Adaptive Video Streaming over OpenFlow Networks with Quality of Service

by

Hilmi Enes Eğilmez

A Thesis Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Electrical and Electronics Engineering

Koç University

July, 2012

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Hilmi Enes Eğilmez

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Prof. Ahmet Murat Tekalp

_____

Prof. Özgür Barış Akan

_____

Assoc. Prof. Öznur Özkasap

Date: _____

To my beloved wife İlknur

To my parents Reyhan and Hulusi Eğilmez

To my grandfather Dr. Hilmi Eğilmez whom I've never met

ABSTRACT

Multimedia streaming applications have stringent Quality of Service (QoS) requirements which cannot be always met by the best-effort Internet. To provide QoS, several QoS architectures have been explored over last two decades, but none of them has been truly successful and globally implemented. This thesis presents a novel QoS architecture for multimedia streaming based on OpenFlow, a Software Defined Networking (SDN) paradigm that has already attracted many commercial vendors and recently being deployed throughout the world. We leverage off OpenFlow's enhanced network control capabilities to deliver multimedia with QoS. On top of OpenFlow, we propose an optimization framework for dynamic QoS routing which fulfills the required end-to-end QoS by dynamically optimizing the routes of the multimedia traffic. Our extensive simulation results show that the proposed architecture and the optimization framework on routing significantly improve the QoS of the multimedia streaming compared to traditional shortest path routing in the Internet. In addition, we extend our framework for large scale multi-domain OpenFlow networks. We propose a distributed control plane architecture and present new methods for dynamic inter-domain QoS routing by addressing the messaging between OpenFlow controllers and the network scalability. We show that the proposed solution to the distributed routing closely approaches the globally optimal routing and nicely scales to large networks. We also implement a controller software to demonstrate the performance of our approach over a real OpenFlow network deployed in our campus. Our experimental results on the real network acknowledge our simulation results and show that we can guarantee seamless video delivery with little or no video artifacts experienced by the end users.

ÖZETÇE

Çokluortam akıtma uygulamalarının kaliteli servis ihtiyaçları en iyi çaba (best-effort) mantığıyla çalışan internet tarafından her zaman karşılanamaz. Servis kalitesini sağlamak amacıyla son yirmi yıl boyunca birçok ağ mimarisi ortaya atıldı, fakat şu ana kadar hiçbiri başarılı olamadı ve yaygınlaşamadı. Bu tezde çokluortam akıtımı için servis kalitesi sağlayan yeni bir mimari ortaya atıyoruz. Sunacağımız mimari bir Yazılım-Tanımlı Ağ (Software-Defined Networking) örneği olan ÖzgürAkış (OpenFlow) üzerine kurulmuştur. ÖzgürAkış'ın ileri ağ kontrol özelliğini kullanarak, mimarimiz üzerine dinamik olarak baştan sona kaliteli çokluortam servis yönlendimesini eniyileyen bir çerçeve öneriyouz. Kapsamlı olarak gerçekleştirdiğimiz simulasyon sonuçlarına göre önerdiğimiz mimari ve eniyileme çerçevesi günümüz internet mimarisinin en kısa yol yönlendirmesinden çok daha iyi performans gösteriyor. Buna ek olarak, sunduğumuz çerçevenin büyük ağlarda ölçeklenebilir kılmak için ÖzgürAkış'a yeni bir dağıtık sistem dizaynı ve buna bağlı dağıtık bir eniyileme çerçevesi öneriyoruz. Yaptığımız çalışmalar gösteriyor ki, önerdiğimiz çözümümüz genel(global) eniyi çözüme çok yaklaşıyor ve çok büyük, çoklu alanlı ağlara da uygulanabiliyor. Ayrıca bu fikirleri gerçek ağlar üzerinde uygulamak için bir kontrol edici (controller) yazılımı ürettik ve üniversitemize gerçek bir ÖzgürAkış ağı kurduk. Gerçek ağ üzerinde yaptığımız testler önceki simülasyonlarımızı doğruladı ve gösterdi ki, bu yeni mimari ile kesintisiz ve kaliteli vidyo servisini garantileyebiliyoruz.

