to truly reach out to last mile environments has also been recognised by CDNs themselves [4].

M. Zhao et Al. [13] proposed a CDN-P2P based solution which addresses the problem and reduces the maintenance and administrative cost of CDN’s inability to reach out to last miles environment. This proposed solution complements CDN services by pushing the content closer to the end user. However, this solution has some disadvantages of the peer to peer networking which requires users to download and install software which takes up the user’s storage space and uploading resources. For these reasons, a more flexible, configurable and transparent in network caching service which is closer to the user, would complement CDN and in fact benefit last mile environment [4].

2.5 Software Defined Networking and OpenFlow

The growth of data centers has brought Software Defined Networking (SDN) to the forefront. SDN is an umbrella term encompassing several kinds of network technology aimed at making the network as flexible and agile as the virtualized server and storage infrastructure of the modern data center [17].According to Open Network Foundation (ONF), SDN is an emerging network architecture where network control is decoupled from forwarding functions enabling the network control to become directly programmable. This migration of control, formerly tightly bound in individual network devices, into accessible computing devices enables the underlying infrastructure to be abstracted for applications and network services, which can treat the network as a logical or virtual entity [18].The goal of SDN is to enable network administrators and engineers respond quickly to changing business requirements. Network administrator can quickly and easily make and push traffic decisions from a centralized control console without having to touch switches on the network, and can deliver services to wherever they are needed in the network, disregarding the specific devices a server or other device is connected to. The key technologies are functional separation, network virtualization and automation through programmability [17]. The most common currently used protocol by SDN to facilitate the communication between the controller and the switches is the OpenFlow.

Figure 3 depicts a logical view of the SDN architecture. In software-based SDN controllers, network intelligence is (logically) centralized which maintains a global view of the network. For this reason, the network appears as a single, logical switch to the application and policy engine.

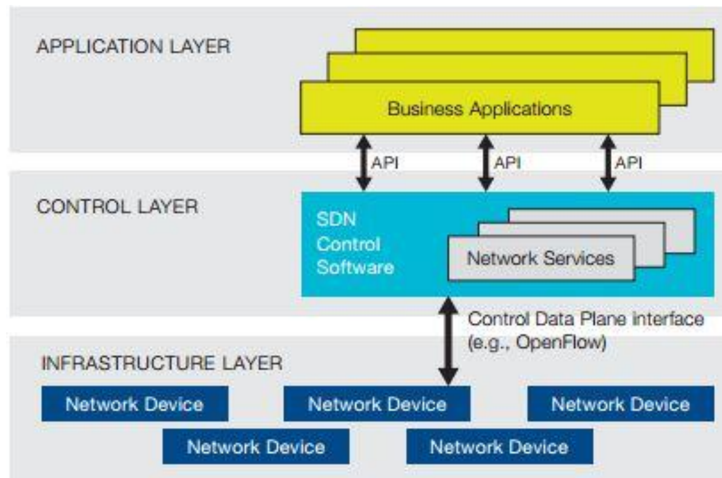

Figure 3 - The SDN System Architecture SOURCE: ONF

The SDN architecture above comprises of three layers as seen on Figure 3. First we have the Application Layer. The application layer contains network applications that can introduce new network features, such as security, forwarding schemes or assist the control layer in the network configuration. Network intelligence is logically centralized in DN controllers, which maintain a global view of the network. As a result, the network appears to the application and policy engines as a single, logical switch. In the application layer lies the Business application. The business application refers to applications that are directly consumable by end users. Possibilities include video conferencing, supply chain management and customer relationship management [19].On the second layer we have the Control layer. The control layer also known as the forward plane makes use of the information provided by the forwarding layer and defines routing and network operation. In the control layer lies the Network services. The network service refers to functionality that allows business applications to perform securely and efficiently. Possibilities include a wide range of L4 – L7 functionality including ADCs, WOCs and security capabilities such as firewalls, IDS/IPS and DDoS protection [19]. The last layer is the infrastructure layer. The infrastructure layer which is also referred to as the data layer comprises of the network forwarding elements.

Enterprises and Carriers with SDN gain vendor-independent control over the entire network from a single logical point, which greatly simplifies the network design and operation. SDN also greatly simplifies the network devices themselves, since they no longer need to understand and process thousands of protocol standards but merely accept instructions from the SDN controllers. Perhaps most importantly, network operators and administrators can programmatically configure this simplified network abstraction rather than having to hand-code tens of thousands of lines of configuration scattered among thousands of devices [18].

OpenFlow is an emerging technology with potentials to increase the value of data center services dramatically. Implementing OpenFlow can provide network administrators with greater control over their resources, integrated network and server management, and an open management interface for routers and switches [20]. OpenFlow is a standard communications interface defined between the control and forwarding layers of an SDN architecture. OpenFlow allows direct access to and manipulation of the forwarding plane of network devices such as switches and routers, both physical and virtual [18].

No other standard protocol does what OpenFlow does, and a protocol like OpenFlow is needed to move network control out of the networking switches to logically centralized control software. The OpenFlow protocol is implemented on both sides of the interface between network infrastructure devices and the SDN control software. OpenFlow uses the concept of flows to identify network traffic based on pre-defined match rules that can be statically or dynamically programmed by the SDN control software. It also allows IT to define how traffic should flow through network devices based on parameters such as usage patterns, applications, and cloud resources. Since OpenFlow allows the network to be programmed on a per-flow basis, OpenFlow-based SDN architecture provides extremely granular control, enabling the network to respond to real-time changes at the application, user, and session levels [18].

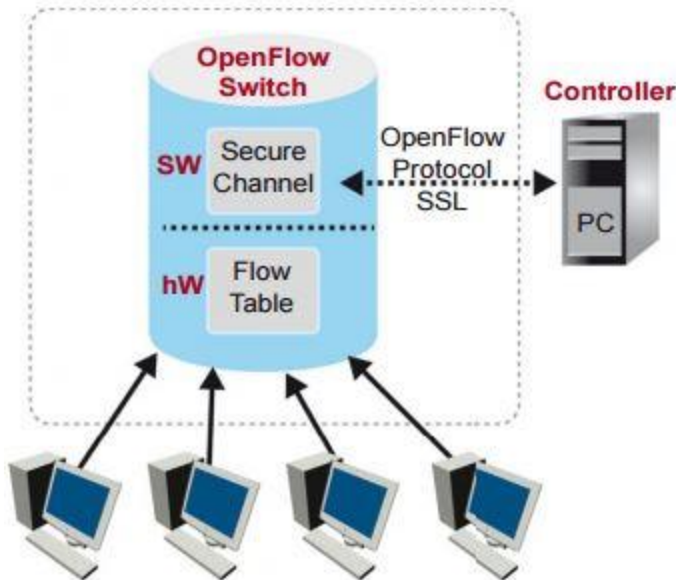

Figure 4 - OpenFlow switch components. Source: [20]

OpenFlow controller is an application that manages flow control in SDN environment [21]. OpenFlow controller can be compared to an operating system (OS) as it acts as an OS for the network. The controller is a passage for communication between applications and devices.

An OpenFlow switch consists of three parts as illustrated in Figure 4 [20]:

i. One or more flow tables and group table that performs packet lookups, tells the switch how to process each data flow by associating an action with each flow table entry

ii. A secure channel Connects the switch to a remote control processor (called the Controller) allowing commands and packets to be sent between the controllers and the switch

iii. An OpenFlow protocol, which provides an open, standardized interface for the controller to communicate with the switch

The OpenFlow Protocol allows entries in the Flow Table to be defined by a server external to the switch, which creates the potential to unify server and network device management.

The OpenFlow Protocol allows entries in the Flow Table to be defined by a server external to the switch, which creates the potential to unify server and network device management. Protocol is implemented on both sides of the interface between network infrastructure devices and the SDN control software. OpenFlow uses the concept of flows to identify network traffic based on pre-defined match rules that can be statically or dynamically programmed by the SDN control software. Using this protocol, a controller can make changes to a flow table (either by adding, deleting, or updating the table) reactively and proactively. Implementation is based on the application programming interface (API), which allows the flow table to be modified which representing a switch's forwarding decision [10, 18, 20].

Packets processed by the forwarder are compared with entries in the Flow table. For every packet matched the flow entry received by the forwarder, the appropriate actions (forward, drop, etc.) will be taken. Flow entries may forward packets to one or more OpenFlow ports. If no match entry is found, the switch sends the packet to the controller then the controller determines how to handle the packets and adds additional flow entries to the flow table on the switch.

The emergence of openflow has brought about the birth of so many openflow controllers, some of which includes opendaylight (ODL), Floodlight, Trema, Beacon, Ryu, NOX,JASON, NodeFlow, Ovs-Controller, and POX. Floodlight Controller is an SDN Controller offered by Big Switch Networks that works with the OpenFlow protocol to orchestrate traffic flows in a software-defined networking (SDN) environment. It is an enterprise-class, Apache-licensed, Java-based OpenFlow Controller. Floodlight is developed by an open community of developers. It is simple to build and run. For this project implementation, we chose to use Floodlight OpenFlow controller because of some of its features which are as following; It offers a module loading system that make it simple to extend and enhance, Easy to set up with minimal dependencies, Supports a broad range of virtual- and physical- OpenFlow switches, Can handle mixed OpenFlow and non-OpenFlow networks; it can manage multiple

“islands” of OpenFlow hardware switches, designed to be high-performance; is multithreaded from the ground up [22].

OpenFlow Specification 1.1 version and later, OpenFlow introduces a new feature that supports more complex forwarding behaviors which are possibly applied to a set of flows and this is called Group Table. Group table contains group entries (as shown on Table 1) and each entry contains a list of action buckets. Action in one or more buckets is applied to packet sent to the group [10]. A group entry may be performed if a flow table entry uses appropriate instruction that refers to its group identifier. Sometimes, multiple flow table entries can point to the same group identifier; in this case, the group table entry is performed for multiple flows. Group table entry contains a group type, a counter field and field for actions buckets [10]. The Counters are used for collecting statistics about packets that are processed by this group. A single action bucket contains a set of actions that may be executed, depending on the group type. There are possibly multiple action buckets for a group table entry. The group types define which of them are applied. There are four group types in group table including all, select, indirect and fast failover [10].

