ABSTRACT

MARMAVULA, RAMACHANDRA KASYAP. Comparative Study of a Proposed OpenFlow Agent Enhancement to support Flexible Payload Match. (Under the direction of Dr.Rudra Dutta).

In recent years, the Internet has carried an ever-increasing variety of application traffic, with diverse application domains including educational, commercial, public service. Both for traffic and network management, and an increasing desire to offer value-added services, ISPs have increasingly sought to provide application-aware traffic processing in their networks. While the current generation of commercially available switches and routers perform many enhanced functions, such as firewalling, NAT and ALGs, most advanced application-aware services such as IDS still require the use of dedicated middleboxes.

The recently emerging paradigm of Software Defined Networking offers the opportunity both for enhanced processing at the forwarding engine level, and more centralized and agile policy management, by clean separation of the point of definition (the controller) and point of application (forwarding engine) of the policy. OpenFlow, an open standard for the communication API between the controller and the forwarding engine, has become the de facto standard for this concept. However, OpenFlow allows policies to be installed for traffic flows defined only in terms of L2-L4 headers, which does not allow application-aware policies to be realized. While such enhancements to the policy can be easily implemented on the controller, this is not scalable.

In this thesis, we consider the general problem of extending the OpenFlow API minimally, to allow definition of flows matching an arbitrary sequence of bytes at an

arbitrary location in the application header or payload of a packet, and extending an existing OpenFlow agent to recognize these API extensions and performing the extended matches in the switch itself. Such an ability will have a very broad range of possible use cases, with different tradeoffs. We articulate the issues necessary to consider, propose an architecture and realize it, and study the tradeoffs in a few specific use cases with distinctly different characteristics. Our results show that our architectural approach is a feasible one, and offers a beneficial tradeoff over using controller applications for such functions.

Comparative Study of a Proposed OpenFlow Agent Enhancement to support Flexible Payload Match

by
Ramachandra Kasyap Marmavula

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfilment of the
requirements for the degree of
Master of Science

Computer Science

Raleigh, North Carolina

2014

APPROVED BY:

_____	_____
Dr. David Thuente	Dr. Tzvetelina Battestilli

_____
Dr. Rudra Dutta
Chair of Advisory Committee

DEDICATION

To my parents and my younger sister

BIOGRAPHY

Ramachandra Kasyap Marmavula is from Nellore, Andhra Pradesh, India. He received his Master of Science (Technology) degree in Information Systems from Birla Institute of Technology and Science, Pilani, India in 2009. After graduation, he worked as a Software Engineer at Embedded Infotech Pvt. Ltd. (TeamF1 Networks) in Hyderabad, India till mid-2010. He also worked at Juniper Networks India Pvt. Ltd. in Bangalore, India as Software Engineer-II, till 2012 before joining the Department of Computer Science at North Carolina State University as a Master's student. During his graduate study, he has worked as a Research Assistant with Dr. Rudra Dutta and Dr.Mihail Sichitiu on various projects related to the Centennial Mesh Network (CentMesh) testbed.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr.Rudra Dutta for his invaluable guidance. I am grateful to him for the opportunities provided and his guidance throughout the graduate program. His insight, encouragement and constructive criticism were vital in completing this work.

I would like to thank Dr.David Thuente and Dr. Tzvetelina Battestilli for serving on my advisory committee.

I would like to thank Dr.Mihail Sichiu for providing me the flexibility to balance my Thesis work with work from other projects that I was working with him.

Special thanks to Rob for introducing me to the various tools used for this Thesis and also offering a helping hand whenever needed. I would also like to thank Anand, Mani Bharathi and Shashaankar for their useful suggestions during various phases of my work.

I would also like to thank all my friends for their help and encouragement in my graduate study.

I am grateful to my parents and my younger sister for their endless love and support throughout my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LISTINGS

CHAPTER 1 - Introduction

With appliances like smartphones and tablets becoming ubiquitous and an estimated one-third of worlds' population online, internet usage has exploded in the recent years. This explosion is not limited to a single domain, but encompasses commercial, education, health and many more. Access to high-definition video, online games and increasing amount of P2P traffic continue to test bandwidth limits. These trends push Internet Service Providers (ISPs) to update their network deployment with higher processing power and more bandwidth capability. This push has in-turn increased focus on application-aware traffic processing to quantify and improve user's experience when accessing bandwidth-intensive operations like streaming high-definition videos or file sharing. Though the existing commercial switches and routers perform enhanced functions (like Network Address Translation or Firewalling) apart from the traditional expected functionality of switching or routing, most networks require specialized network appliances (middleboxes) for certain services. The term middlebox covers a broad range of devices including Wide Area Network (WAN) optimizers, load balancers, Intrusion Detection System (IDSs). A discussion on the significance of such middleboxes in the network is presented in [4]. It is also noted that the number of middleboxes in an enterprise network is comparable to the number of switches and routers in the network [3].

Companies interested in introducing new functionality in their network typically have to wait for device manufacturers to support that functionality. This can involve multiple development cycles. Through the Software Defined Networking (SDN) paradigm,

opportunities for agile policy management and enhanced processing at the level of forwarding engine are provided. This is made possible by the separation of the data plane & control plane and usage of commodity hardware. Using SDN, organizations can introduce the desired functionality making use of commodity hardware and avoiding long waiting times. On the lines of many other network elements, the application of SDN to middleboxes has been explored and the rate of functionality provision in a middlebox can be improved [7] [8] [9] [10]. OpenFlow is a SDN standard [1] that has the semantics to match header fields of an incoming packet and apply a particular action if the match is found. The current specification allows match-action operations for Layers 2-4 in the Open Systems Interconnection (OSI) model. In this thesis, an enhancement to the existing OpenFlow specification is proposed, using the 'experimenter match' allowed by OpenFlow protocol. This enhancement allows specifying matches in the application layer traffic (Layers 5-7) and would provide complete control over the network packet for Deep Packet Inspection (DPI) functionality. The necessity of providing such a capability for a SDN architecture and the possible ways to support this are discussed in [28]. The intelligence to perform DPI on application traffic can be co-located with the control plane or it can be part of the data plane. While co-locating this capability in the control plane can be easily supported, it is not scalable. In this work, we consider the latter approach of enhancing the OpenFlow agent to support matches in the application payload. The enhancement supports defining flows to match an arbitrary sequence of bytes in the application payload and performing these extended matches inside the datapath.

We demonstrate the usefulness of this enhancement by implementing two

applications that require information gleaned from Layers 5-7 to function. One application is an FTP Application Level Gateway (ALG). The other is 'HTML scanner', an application which tracks the number of HTTP connections that render HTML5 pages. These applications are implemented using both - the current OpenFlow specification and the proposed enhancement. We make a comparison between the two approaches by considering metrics such as response latency and the additional traffic generated by the datapath. With OpenFlow's inherent semantics to perform matches, we demonstrate that enhancing OpenFlow to allow a generic payload match would be a better approach than using the current OpenFlow specification for the applications under consideration.

1.1 Software Defined Networking and OpenFlow

In recent years, Software Defined Networking (SDN) has gained traction as a paradigm that allows rapid deployment of services by providing a clean abstraction between control plane and data plane [2]. OpenFlow is a leading SDN standard that has been introduced by Open Networking Foundation, a user-driven organization that focuses on adaption of SDN. Working with OpenFlow allows researchers to develop network protocols and experiment the same using production traffic [2]. OpenFlow-based Software Defined Networks have been deemed useful in many scenarios and networks owing to their centralized control, provision of programmability and ability to rapidly deploy services [5] [6]. An OpenFlow network typically consists of a controller and one or more switches. An OpenFlow switch consists of an agent that communicates with the controller using the OpenFlow protocol and a datapath. OpenFlow provides an interface that allows programmers to develop applications. These applications interact with the switches by installing flows to the switches.

Installation of these flows can occur proactively or reactively.

Flows are policies that determine how a switch functions. A standard forwarding engine can function as a switch (in which case it forwards based on Ethernet header), a VLAN aware switch (in which case it also uses VLAN tags), an IP router (in which case it forwards based on IP headers) and so on. It can also function a multi-protocol router, which implies that it is capable of doing all of the operations listed above, but it has to be configured accordingly, which may result in an increased cost for such devices. OpenFlow allows the forwarding engine to be commoditized so that it is not required to decide beforehand how the forwarding engine functions - as a switch, as a router or as a firewall. In fact, any of these functions can be mixed and matched. This 'policy' can be changed at run-time without the need of resetting either the forwarding engine or the controller, by simply changing the software on the controller. It is agile, because it is software-defined. The controller is highly specialized software, but it can run on commoditized hardware because it is not operating at wire speed.

Though referred generally as a single application, a controller typically has two components:

- 'Controller software' that implements OpenFlow protocol and communicates with the switch
- An application that uses the interfaces provided by the controller software and determines the policies that controls the actions of the switch. This application defines how the OpenFlow switch functions (i.e. as a learning switch, as a router etc.)

There can be one or more controller applications that use the controller software to communicate with the switch. Following figure is a representation of the same:

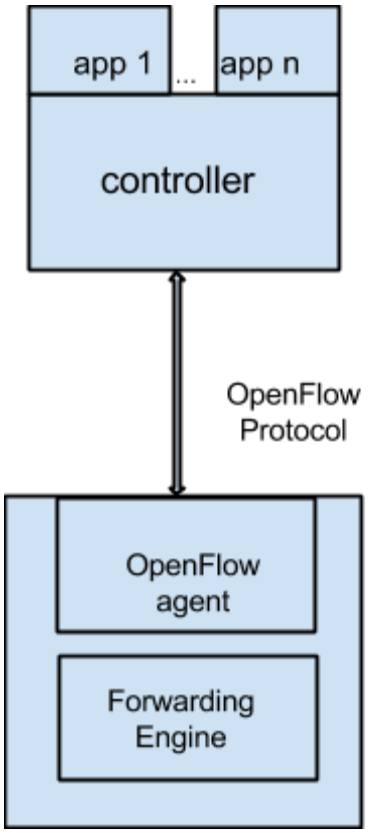

Figure 1: Representation of OpenFlow components

An OpenFlow switch maintains ‘flow tables’. These flow tables contain flows that are installed by the controller. Each flow has six fields which include a set of ‘match’ parameters and an action. Following is a brief description of each field:

1. match fields: fields to match against incoming packet.
2. priority: match’s precedence.
3. instructions: actions to apply on matched packets. These include pipeline processing, modification of packet headers and action set of the packet.
4. counters: statistics that track the number of times a packet has matched a particular flow.

5. timeouts: maximum duration a flow has until expiration.
6. cookie: identifier used by the controller to filter flow statistics and modification or deletion of flows.

An incoming packet in the datapath is monitored to check if it matches any of the flows and if matched, the corresponding instruction in the flow is applied on that particular packet. The datapath uses the highest-priority matching rule to determine a matching flow. The behavior of the forwarding engine depends on the flow installed by the controller application. Thus, the controller application can enforce that the forwarding engine behave as a learning switch, IP router etc. The behavior of this controller application (and thus the actions of the OpenFlow agent) can be changed during run-time. This change of behavior can be facilitated by adding new flows or deleting/modifying existing flows. OpenFlow has the semantics to perform each of these operations [1].

We include a more detailed description of OpenFlow in Chapter 2 and Chapter 3.

1.2 Problem Statement

The SDN paradigm encourages rapid deployment of new services and reduction in capital and operating expenses (CapEx/OpEx) for companies. The flexibility offered by SDN allows network operators to introduce new features using commodity hardware rather than depend on device vendors. OpenFlow is the de facto early standard for SDNs and offers match-action semantics. The current specification of OpenFlow supports a multitude of these ‘match fields’ [1]. These include matches for IP addresses (IPv4/IPv6), MAC addresses, VLAN IDs, TCP and UDP port numbers, MPLS related fields etc. Invariably all these matches support fields only in the packet headers (L2-4). There is also an

‘experimenter match’, which is in place to support matches for new protocol fields. An experimenter match can be supported by any controller-switch combination that can parse and interpret the match.

To realize application-aware traffic processing, there is a requirement to support matches that can work on application traffic (Layers 5-7). For example, a middlebox performing the functionality of a FTP Application Level Gateway (ALG) would need to determine when a FTP server in passive mode is opening a new port for data traffic and allow traffic on that port for the duration of the connection [17]. Similarly, a system that tracks phases of interest needs to perform DPI in the application payload for signature recognition. To perform DPI in the application payload of the packet, one possible solution is to forward packets that match a basic requirement to an external ‘packet processor’. The packet processor can be a standalone machine or just another OpenFlow managed box. In case of the FTP ALG, every control packet from the server should be forwarded to this packet processor for verification of the signalling information. This packet processor would then need to perform the following actions:

1. Determine if the packet matches an extended set of requirements. For example, in case of the FTP ALG, this involves verifying the FTP action code to determine if the current packet contains data port information.
2. For packets that match the extended set of requirements, apply the pre-decided action. For example, in case of the FTP ALG, this involves adding a flow for allowing the new data connection.

The first part of the two actions is essentially an extension of the match that the OpenFlow agent performs. The idea presented in this work is that it is beneficial to perform this extended match in the datapath of the OpenFlow switch. This is realised by enhancing the existing OpenFlow agent to allow definition of flows matching an arbitrary sequence of bytes at an arbitrary location in the application header or payload of a packet, and extending an existing OpenFlow agent to recognize these API extensions. We evaluate the performance impact of such matches in the datapath of the switch and analyse applications that suit this criteria. We demonstrate that the extension improves utility with little or no change in complexity for a broad range of applications.

1.3 Thesis organization

The rest of the thesis is organized as follows. Chapter 2 provides a brief overview of OpenFlow as it is standardized today. Chapter 3 presents the challenges in designing the proposed enhancement. Chapter 4 describes the architecture and implementation of the desired enhancement. Chapter 5 presents the use cases (FTP ALG and in-transit HTML scanner) and some results based on these. Chapter 6 concludes the thesis and briefly discusses future research directions.

CHAPTER 2 - Background

2.1 OpenFlow-introduction

In this section, we present the key components of the OpenFlow protocol. This introduction forms the basis for the changes involved in the proposed enhancement.

2.1.1 OpenFlow - matches and actions (specification 1.3.2)

An OpenFlow switch contains multiple flow tables, each containing zero or more flows. Each flow has a set of match fields along with a set of instructions and counters. Some of the required match fields that must be supported include the following [1]:

- Ingress port
- Ethernet Destination address
- Ethernet Source address
- Type of protocol in Ethernet header
- IPv4 source address
- IPv4 destination address
- IPv6 source address
- IPv6 destination address
- IPv4 or IPv6 protocol number
- TCP source port
- TCP destination port
- UDP source port
- UDP destination address

The flow match fields are specified using a Type-Length-Value (TLV) format called the OpenFlow Extensible Match (OXM) [1]. Each OXM TLV has a header that is 4 bytes long and a payload that is 1 to 255 bytes long. The payload contains the value of the OXM field. The header fields of the OXM TLV and their description is given in the following table [1]:

Table 1: OXM TLV Header fields (from [1])

Name	Width (bits)	Description
oxm_class	16	Match class: member class or reserved class
oxm_field	7	Match field within the class
oxm_hasmask	1	This bit is set if OXM includes a bitmask in payload
oxm_length	8	Length of OXM payload

The default set of match fields are specified by OFPXMC_OPENFLOW_BASIC class [1]. For OFPXMC_OPENFLOW_BASIC class, the OXM payload specifies the match and optionally, a bit mask. For example, if 'oxm_type' refers to Source IPv4 address and 'oxm_hasmask' is true, then the value of 'oxm_length' is eight bytes - four bytes of IPv4 address and four bytes of the network mask. The specification of a mask is not allowed for certain 'oxm_type' (example: TCP port number). The other class is referred as the 'OFPXMC_EXPERIMENTER' and it is used for specifying experimenter matches. This class is used in implementing the proposed enhancement to the existing OpenFlow agent.

Each flow has also an associated instruction set. When a packet matches the flow entry, this

instruction set is applied and it may result in changes to the packet (header modification) or the action set of the packet. The packet's 'Action set' is explained in Section 2.1.2. It may also result in changes to pipeline processing. OpenFlow specification recommends many instructions, some of which are mandatory to be supported by all OpenFlow switches:

- (Required) Goto-Table <next-table-id>: A packet that matches the entry is forwarded to another table.
- (Required) Write-Action: Merges the actions to the existing action set of the packet.
- (Optional) Clear-Action: Clears all actions in the action set of the packet.

An instruction set can contain both 'Write-Action' and 'Goto-Table' instructions. In such a scenario, the 'Write-Action' is applied first followed by a 'Goto-Table' instruction.

Our work does not involve enhancing the existing supported instructions. The focus is only on enhancing the match capability.

2.1.2 Action Set

Each packet that enters the datapath has an associated action set, which is empty by default. This action set can be updated by either a 'Write-Action' instruction or a 'Clear-Action' instruction. When a packet matches a flow entry and the instruction set of the flow entry does not contain a 'GoTo-Table' entry, the actions in an action set are applied to the packet. The OpenFlow specification specifies several recommended actions, some which must be supported by all OpenFlow switches [1]:

- (Required) Output: Directs the OpenFlow switch to direct the packet out of a particular port.
- (Required) Drop: Directs the OpenFlow switch to drop the packet.

- (Optional) Set-Field: Directs the OpenFlow switch to modify fields of a packet's header.

2.1.3 Pipeline processing

Multiple flow tables exist in the OpenFlow switch and each table is referred by a flow table number. These tables are sequentially numbered starting at zero. The flow table number to which a particular flow is to be installed is configured by the controller application. Flow processing for a packet that enters the datapath starts at table number zero. The packet is checked against each flow entry to verify if there is an exact match. If a match is found in a particular table, then the packet may be directed to another flow table (or not) depending on the instruction set of the flow entry. A packet that has found a match in a flow table number 'X' can be directed only to a table with a higher flow table number.

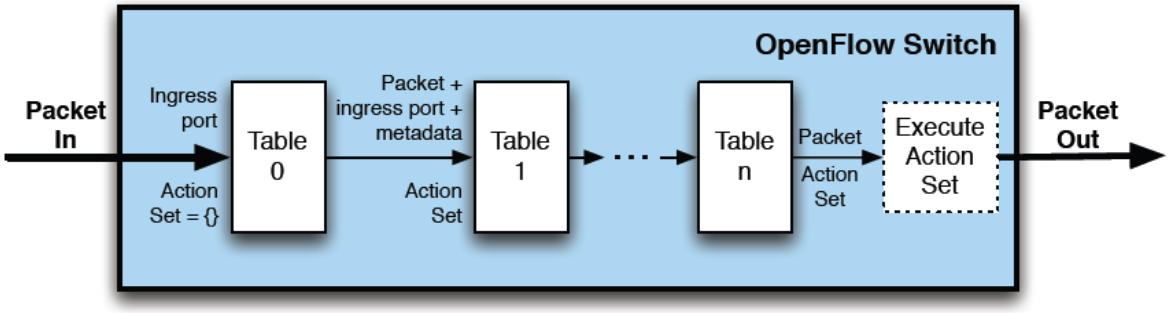

Figure 2: Pipeline processing in the OpenFlow agent (reproduced from [1])

2.1.4 Ordering of flows in a flow table

Flows are ordered based on decreasing order of a field called priority [1]. Before installing a flow that is configured by the controller, the OpenFlow agent verifies if there is a flow with the same match fields and same priority. If there is a conflicting flow, the entry can be

replaced. If there are multiple matching flows with same priority, then the flow selection is undefined. Thus, it is up to the controller application to differentiate flows by specifying the desired priority and avoid overlapping flows.