ACKNOWLEDGMENTS

I would like to express my endless gratitude to Prof. A. Murat Tekalp for his excellent advisory, reliable guidance and full support. This thesis have not been written without his profound knowledge. I thank to Dr. Seyhan Civanlar for her guidance, constructive comments, and help in preparation of Chapter 2. She has been like a co-advisor of me during my master's work. Moreover, I would like to express my special thanks to Prof. Alper T. Erdoğan who inspired me first to research. I also thank to Prof. Özgür B. Akan and Prof. Öznur Özkasap for being in my thesis committee and for their valuable time.

I thank to my friends Tolga Bağcı, the angry; Ümit Baş, the trojan brother; Enes Özel, the other trojan brother; Yalçın Şadi, the reporter; Serkan Özkul, the acceptor; Tuğtekin Turan, the office-boy; and Tahsin Dane, the contactless, for making these two years enjoyable. I would like to show my thanks again to Tahsin and Tolga for helping me in preparing Chapter 5.

Last but not least, I thank to my mother, Reyhan Eğilmez, my father Hulusi Eğilmez and my brother Bedi Eğilmez for their everlasting love and support. Obviously, I thank to the first light of my life, İlknur Eğilmez for her eternal love and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

The Internet design is based on end-to-end arguments [1] where the network support is minimized and the end hosts are responsible for most of the communication tasks. This design sufficiently covers two fundamental goals of the Internet architecture [2]:

- Resiliency to failures: Internet communication must continue despite the loss of networks or gateways.
- Unified service model: The Internet must support multiple types of communications service.

End-to-end principles of the Internet allow a unified best-effort service for any type of data at the networking layer where service definitions are made at the upper layers (hosts), and it reduces the overhead and the cost at the networking layer without losing reliability and robustness. This type of architecture fits perfectly to data transmission where the primary requirement is reliability. Yet, in multimedia transmission, timely delivery is preferred over reliability. Multimedia streaming applications have stringent delay requirements which cannot be guaranteed in the best-effort Internet. So, it is desirable that the network infrastructure supports some means to provide Quality of Service (QoS) for multimedia traffic. Therefore, the Internet Engineering Task Force (IETF) has proposed several QoS architectures such as IntServ [3] and Diffserv [4], but none has been truly successful and globally implemented. This is because of two main reasons:

- They require some fundamental changes on the Internet design.
- They are built on top of the current Internet's completely distributed hop-by-hop routing architecture lacking the end-to-end information of available network resources.

Without requiring fundamental changes, Multi-protocol Label Switching (MPLS) [5] provides a partial solution via its ultra-fast switching capability, but it lacks real-time reconfigurability and adaptivity.

Software Defined Networking (SDN) [6] is a paradigm shift in network architecture where the network control is decoupled from forwarding and is directly programmable. This migration of control provides an abstraction of the underlying network for the applications residing on upper layers, enabling them to treat the network as a logical or virtual entity [7]. Among several attempts, OpenFlow is the first successful implementation [8] of SDN which has recently started being deployed throughout the world and has already attracted many commercial vendors [9]. As proposed in SDN, OpenFlow moves the network control to a central unit, called controller while the forwarding function remains within the routers, called forwarders. On the other hand, in the Internet, the routers perform both routing control and packet forwarding functions. Fig.1.1 illustrates the migration of the routing control function in OpenFlow and its difference from the Internet.

The OpenFlow controller is the brain of the network where packet forwarding decisions are made on per-flow basis and the network devices are configured accordingly via the OpenFlow protocol, which defines the communication between the controller and the underlying devices. We believe that this technology will lead to many innovative networking solutions including new QoS mechanisms. The major advantage of OpenFlow is that, unlike current QoS architectures, its implementation does not require fundamental changes on the Internet design. Many network device vendors have already started to produce OpenFlow-enabled switches/routers which are backward compatible and can also work in legacy mode. As depicted in Fig.1.2, these routers