Group Identifier	Group Type	Counters	Actions Buckets

Table 1 - Group Table Entries for OpenFlow 1.1 and later versions

For enterprises and carriers alike, SDN makes it possible for the network to be a competitive differentiator, not just an unavoidable cost center. OpenFlow-based SDN technologies allow IT to address the high bandwidth, dynamic nature of today’s applications, adapt the network to ever-changing business needs, and significantly reduce operations and management complexity [18].

Some of the benefits that enterprises and carriers can achieve through an OpenFlow-based SDN architecture are [18]:

- **Centralized control of multi-vendor environments**: SDN control software can control any OpenFlow-enabled network device from any vendor, including

switches, routers, and virtual switches. Rather than having to manage groups of devices from individual vendors, IT can use SDN-based orchestration and management tools to quickly deploy, configure, and update devices across the entire network.

- **Complexity Reduction through automation**: OpenFlow-based SDN offers a flexible network automation and management framework, which makes it possible to develop tools that automate many management tasks that are done manually today. These automation tools will reduce operational overhead, decrease network instability introduced by operator error, and support emerging IT-as-a-Service and self-service provisioning models.
- **Increased network reliability and security**: SDN makes it possible for IT to define high-level configuration and policy statements, which are then translated down to the infrastructure via OpenFlow., Ensured access control, traffic engineering, quality of service, security, and other policies are enforced consistently across the wired and wireless network infrastructures because SDN controllers provide complete visibility and control over the network.
- Better user experience: By centralizing network control and making state information available to higher-level applications, an SDN infrastructure can better adapt to dynamic user needs.

For this project, our in-network caching service uses OpenFlow to dynamically cache and distribute media content within a network in a highly efficient, vendor-agnostic and transparent manner. The transparency aimed to be achieved through the use of SDN is particularly important as no end-client applications or media delivery services have to be updated.

3. OPENFLOW BASED VIDEO ON DEMAND CACHING NETWORK ARCHITECTURE

For this section, the description of our openflow based VoD caching solution is discussed. It discusses the design of the openflow based VoD caching solution, its entities and where they are placed on the network.

3.1 Design

Openflow based VoD caching offers a powerful interface that provides "cache" as a service. This interface is not intrinsically linked to a particular type of content (i.e. it is content-agnostic), or to a specific hardware or software implementation. The control and decision of what content should be cached is made by the network administrator, who now has the ability to optimise his network's utilisation and external link usage. This can be achieved by enabling in-network caching for specific content via OpenCache's designated interfaces. Openflow based VoD caching exposes these interfaces through powerful and flexible HTTP-RPC web services with JSON, which allow VoD content to be cached closer to the end-user. This placement of cached content also increases the user's quality of experience when streaming it. Given appropriate SLAs, OpenCache's interface could also be used by content providers (e.g. CDNs) to declare their content as cacheable on last mile environments, without having to physically deploy and administer their own caching hardware. [4]

The functionalities of the entities involved in the openflow based VoD caching are carefully revised and also renamed to match their new functionality and interfaces extension. From my design point of view, openflow based VoD caching architecture includes the following components (Figures 5).

1. OpenFlow Switch: OpenFlow switch is a software program or hardware device that consists of a flow table, which performs packet lookup and forwarding, and a secure channel to an external controller in a software-defined networking (SDN) environment [24, 25]. OpenFlow switches are either based on the OpenFlow protocol or compatible with it. It

communicates with the Video-on-Demand (VoD) Server, the openflow based VoD caching (opencache) Nodes, and the OpenFlow Controller.

2. Video-on-Demand (VoD) Server: A VOD server (video-on-demand server) is a specialized server that delivers video content such as movies and television shows over the internet, in response to requests from users with internet-connected devices such as PCs, tablets, or smart phones [26]. In this case of my scenario, the VoD server was placed on the internet with an IP reachable address as shown in figures 5.

3. OpenFlow Controller: An OpenFlow Controller is a type of SDN Controller that uses the OpenFlow Protocol. An SDN Controller is the strategic point in software-defined network (SDN). An OpenFlow Controller uses the OpenFlow protocol to connect and configure the network devices (routers, switches, etc.) to determine the best path for application traffic [27]. For an openflow based video on demand caching solution, there are varieties of controller to choose from for implementation. Some of which includes POX, NOX, Floodlight etc. For this project's openflow video on demand caching solution scenario, I have chosen to use the Floodlight Openflow Controller for its performance and robustness. The controller behaves as a standard L2 Learning Switch: instructing the switch to forward packets based upon a learned MAC-to-port pairing.

4 OpenFlow based VoD caching (opencache) Controller: The Opencache Controller is the orchestrator of the VoD caching and distribution functionalities

5. OpenFlow based VoD caching (opencache) Node: The Opencache Node(s) are essentially the cache entities of our VoD in-network caching service

6. Non-Openflow Switch: is a computer networking device that connects devices together on a computer network by using packet switching to receive, process, and forward data to the destination device.

7. Database: This database is used for maintaining a list of all the video assets that have been requested for caching and the ones that have already been cached. Openflow based VoD caching uses mongoDB [mongoDB] , due to its open-source, document based and

NoSQL nature. MongoDB offers great benefits compared to database systems, such as fast in-place updates, scalability, flexible aggregation and data processing, full index support and replication and high availability. The OCC regularly updates the information that mongoDB stores using the pymongo python library [mongoDB-PY] [4].

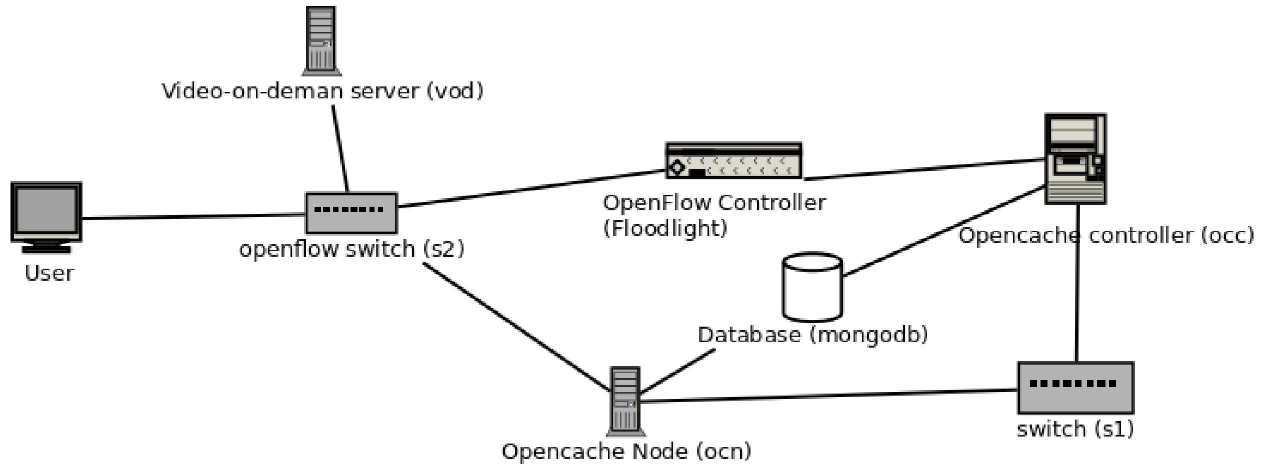

Figure 5 - Simplified Simulation Topology

A single, widely reachable Opencache controller would be able to coordinate caching amongst a number of opencache node instances (i.e. caches). We created simulation topology in Mininet to meet our scenario. A user, a VOD, a OpenFlow switch, and OCN are in Mininet as a single node in the simulated environment.). It is important to note that it is not a requirement that OpenFlow be deployed throughout the network; the connecting network hardware could also be entirely non-OpenFlow. The only specific OpenFlow switch requirement is at the last hop, closest to the user. The Opencache node is connected to the outside of simulation to the main computer through a simulated switch but without OpenFlow capability. The main computer has OCC, database and OpenFlow controller. The OpenFlow controller also has a special connection to OpenFlow switch inside the simulated environment. Deployment of an OCN instance on the same VM as the end-user clients would offer the lowest latency and fastest response time, and thus ensure higher quality of

experience for the end-users. However, generally speaking, the use of multiple OCNs at different points in a network (e.g. attached to aggregation switches in an enterprise or University campus network) is also entirely feasible. In fact, a hierarchical approach has the potential to provide further benefits if a greater proportion of requests can be fulfilled without leaving the LAN.

4. OpenFlow based Video on demand Caching Core Entities

This section describes the main entities of the openflow based VoD caching solution which are the openflow based VoD caching (opencache) controller and the openflow based VoD caching (opencache) node and explains their extended functionalities. Also, in this section, we describe the interfaces designed for intercommunication between the openflow based VoD caching entities with the help of openflow for an overall efficient and transparent in-network caching of video assets.

4.1 OpenCache Controller

The OpenCache Controller is the main orchestrator of the in-network caching functionality that openflow based VoD caching provides, and implements some fundamental functionalities that will be described.

4.1.1 Handling Request For Content of Interest

The Opencache controller receives requests for content of interest that should be cached in the network's opencache nodes (Figure 7). A network administrator starts caching the video on demand server content using Software defined networking. The network administrator instructs the Opencache controller to cache any new video being requested from the user to a video on demand server. The network administrator sends an IP address of the video on demand server to the Opencache controller as a parameter of which the video on demand should be cached.

The Opencache controller processes these instructions and saves the IP address to its database. The Opencache controller tells Opencache node to prepare and cache any request redirected to it.

Opencache node prepares a dedicated caching processor and dedicated port for receiving the request. Then, the Opencache node tells the Opencache controller that it is ready and sends Opencache controller the TCP port number it will receive the request on that is redirected to it.

The Opencache controller receives the port number information then orders the OpenFlow controller to redirect any traffic flow from user to the video on demand server in the OpenFlow switch to be redirected to Opencache node and the dedicated port.