2.1.5 Table-miss flow entry

When a packet is matched against all the flows in a flow table and if there is no match, it is referred as a 'table-miss' [1]. This entry has a priority of 0 (the least) and the match fields are wild-carded. A 'table-miss' operation has an associated action. Following is the list of possible actions taken on the packet:

- the packet is dropped
- the packet is forwarded to the controller over the OpenFlow channel
- the packet is directed to another table (whose ID is greater than the ID of the current table).

This action is configured in a 'table-miss' flow that should exist in each table. If a table does not contain a 'table-miss' flow entry and a packet does not match any of the other flows in the table, the OpenFlow switch drops the packet.

2.2 Use case background

2.2.1 File Transfer Protocol (FTP)

File Transfer Protocol (FTP) is a standard network protocol that specifies the transfer of files over a TCP connection [18]. An FTP request involves two connections between the client and the server– the control connection and the data connection. The control connection is established first and is used, among other purposes, for authentication. Data connection is only used whenever the client requires to download or transfer files. FTP works in two modes: 'active open' and 'passive open' [17]. The two modes are distinguished in the way the data connection is established. In case of 'active open', the client opens a TCP port for data connection and the server connects to that port over TCP port 20. In case of 'passive open', the server opens a TCP port for data connection and the client connects to that port. We focus on FTP passive mode in this work as that is more relevant for servers and whenever Network Address Translation (NAT) is involved [17]. 'Passive open' is initiated by the client by sending 'PASV' command to the server. In the response to the 'PASV' command, the server specifies the port number to be used for data connection [18].

We use the FTP Application Level gateway as one of the use cases for the enhanced OF agent.

2.2.2 HTML5

HTML is a mark-up language that is used for structuring and presenting web pages. It consists of a set of tags (referred as the HTML elements), that can be rendered by web browsers. HTML5 is the fifth revision of the HTML standard [19]. HTML5 adds new tags to the existing set and these include the <video> and <audio> tags. The addition of these

tags allows the integration and rendering of multimedia and graphical content without having to work with proprietary plug-ins [30]. An application-aware use case might be to recognize HTML5 and funnel them through in-network services or a different server. We use 'HTML scanner', an application that tracks the number of HTTP connections that render HTML5 pages as a use case.

CHAPTER 3 Proposed architecture of desired enhancement

3.1 Challenges

An Experimenter Match that allows specification of matches in the application payload requires changes both to the controller software and the OpenFlow agent. The controller software is to be enhanced to support configuration of flows with the extended match. Similarly, the OpenFlow agent should detect the match specifications in the configured flow(s) and match packets in the datapath appropriately. The real challenge for supporting such a match is making it OpenFlow-compatible. The architecture for the enhanced OpenFlow agent and the controller software is designed ensuring the existing OpenFlow specifications are met.

3.2 Design

OpenFlow match fields are specified using a Type-Length-Value (TLV) format called the OpenFlow Extensible Match (OXM). The OXM TLV header is four bytes in length and has four fields [1]:

- oxm_class
- oxm_field
- oxm_hasmask
- oxm_length

The oxm_class can either have 'OFPXMC_OPENFLOW_BASIC' or 'OFPXMC_EXPERIMENTER' as its values. For the proposed enhancement, we make use of 'OFPXMC_EXPERIMENTER' class. An Experimenter Class requires an Experimenter

ID to be issued. This is a 32-bit value that uniquely identifies the experimenter [1]. An arbitrary Experimenter ID (3735928559) is selected, which can be easily recognized during communication by its hex equivalent (0xDEADBEEF). Experimenter IDs with zero as the most significant byte have the following three bytes as the experimenter's IEEE OUI and this particular ID was selected outside the range of IEEE OUI designated naming space.

'oxm_field' is a class-specific entity. For example, with the basic class for OpenFlow (OFPXMC_OPENFLOW_BASIC), following are the values for different fields (this is not an exhaustive list):

Table 2: Field Mappings for the OXM TLV basic class

Description	oxm_field value
Switch input port	0
Switch physical input port	1
Ethernet destination address	3
Ethernet source address	4
IP Protocol	10

For the Experimenter Match used in this enhancement, we use an arbitrary value of '100' for the field. Since this field is interpreted based on the value of the Experimenter ID, there is flexibility to choose a value from the range 0-127 (the number of bits used for oxm_field is seven). The syntax of the Flexible Payload Match (FPM) is actually specified in the OXM payload and is discussed in detail in the following section.

3.2.1 Flexible Payload Match (FPM) fields

The FPM contains four fields which are described in the following table:

Table 3: Description of FPM fields

Name of the FPM field	Description
start_offset	Offset relative to the start of the payload from which the search is to be performed.
end_offset	Offset relative to the start of the payload beyond which search need not be performed.
length_match	Length of the payload match.
value	The actual value that is to be matched.

For example, we consider a FPM which has the following values:

- start_offset: 10
- end_offset: 15
- length_match: 4
- value: ASCII equivalent of ‘Port’

If a packet with payload containing 500 bytes has to be matched, the search is performed from the 10th byte to the 15th byte. Thus the following sets of bytes are matched for the availability of the pattern (‘Port’):

- 10-13

- 11-14
- 12-15
- 13-16
- 14-17
- 15-18

There can be cases for which the exact byte offsets are unknown. To support such matches, length_match is set with a value of zero. Specifying thus would indicate that the rest of the packet is to be searched. For example, consider a FPM which has the following values:

- Start_offset: 10
- End_offset: Not considered
- Length_match: 0
- Value: ASCII equivalent of 'Port'

If a packet with payload containing 500 bytes has to be matched, the search is performed from the 10th byte till the end of the packet. Thus the following sets of bytes are matched for the availability of the pattern ('Port'):

- 10-13
- 11-14
- 12-15

  ...

- 497-500

If the size of the match is greater than the difference between start offset and end of the packet, then no set of bytes is matched. For the above example, if a packet containing a

twelve-byte payload enters the datapath, then no byte set is searched (as the start offset is ten and the length of the match is four bytes).

3.2.2 Format of the OXM TLV for FPM

The first four bytes of the payload of an OXM TLV that contains an Experimenter Class must contain the Experimenter ID [1]. As noted earlier, the Experimenter ID has been chosen as the value 3735928559. Thus, for the proposed enhancement, the format of the OXM TLV looks as follows:

oxm_class	oxm_field	HM	Payload length	Experimenter ID	Start_offset	End_offset	Length of match	Actual match
2 bytes	7 bits	1 bit	1 byte	4 bytes	2 bytes	2 bytes	1 byte	variable length

Figure 3: Format of the OXM TLV for FPM

The maximum length of an OXM TLV is 259 bytes [1]. Since the OXM header is four byte-long, the OXM payload can be no longer than 255 bytes. With the FPM, the permissible length of the actual value is constrained by the fact that other fields are specified in the OXM payload and occupy space - the experimenter ID is four byte long, the start and end offset are two bytes long each and a byte is required for the length_match field.

Thus, the maximum length of the value that can be specified using FPM is 246 bytes (255-4-2-2-1).

CHAPTER 4 Implementation

Implementing the proposed architecture involves changes to the existing controller software and the OpenFlow agent. Existing controller applications require enhancement to specify matches using the Flexible Payload Match (FPM).

4.1 Controller enhancements

The controller software is enhanced to support the following:

- Provision of APIs (for the controller application) to specify matches using FPM.
- Encapsulation of the OXM TLV with the Experimenter Match according to OpenFlow specification [1]

4.1.1 Enhancing the controller 'Ryu'

The controller platform for this work is Ryu [11]. Though there are many ongoing projects in various languages offering controller software, Ryu was chosen as it supports OpenFlow 1.3, which was the latest OpenFlow version available when we started evaluating controllers. OpenFlow 1.4 specification has been announced later, but the format of OXM TLVs has not changed [22]. Thus, the architecture proposed in this work still holds relevance. Ryu is written in python. All changes have been made using [12] as the reference branch.

A new class 'MTExpField' that extends 'OFPMatchField' has been introduced. This class contains the methods to 'pack' the specified payload match to communicate with the switch over the OpenFlow channel.

Two new methods have been added to the class 'OFPMatch':

- append_exp_field (start_offset, end_offset, length_match, value) - This method adds

a new experimenter field to the given match.

- validate_exp_fields (start_offset, end_offset, length_match, value) - This performs relevant validation on the input fields specified in the FPM. The validations include verifying if the length of the match (specified by length_match) matches the actual length of the payload value. Also, the 'length_match' field is evaluated to determine if it exceeds 246 bytes (as referred in Section 3.2.2, this is the maximum allowed value for the length of the FPM).

4.2 OpenFlow switch enhancements

The existing OpenFlow switch is enhanced to support the following:

- Decoding the OpenFlow FPM communicated by the controller when installing a flow.
- Saving the decoded FPM after relevant validation to the configured flow table.
- Performing the match operation in the datapath by interpreting start_offset, end_offset, length_match and value fields.

4.2.1 Enhancing the soft-switch 'LINC'

Software switches provide a programmable platform which is not usually found with hardware switches (unless the vendors explicitly support it). Currently, there are many ongoing projects in various languages that provide soft-switches. The OpenFlow switch platform used for this work is Link Is Not Closed (LINC) from 'FlowForwarding' [13]. LINC is a software switch written in Erlang. LINC was chosen because of its support for OpenFlow 1.3. All changes have been made using [14] as the reference branch.

As explained in Section 2.1.1, the match fields in OpenFlow specification are described by the OXM TLV. LINC represents the match fields as a list of 'ofp_field' record in Erlang.

An Erlang record is similar to a C Structure [15]. Listing 1 illustrates the code snippet of 'ofp_field' as an Erlang record.

```
-record(ofp_field, {
        class = openflow_basic :: ofp_field_class(),
        name :: ofp_field_type(),
        has_mask = false :: boolean(),
        value :: bitstring(),
        mask :: bitstring()
        }).
```

Listing 1: Representation of OXM TLV as an Erlang record

Comparing this listing with the contents of Table 1, the mapping is self-explanatory. For the FPM, a few fields are added to the existing record structure. Listing 2 illustrates the code snippet for the updated Erlang record.

```
-record(ofp_field_exp, {
        class = ofp_field_class(),
        name :: ofp_field_type(),
        has_mask = false :: boolean(),
        value :: bitstring(),
        mask :: bitstring(),
        exp_id :: bitstring(),
        start_offset :: integer(),
        end_offset :: integer(),
        length_match :: integer()
        }).
```

Listing 2: Representation of OXM TLV as an Erlang record with added fields for FPM.

The FPM communicated by the controller is decoded and saved into the relevant fields of this record. In the datapath, these fields are compared against the fields extracted from the packet for a match.

4.2.2 Enhancing dependent libraries

LINC makes use of a packet library written in Erlang called 'pkt' [26]. This library supports the datapath operations. When a packet enters the datapath, various header fields of the packet are extracted and stored as 'match fields'. When the packet traverses through the flow tables, the extracted fields are matched against the corresponding fields present in each flow [1]. The 'pkt' library is enhanced to support extraction of application payload along with the

header fields for TCP traffic. All changes have been made using [26] as the reference branch.

4.3 Enhancing the controller-application

The learning switch implementation for OpenFlow 1.3 [16] is used as the building block for implementing the other use cases. In this section, we describe the existing learning switch implementation.

4.3.1 Description of the existing learning switch functionality

Figure 4 illustrates a logical topology for the setup used. Using a PC-like representation for the controller and the agent is to imply the usage of commodity hardware for both these components.

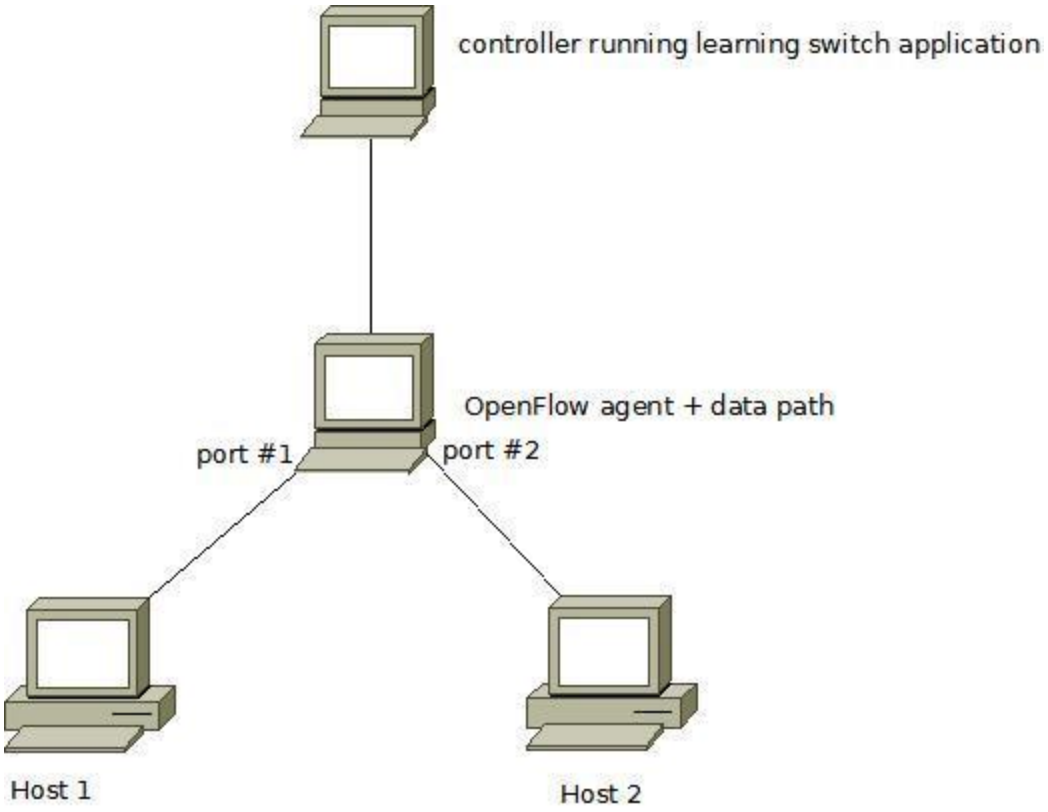

Figure 4: Logical Topology

The controller application is a simple learning switch that inserts rules reactively. When the OpenFlow switch is powered on, it has no flows in any of its flow tables. When the switch establishes a connection with the controller, the 'table-miss' entry in each table is configured to forward unmatched packets to the controller over the OpenFlow channel. Following is the sequence of steps that take place when Host 1 tries to communicate with Host 2:

1. The ARP request generated by Host 1 enters the datapath of the switch and pipeline processing begins at flow table number zero.
2. As no flows exist, the 'table-miss' entry directs the packet to the controller from flow table number zero.

3. The controller application extracts the source Ethernet address from the packet and inserts a flow in flow table number zero. This flow matches all packets that ingress via input port number one and have source Ethernet address of the interface connected to port number. The instruction associated with this flow is 'GoTo table-id one', that directs all matching packets to table with ID one.
4. The controller application also inserts a flow to table-id one. This flow matches all packets that have the destination Ethernet address of the interface connected to port number one. The instruction is 'output the packet using port number two as the egress port'.
5. The packet is then flooded out of all switch ports except the input port (which, in the current case, is port number one).
6. When Host 2 sends a packet in response, Steps (2), (3) and (4) repeat.

After step (5), we have the relevant flows for communication between Host 1 and Host 2. The controller is not involved unless there is traffic between a different set of hosts (for example, if a new system 'Host 3' is connected to the switch and if tries to communicate with either Host 1 or Host 2).

In hindsight, Step (2) may not be necessary. In the absence of this step, the switch ends up directing more packets to the controller. This is explained by the following possible sequence when Host 1 tries to communicate with Host 2 (in absence of step 2):

1. The ARP request generated by Host 1 enters the datapath of the switch and pipeline processing begins at flow table number zero.
2. As no flows exist, the 'table-miss' entry directs the packet to the controller from flow

table number zero.

3. The controller application extracts the source Ethernet address from the packet and inserts a flow to table-id one. This flow matches all packets that have the destination Ethernet address of the interface connected to port number one. The instruction is 'output the packet using port number two as the egress port'.
4. The packet is then flooded out of all switch ports except the input port (which, in the current case, is port number one).
5. When Host 2 sends a packet in response, the packet is sent out of port number one as a flow exists to identify the destination Ethernet address and to choose the appropriate port.

In this entire process, the (destination ethernet address, output port) mapping has not been saved for Host 2. So, each time Host 1 tries to send a packet to Host 2, steps (1), (2) and (3) repeat resulting in more traffic to the controller. Thus, the preferred way is to install two flows per Ethernet address in the flow tables. This functionality is achieved by having a controller application that inserts the following flows to the OpenFlow switch:

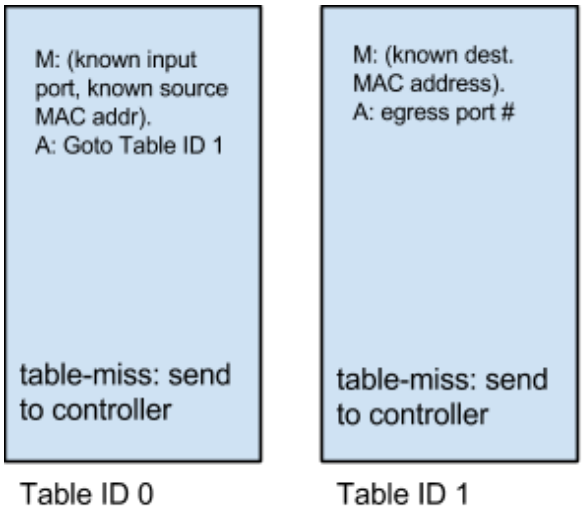

Figure 5: Flow table representation for an OF agent that acts as a learning switch

Listing 3 demonstrates the notation for illustrating the flow entries for a particular application. This notation is used throughout this work.

- Flows are listed in the order of their precedence.
- The alphabet 'M' specifies the match criterion for the particular flow.
- The alphabet 'A' specifies the action applied to packets matching this flow.
- The table-miss flow entry is specified at the end of each flow table.

Listing 3: Notation for representation of flow entries

4.3.2 Use of multiple tables

The steps given in Section 4.3.1 can be implemented using a single table. Following flows would be inserted in the OpenFlow switch for communication between Host 1 and Host 2:

- Match: (source Ethernet address = Ethernet address of Host 1, destination Ethernet

address = Ethernet address of Host 2). Action: output port #2

- Match: (source Ethernet address = Ethernet address of Host 2, destination Ethernet address = Ethernet address of Host 1). Action: output port #1

If there is another system attached to the switch, then four more flows are required for that system to communicate with both Host 1 and Host 2. For example, if 'Host 3' is connected to port #3 (not shown in the Figure), then the additional flows inserted would include the following:

- Match: (source Ethernet address = Ethernet address of Host 1, destination Ethernet address = Ethernet address of Host 3). Action: output port #3
- Match: (source Ethernet address = Ethernet address of Host 2, destination Ethernet address = Ethernet address of Host 3). Action: output port #3
- Match: (source Ethernet address = ethernet address of Host 3, destination ethernet address = Ethernet address of Host 2). Action: output port #2
- Match: (source Ethernet address = Ethernet address of Host 3, destination ethernet address = Ethernet address of Host 1). Action: output port #1

This would severely affect the scalability of this application as it would result in a Cartesian product explosion of rules. This can be contrasted with the use of multiple tables, in which case a new system connected to the switch would result in just two additional flows (one in table-id zero and another in table-id one).

CHAPTER 5 Verification and Validation

We implement two applications that require the payload matching using both – the OpenFlow specification as it stands today and the proposed enhancement. The applications are FTP ALG and HTML scanner. We then compare the performance of each of these applications with the scenario that does not have the proposed enhancement.