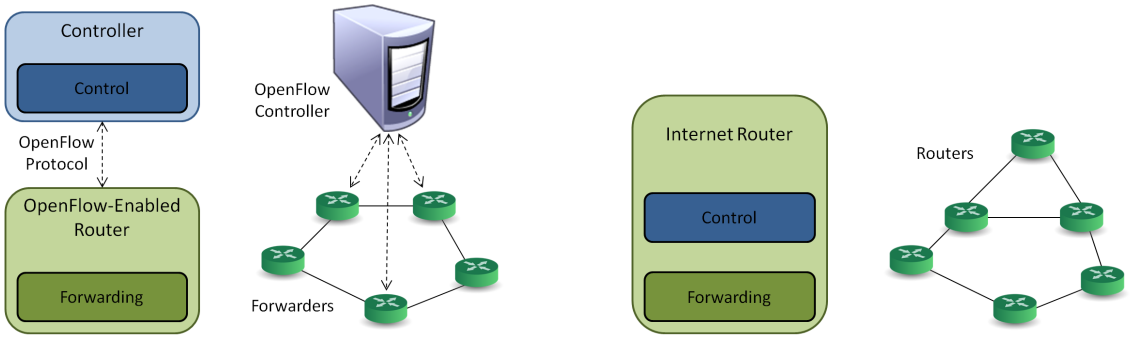


Figure 1.1: (a) OpenFlow decouples control and forwarding layers. (b)Current Internet operate both control and forwarding layers distributively

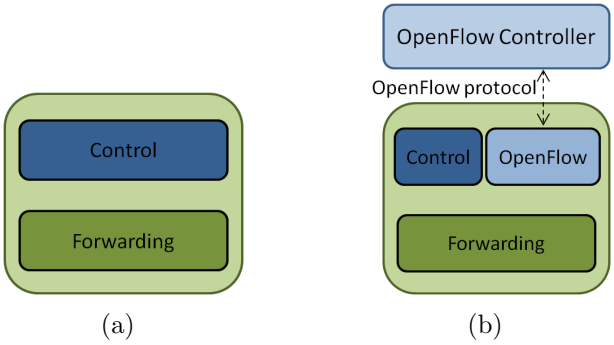


Figure 1.2: (a) An Internet router with embedded (default) control software. (b) An OpenFlow-enabled router inter-operating with the default and OpenFlow control software.

have an additional option for enabling the OpenFlow control on the network which can be switched to default control (legacy) mode. Hence, inevitably, OpenFlow will gradually spread throughout the world in the near future as new OpenFlow-enabled routers are deployed in the Internet.

The state-of-the-art streaming multimedia applications are over the Internet Protocol (IP) [10], so they do not guarantee any QoS requirements. But, they are managed to provide QoS with a high probability by exploiting application-layer QoS control techniques [11–18] along with the sophisticated multimedia compression schemes [19–24]. The application-layer techniques are well-studied in the literature

and the reader is referred to overview papers [25–28] for general information. In this thesis our focus is on network-layer adaptation strategies for multimedia streaming and application-layer QoS control techniques are not in our scope.

In this thesis, we propose a new QoS architecture by taking OpenFlow as a basis. On top of the OpenFlow controller, we construct an optimization framework for dynamic QoS routing so that the multimedia traffic is dynamically rerouted to provide required QoS while the other traffic packets follow the traditional shortest path [29,30]. Then, we extend this framework to enable QoS over multi-domain large scale OpenFlow networks which is not possible with a single controller. Since the current OpenFlow [31] only supports a single controller, we also propose a distributed control plane architecture supporting multiple controllers [32]. In addition, we implement our dynamic QoS routing approach and test over a real OpenFlow network deployed in our campus [33].

The remainder of this thesis is organized as follows: Chapter 2 reviews current QoS architectures and presents the proposed QoS architecture. Chapter 3 provides a review on QoS routing and discusses the proposed optimization framework for dynamic QoS routing which runs on top of our proposed QoS architecture. Chapter 4 proposes the distributed extension of our optimization framework for inter-domain QoS routing and we also discuss the current Internet's inter-domain routing which is based on Border Gateway Protocol (BGP) in Chapter 4. The controller software implementation and the real OpenFlow test network environment are presented in Chapter 5. Chapter 6 includes some future directions and application areas of the proposed QoS architecture. Finally, concluding remarks are given in Chapter 7.

Chapter 2

QOS ARCHITECTURES

The Internet architecture is designed to provide a unified (i.e. classless) best-effort service. So, it cannot make any promises about the end-to-end delay of a packet and the delay variation between consecutive packets in a packet stream. Due to the lack of support to deliver multimedia traffic within its timing constraints, it is an extremely challenging task to develop successful streaming multimedia applications over the Internet.