The OpenFlow controller does the following. It installs flow rule into the OpenFlow switch as follows:

1. If a user requests any packet on the network and has its destination is the video on demand server, the packet should be redirected to the Opencache node and then it rewrites the IP header of the packet to be destination of the Opencache node with the dedicated port number in the Opencache node.
2. If any packet on the network is from Opencache node to user, before the packet is been forwarded to the user, it rewrites the packet header of IP address to be the source from video on demand server and port 80, as such, the user thinks the packet came from video on demand server when it receives the packet.

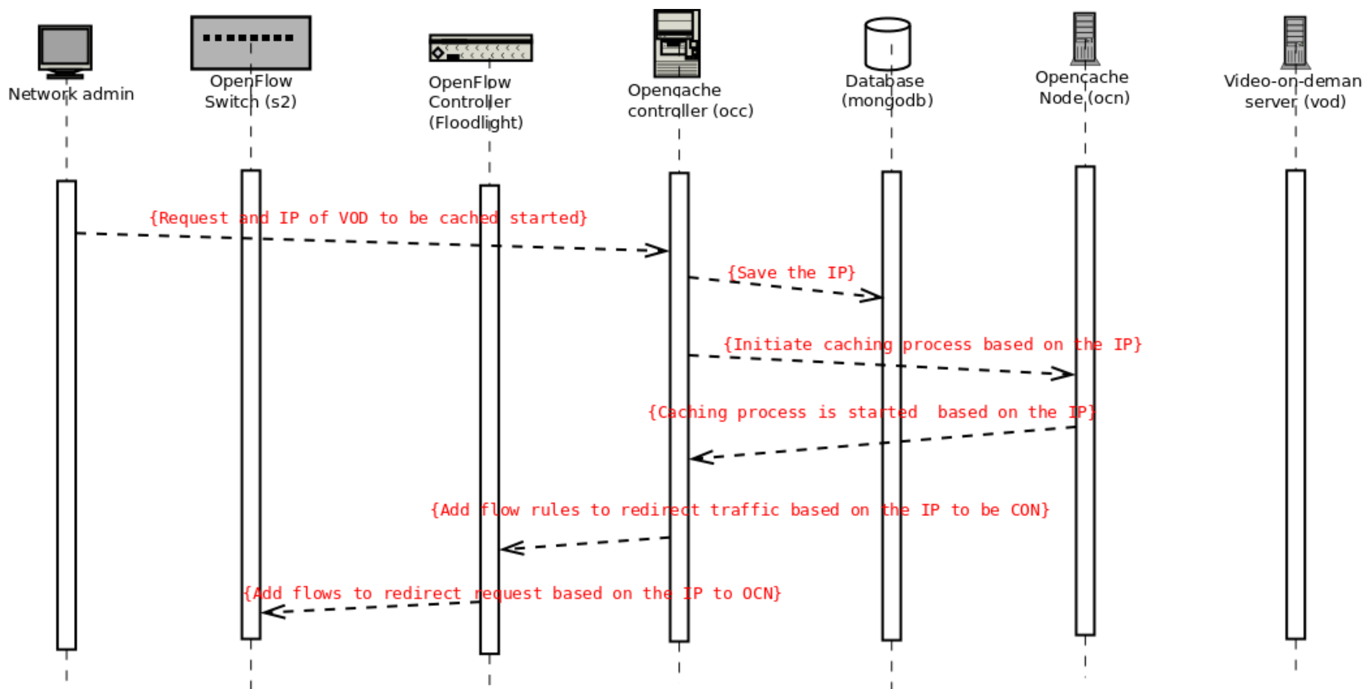

Figure 6 - Declaring Content of Interest as Cacheable

4.1.2 Implementing Caching

Since the Opencache controller is the orchestrator of the in-network caching functionality, it is the central entity that is responsible to implement the caching logic. The caching logic essentially dictates what content should be cached and where (i.e. to which Opencache node) at each point in time. The main goal and advantage of Openflow based video on demand caching is to be able to provide (through its Opencache controller) centrally controlled caching and allow the network administrator to program and deploy his caching logic at will. The implementation of a caching logic could be based on the specific parameters that a network administrator wants to optimise in his network. For example, an administrator could program the caching logic to minimise the streaming latency of recorded video lectures on a University's network, or to implement the pre-caching of popular content closer to end-users overnight, when the network is underutilised. It is important to note the ease and speed at which the network administrator can actually implement the caching logic with the use of OpenFlow in OpenCache [4].

4.1.3 Managing Caching Resources

The responsibility of Opencache controller is to manage the available Opencache nodes' resources in the network. This includes two important functionalities. The first one concerns the ability to understand the status of the OCNs (if they are online or offline) and the second one concerns the propagation of the caching decisions to the OCNs.

An important part of the caching resource management is to be able to handle the addition and removal of caches in a network dynamically. For this reason, the Opencache controller exposes three methods (namely hello, keep-alive and goodbye) that allow the communication of the Opencache controller with a number of Opencache nodes. When an Opencache node is added to the network, it invokes the hello method to let the Opencache controller know that it is now available on the network. In turn, the Opencache controller replies with a node-id that is assigned to this particular Opencache node. From that point on, the Opencache node periodically sends a keep-alive message telling it that it is still in the network and functioning properly. If the Opencache controller does not receive a keep-

alive message from an Opencache node, then it assumes that the Opencache node is not reachable, which could be a result of network congestion or because it has been taken offline. Consequently, the OCC will remove the “unreachable' Opencache node from the list of caching resources that it has at its disposal. Figure 8 illustrates with a sequence diagram the removal of a cached content.

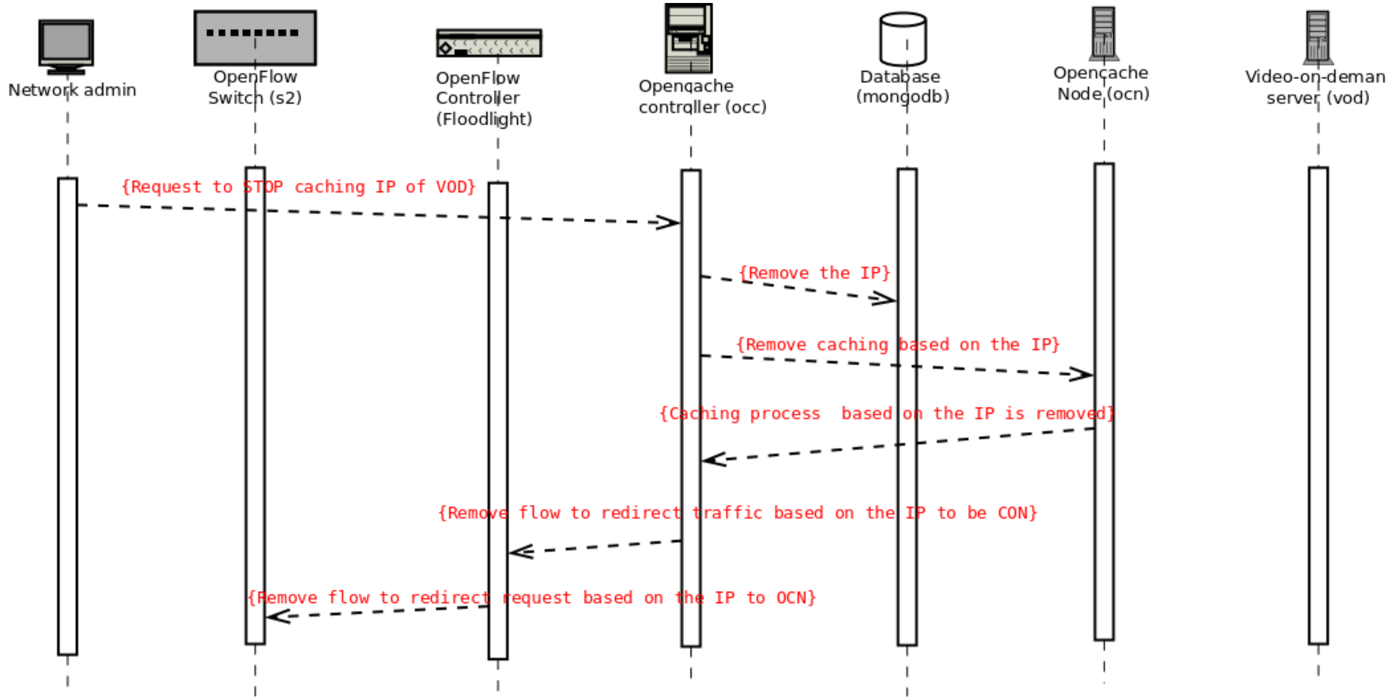

Figure 7 - Removal of Cached Content

4.2 OpenCache Node

The OpenCache Node is responsible for caching the appropriate video content, and delivering it to users if they request it. When the OpenCache Node comes online in a network, it communicates with the OpenCache controller using the hello method and makes its resources available to it [4].

4.2.1 Cache-miss Scenario

In the cache-miss scenario as illustrated on figure 9, a video is delivered to the user by OpenCache Node after it forwards the requests to Video on Demand Server and caches the video for future use. First, a user sends a request to get a video in Video on Demand Server. The OpenFlow switch checks this request and finds it match one of their flow rules which are destination IP address to Video on Demand Server and port is 80. The switch, then, rewrites the header of IP on this package request to be the destination to IP of OpenCache Node and transmission control protocol (TCP) port number dedicated for this Video on Demand Server request. Then, the switch sends it to OpenCache Node.

OpenCache Node checks this request and if it does not find that OpenCache Node has cached the video being requested, the OpenCache Node request Video on Demand Server to get this video. The Video on Demand Server delivers to the OpenCache Node then the OpenCache Node caches this video and sends the video to the user via the OpenFlow switch.

OpenFlow switch checks the packets of this video and the packets match one of its flow rules. So, the switch rewrites the IP header of these packets to be the source IP address of the IP address of Video on Demand Server and port 80. The switch then sends these packets containing requested video to the user. The User receives the video thinking it is directly from Video on Demand Server meanwhile, the video is being cached first in OpenCache Node.