5.1 FTP Application-level gateway

An Application Level Gateway (ALG) augments a firewall or a NAT service. In this work, we focus on firewall functionality. The learning switch described in Section 4.3.1 is enhanced to work as a layer-4 firewall where only access to specific TCP services is allowed. For all the examples that are discussed further, only traffic to-and-from FTP and SSH services is allowed. Access is restricted by filtering traffic using the port numbers for SSH and FTP. This functionality is achieved by having a controller application that inserts the following flows to the OpenFlow switch:

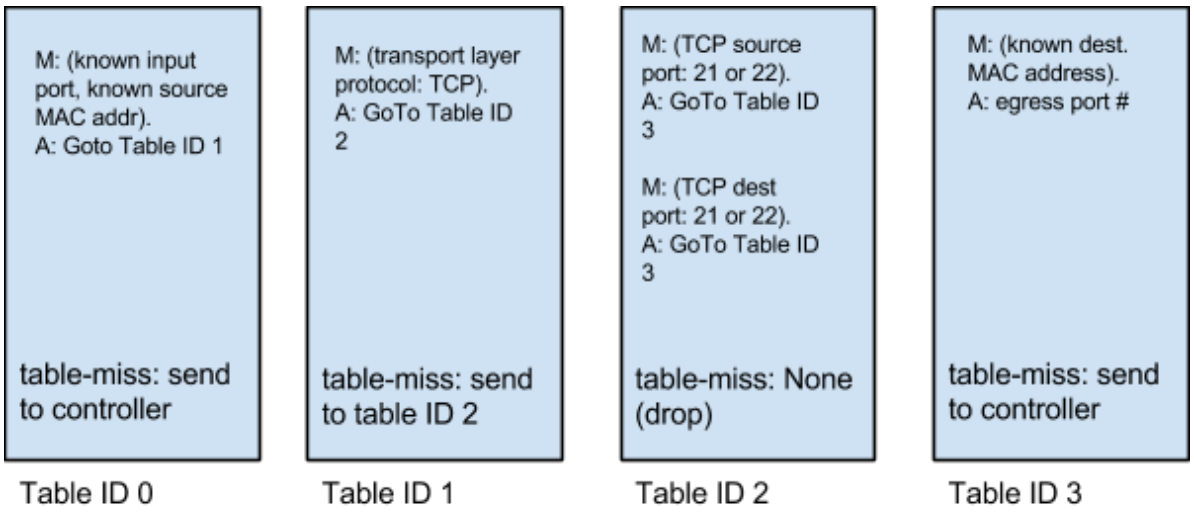

Figure 6 Flow table representation for a firewall that allows only SSH, FTP services

It is evident that from the flow table representation that accessing SSH would work. Even the FTP control connection would get established, but the data connection would not go through. For FTP to work (in passive mode), we need additional functionality to handle the data connection. The firewall would need to determine that the FTP server intends to open a data port and allow traffic on that port. Listing 4 specifies the format of the command [18]. We compare the implementation for the same using the regular OF agent and the proposed enhancement in the following sections.

```
227 Entering Passive Mode.  h1,h2,h3,h4,p1,p2
```

Listing 4: Format of the response by the server to FTP 'PASV' command

5.1.1 FTP ALG using regular OF agent

The existing flows in the Figure 6 do not allow the FTP data connection. When the server

signals the data port information, a flow should be added to allow traffic to/from this port. The traditional OpenFlow specification does not allow specifying matches in the application payload. Thus, it is imperative that packets sent by the server over the control connection must be sent to a 'packet processor' to validate if that packet contains the data port information. Since OpenFlow has semantics to send packets to the controller (using a PACKET_IN message) [1], we make use of it (instead of defining new semantics). So, whenever a packet is sent by the FTP control port, the following sequence of events occur:

- packet is sent to the controller over the OpenFlow channel
- Controller validates the packet to determine if it contains the data port information.
- If it does, a flow is installed by the controller to allow packets to and from that data port. If not, no flow is installed.
- The packet is sent back to the switch and pipe-line processing restarts.

Following figure represents the FTP control traffic flow from client to server, when a regular OF agent is used:

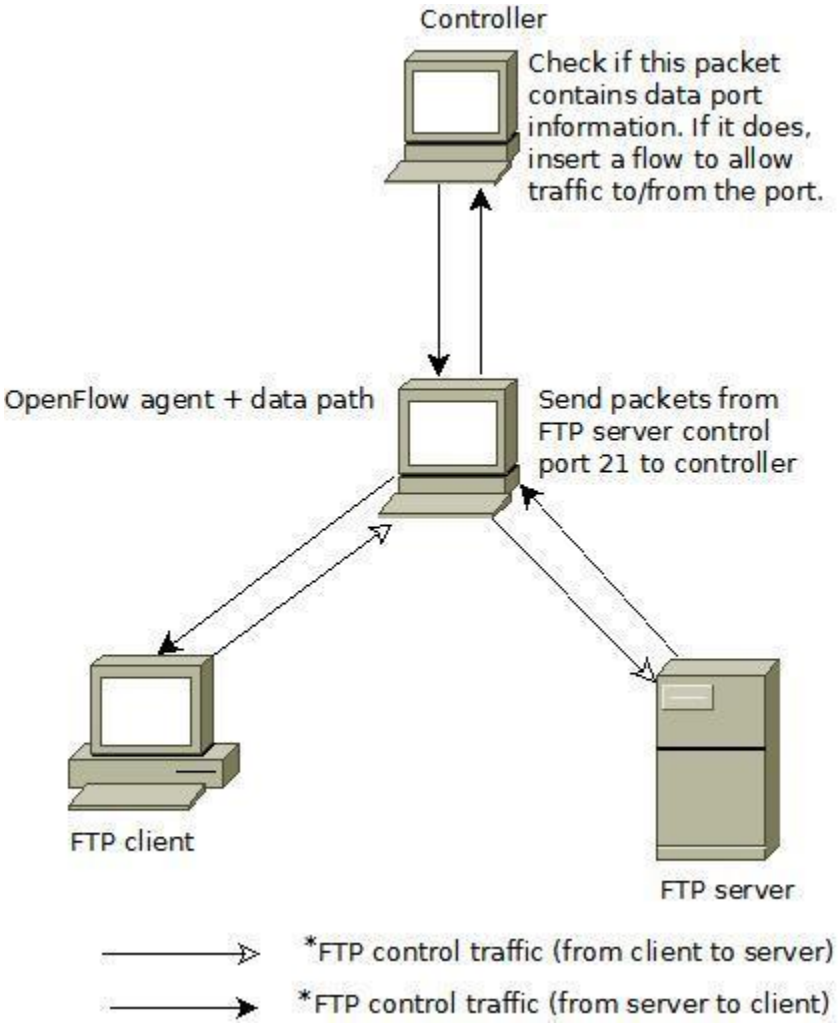

Figure 7: Illustration of the flow of FTP control traffic with the regular OF agent

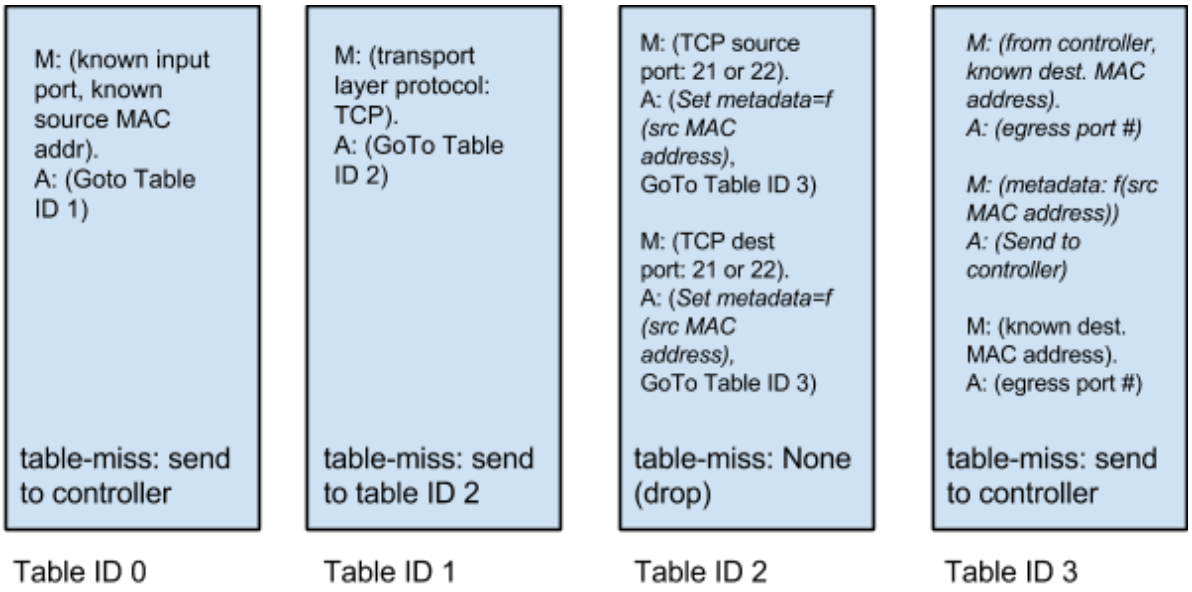

Figure 8: Flow table representation for a FTP ALG using the regular OF agent

To obtain this behaviour and facilitate the working of a FTP ALG, a few additional flows are required to the ones in Figure 6, which are italicized in the Figure 8.

The important point to note is that 'matching' is happening twice for each packet sent from the FTP control port- first at the OF agent and then at the controller to determine if that packet contains data port information. This additional step (of performing a match at the controller) can be avoided if the OpenFlow agent can be enhanced to support matching in the application payload.

5.1.2 FTP ALG using the proposed enhancement

In case of the regular OF agent, every control packet sent by the FTP control port is sent to the controller and the FTP payload is examined to determine if that packet contains data port information. With the proposed enhancement, this is avoided by sending only the particular control packet with the data port information to the controller. This is achieved by

monitoring each packet sent from the FTP control port and checking if it contains '227' (ASCII) in the first three bytes of payload. If a match is found, the packet is sent to the controller for installing the flow to allow data traffic. Following figure represents the FTP control traffic flow from client to server when an OF agent with the proposed enhancement is used:

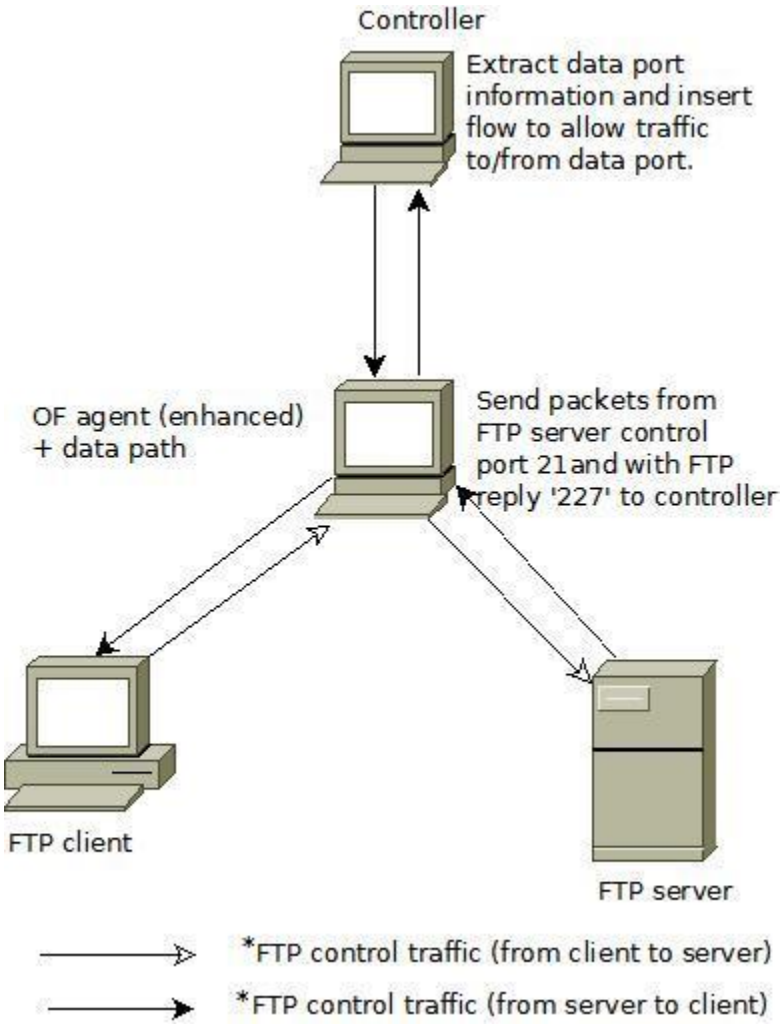

Figure 9: Illustration of FTP control traffic flow with the enhanced OF agent

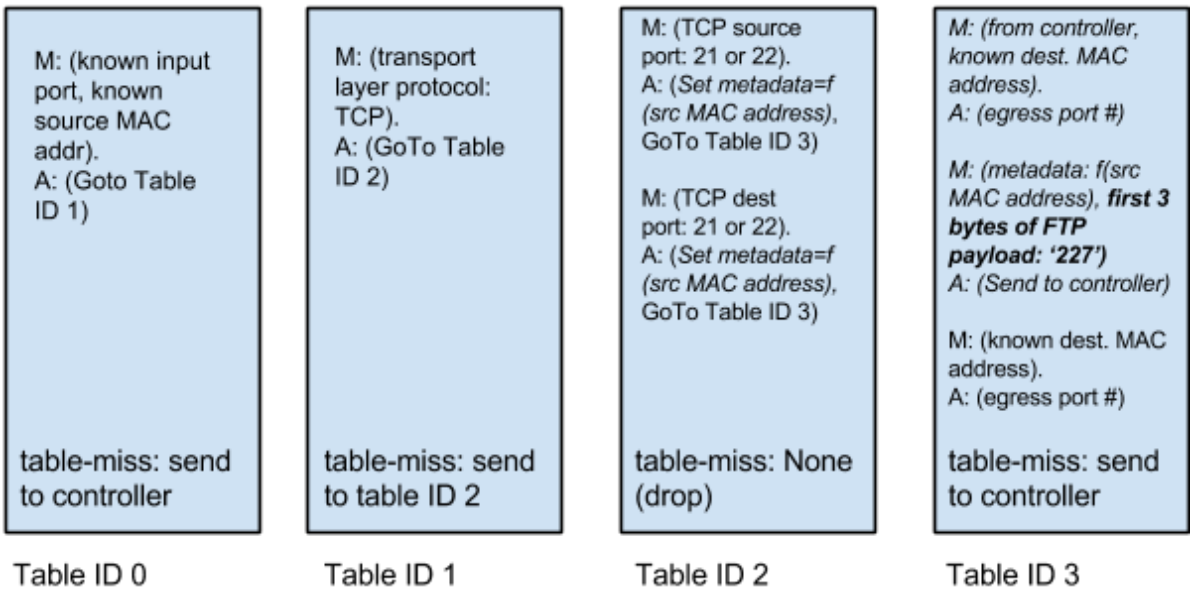

Figure 10: Flow table representation for a FTP ALG using the enhanced OF agent

To obtain this behaviour in the firewall and facilitate the working of a FTP ALG, a few additional flows are required, which are italicized in the following Figure 10. From the figure, it can be observed that the additional matching of the FTP command (first 3 bytes of the FTP payload) is being performed in the OF agent itself using an experimenter match. In case of the regular OpenFlow agent, the same match is performed in the controller rather than the switch.

5.1.3 Setup and testing tools

While the logical setup for this experiment is shown in figure 5.2, the configuration for each of the systems used is given in the following table:

Table 4: Configuration used for FTP ALG

Purpose	System configuration
Controller	A Dell Precision 390 [25] running Ubuntu 12.04.4 LTS
OF agent + Datapath	A Dell PowerEdge T610 [26] running Ubuntu 12.04.4 LTS
FTP Server	A Dell Precision 370 [27] running Ubuntu 10.04.4 LTS
FTP Client(s)	A Dell Precision 370 [27] running Ubuntu 10.04.4 LTS

To perform assessment of FTP connection establishment latency, we used the FTP client that is available with Ubuntu 10.04.4 LTS installation. The Very Secure FTP Daemon (vsftpd) is installed and used as the FTP server [36]. We compiled a simple script that connects to the FTP server and disconnects after a data connection is established. Listing 4 illustrates the code snippet for the script used. The data connection is established when a list ('ls') command is sent by the client over the control connection. Once the data connection is established, the client disconnects from the server.

```
HOST='192.168.1.100'
USER='Username'
PASSWD='Password'


ftp -n -v $HOST << EOT
ascii
user $USER $PASSWD
prompt
passive
ls
bye
EOT
```

Listing 5: Code snippet for a script that connects to the FTP server and disconnects after a data connection is established

Ten Linux processes each running the above script in a loop are initiated simultaneously. The inter-arrival time between invoking the above script (by a single process) is modelled using an exponential distribution. The FTP connection establishment latency for each request is recorded and the average is used for comparison. The amount of additional traffic handled by the controller (in terms of number of packets) is also recorded. To obtain the TCP throughput of connections through the OF agent, Iperf is used. Iperf is a commonly used network testing tool that can create TCP and UDP streams and measure the throughput of a network in which the streams flow [38].

5.1.4 Results

Results indicate that the FTP connection establishment latency does not vary much between the regular OF agent and the proposed enhancement for the FTP ALG. This trend is clearly seen in Figures 11 and 12 where the X-axis plots the average number of FTP requests per second (per process). However, the amount of additional traffic directed to the controller is about 10 times more when the regular OF agent is used (Figure 13). From figure 14, it can also be observed that the average throughput for TCP applications does not vary much between the regular OF agent and the proposed enhancement.