There is a continuing debate on how to evolve the Internet in order to provide QoS for multimedia traffic. Some researchers argue that fundamental changes should be done to fully guarantee QoS via end-to-end bandwidth reservations, while others think moderate changes are enough to have soft guarantees via unequal treatment of packets using new scheduling policies which will provide the requested QoS with very high probability. Another group of researchers argue that best-effort service of the Internet is adequate and networking solutions such as content distribution networks (CDNs), peer-to-peer (P2P) systems and multicast overlay networks can provide sufficient QoS. Even though multimedia applications such as YouTube and Skype have achieved considerable success, during the peak usage times of the Internet, their performance may be unsatisfactory due to variable queuing delays and congestion losses. Moreover, according to recent studies, the amount of Internet traffic has already reached up to its limits [34]. Hence, the Internet design will require new QoS mechanisms as the demand on multimedia contents continue to grow. In this chapter, we first review current QoS architectures in Section 2.1, then in Section 2.2 we present our proposed QoS architecture which is based on OpenFlow.

2.1 Review of QoS Architectures

In a nutshell, we can group the QoS architectures into two major categories:

- *Integrated Services (IntServ)* – provide hard QoS guarantees via resource reservation (bandwidth, buffer) techniques. They require fundamental changes in the network core.

- *Differentiated Services (DiffServ)* – provide soft QoS guarantees via scheduling (priority queuing) techniques. Unlike IntServ-like architectures, they require changes in the edge routers of the network.

Besides, Multiprotocol Label Switching (MPLS) [5] is a mechanism that allows ultra-fast routing. By itself, MPLS is not a QoS architecture, but it benefits many QoS architectures with its fast switching capability. In the subsequent sections, we discuss these QoS mechanisms in detail.

2.1.1 IntServ

IntServ is an architecture that specifies the elements to guarantee QoS for individual flows. In IntServ, a flow is defined as a unidirectional data stream between two applications and is uniquely identified by source and destination address pair, port numbers and the transport protocol. Intserv defines three classes of traffic (services) based on application's delay requirements (from highest to lowest priorities):

- *Guaranteed service* [35] – achieves firm (mathematically provable) bounds on end-to-end datagram queuing delays and therefore provides hard (delay and bandwidth) guarantees for each flow.

- *Controlled Load service* [36] – provides approximate QoS guarantees in a lightly loaded network. The delay service agreement is made statistically which is not violated under unloaded network conditions.

- *Best-effort service* – is similar to the service that the Internet offers.

The guaranteed service and controlled load traffic classes are based on quantitative service requirements, and both of them require signaling to reserve network resources (e.g. link bandwidth and buffers). Even though the IntServ architecture is not tied to any particular signaling protocol, Resource Reservation Protocol (RSVP) [37], is often regarded as the signaling protocol in IntServ. RSVP is used to reserve bandwidth for the applications requesting QoS for their unicast of multicast data flows. It is used by a host to request specific QoS requirements from the network and also used by routers to distribute the QoS requirements of the host. This mechanism is similar to circuit switched networks (e.g. ATM [38]) where the data transmission starts after an end-to-end connection is established. RSVP is not a routing protocol; it inter-operates with any unicast/multicast routing protocol. Routing protocols determine where packets get forwarded and RSVP is only concerned with the QoS management of those packets that are forwarded in accordance with routing. It is important to note that the routing protocol should be QoS-aware, so that the calculated routes satisfy the QoS requirements.

An IntServ-based router has to implement following traffic control mechanisms to satisfy appropriate QoS for each flow:

- *Admission control* – decide whether accept or reject an application's request depending on available resources. Admission control needs that the router understands the demands that are currently being made on its assets, so that it can predict the worst-case bounds on each service.

- *Packet scheduler* – manages the forwarding of different packet streams using a set of queues or other mechanisms such as timers.

- *Packet classifier* – To handle traffic control, each incoming packet must be mapped into some traffic class. This mapping is performed by the classifier. A class might correspond to a broad category of flows, e.g., all video flows or all flows coming from a particular source. On the other hand, a class might hold only a single flow.