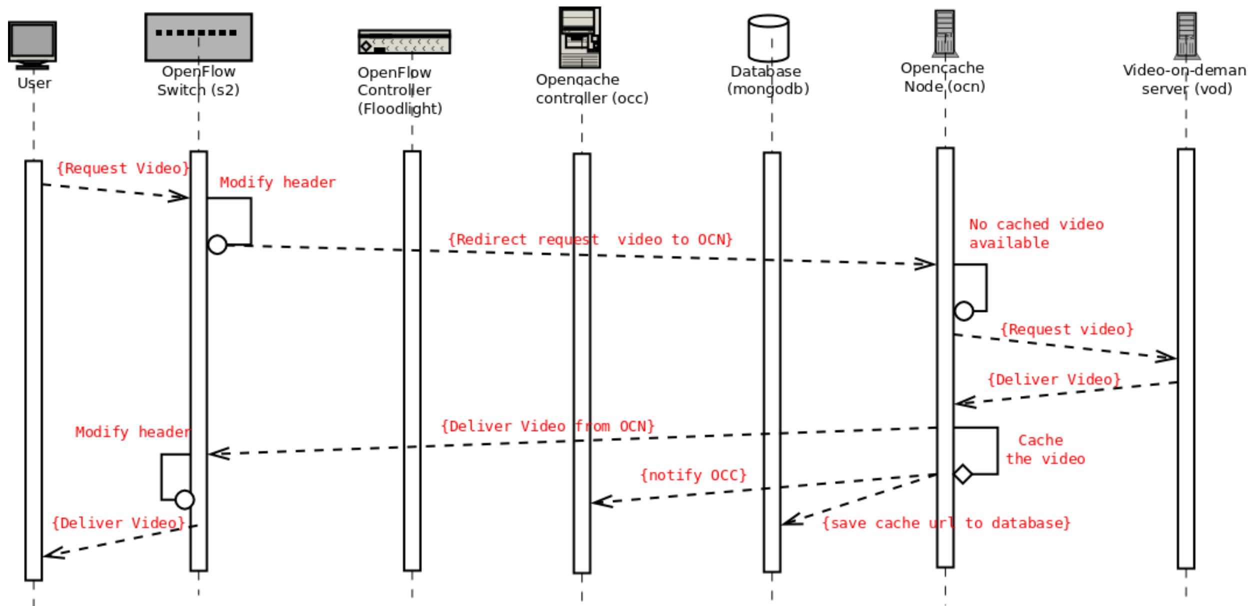

Figure 8 - A Cache-miss Scenario

4.2.2 Cache-hit Scenario

In the cache hit as illustrated in figure 10, a video delivers to the user directly using their cached video. Similar to what have been done in the cache-miss scenario, first, a user sends a request to get a video in Video on Demand Server. The OpenFlow switch checks this request and find it match one of their flow rules which are destination IP address to Video on Demand Server and port is HTTP. OpenFlow switch, then, rewrites the header of IP on this package request to be the destination to IP of OpenCache Node and TCP port number dedicated for this Video on Demand Server request. Then, the switch sends the request packet to OpenCache Node. OpenCache Node check this request and finds that the OpenCache Node has cached the video being requested. Instead of requesting the Video on Demand Server again, OpenCache Node directly delivers the cached video to the user via OpenFlow switch.

OpenFlow switch checks the packets of this video and the packets match one of its flow rules. So, the switch rewrites the IP header of these packets to be the source IP address is IP address of Video on Demand Server and port 80. The switch sends these packets containing

requested video to the user. The user receives the video as if it is from Video on Demand Server meanwhile it is the cached video of from OpenCache Node.

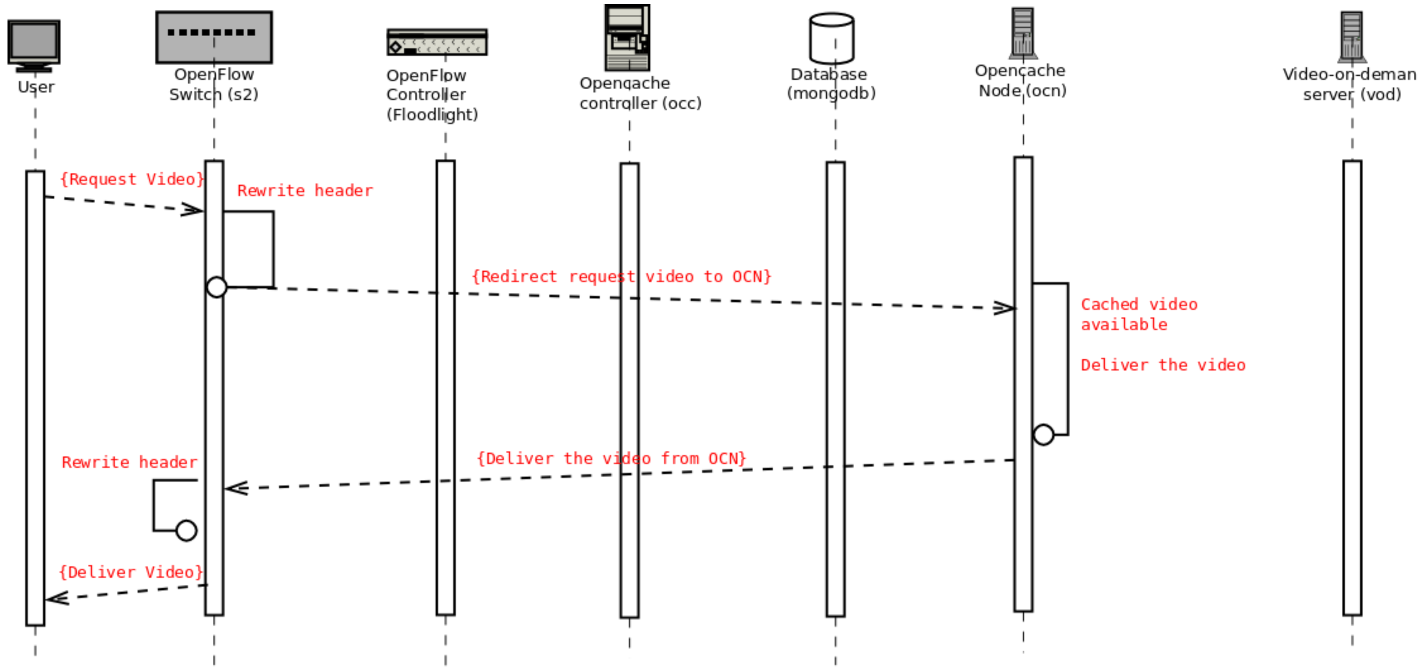

Figure 9 - A Cache-hit Scenario

4.2.3 Non-Caching Scenario

For a Non-Caching Scenario as illustrated in figure 11, it follows the everyday pattern of how the internet behaves normally. The user requests to Video on Demand Server for a video. The Video on Demand Server delivers the video to the user.

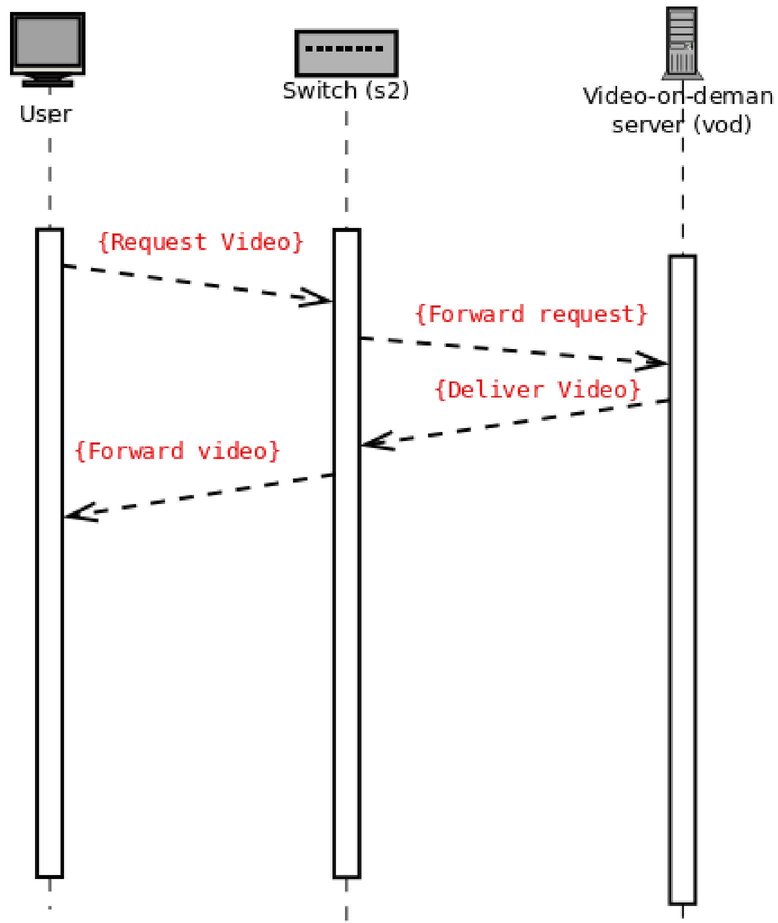

Figure 10 - Non-Caching Scenario

5. SIMULATIONS AND RESULTS

5.1 CONTROLLER

The first part of our experiment is determining what type of controller to use for the implementation. There are different known controllers available. The one that is used in our solution is POX Floodlight Openflow Controller. Floodlight Openflow Controller is an SDN Controller offered by Big Switch Networks that works with the OpenFlow protocol to orchestrate traffic flows in a software-defined networking (SDN) environment. It is an enterprise-class, Apache-licensed, Java-based OpenFlow Controller. Floodlight is developed by an open community of developers. It is simple to build and run. For this project implementation, we chose to use Floodlight OpenFlow controller because of some of its features which are as following; It offers a module loading system that make it simple to extend and enhance, Easy to set up with minimal dependencies, Supports a broad range of virtual- and physical- OpenFlow switches, Can handle mixed OpenFlow and non-OpenFlow networks; it can manage multiple “islands” of OpenFlow hardware switches, designed to be high-performance; is multithreaded from the ground up [22].

For entirety, some other controllers must be cited. POX controller provides a framework for communicating with SDN switches using either the OpenFlow or OVSDB protocol. It has a high-level SDN API including a queriable topology graph and support for virtualization. Developers can use POX to create an SDN controller using the Python programming language. It is a popular tool for teaching about and researching software defined networks and network applications programming [23, 28]. Beacon is another Java-based controller, open-source, which has high throughput and low latency [29]. OpenDaylight controller is the most latter addition to Openflow controllers. OpenDaylight announced its first release: Hydrogen. It was the first simultaneous release of OpenDaylight and features three different editions to help users get started: the Base Edition, the Virtualization Edition, and the Service Provider Edition [30].