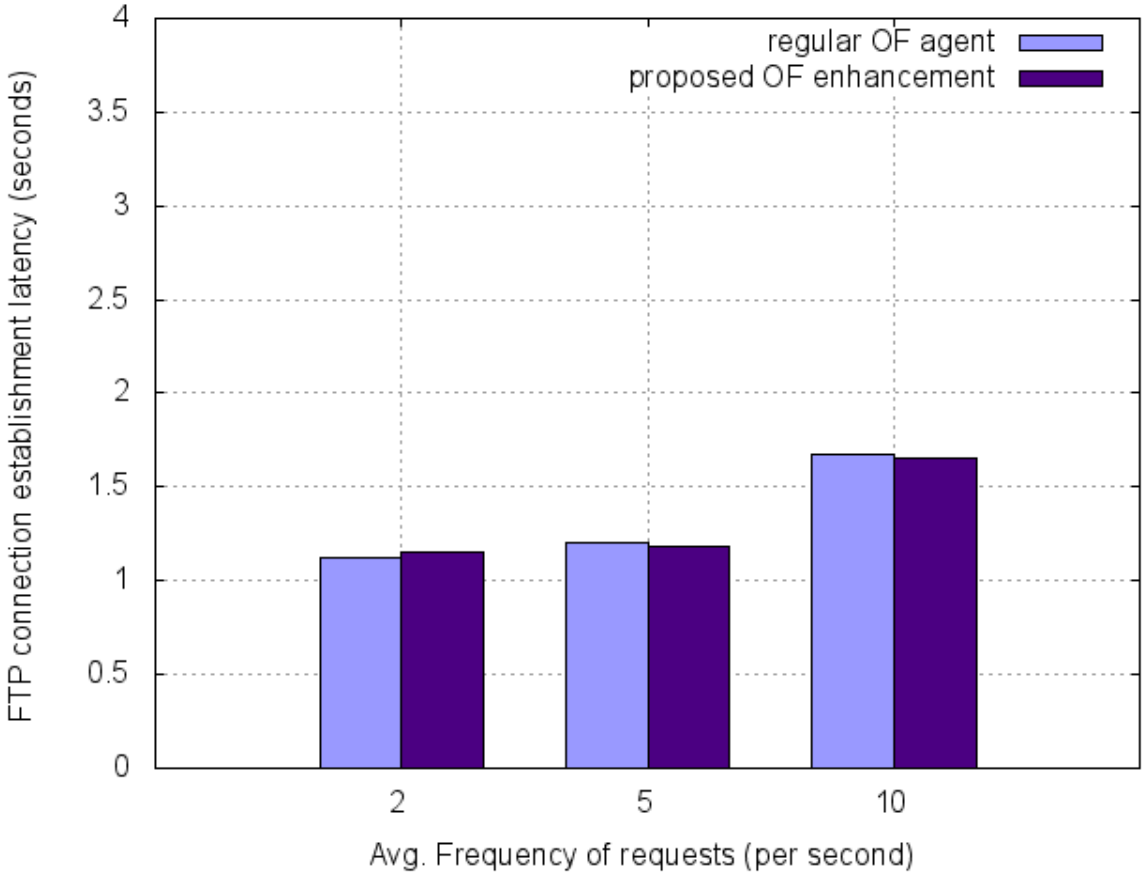

Figure 11: FTP connection establishment latency for various inter-arrival request times

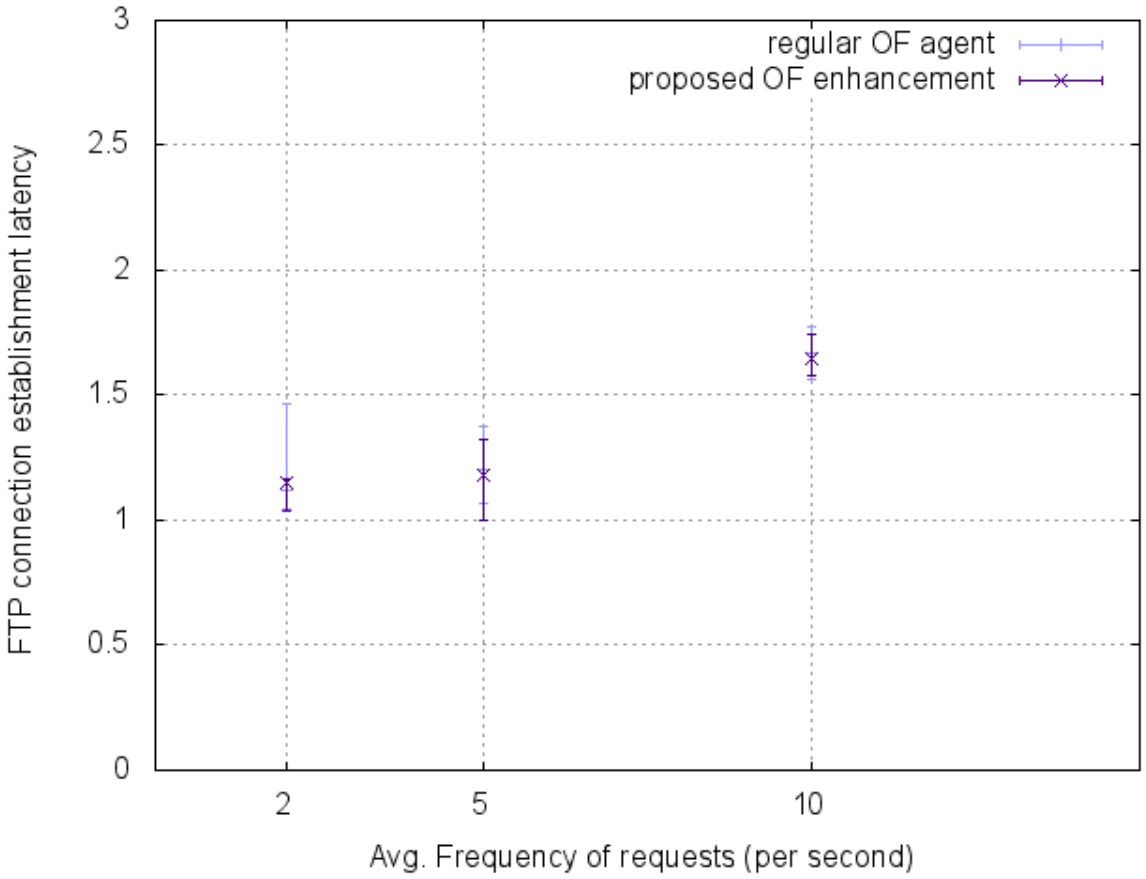

Figure 12: FTP connection establishment latency for various inter-arrival request times (same data as Figure 11, plotted with error bars)

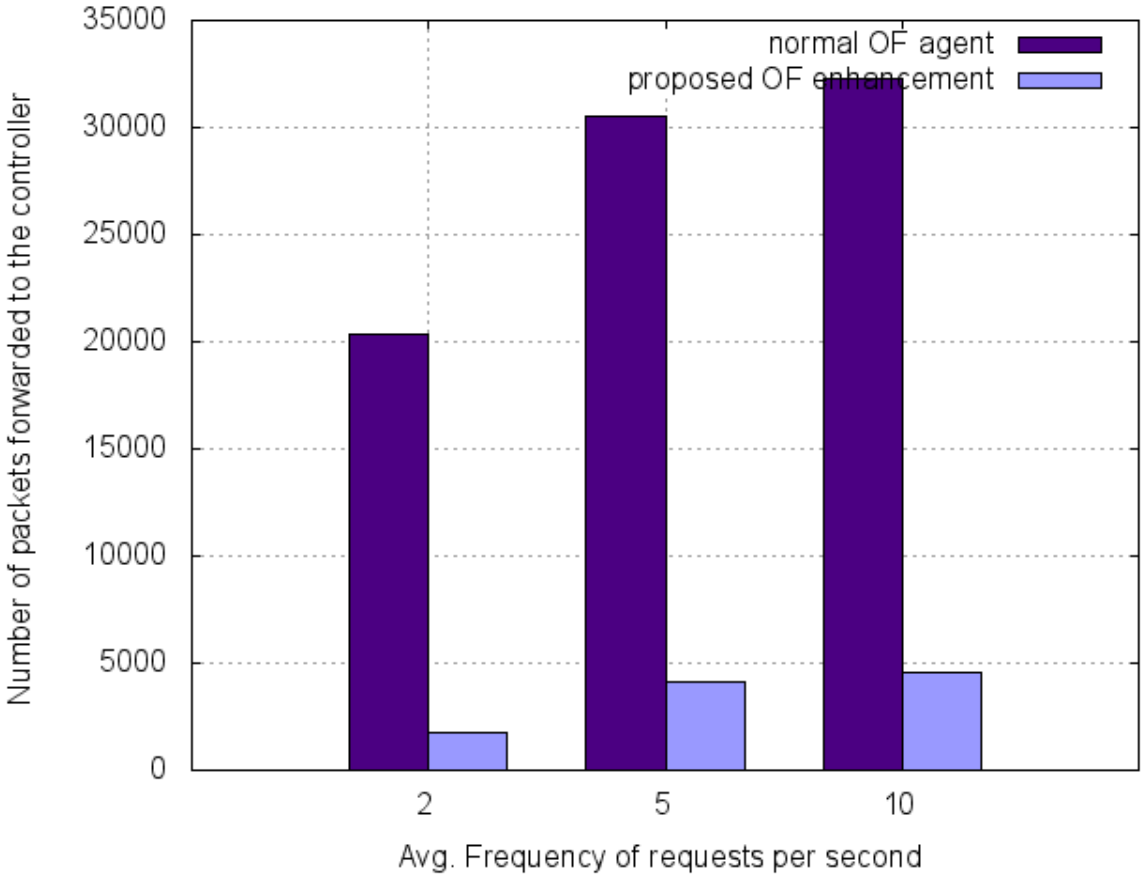

Figure 13: Additional traffic forwarded to the controller (in terms of number of packets) for various inter-arrival request times

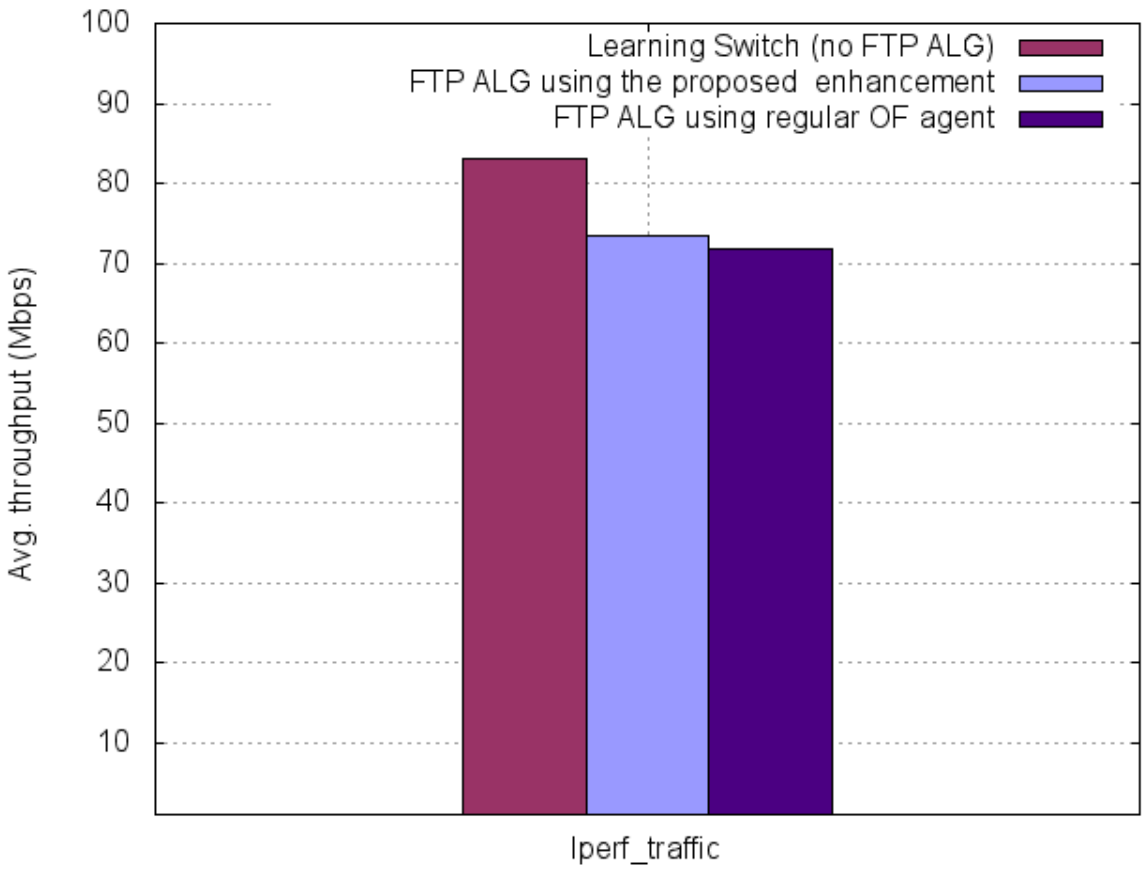

Figure 14: Average throughput for TCP traffic (comparison)

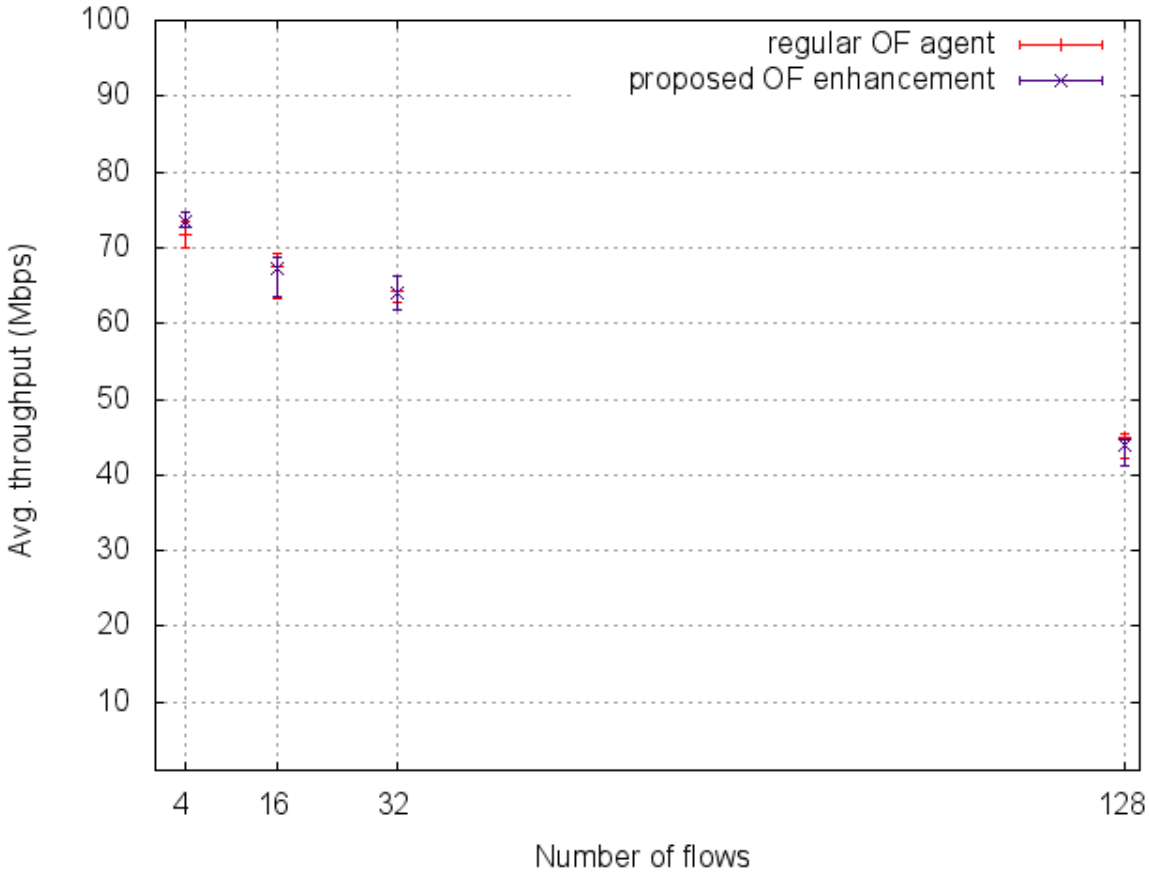

Figure 15: Throughput variation with number of flows

The similar throughput observed for the regular OF agent and the proposed enhancement can be traced down to the number of flows installed in the switch. If additional services are allowed to be accessed using the FTP ALG, it would increase in the number of flows in flow table number two. Figure 15 illustrates that with increase in number of rules, the average throughput decreases about the same rate for both the regular OF agent and the proposed enhancement. Thus, the factor in throughput determination is the number of flows in the switch.

5.2 in-transit HTML scanner

This application counts the number of HTTP connections that render HTML5 pages through the switch in a given time frame. Though the HTML5 specification is a working candidate and is not yet a World Wide Web Consortium (W3C) recommendation, it has been gaining popularity. A report released in August 2013 shows that 153 of the Fortune 500 U.S. companies support HTML5 on their corporate sites [35]. Though the purpose of HTML scanner in this work is purely statistical, its use is not restricted to the same. One significant advantage of recognizing a HTML5 stream is to determine if it has video content (by checking for the <video> tag) and provide a guaranteed quality of service (QoS) to the recognized stream. This can be achieved by adding relevant flows to the switch using the Flexible Payload Match.

HTML5 documents can be recognized by making use of the DOCTYPE [19]. The preamble for HTML5 consists of the following components in that order:

- '<!DOCTYPE (case insensitive)' followed by
- 'one or more spaces' followed by
- 'html (case insensitive)'followed by
- 'zero or more spaces' followed by
- (optional) a DOCTYPE legacy string or an obsolete permitted DOCTYPE STRING
- '>'

Multiple combinations are possible based on the syntax. For example, '<!Doctype html>', <!doctype html>, <!doctype HTML> etc. are possible. However, based on the syntax of many existing HTML pages, we ignore other types of possible combinations and focus on

‘<!DOCTYPE html> ‘, ‘<!DOCTYPE HTML>’ and ‘<!doctype html> ‘only. This may result in slightly lesser accurate results, but it does not limit the overall usefulness of this approach. We demonstrate that supporting more combinations is still possible.

5.2.1 Topology

Figure 16 illustrates the logical topology used for both the regular OF agent and the proposed enhancement. The only difference is the controller application, which inserts different rules in each case. The HTML scanner application does not need to act as a firewall and thus it has fewer rules than the ones depicted in Figure 6.

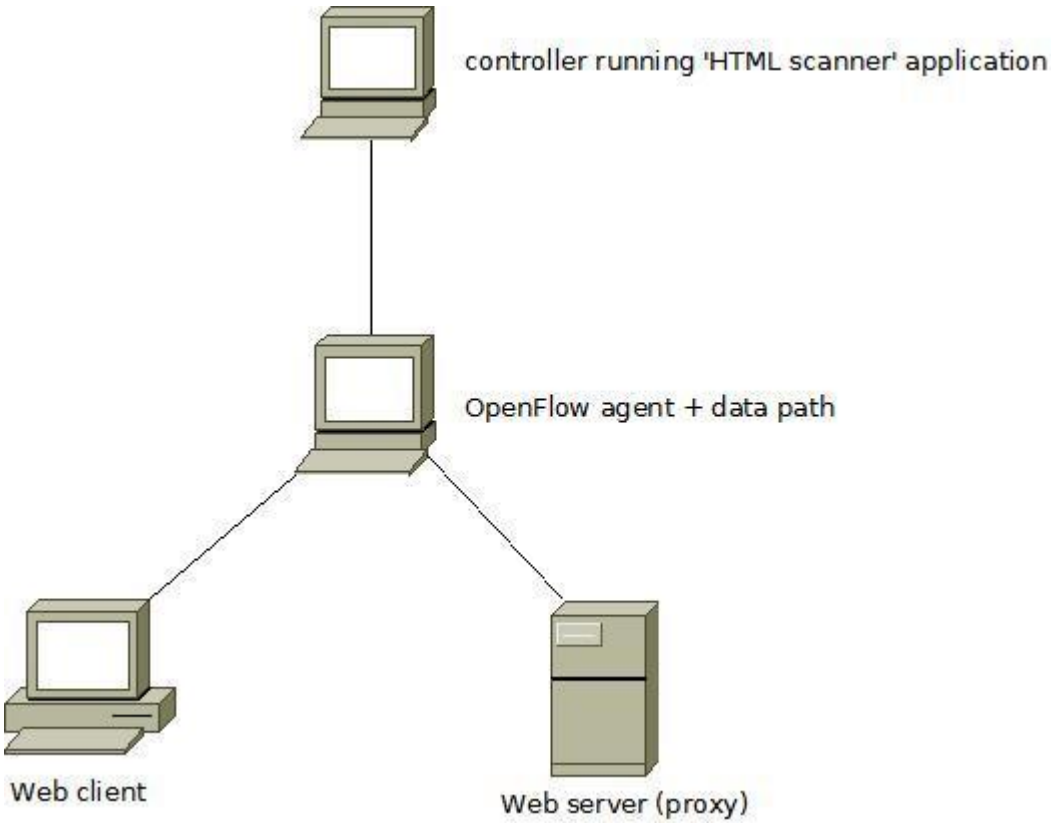

Figure 16: Logical topology for ‘HTML scanner’ application

A web proxy server is used in the setup used for validation. Many companies employ web proxies that support caching to reduce bandwidth usage and improve latency access [20]. The proxy server used is Squid [21], which is a popular web-content caching proxy service. GNU wget program on Linux is used as a web client to download files from the web. It can also work with proxy services with appropriate configuration. In this setup, Squid is configured to run on TCP port 8080. Using the caching service offered by Squid, a set of URLs is accessed to gather results and compare the two different approaches. Each URL may consist of other elements like images along with HTML content. Though wget has the capability, in the current setup it is not used to parse the HTML file and does not generate requests for images or other objects.