5.2 NETWORK EMULATOR

Mininet is the network emulator that we used in our work for running the experiments. It is the standard network emulation tool that can be used for SDN. Network namespace gives personal processes with their own network interfaces, routing tables and Address ARP tables. Mininet takes advantage of this feature of the kernel. It uses process-based virtualization to run hosts and switches on a single operating system (OS) kernel. Large networks (up to 4096 host on a single operating system) with different topologies can be tested and emulated. It is commonly achievable to develop a Mininet network simulates a hardware network, or a hardware network that simulates a Mininet network, and to compute the same application and binary code on either platforms [31].

5.3 NETWORK SETUP

This experiment was done on a HP ProBook 4430s laptop with an Intel(R) Core(TM) i5-2450M CPU 2.50GHz Processor, 2.50GHz of power, 4GB of RAM. The Operating system is Linux Lubuntu 14.0.4 and Mininet Version 2.2.1 was run native on Windows 7. Mininet 2.2.1 supports openflow 1.3.

5.4 NETWORK TOPOLOGY

We created simulation topology in Mininet to meet our scenario. A user, a VOD, a OpenFlow switch, and OCN are in the Mininet as a single node in the simulated environment. Meanwhile, the OCN is connected to the outside of simulation to the main computer through a simulated switch but without OpenFlow capability. The main computer has OCC, database and OpenFlow controller. The OpenFlow controller also has a special connection to OpenFlow switch inside the simulated environment.

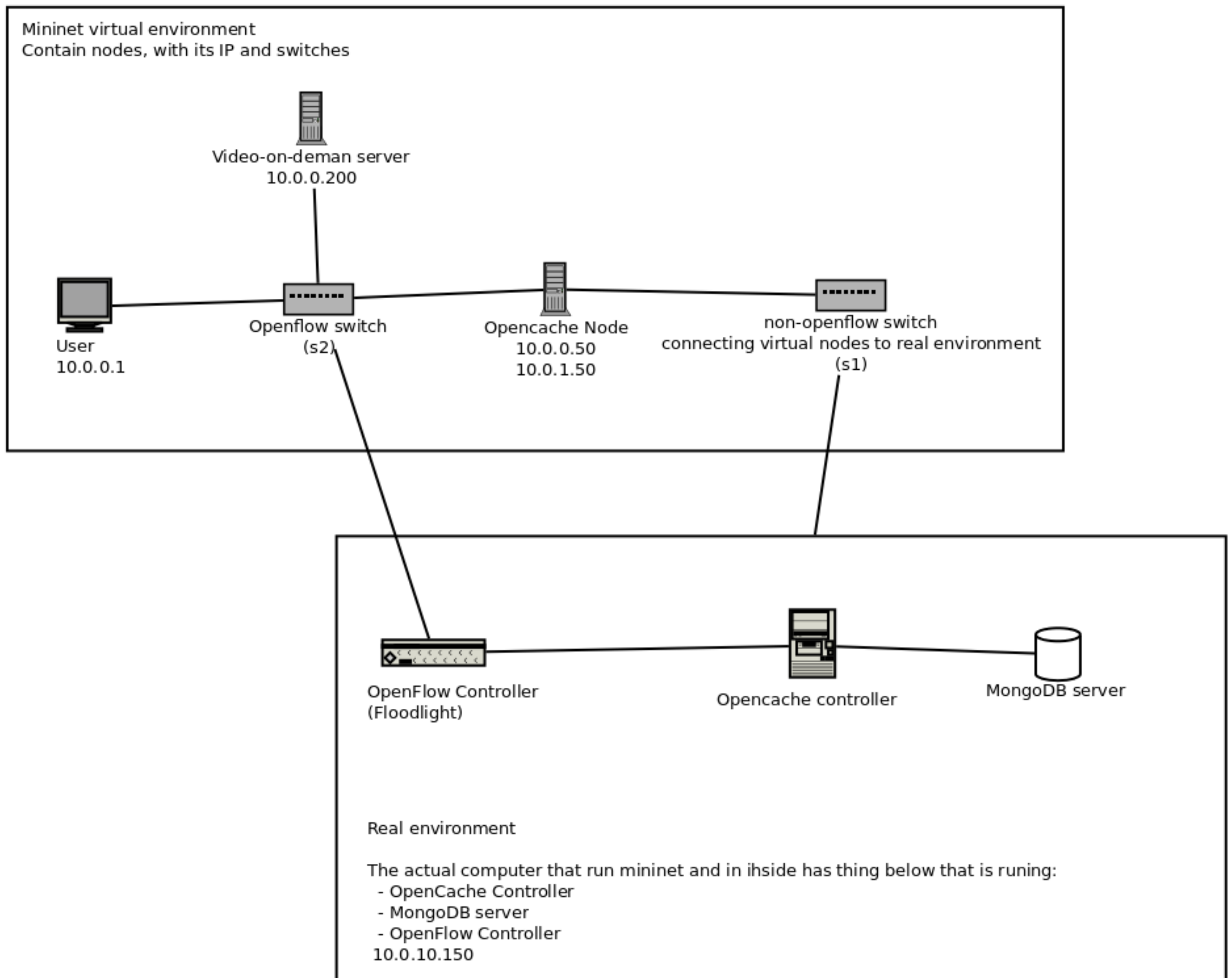

Figure 11 - Detailed Simulation Topology

In our implementation code, we write the code using Python and Bash. We also write a simulation code for Mininet containing the topology of our scenario. The code also invokes and reuses several open source code libraries such as floodlight, Opencache, flask, pymongo and several other libraries. The Opencache library is modified to fit current compatibility and to fixing several bugs. We also write several codes to run the scenario. More detailed is explained in table of code and folder. The activities diagram explains which code are executing by which action for the scenario.