5.2.2 HTML scanner using regular OF agent

To determine if there is a HTML5 connection, the HTTP payload should be analysed for the DOCTYPE. With the limitation of the regular OF agent, an external packet processor is needed. Since OpenFlow has semantics to send packets to the controller using a 'PACKET_IN' message [1], we make use of the same instead of defining new semantics. The controller has a counter for the number of HTML5 documents rendered by the switch, which is initialized to zero. So, whenever a packet is sent from TCP port 8080, the following sequence of events occur:

- The packet is forwarded to the controller over the OpenFlow channel.
- Controller checks the packet to determine if it contains the DOCTYPE for HTML5.
- If the DOCTYPE exists, the counter keeping track of the same is updated.
- The packet is sent back to the switch and pipe-line processing restarts.

Since a single HTTP request may result in multiple packets being sent by the web server (depending on size of the document), the number of packets sent to the controller for each request varies depending on the URL being accessed. Following figure represents the traffic flow from client to server, when a regular OF agent is used:

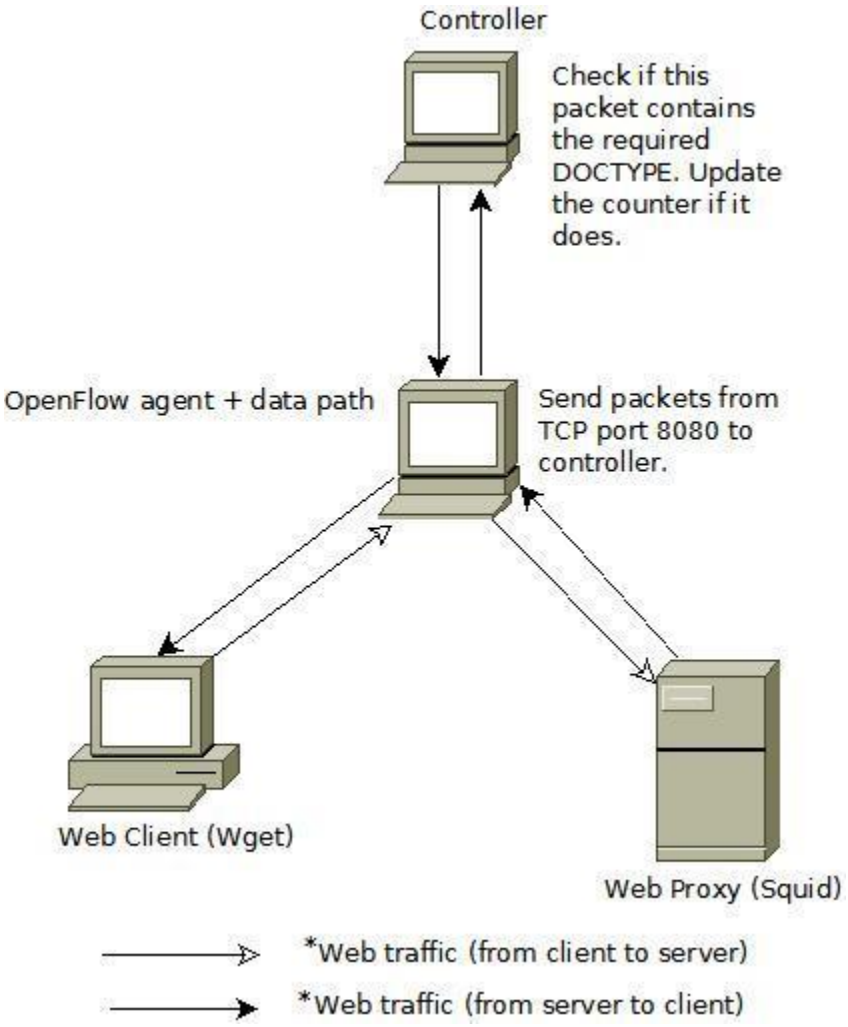

Figure 17: Illustration of HTTP traffic flow for HTML scanner with the regular OF agent.

To obtain this behaviour, the following flows are inserted to the switch by the controller

application:

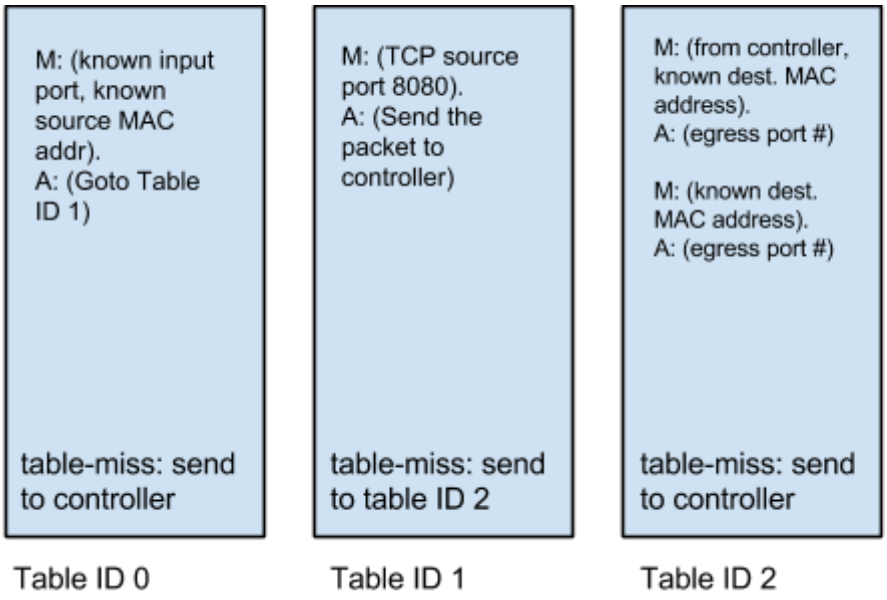

Figure 18: Flow table representation for 'HTML scanner' using the regular OF agent

5.2.3 HTML scanner using the proposed enhancement

The proposed enhancement removes the need for a packet processor as relevant flows can be inserted into the OF agent to count HTML5 connections. As described in Section 5.2, we focus on only three possible combinations. The rules inserted into the OF agent correspond to these three combinations: '<!DOCTYPE html>', '<!DOCTYPE HTML>' and ''<!doctype html>'. Unlike the case with the regular OF agent, no counter is maintained by the controller. Instead, the 'Received Packets' counter that is maintained by the OF agent is used for keeping track of connections [1]. The controller application can obtain the counter value by querying the switch. This eliminates the need for sending any packet to the controller. This functionality is achieved by having a controller application that inserts the following

flows to the OpenFlow switch:

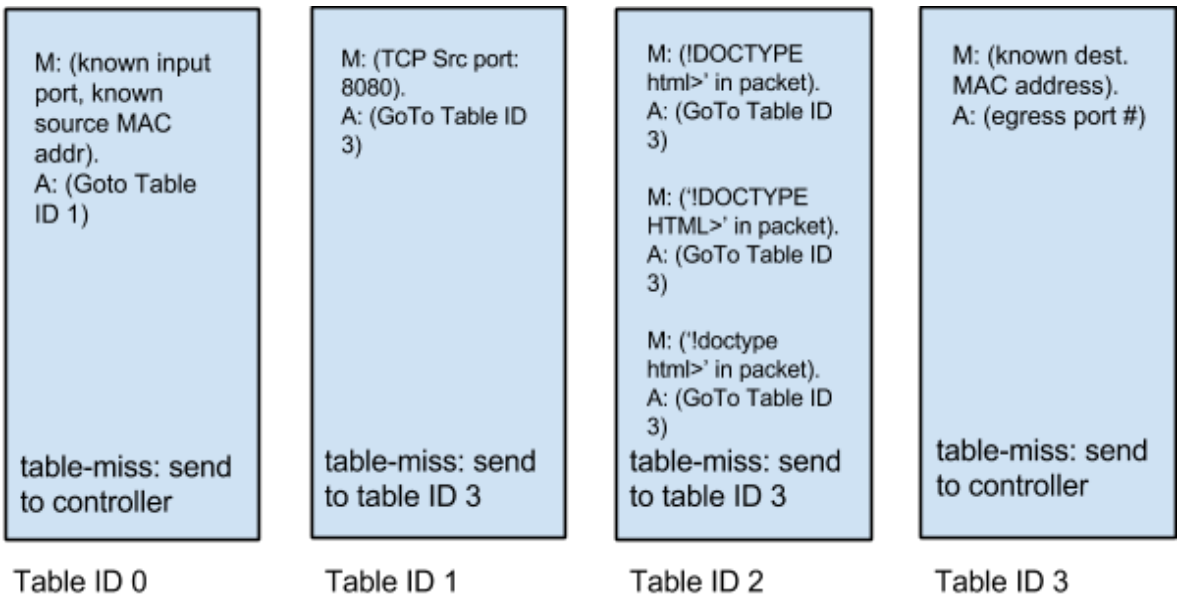

Figure 19: Flow table representation for HTML scanner using the proposed enhancement

The three rules in flow table number 1 check if '!DOCTYPE html>', '!DOCTYPE HTML>' or ''<!doctype html>' exist anywhere in the application payload. These differ from the rules used for FTP ALG where the location of the action code is known to be in the first three bytes of the FTP payload. Since the location of the DOCTYPE can be at variable length from the beginning of the application payload, we use this sort of a match. More 'DOCTYPE' combinations can be supported by using more flows, though it may not be possible to accommodate all possible combinations with the current implementation.

5.2.4 Accuracy

Using the setup given in figure 5.7, multiple HTML5 pages have been accessed to determine the accuracy of the 'HTML scanner'. Using [23] as a reference, the number of HTML5 pages

has been determined using both regular OF agent and the proposed enhancement. Of the list of active sites, only five were not recognized as HTML5 sites by the HTML scanner (because of its limitation). A few connections timed out (tried with a timeout of 10 seconds) and a few sites were inactive. The list of URLs that was not recognized by the HTML scanner or was found to be inaccessible is specified in Appendix A.

5.2.5 Setup and testing tools

Figure 5.12 gives the logical setup for this experiment, while the configuration for each of the systems used is given in the following table:

Table 5: System configuration used for HTML scanner

Purpose	System configuration
Controller	A Dell Precision 390 [25] running Ubuntu 12.04.4 LTS
OF agent + Datapath	A Dell PowerEdge T610 [26] running Ubuntu 12.04.4 LTS
Web Proxy Server (Squid)	A Dell Precision 370 [27] running Ubuntu 10.04.4 LTS
Web Client(s)	A Dell Precision 370 [27] running Ubuntu 10.04.4 LTS

Ten Linux processes each generating a wget request in a loop are initiated simultaneously.

The inter-arrival time between invoking the above script (by a single process) is modelled using an exponential distribution. The response latency for each request is recorded and the average is used for comparison. The amount of additional traffic handled by the controller (in terms of number of packets) for the regular OF agent is also recorded. To obtain the TCP throughput of connections through the OF agent, Iperf is used.

5.2.6 Results

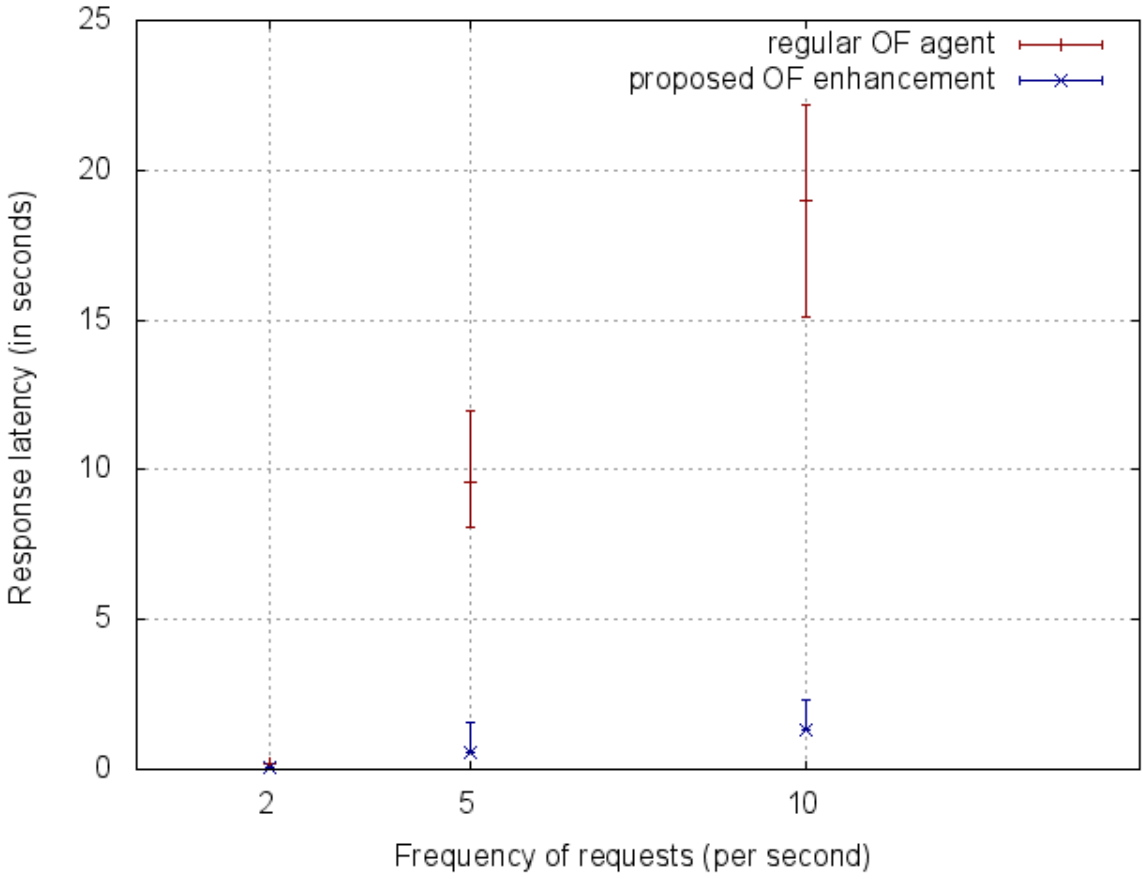

Figure 20: Comparison of response latency between the regular OF agent and the proposed enhancement (for various request frequencies)

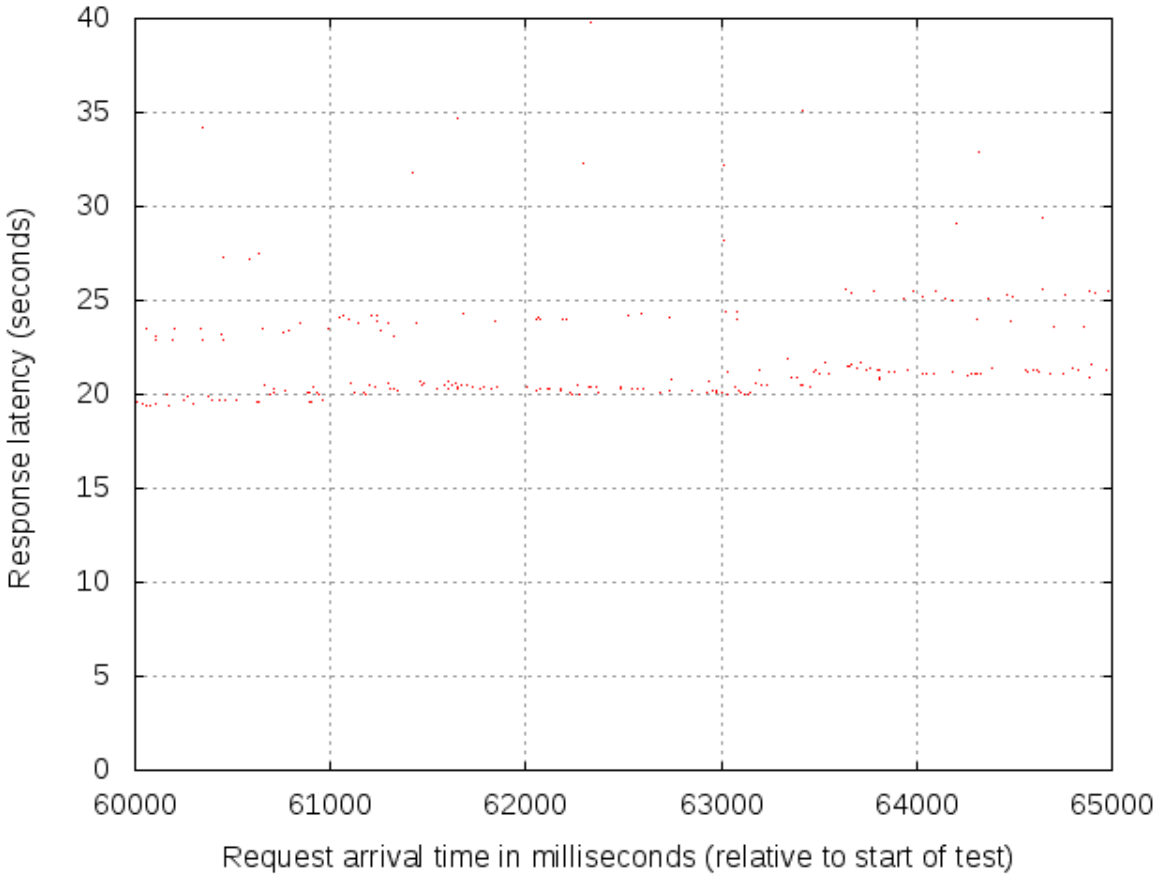

Figure 21: Illustration of response latency variation for a five second interval with the regular OF agent (average frequency of requests per second: ten)

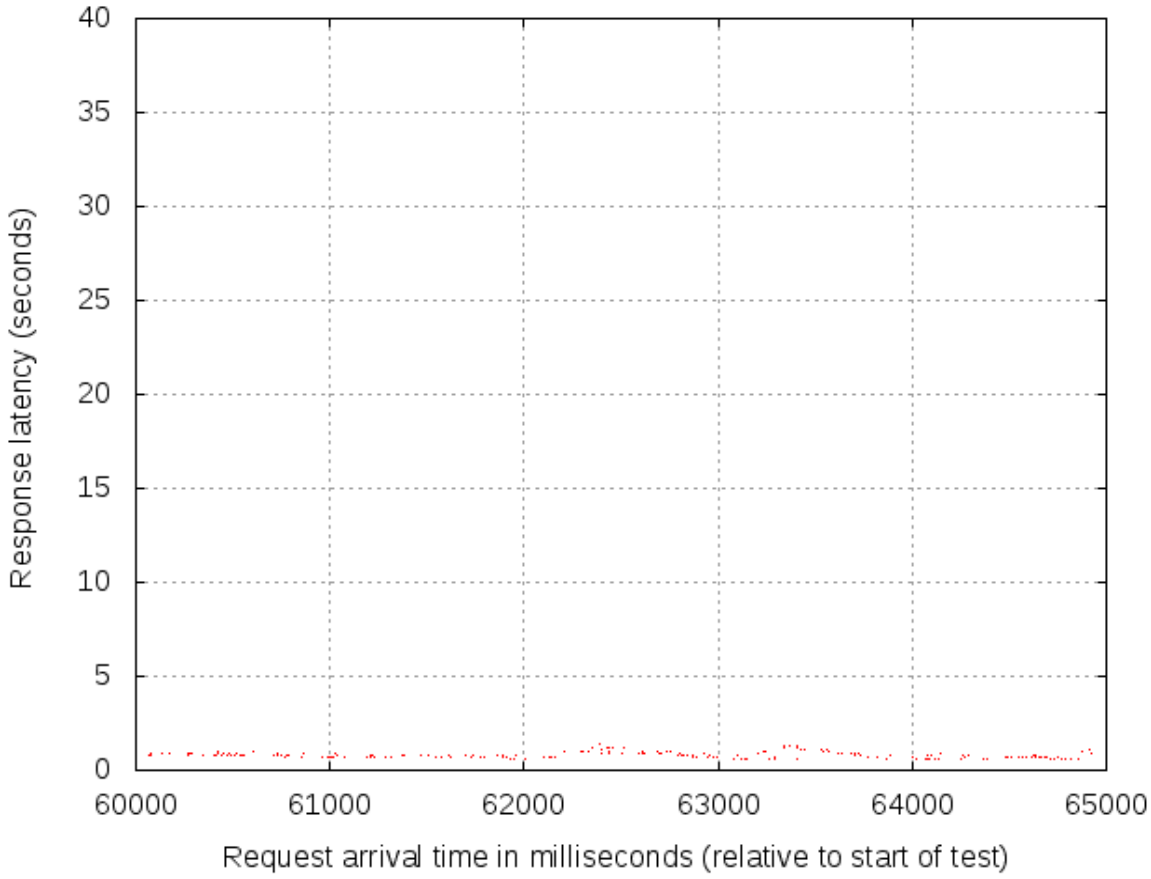

Figure 22: Illustration of response latency variation for a five second interval with the enhanced OF agent (average frequency of requests per second: ten)

Results indicate that the response latency increases with the increase in frequency of requests. Though the increase is minimal with the enhanced OF agent, there is a significant increase in latency with the regular OF agent. Figure 20 illustrates this comparison. Figures 21 and 22 illustrate the variation of response latency during a five-second interval with the regular OF agent and the enhanced OF agent. It can be clearly observed that the response latency with the regular OF agent is around 20 seconds, while the same for the enhanced OF agent is less than 3 seconds. Multiple possibilities exist for the bottleneck that results in this

behaviour:

1. The controller getting swamped by the HTTP traffic forwarded by the OpenFlow switch
2. Inability of the switch to process requests at the rate generated by the proxy server.
3. Inability of the proxy server to match the request rate

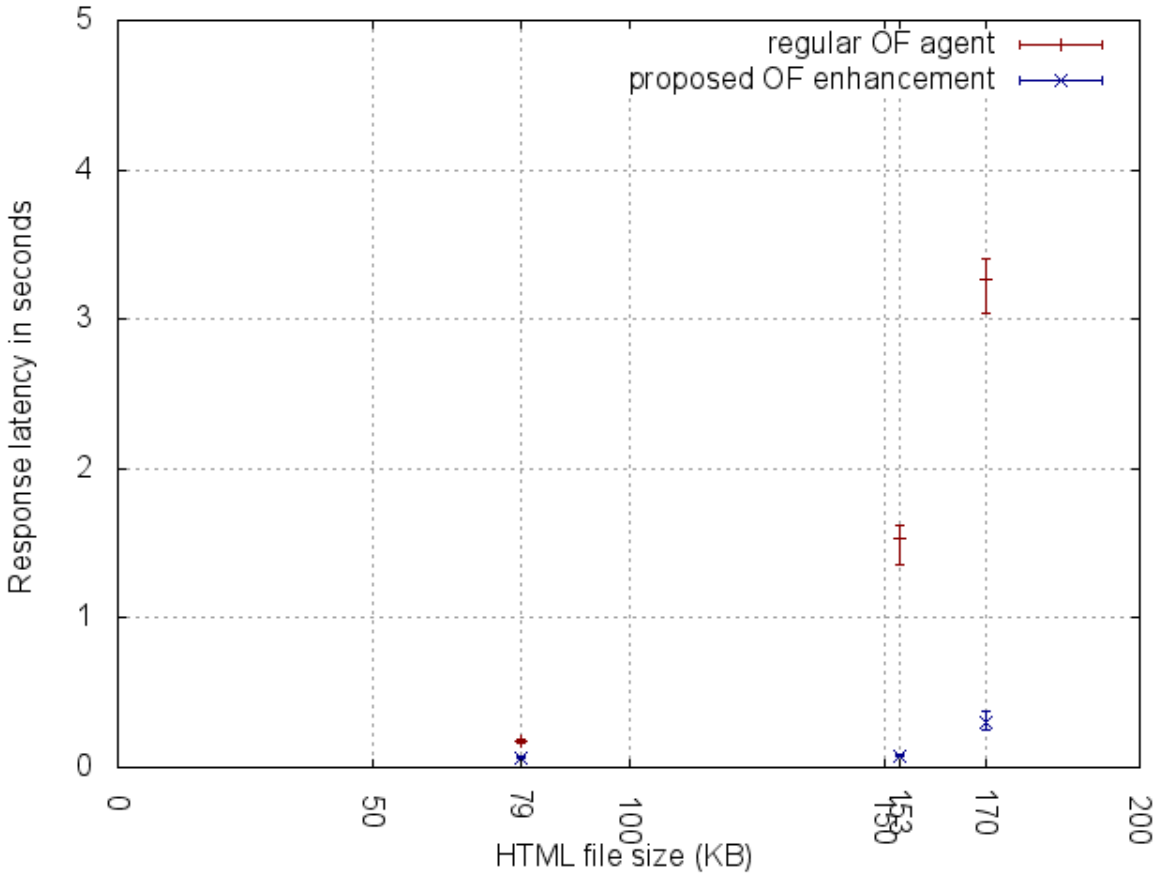

Figure 23: Comparison of response latency with increase in file size

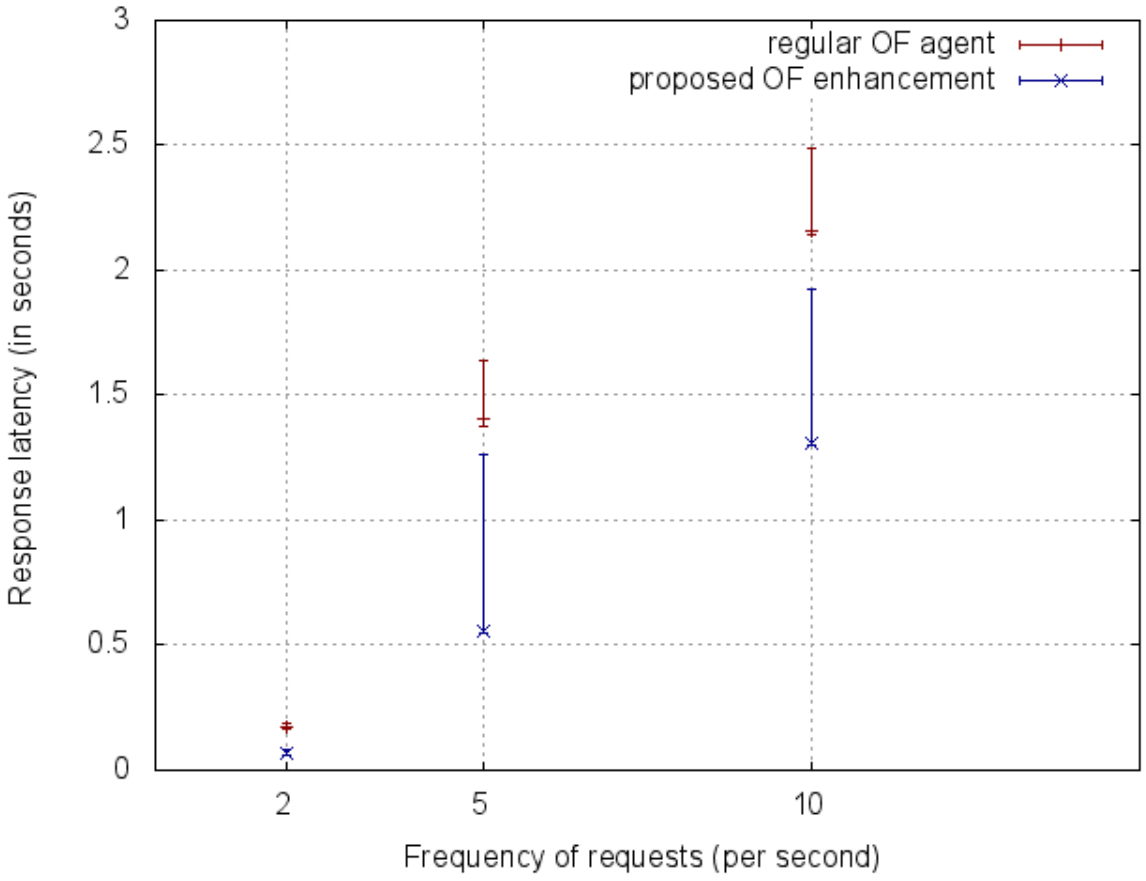

Figure 24: Comparison of response latency using the T610 as the controller platform

The third reason can be eliminated as there is no visible change with the enhanced OF agent. Figure 23 hints that the bottleneck is the controller. The data for this figure is obtained by requesting HTML files of different sizes and recording the average response latency. The average frequency of requests per second for this plot is constant (two). With increase in file size (and thus the number of packets forwarded by the switch to the controller over the OpenFlow channel), the average response latency increased. That the controller is the bottleneck is confirmed by moving the controller software to a Dell T610. The T610 has greater processing power and memory (Intel Xeon processor and 24 GB of RAM) than Dell

Precision 390. This is the best-case scenario in which the controller operates as fast as the OpenFlow switch. Using the T610 as a controller, we can observe lesser response rates with the regular OF agent. This validates the theory that the controller is proving to be the bottleneck with increase in the amount of traffic forwarded by the switch over the OpenFlow channel. Even with the T610 acting as the controller, the response latencies observed in case of the enhanced OF agent are much better. In practice, the system used for running the controller is commodity hardware and is not expected to have processing power equal to that of the OpenFlow switch. Also, a controller typically manages multiple switches. In such a scenario, it is only trivial that the response latency increases even when a powerful system like the T610 is used. This demonstrates that DPI in the application payload with the existing OF specification is not scalable.

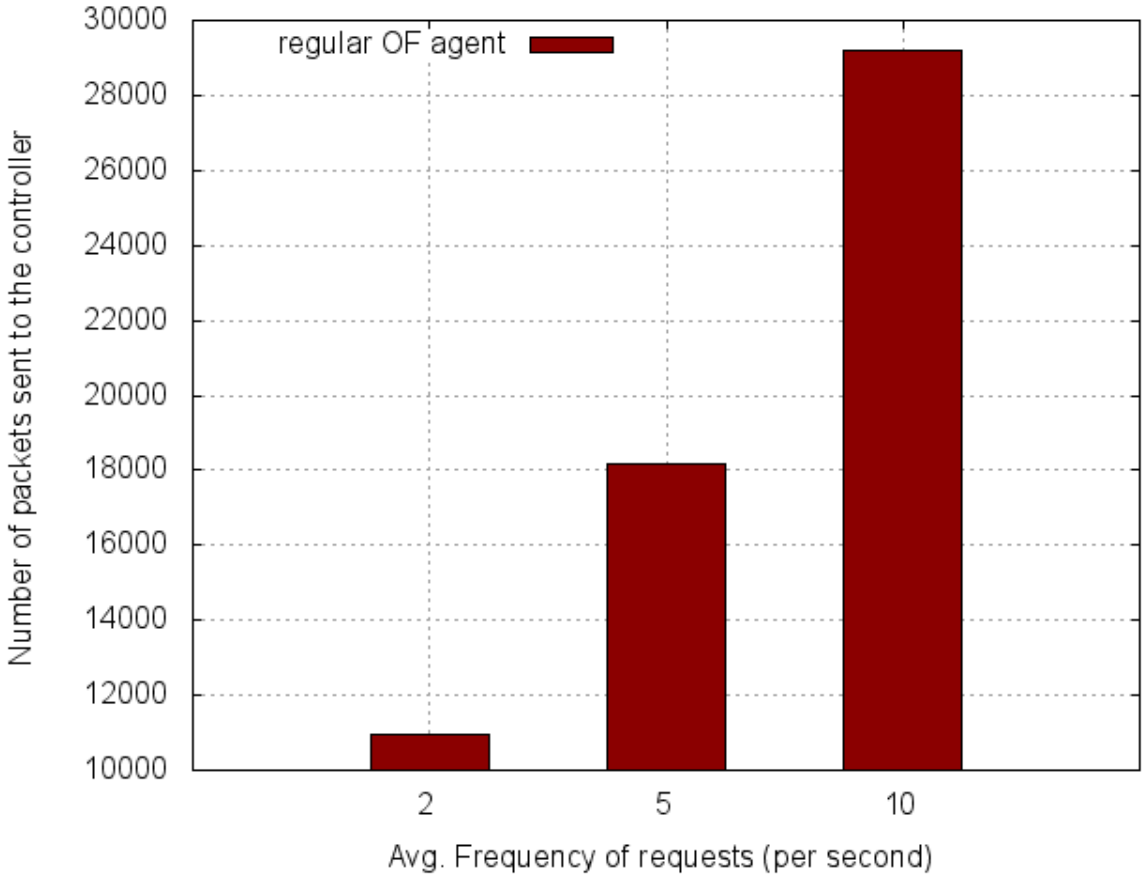

Figure 25: Illustration of additional traffic forwarded to the controller by the OpenFlow switch

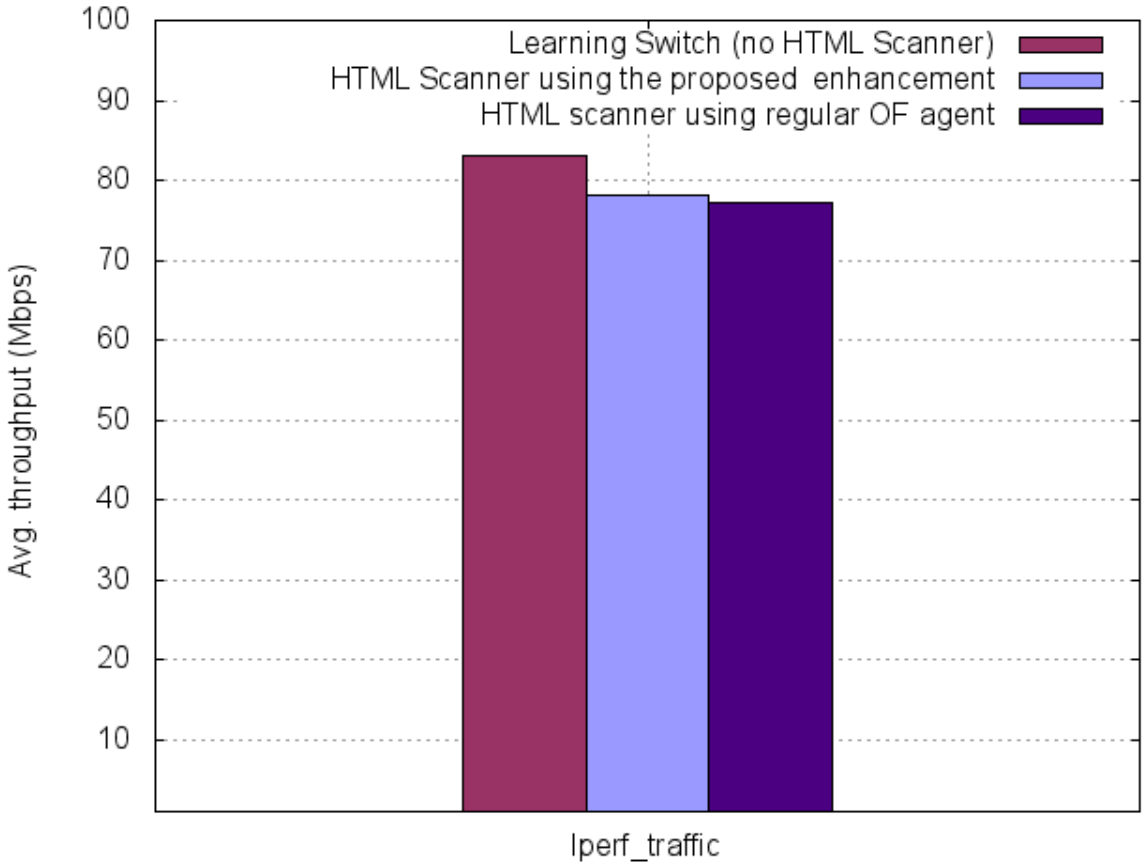

Figure 26: Average throughput for TCP traffic (comparison)

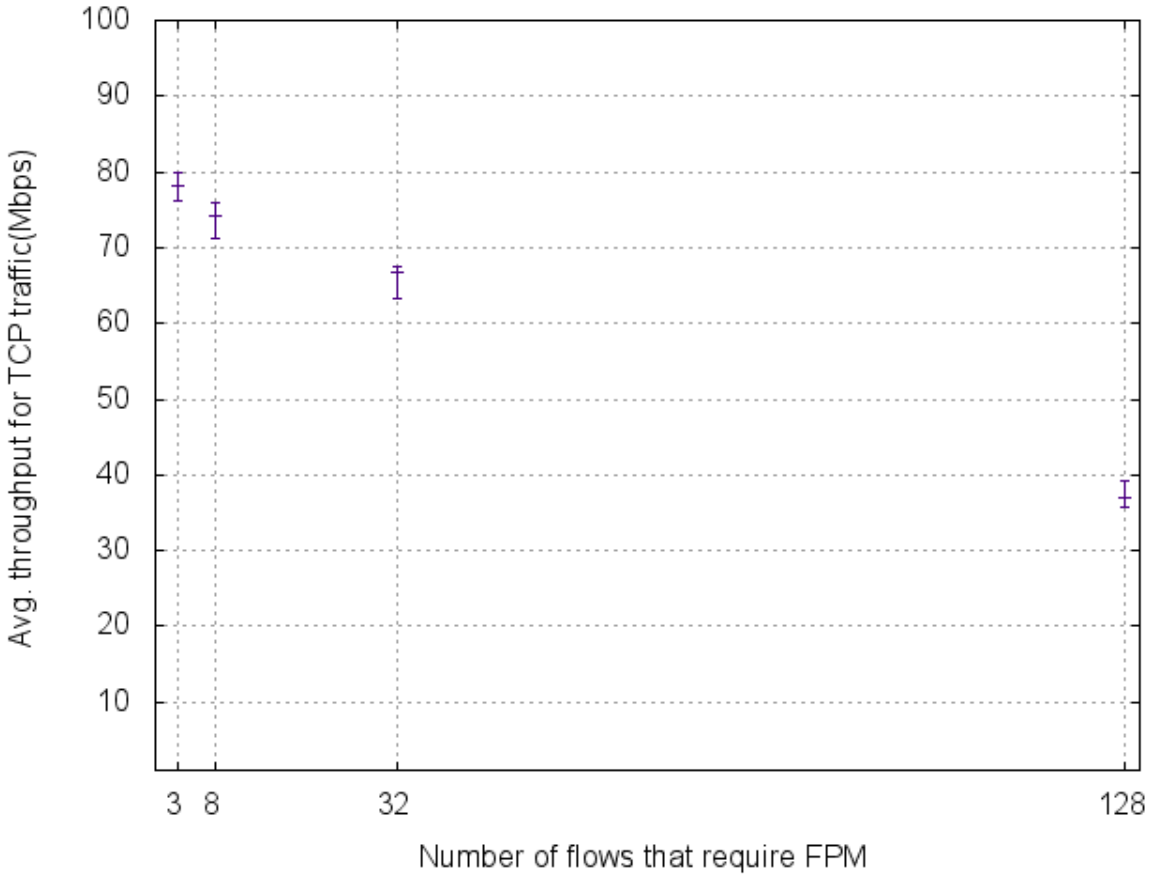

Figure 27: Illustration of throughput variation with increase in number of FPM rules (for the enhanced OF agent)