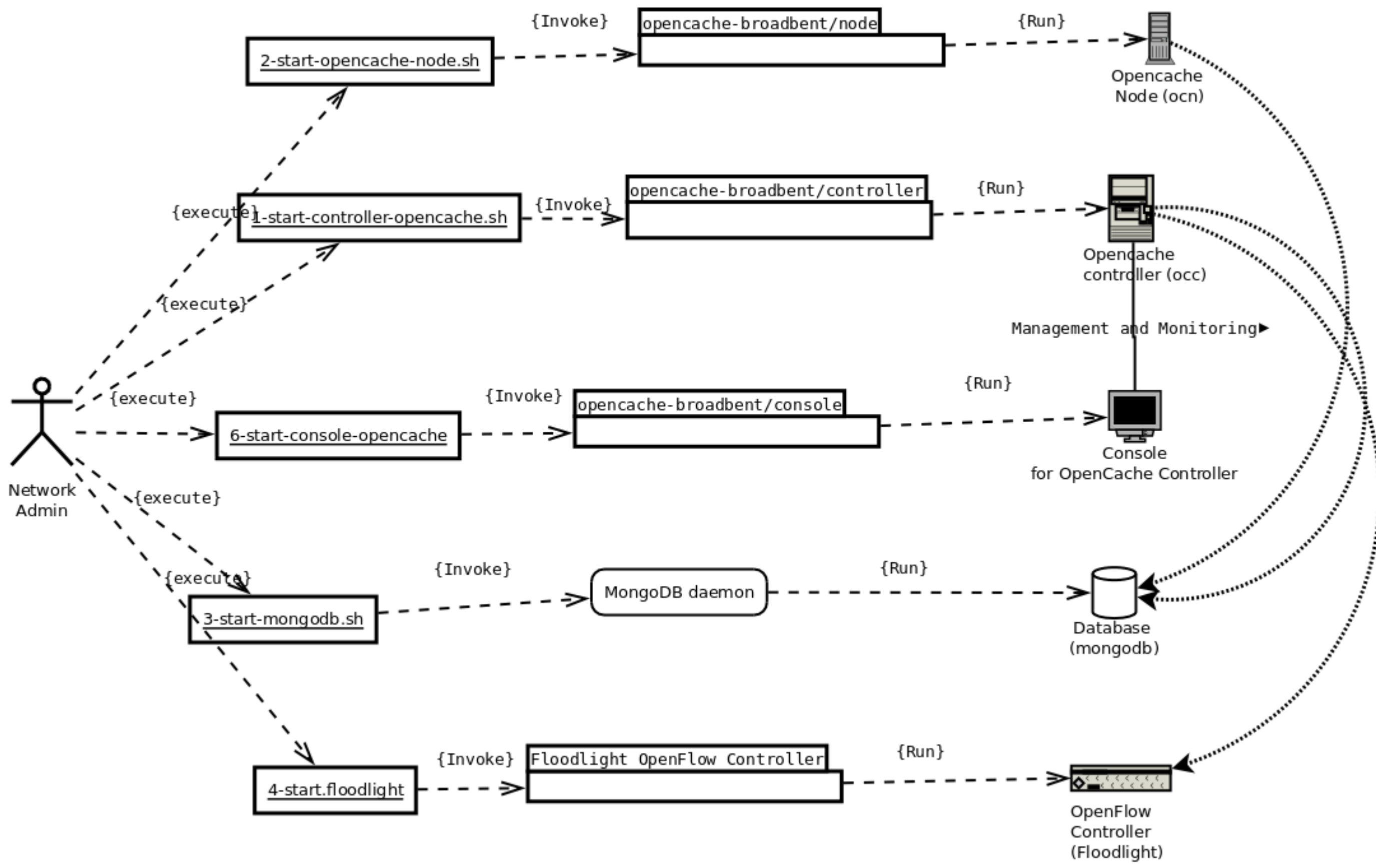

Figure 12 - Activity Diagram For Network Administrator

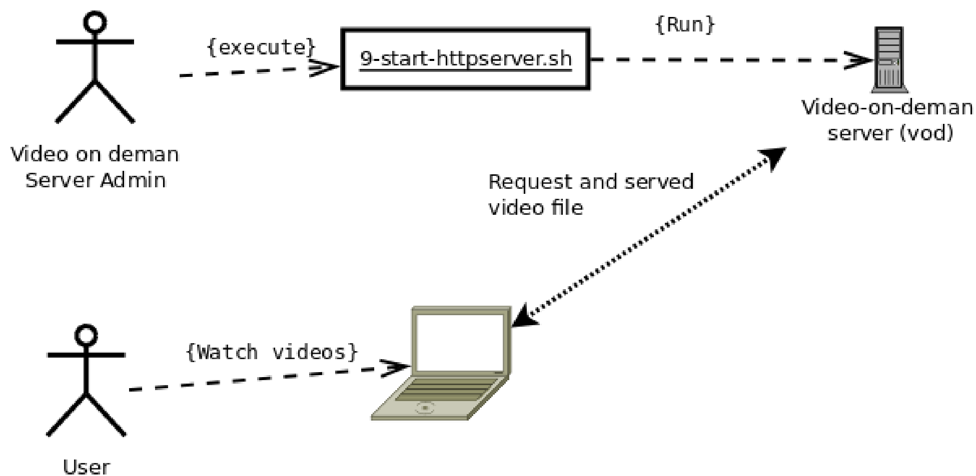

Figure 13 - Activity Diagram

5.5 EXPERIMENTS AND RESULTS

5.5.1 Scenario 1- screenshot of instance to declare caching

Figure 14 - screenshot of instance to declare caching

5.5.2 Scenario 2- Screenshot of Cache-hit

Figure 15 - Screenshot of Cache-hit

5.5.3 Scenario 2- Screenshot of Cache-miss

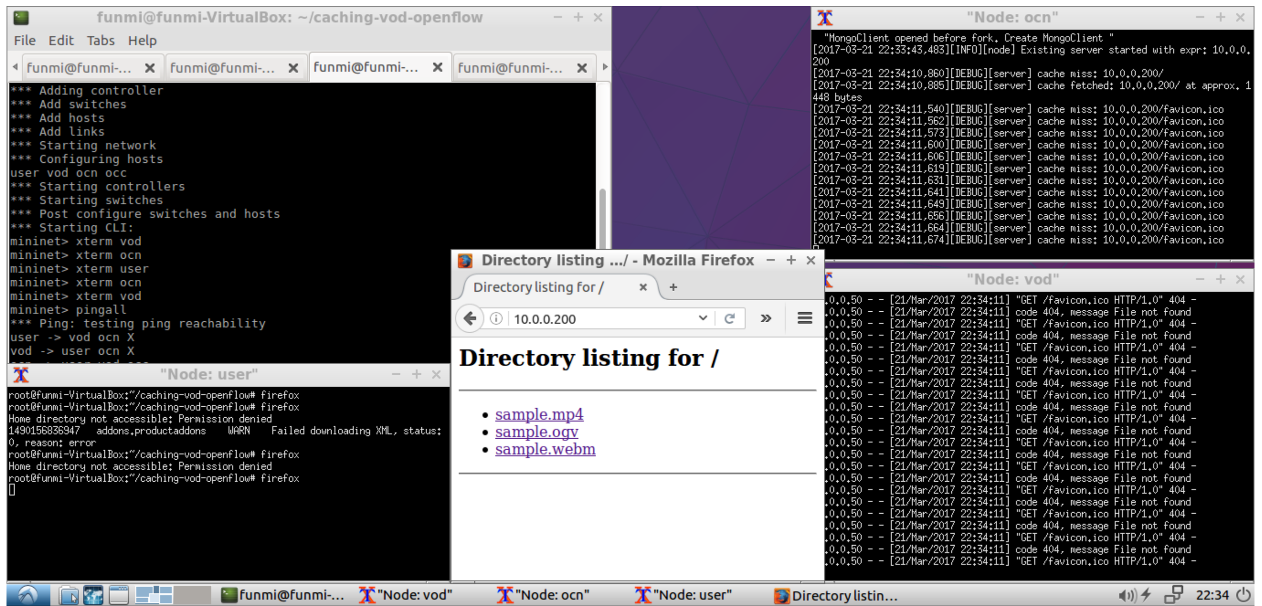

Figure 16 - Screenshot of Cache-miss (i)

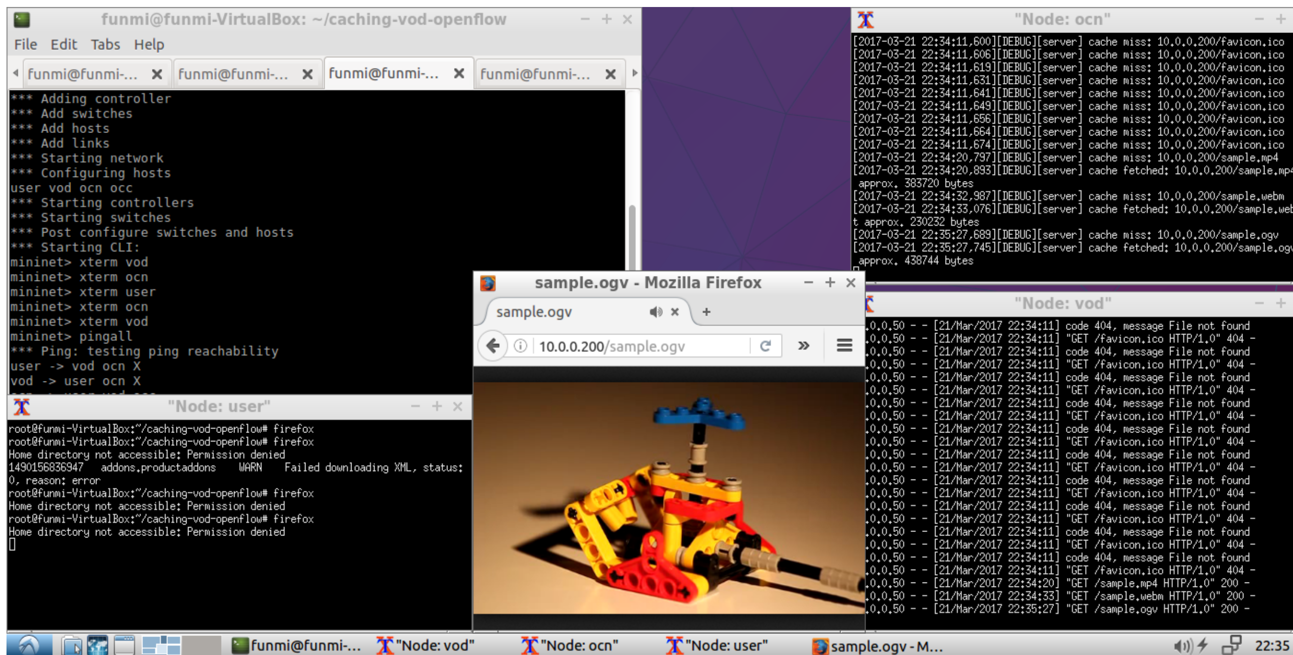

Figure 17 - Screenshot of Cache-miss (ii)

5.5.4 Scenario 2- Screenshot of no-caching

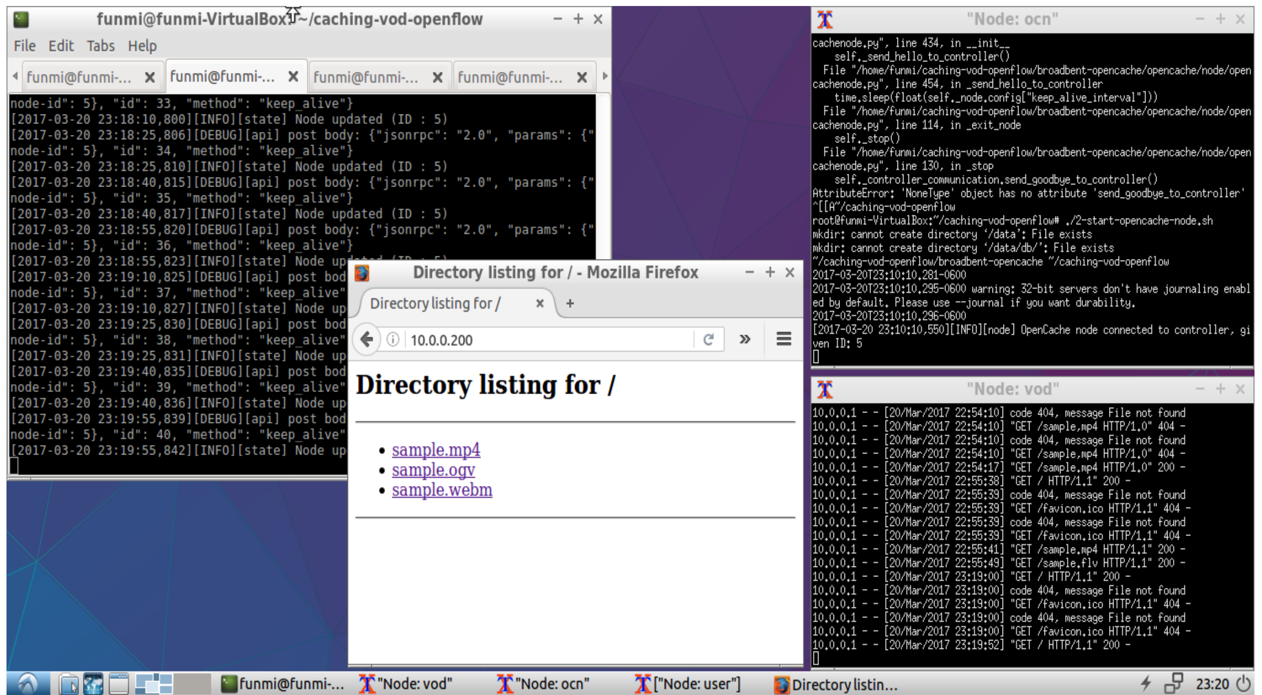

Figure 18 - Screenshot of no-caching (i)