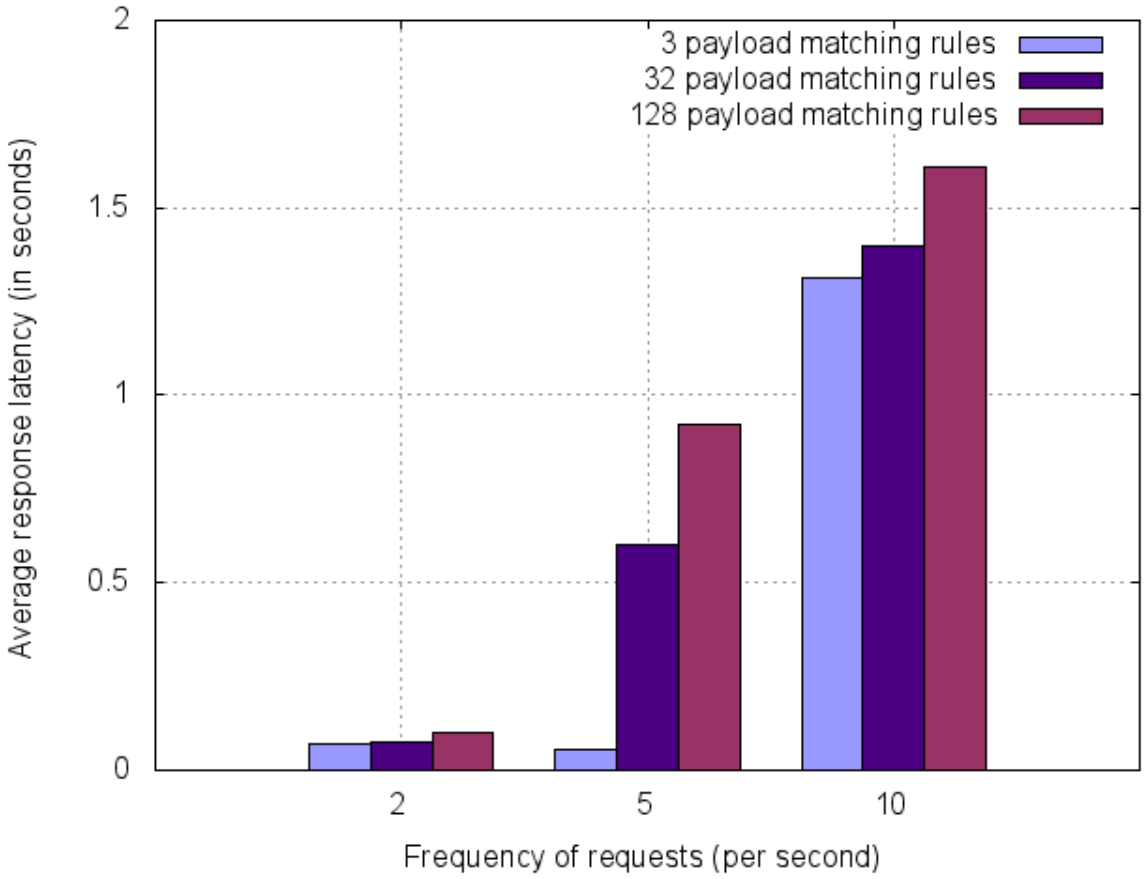

Figure 28: Comparison of response latency with increase in number of FPM rules (for the enhanced OF agent)

Another advantage with the enhanced OF agent is that no traffic is directed to the controller. Figure 25 illustrates the increase in traffic (in terms of number of packets) to the controller with the regular OF agent. Though there is significant gain in terms of response latency for web clients, the effect on throughput for TCP connections has not been discussed so far. Figure 26 illustrates the throughput comparison between the regular OF agent and the proposed enhancement. With the FTP ALG, it has been demonstrated that the throughput

obtained depends on the number of rules. The same result is confirmed with Figure 26. Another interesting comparison is the effect of FPM flows on throughput. As demonstrated earlier, the throughput decreases gradually with increase in number of flows (Figure 27). However, the response latency is not particularly affected with the increase in number of rules. Figure 28 confirms the same. This demonstrates the ability of the enhanced OF agent to operate with minimum impact in latency for increased number of flows.

5.3 Set-up issues

During performance evaluation for the use cases, we ran into inexplicable setup issues. Though they are not related to OpenFlow (and thus the proposed enhancement), we feel it is worthwhile to document the same. As described in Section 5.2.7, Dell Precision 390 [27] systems were used as the hosts (each running Ubuntu 10.04.4 LTS). Each of them had two PCI Network Interface Cards (NICs) – a Broadcom 'NetXtreme BCM5751' (on-board NIC) and an 'Intel 82557'. When the host was connected to the OpenFlow switch using Intel NIC, the traffic that was being forwarded had a jittery quantization. This pattern did not depend on whether the traffic was OpenFlow or not. The same behavior was observed when there was no soft-switch between the hosts and traffic was exchanged over a hardware switch (from Cisco). Figures 29, 30 and 31 demonstrate this phenomenon with ping traffic.

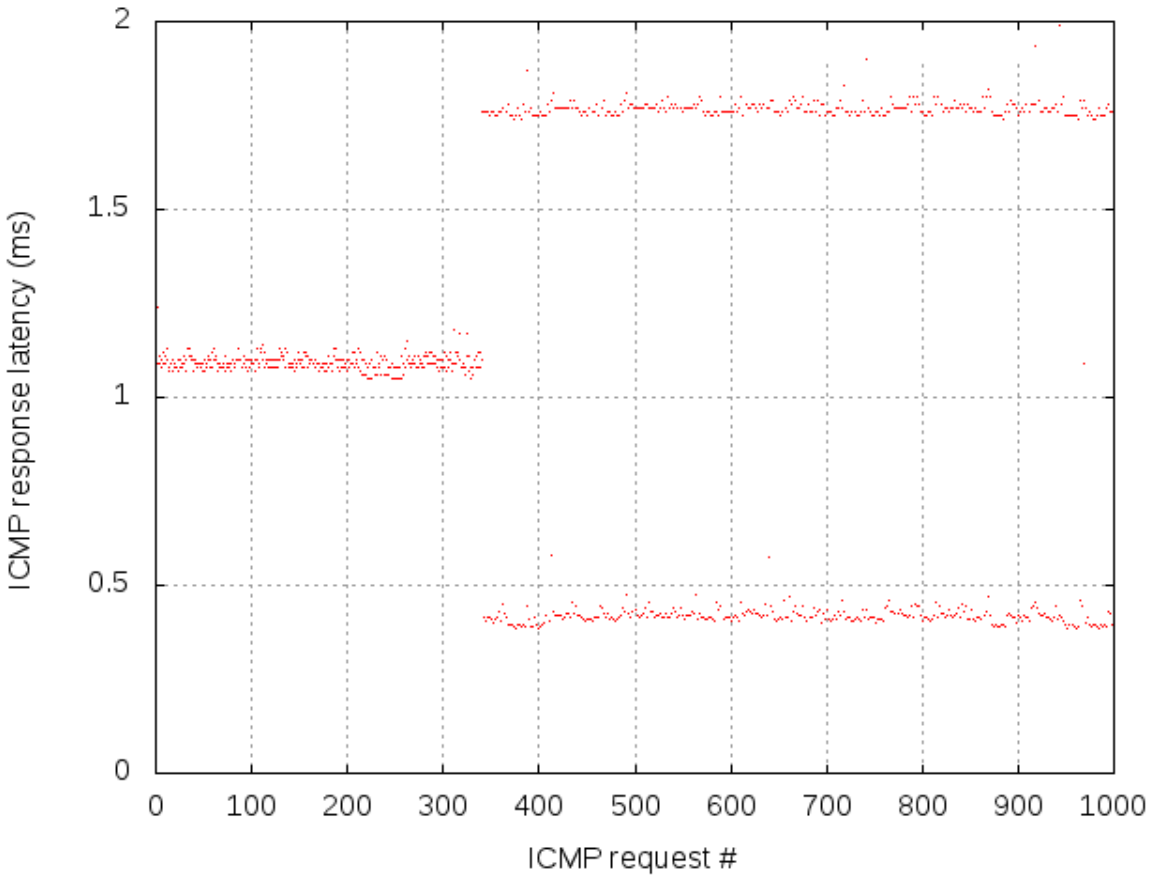

Figure 29: Figure illustrating multiple patterns for ping responses (with 200 requests per second frequency). Hosts are connected with the enhanced OF agent acting as a switch.

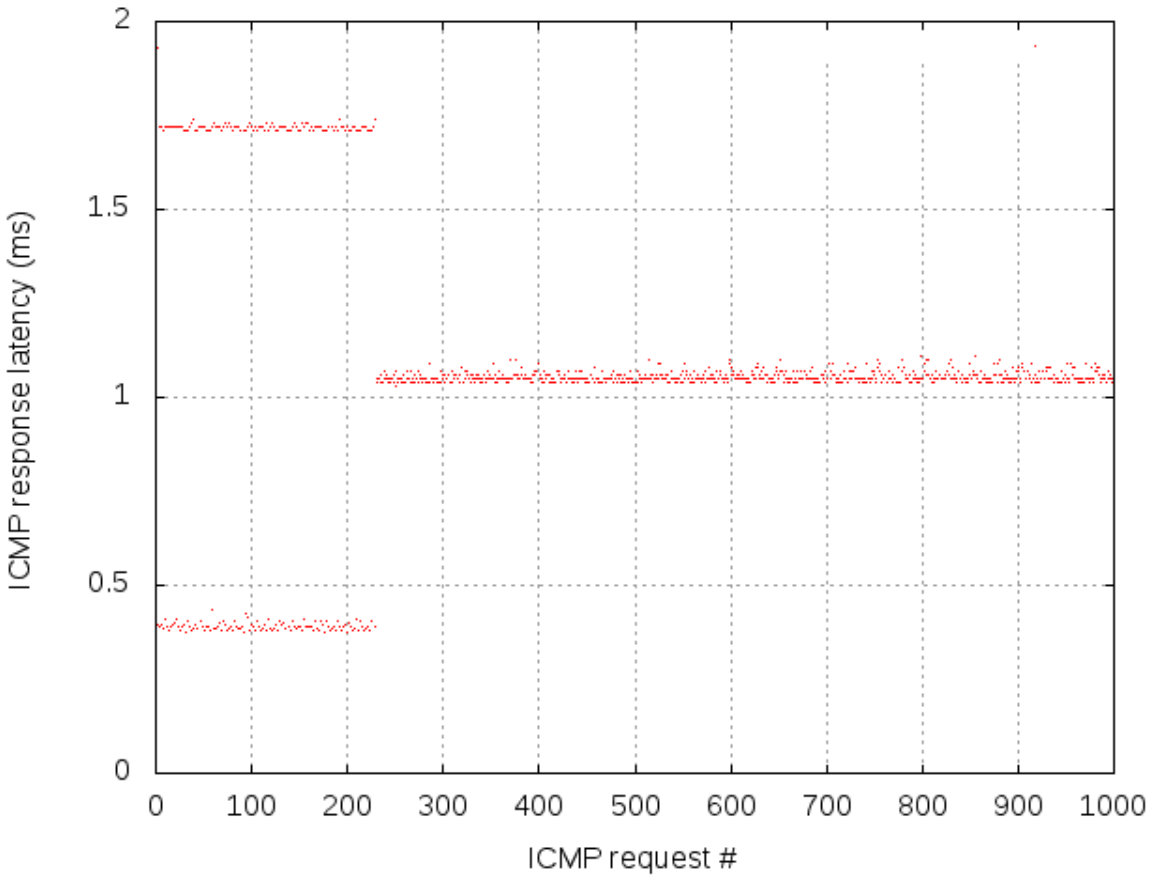

Figure 30: Figure illustrating multiple patterns for ping responses (with 200 requests per second frequency). Hosts are connected with the regular OF agent acting as a switch.

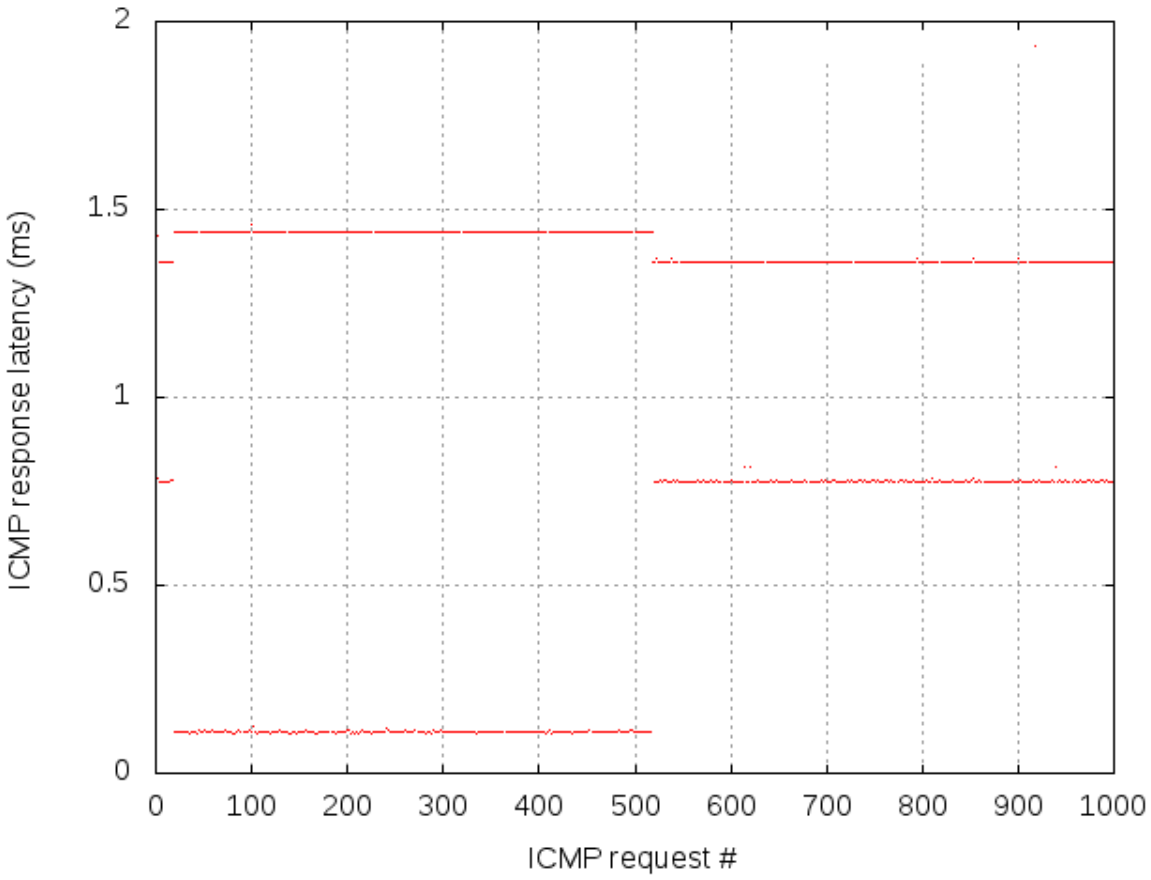

Figure 31: Figure illustrating multiple patterns for ping responses (with 200 requests per second frequency). Hosts are connected using a hardware switch.

We believe it is an issue with the Operating System-driver combination. When the connection was flipped to the on-board NIC, this behavior vanished. Figures 31 and 32 demonstrate the results for the same test when the on-board NIC was used on both the hosts.

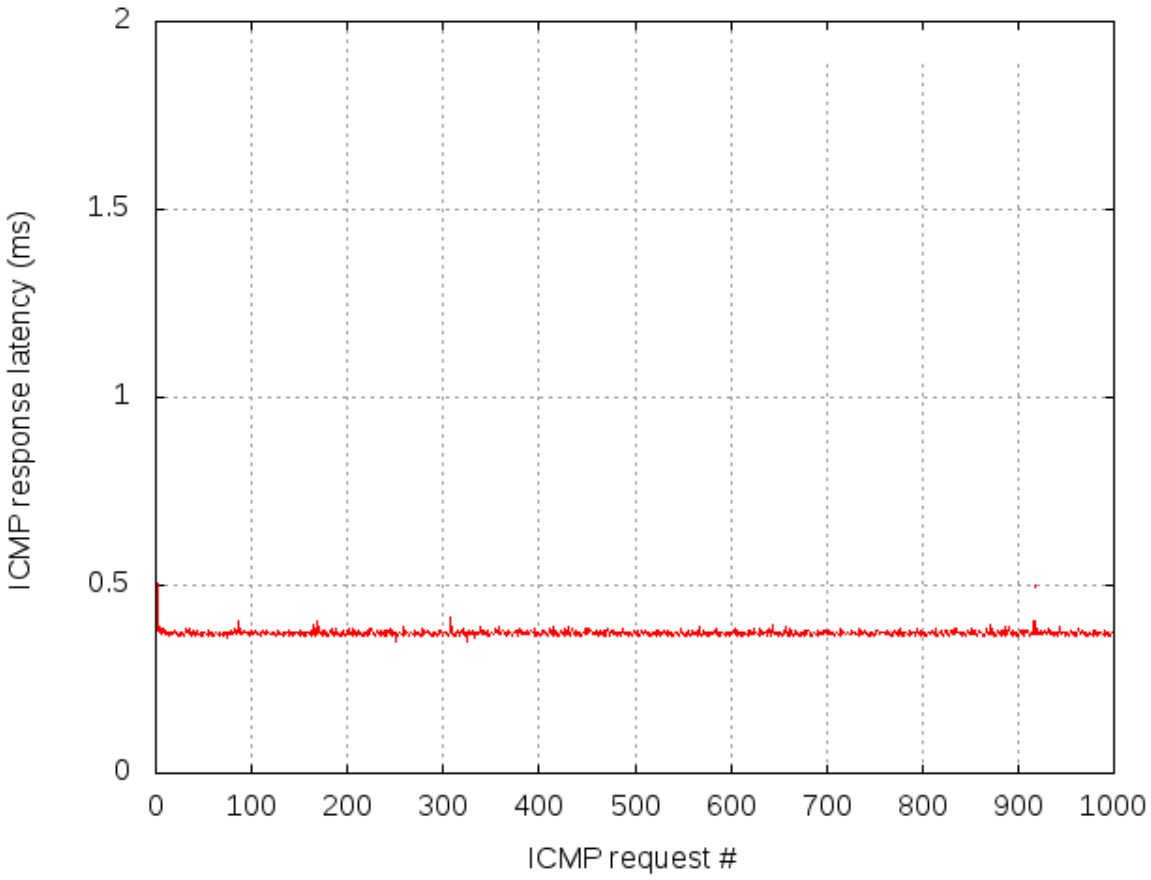

Figure 32: Response latency with the on-board NIC for ping (with 200 requests per second frequency). Hosts are connected using the enhanced OF agent acting as a switch.

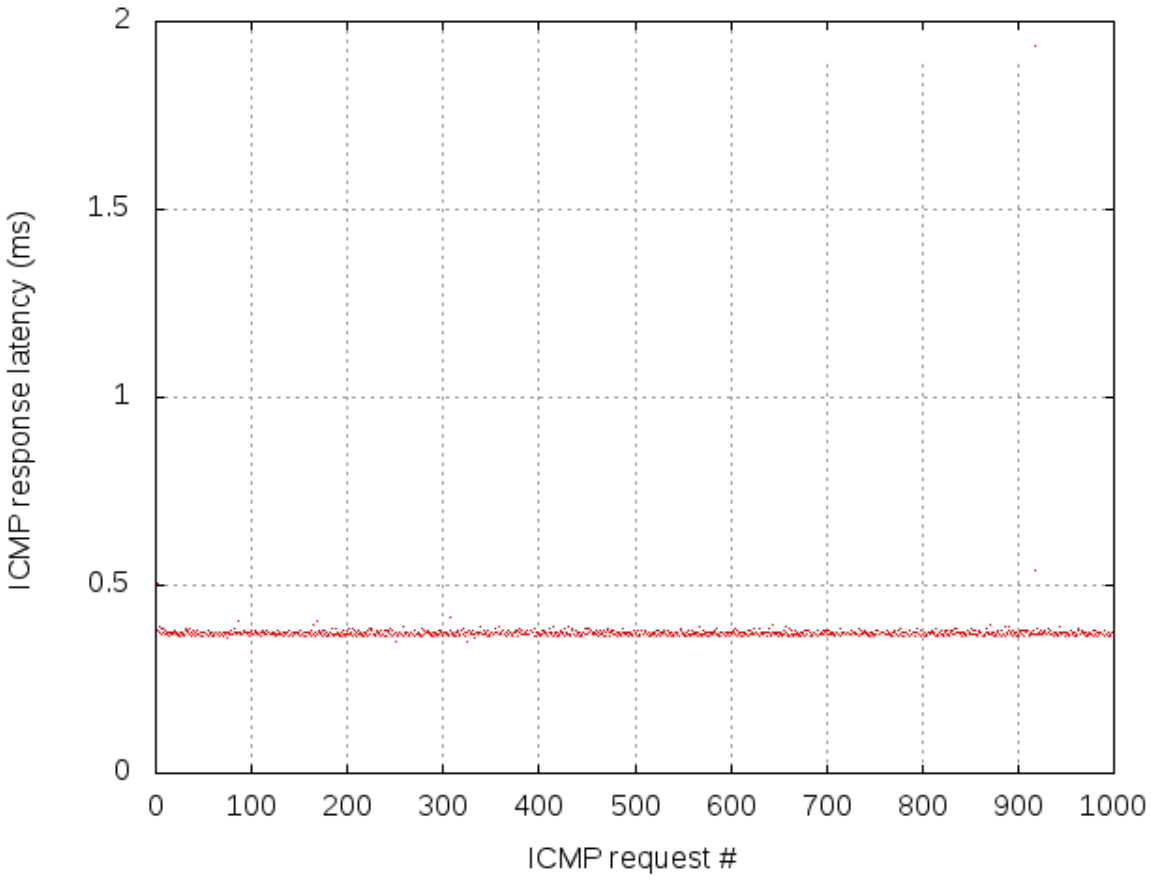

Figure 33: Response latency with the on-board NIC for ping (with 200 requests per second frequency). Hosts are connected using the regular OF agent acting as a switch.

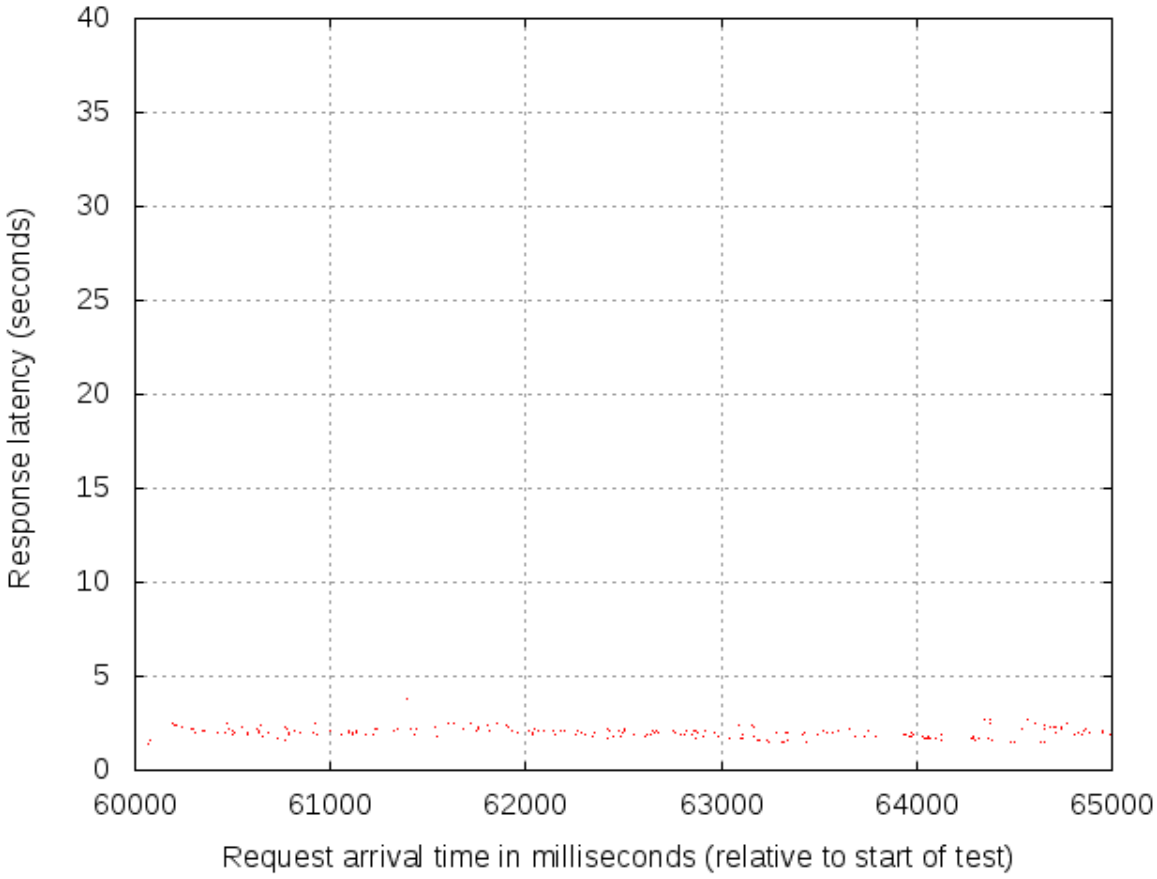

Figure 34: Illustration of response latency variation for a five second interval using T610 as the controller platform (average frequency of requests per second: ten)

From our current understanding, another driver-operating system combination is the reason for the observed pattern shown in Figure 21 (with the regular OF agent). For the same frequency of requests per second, such a behaviour is not observed when T610 is used as a controller platform. Figure 34 confirms this hypothesis as no such pattern is observed (with all other conditions remaining the same).

CHAPTER 6 Conclusion and future work

6.1 Conclusion

This work demonstrates the utility of enhancing OpenFlow to support flexible payload matches. Performing application-aware service experiments in a network using vendor's hardware would require support from the vendor and can involve multiple development cycles. Our proposal offers an effective programming interface that allows application-aware routing experiments by using OpenFlow. The value of this enhancement lies in seamless integration into the open multi-vendor nature of OpenFlow, while providing flexibility. We have realized a practical prototype that demonstrates the realistic nature of the solution, and its potential feasibility in practice. Quantitative results from a study of the prototype demonstrate the importance in terms of performance to a set of applications. Absence of such an enhancement requires the need of an external packet processor, which would result in inefficient use of network bandwidth and for certain applications, like the HTML scanner, a drastic impact to the offered service (HTTP). The negative impact may be mitigated to a certain extent by using faster packet processors. However, unlike the best-case scenario of having a controller managing a single switch (which was used to gather results in this work), a network typically has multiple switches and the packet processor could still be swamped by the arriving requests unless there is a packet processor for each switch.

The flexible payload match avoids the need of introducing more elements into the network by handing operations in the datapath. Making use of the proposed architecture can allow inspecting packet payload for traffic steering purposes or detecting threats using

signature matching. The only limitation with this solution in its current form is it can be used only for those applications that do not require storage and retrieval of meta-data. In other words, it is ideal for applications that can perform work with DPI on a per-packet basis.

6.2 Future Work

The 'HTML scanner' application used for tracking HTTP connections that render HTML5 pages does not detect all possible combinations of the allowed DOCTYPE (as noted in section 5.2). This is a direct effect of the inability of the current architecture to support matches that make use of regular expressions. This can possibly extend the utility of this approach to support a broader set of applications like the Intrusion Detection System (IDS). Extending support to such matches and evaluating the performance impact would be part of a future work.

References

[1] O. N. F. “OpenFlow Switch Specification 1.3.2,” 2013. [Online].

[2] N. Mckeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, “OpenFlow: Enabling Innovation in Campus Networks,” *ACM SIGCOMM Computer Communication Review,* vol. 38, no. 2, 2008.

[3] Z. Qazi, C.-C. Tu, R. Miao, L. Chiang, V. Sekar and M. Yu, “Practical and incremental Convergence between SDN and middle-boxes,” in *Open Network Summit*, Santa Clara, 2013.

[4] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris and S. Shenker, “Middleboxes no longer considered harmful,” in *Operating Systems Design and Implementation*, San Francisco, 2004.

[5] K.-K. Yap, R. Sherwood, M. Kobayoshi, T.-Y. Huang, M. Chan, N. Handigol, N. Mckeown and N. Parulkar, “Blueprint for Introducing Innovation into Wireless Mobile Networks,” in *Proceedings of of the Second ACM workshop on Virtualized Infrastructure Systems and Architectures*, New Delhi, 2010.

[6] A. Mahmud and R. Rahmani, “Exploitation of OpenFlow in Wireless Sensor Networks,” in *International Conference on Computer Science and Network Technology*, Harbin, 2011.

[7] A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, L. Mathy and P. Papadimitriou, “Flow Processing and the Rise of commodity network hardware,” *ACM SIGCOMM Computer Communication Review,* 2009.

[8] S. K. Fayazbakhsh, V. Sekar, M. Yu and J. C. Mogul, “Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions,” in *Proceedings of HotSDN*, Hong Kong, 2013.

[9] J. W. Anderson, R. Braud, R. Kapoor, G. Porter and A. Vahdat, “xOMB - Extensible Open Middleboxes with Commodity Servers,” in *Architectures for Networking and Communication Systems*, Austin, 2012.

[10] J. Sherry, H. Shaddi, C. Scott, A. Krishnamurthy, S. Ratnasamy and V. Sekar, “Making Middleboxes Someone Else’s Problem: Network Processing as a Cloud Service,” in *SIGCOMM*, Helsinki, 2012.

[11] “Ryu SDN Framework,” [Online]. Available: http://osrg.github.io/ryu/. [Accessed 24

May 2014].

[12] "Ryu July 24 commit," [Online]. Available: https://github.com/osrg/ryu/commit/cd49f5495617a2c69c365cf44c7c659b8bb63f3b. [Accessed 24 May 2014].

[13] "LINC-Switch," FlowForwarding, [Online]. Available: https://github.com/FlowForwarding/LINC-Switch. [Accessed 24 May 2014].

[14] "LINC Jan 28 commit," FlowForwarding, [Online]. Available: https://github.com/FlowForwarding/LINC-Switch/commit/e382830f16c83ceb9e39dff46fcdced02589d5e2. [Accessed 24 May 2014].

[15] "Erlang records," [Online]. Available: http://www.erlang.org/doc/reference_manual/records.html. [Accessed 24 May 2014].

[16] "Learning switch based on OpenFlow 1.3," [Online]. Available: https://github.com/FlowForwarding/LINC-Switch/blob/master/scripts/ryu/l2_switch_v1_3.py. [Accessed 24 May 2014].

[17] Bellovin, S., "Firewall-Friendly FTP," [Online]. Available: http://tools.ietf.org/html/rfc1579. [Accessed 24 May 2014].

[18] J. POSTEL, "FILE TRANSFER PROTOCOL," [Online]. Available: http://tools.ietf.org/html/rfc765. [Accessed 24 May 2014].

[19] "HTML5," [Online]. Available: May 20, 2014. [Accessed 24 May 2014].

[20] S. Sulaiman, S. M. Shamsuddin and A. Abraham, "A Survey of Web Caching Architectures or Deployment Schemes," *International Journal of Innovative Computing*, vol. 3, 2013.

[21] "Squid Introduction," [Online]. Available: http://www.squid-cache.org/Intro/. [Accessed 24 May 2014].

[22] O. N. F. "openflow-spec-v1.4.0," 2013. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf.

[23] "101 Best HTML5 Sites," [Online]. Available: http://101besthtml5sites.com/. [Accessed 24 May 2014].

[24] "pkt," [Online]. Available: https://github.com/esl/pkt/commit/68f5b053e05e52055295cb317f9b64ffa5bc5d76. [Accessed 24 May 2014].

[25] "Dell Precision 390," [Online]. Available: http://www.dell.com/downloads/global/products/precn/en/spec_precn_390_en.pdf. [Accessed 24 May 2014].

[26] "Dell PowerEdge T610," [Online]. Available: http://www.dell.com/us/business/p/poweredge-t610/pd. [Accessed 24 May 2014].

[27] "Dell Precision 370," [Online]. Available: http://www.dell.com/downloads/global/products/precn/en/spec_precn_370_en.pdf. [Accessed 24 May 2014].

[28] Finnie (Heavy Reading), Graham ; QOSMOS, *The role of DPI in an SDN World,* 2012.

[29] Z. Qazi, J. Lee, T. Jin, G. Bellala, M. Arndt and G. Noubir, "Application-Awareness in SDN," in *Proceedings of ACM SIGCOMM 2013*, New York, 2013.

[30] F. Daoust, P. Hoschka, C. Z. Patrikakis, R. S. Cruz, M. S. Nunes and D. S. Osborne, "Towards Video on the Web with HTML5," in *NEM Summit*, Barcelona, 2010.

[31] "HTML5 Popularity Among Fortune 500 Companies," [Online]. Available: http://www.incore.com/Fortune500HTML5/#infographic. [Accessed 24 May 2014].

[32] "vsftpd," [Online]. Available: https://security.appspot.com/vsftpd.html. [Accessed 24 May 2014].

[33] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu and M. Tyson, "FRESCO: Modular Composable Security Services for Software-Defined Networks," in *Proceedings of the 20th Annual Network and Distributed System Security Symposium (NDSS)*, San Deigo, 2013.

[34] Velrajan, Saro, *Application-Aware routing in Software-Defined Networks,* Aricent.

APPENDIX

APPENDIX A – HTML Scanner Accuracy

In this section, we present the results that determine the accuracy of the HTML scanner. We tried to access all the URLs specified in [24] using wget. Following statistics were gathered using a script:

Type	Number of Matching URLs
Sites that render HTML5 pages (recognized by detecting '<!DOCTYPE HTML')	2
Sites that render HTML5 pages (recognized by detecting '<!DOCTYPE html>')	60
Sites that render HTML5 pages (recognized by detecting <!doctype html'>)	14
Number of Timeouts (timeout for a response: 10 seconds)	17
Number of inactive sites	2
Number of sites that redirected the request (not handled by wget)	1
Sites that render HTML5 pages, but not recognized by HTML Scanner	***5***
Total	101

The logic used by the script that reported the above statistics is the same logic used for the regular OF agent. The enhanced OF agent reported the exact statistics:

Count of !DOCTYPE html>: 60

Count of !DOCTYPE HTML>: 2

Count of !doctype html>: 14

total count is 76

Ignoring the URLs that timed out/or were inaccessible, the total percentage of URLs that could not be recognized to render HTML5 pages is approximately 6% (5 out of 81).

Following listing gives the appropriate output for each URL in the format '<URL><space><category>', where category can be one of

- type_1 (refers to '<!DOCTYPE HTML>')
- type_2 (refers to '<!DOCTYPE html>')
- type_3 (refers to '<!doctype html>')
- NOT_RECOGNIZED (refers to sites that were not recognized by the HTML scanner logic)
- SITE_INACTIVE (refers to sites that are inactive)
- SITE_REDIRECT_NOT_HANDLED (refers to sites that redirected the request)
- Timeout (refers to connections that timed out)

- http://michaelacevedo.com type_3
- http://www.gaga-debki.pl type_2

- http://html5lab.pl type_2
- http://magicfabric.trefl.com type_3
- http://lucyk.pl NOT_RECOGNIZED
- http://www.estrada.fr type_2
- http://gruporastro.com.br Timeout
- http://www.am-shuuemura.jp type_2
- http://www.pelletized.com type_3
- http://thedesignsuperhero.com type_2
- http://www.adinathweb.com type_3
- http://arno.hoog.ma type_2
- http://pixelblast.net type_1
- http://wildgames.es Timeout
- http://www.fit-equip.com NOT_RECOGNIZED
- http://destroyafteruse.com type_2
- http://www.piotrkwiatkowski.co.uk type_2
- http://www.mediacross.com.br Timeout
- http://www.apple.com/safari type_2
- http://www.20thingsilearned.com type_2
- http://nikebetterworld.com type_2
- http://vua.la Timeout
- http://www.ryandelaney.co.nz NOT_RECOGNIZED

- http://agent8ball.com type_2
- http://www.mademyday.de type_2
- http://www.garbage.ro type_2
- http://devinist.de type_2
- http://www.aquaticayachtservices.com type_2
- http://gbwd.seodivers.com SITE_INACTIVE
- http://magicfabric.trefl.com type_3
- http://www.pixelbender.ca type_2
- http://www.ngenworks.com type_2
- http://www.vivid-ness.co.uk type_2
- http://mckinney.com type_2
- http://nimbupani.com type_2
- http://www.getyouram.com Timeout
- http://visuadesign.com Timeout
- http://gramy.trefl.com type_3
- http://thisisnation.com Timeout
- http://www.singleframedesign.co.uk type_3
- http://www.vision18.co.in/studio type_2
- http://waytoogood.ca type_2
- http://benthebodyguard.com type_3
- http://www.qlassik.com Timeout

- http://ts3.trefl.com type_3
- http://www.edgarleijs.nl type_3
- http://modellino.trefl.com Timeout
- http://sergiooliveira.eu type_2
- http://www.am-shuuemura.jp type_2
- http://me.evanblack.com type_2
- http://www.ivebo.net type_2
- http://pixelblast.net Timeout
- http://www.henryblake.co.uk type_2
- http://rumpetroll.com type_3
- http://colorkitchen.net type_1
- http://gruporastro.com.br Timeout
- http://grey-ang.com type_2
- http://seogadget.co.uk type_3
- http://gandrweb.com type_2
- http://www.picoinco.com SITE_INACTIVE
- http://www.ads-plus.org NOT_RECOGNIZED
- http://ainiesta.com type_2
- http://www.scientificname.net NOT_RECOGNIZED
- http://www.netlinkdemo.com/pellegrini Timeout
- http://www.installhdtv.com type_2

- http://labs.mandogroup.com type_2
- http://www.subcide.com type_2
- http://www.cazinc.co.uk type_2
- http://www.fit-equip.com SITE_REDIRECT_NOT_HANDLED
- http://html5lab.pl type_2
- http://mollar.me Timeout
- http://mebassett.gegn.net type_2
- http://thedesignsuperhero.com type_2
- http://arashzad.net type_2
- http://wantist.com type_2
- http://www.html5rocks.com type_2
- http://www.carloscabo.com type_2
- http://www.webmasterpoint.org type_2
- http://webinperu.com type_3
- http://www.eend.nl type_2
- http://www.html-5-tutorial.com type_2
- http://smultronlab.com type_2
- http://blog.daichifive.com type_2
- http://www.better2web.com type_2
- http://www.cursoclio.com.br/o-curso-clio type_2
- http://fifthstreetcreative.com type_2

- http://intenseminimalism.com Timeout
- http://munch5aday.com type_3
- http://www.publiland.com.ar Timeout
- http://garrettwinder.com type_2
- http://designc7.com type_2
- http://www.cekerholic.com type_2
- http://www.arqpsa.com type_2
- http://72ave.com type_2
- http://html5demos.com type_2
- http://www.pinkturkey.com Timeout
- http://www.justtype.de type_2
- http://www.dynakode.com type_2
- http://www.hboaventura.com type_2
- http://doubleleft.com/garbageman/ type_2
- http://vtech.trefl.com/ Timeout