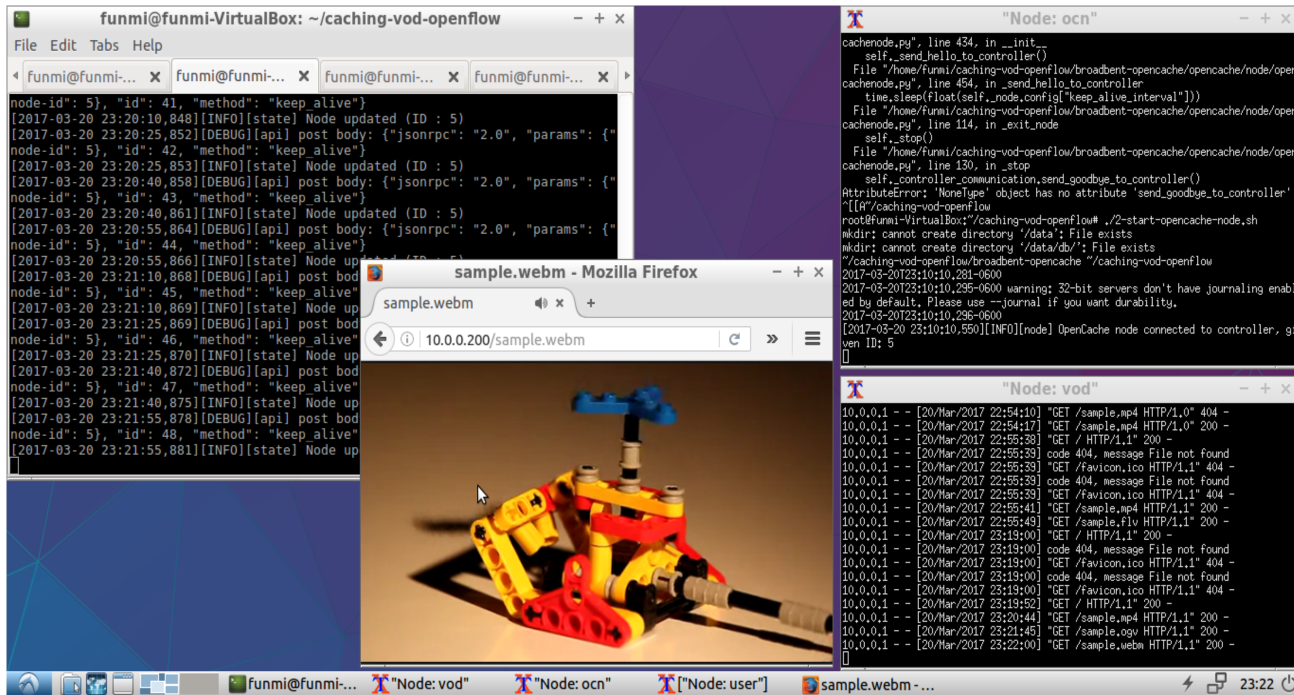

Figure 19 - Screenshot of no-caching (ii)

6. CONCLUSION AND FUTURE WORK

This chapter presents the summary of the project, discusses future work.

6.1 CONCLUSION

In this deliverable, OpenFlow based Video on Demand caching solution was described as being designed and implemented for multi-site operation. OpenFlow based Video on Demand caching solution is determined to bring an efficient, transparent and highly configurable caching service for Video on Demand content and address the underlying challenge that the network faces when the same video files are streamed to end-users repeatedly using independent unicast flows. Also, we augmented OpenCache's functionality to handle the addition and removal of multiple caches and also added functionality to pre-cache and load-balance multiple caches' contents.

OpenFlow based Video on Demand caching solution aims to provide the following key benefits;

- Provision of cache as a service by offering an interface to declare cacheable content of interest in an open, highly configurable and flexible manner.
- It supports centrally controlled caching that provides the forum for many additional services to be programmed on top of it with ease (e.g. load balancing, pre-caching)
- It is easily deployable in a production network; there are no changes required in the underlying delivery video mechanisms and all existing hardware and software can be retained.
- It is fully transparent to the end-user. The user does not need to install any extra software, or have to sacrifice any of his local network or storage resources to stream video content with high efficiency, which other technology requires.
- It provides caching very close to the user that allows the external link usage to get reduced and the overall network utilisation to be improved as end-user requests will now be served locally. However, since the content is served locally, the video client will observe higher throughput, lower latency, higher video

quality and smaller start up and buffering times, eventually leading to higher quality of experience for the end-user.

6.2 FUTURE WORK

As part of our future work, we plan to improve OpenFlow based Video on Demand caching solution by exploring the cache placement problem in conjunction with different caching policies. We hope to identify the beneficial position of being in close proximity to end-users whilst maximising the use of the caching resources. In addition, we plan to extend OpenFlow based Video on Demand caching functionality and evaluate it further in different environments and against other commercial or research based caching services.

REFERENCES

[1] Is-Haka Mkwawa, Alcardo Alex Barakabitze and Lingfen Sun "Video Quality Management over the Software Defined Network" 2016

[2] "Software Defined Networking" https://www.opennetworking.org/sdn-resources/sdn-definition

[3] W.Braun and M Menth "Software Defined Networking Using OpenFlow: Protocols, Applications and Architecture Design Choices", Future Internet, 12 May, 2014

[4] Panagiotis Georgopoulos, Matthew Broadbent, and Nicholace Race "Open Call Deliverable OCC-DS2.1 Design of a Multi-site OpenFlow-assisted Video-on-Demand Distribution Architecture (CEOVDS)" 20 May, 2014

[5] Ting Li, Nathanael Van Vorst, Rong Rong, and Jason Liu "Simulation Studies of Openflow-Based In-Network Caching Strategies" 11 January, 2017

[6] The Zettabyte Era – Trends and Analysis, Technical report, CISCO, 2012.

[7] Visual Networking Index: Forecast and Methodology, 2012-2017. Technical report, CISCO, May 2013

[8] Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2013-2018. Technical report, CISCO, February 2014

[9] W.Simpson and H.Greefied, "IPTV and Internet Video: Expanding the Reach of Television Broadcasting", Focal Press, UK, 2009

[10] Bernard Niyonteze"Provider-side VoD Content Delivery Using Openflow Multicast" 30 June, 2014

[11] Paul Goransson and Chuck Black, "Software Defined Networks - A Comprehensive Approach" June, 2014

[12] Vangie Beal "CDN - Content Delivery Network" http://www.webopedia.com/TERM/C/CDN.html

[13] M. Zhao, P. Aditya, A. Chen, Y. Lin, A. Haeberlen, P. Druschel, B. Maggs, B. Wishon, and M. Ponec "Peer-assisted Content Distribution in Akamai Netsession" , 13th ACM SIGCOMM IMC 2013, pages 31-42, 2013

[14] Microsoft "Delivering Content as a Multicast Stream" https://technet.microsoft.com/en-us/library/cc754435(v=ws.11).aspx

[15] Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue Moon "I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System" 2007

[16] Maureen Chesire, Alec Wolman, Goeffrey M. Voelker and Henry M. Levy "Measurement and Analysis of a Streaming-Media Workload" 2001

[17] Software Defined Networking http://searchsdn.techtarget.com/definition/software-defined-networking-SDN

[18] Open Networking Foundation "Software Defined Networking: The New Norm for Networks", ONF White Paper, 12 April, 2012

[19] "SDN 101: An Introduction to software defined networking" https://www.citrix.com/content/dam/citrix/en_us/documents/products-solutions/sdn-101-an-introduction-to-software-defined-networking.pdf

[20] OpenFlow: The next generation in networking interoperability", white paper, May 2011

[21] OpenFlow Controller http://searchsdn.techtarget.com/definition/OpenFlow-controller

[22] Open-Source software for building Software defined networks http://www.projectfloodlight.org/floodlight/

[23] Guillermo Romero de Tejada Muntaner "Evaluation of OpenFlow Controllers" October 15, 2012

[24] "OpenFlow Switch Specification", Version 1.0.0 (Wire Protocol 0x01), December 31, 2009. http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf

[25] "Open Networking Foundation" https://www.opennetworking.org/sdn-openflow-products

[26] https://www.akamai.com/us/en/resources/vod-server.jsp

[27] https://www.sdxcentral.com/sdn/definitions/sdn-controllers/openflow-controller/

[28] Open-Source Routing and Network Simulation: Using the POX SDN Controller http://www.brianlinkletter.com/using-the-pox-sdn-controller/

[29] M. McCauley, /, November, 2013 http://www.noxrepo.org

[30]https://www.sdxcentral.com/sdn/resources/sdn-controllers/opendaylight-controller/

[31] http://mininet.org/

APPENDIX: SOME OF THE IMPLEMENTATION CODES AND CODES FUNCTIONALITIES

CODE FILES

Code Files	Explanations
1-start-controller-opencache.sh	Starting the opencache controller (OCC) which runs on the main computer
2-start-opencache-node.sh	Starting the opencahe node (OCN) which is run in the `ocn` node in mininet
3-start-mongodb.sh	Starting the database of mongodb for ocn and occ
4-start-floodlight.sh	Starting the floodlight OpenFlow controller on the main computer. It runs floodlight module `webservice` for connecting with opencache controller and `forwarding.hub` to make the switch run as normal when no traffic is redirected to OCN
6-start-console-opencache.sh	Starting the console of opencache controller to add and remove the caching of an IP video-on-demand(VOD) the opencache
7-start-scenario-mininet.sh	Starting the mininet scenario based configuration in `scenario_topology_open_cache.py`
8-start-client.sh	Run by the user inside to simulate the scenario a user requesting a video to VOD
9-start-httpserver.sh	Run inside VOD server to simulate a video-on-demand functionality

1-start-controller-opencache.sh

```
#!/usr/bin/env bash

pushd broadbent-opencache/
```

sudo python ./opencache.py -c --config=config/controller.conf

popd

2-start-opencache-node.sh

#!/usr/bin/env bash

mkdir /data

mkdir /data/db/

mongod --quiet --logpath /dev/null &

pushd broadbent-opencache/

python ./opencache.py -n --config=config/node.conf

popd

3-start-mongodb.sh

#!/usr/bin/env bash

mongod --dbpath=/var/lib/mongoDB

4-start-floodlight.sh

#!/usr/bin/env bash

pushd floodlight

java -jar floodlightV1.2.jar -cf floodlight.properties

popd

6-start-console-opencache.sh

#!/usr/bin/env bash

```
python ./broadbent-opencache/console/console.py
```

7-start-scenario-mininet.sh

```
#!/usr/bin/env bash

sudo python topology/scenario_topology_open_cache.py
```

8-start-client.sh

```
#!/usr/bin/env bash

pushd file_client

wget http://10.0.0.200/sample.mp4

ls

rm sample.mp4

popd
```

9-start-httpserver.sh

```
#!/usr/bin/env bash

pushd file_server

python -m SimpleHTTPServer 80

popd
```

CODE for SCENARIO TOPOLOGY

```
def scenario_topology_open_cache():

  net = Mininet( topo=None,

               build=False,
```

```
            ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )

    ofc=net.addController(name='ofc', controller=RemoteController,

                ip='127.0.0.1',

                protocol='tcp',

                port=6633)

    info( '*** Add switches\n')

    switchlegacy = net.addSwitch('s1', cls=OVSKernelSwitch, failMode='standalone')

    switchOf = net.addSwitch('s2', cls=OVSKernelSwitch, protocols='OpenFlow13' , dpid='1')

    #switchOf = net.addSwitch('s2', cls=OVSKernelSwitch,  dpid='1')

    info( '*** Add hosts\n')


    user = net.addHost('user', cls=Host, ip='10.0.0.1', defaultRoute=None)

    vod = net.addHost('vod', cls=Host, ip='10.0.0.200', defaultRoute=None) #video on-demand
server

    ocn = net.addHost('ocn', cls=Host, ip='10.0.0.50', defaultRoute="10.0.0.150") #open cache
node

    info( '*** Add links\n')

    net.addLink(user, switchOf, port2=1)

    net.addLink(vod, switchOf, port2=2)

    # net.addLink(ocn, switchOf, port2=3)
```

```
    Link(ocn, switchOf, intfName1='ocn-eth0')

    # net.addLink(ocn, switchlegacy)

    Link(ocn, switchlegacy, intfName1='ocn-eth1')

    ocn.cmd('ifconfig ocn-eth1 10.0.10.50 netmask 255.255.255.0')

    occ = net.addNAT(name="occ", connect=switchlegacy, ip="10.0.10.150")

    info( '*** Starting network\n')

    net.build()


    info( '*** Starting controllers\n')

    for controller in net.controllers:

        controller.start()

    info( '*** Starting switches\n')

    net.get('s1').start([])

    net.get('s2').start([ofc])

    info( '*** Post configure switches and hosts\n')

    CLI(net)

    net.stop()

if __name__ == '__main__':

    setLogLevel( 'info' )

    scenario_topology_open_cache()
```

CODE for opencachemongodb.py - Manages the state of the node contents using a MongoDB

```
TAG = 'state'

class State:

"""Initialise state instance with useful objects.

    Instantiated controller and configuration objects are passed for use within this instance.

    Try connecting to the database. Continue to do so until database information is returned

    (and the connection is therefore successful).


    """

  def __init__(self, node):

    self._node = node

    database_test = None

    while (database_test == None):

      try:

        self._client = pymongo.MongoClient(self._node.config['database_host'],
int(self._node.config['database_port']))

        self._database = self._client[self._node.config['database_name']]

        database_test = self._database.command("serverStatus")

      except Exception as e:
```

```
            self._node.print_warn(TAG, "Could not connect to MongoDB database, retrying in 15 seconds.")
            time.sleep(15)


    def create(self, document):
        # return self._database.content.insert_one(document, upsert=True)
        # print document
        result = self._database.content.insert_one(document)
        # pymongo.MongoClient()[].get_collection().insert()
        return result
    def remove(self, document):
        return self._database.content.remove(document)
    def lookup(self, document):
        result = self._database.content.find(document)
        result_obj = []
        for test in result:
            result_obj.append(test)
        return result_obj
```

Code for opencachelib.py: Core OpenCache functionality shared between controller and node.

```
TAG = "lib"
```

```
def setup_logger(log_path, name, verbosity):
    """Setup logging functionality to both console and a rotating log file."""
    logger = logging.getLogger('opencache')
    logger.setLevel(logging.DEBUG)
    create_directory(log_path)
    try:
        fh = logging.handlers.RotatingFileHandler(log_path + name + ".log",
maxBytes=1073741824)
    except IOError:
        fh = logging.handlers.RotatingFileHandler('node.log', maxBytes=1073741824)
    fh.setLevel(logging.DEBUG)
    ch = logging.StreamHandler()
    ch.setLevel(logging.DEBUG)
    formatter = logging.Formatter('[%(asctime)s][%(levelname)s]%(message)s')
    ch.setFormatter(formatter)
    fh.setFormatter(formatter)
    logger.addHandler(ch)
    logger.addHandler(fh)
    logger.verbosity = verbosity
    return logger
```

```
def create_directory(path):
    """Create a new directory if it doesn't exist."""
    while not os.path.isdir(path):
        try:
            os.makedirs(path)
        except Exception:
            raise


def delete_directory(path):
    """Removed an existing directory used for storing cached content specific to this HTTP server's expression."""
    if os.path.isdir(path):
        try:
            shutil.rmtree(path)
        except Exception:
            raise


def expr_split(expr):
    expr_split = expr.split("/", 1)
    root = expr_split[0]
    try:
```

```
        path = expr_split[1]
    except Exception:
        path = ''
    return root, path


def log_debug(tag, string):
    """Print a formatted error message if verbosity level permits."""
    logger = logging.getLogger('opencache')
    if int(logger.verbosity) >= 4:
        logger.debug("[" + tag + "] " + string)


def log_info(tag, string):
    """Print a formatted information message if verbosity level permits."""
    logger = logging.getLogger('opencache')
    if int(logger.verbosity) >= 3:
        logger.info("[" + tag + "] " + string)


def log_warn(tag, string):
    """Print a formatted warning message if verbosity level permits."""
    logger = logging.getLogger('opencache')
```

```
    if int(logger.verbosity) >= 2:
        logger.warn("[" + tag + "] " + string)


def log_error(tag, string):
    """Print a formatted error message if verbosity level permits."""
    logger = logging.getLogger('opencache')
    if (logger.verbosity) >= 1:
        logger.error("[" + tag + "] " + string)


def load_config(config, parser):
    """Load configuration settings from the configuration file"""
    for name, value in parser.items('config'):
        config[name] = value
    return config


def do_json_rpc_post(_id, method, host, port, _input=None):
    """Format JSON request as JSON-RPC with HTTP POST."""
    if _input==None:
        try:
            post_data = json.dumps({"id":_id, "method":method, "jsonrpc":"2.0"})
```

```
            return do_json_post(host=host, port=port, post_data=post_data)
        except Exception as exception:
            log_error(TAG,  "Could not encode JSON: %s" % exception)
    else:
        try:
            post_data = json.dumps({"id":_id, "method":method, "params":_input, "jsonrpc":"2.0"})
            return do_json_post(host=host, port=port, post_data=post_data)
        except Exception as exception:
            log_error(TAG,  "Could not encode JSON: %s" % exception)


def do_json_rest_post(host, port, _json, path):
    """Format JSON request as REST with HTTP POST."""
    try:
        return do_json_post(host=host, port=port, post_data=_json, path=path)
    except Exception as exception:
        log_error(TAG,  "Could not encode JSON: %s" % exception)


def do_json_rest_delete(host, port, _json, path):
    """Format JSON request as REST with HTTP DELETE."""
    conn = httplib.HTTPConnection(host + ':' + port)
```

```
    conn.request('DELETE', path, _json)
    resp = conn.getresponse()
    content = resp.read()
    return content


def do_json_post(host, port, post_data, path=None):
    """Make a JSON call to given destination."""
    httplib.HTTPConnection._http_vsn = 10
    httplib.HTTPConnection._http_vsn_str = 'HTTP/1.0'
    if path == None:
        url =  "http://%s:%s" % (host, port)
        try:
            response_data = urllib2.urlopen(url, post_data, 30).read()
            try:
                response_json = json.loads(response_data)
                return response_json
            except Exception as exception:
                log_error(TAG,  "Could not decode JSON from response: %s" % exception)
        except Exception as exception:
            log_error(TAG,  "Could not connect to %s:%s: %s" % (host, port, exception))
```

```
            raise
        else:
            url =  "http://%s:%s%s" % (host, port, path)
            try:
                response_data = urllib2.urlopen(url, post_data, 30).read()
                try:
                    response_json = json.loads(response_data)
                    return response_json
                except Exception as exception:
                    log_error(TAG,  "Could not decode JSON from response: %s" % exception)
            except IOError as exception:
                log_error(TAG,  "Could not connect: %s" % exception)
                raise
    def send_json_rpc_result(JSONHandler, request_id, result):
        """Send JSON result message in response to successful call."""
        JSONHandler.send_response(200)
        JSONHandler.send_header("Content-type", 'application/json')
        JSONHandler.end_headers()
        try:
            message = json.dumps({"id":request_id, "result":result, "jsonrpc":"2.0"})
```

```
    except Exception as exception:
        log_error(TAG,  "Could not encode JSON: %s" % exception)
    JSONHandler.wfile.flush()
    JSONHandler.wfile.write(message + '\n\r\n')
    JSONHandler.wfile.flush()
    JSONHandler.wfile.close()
    return


def send_json_rpc_error(JSONHandler, request_id, code, data=None):
    """Send JSON error message in response to failed call."""
    error = {}
    error["code"] = code
    if code == '-32700':
        error["message"] = 'Parse error'
    elif code == '-32600':
        error["message"] = 'Invalid Request'
    elif code == '-32601':
        error["message"] = 'Method not found'
    elif code == '-32602':
        error["message"] = 'Invalid params'
```

```
    elif code == '-32603':
        error["message"] = 'Internal error'
    JSONHandler.send_response(200)
    JSONHandler.end_headers()
    if data != None:
        error["data"] = str(data)
    message = json.dumps({"id":request_id, "error":error, "jsonrpc":"2.0"})
    JSONHandler.wfile.write(message)
    JSONHandler.wfile.close()
class RemoteProcedureCallError(Exception):
    def __init__(self, data, code):
        self.data = data
        self.code = code
    def __str__(self):
        return repr(self.data)
```