The advantages of IntServ can be summarized as follows,

- IntServ's guaranteed service fully provides QoS with firm bounds on delay.
- It guarantees end-to-end QoS on per-flow basis.

However, IntServ architecture has the following drawbacks,

- Scalability; IntServ works well on small-scale networks, but it is difficult to keep track of all reservations and end-to-end signaling in a large scale network, like the Internet.
- IntServ requires fundamental changes in the network core, since all routers along the traffic path have to support it.

2.1.2 DiffServ

DiffServ is an architecture that specifies the mechanisms for classifying, managing data traffic and providing QoS for aggregated traffic classes. Unlike IntServ's fine-grained, flow-based mechanism, DiffServ is a coarse-grained, class-based mechanism for traffic management. Packets are classified and marked to receive a particular per-hop forwarding behavior on routers along their path. The main advantage of DiffServ over IntServ is the scalability which is achieved by implementing operations such as packet classification, packet marking, traffic shaping and policing at network boundaries (edges) or hosts. It does not require fundamental changes in the network core. Each router in the network only need to be configured to differentiate traffic based on its class, so that each traffic class can be managed differently based on its priority preferences. In a nutshell, a DiffServ-based network should implement,

- *Packet classification* and *packet marking* on the edge routers.
- *Per-hop behaviours (PHBs)* (e.g. priority queueing, traffic shaping) on all network routers.

Per-hop behaviours can only provide soft guarantees with statistical bounds on end-to-end delay. Although, it does not fully guarantee end-to-end QoS, it provides the requested QoS with high probability. Moreover, DiffServ does not define standard traffic classes like IntServ does. This allows flexible class prioritization definitions and service differentiation, therefore; differentiated pricing strategies to the Internet Service Providers (ISPs). However, in practice, most networks use the following commonly-defined per-hop behaviours (PHBs) in several RFCs (from highest to lowest priorities):

- *Expedited Forwarding (EF) PHB* [39] – dedicated to low-loss, low-latency traffic (e.g. video, VOIP). The mechanism is similar to RSVP via IntServ model. It is implemented using priority queuing along with rate limiting on a traffic class.

- *Assured Forwarding (AF) PHB* [40] – gives assurance of delivery under prescribed conditions. This is rough equivalent of IntServ's controlled load service.

- *Class Selector PHBs* [41] – which maintain backward compatibility with the IP Precedence field for network nodes implementing IP-based precedence based classification and forwarding.

- *Default PHB* [41] – which is typically best-effort traffic

In summary, some major advantages of DiffServ are as follows,

- In contrast to IntServ, DiffServ achieves scalability in guaranteeing QoS by implementing policing and classification functions at the boundaries of DiffServ domains. So, it proposes minimal changes in the network core.

- DiffServ does not require advance setup, reservation and time consuming end-to-end negotiation for each flow.

- DiffServ allows flexible class definitions and differentiated pricing of Internet service.

However, DiffServ architecture has following drawbacks,

- Since DiffServ-aware routers apply per-hop behaviours (PHBs) to traffic classes, it is difficult to predict end-to-end behaviour of the network. The problem is even more complicated if there are more than two DiffServ domains.

- Unlike IntServ, DiffServ does not provide hard guarantees for QoS requirements. It can only promise statistical bounds on QoS parameters (e.g. delay, throughput).

2.1.3 MPLS

MPLS is not a QoS architecture but its fast data switching capability for high speed networks makes MPLS suitable for multimedia delivery. It is a hybrid mechanism that combines the benefits of ATM's circuit switching and IP's packet routing. The fast switching is achieved by assigning packets to short MPLS labels which avoids complex routing table lookups. So, an MPLS labeled packet is switched after a label instead of a lookup into the IP table. In conventional IP, the next hop is determined based on packet's long header information which requires more effort than short MPLS labels. In addition, label lookup and switching is much faster than a routing table lookup since it can take place directly within the switched fabric and not the CPU.

In MPLS, a packet is labeled by assigning that packet to a forwarding equivalent class (FEC). This is done only once when an unlabeled packet enters to an MPLS domain. The routing is performed by label switch routers (LSRs), and labeled packets are routed according to label switched paths (LSPs). The route (LSP) selections can be determined by hop-by-hop routing or explicit routing. Hop-by-hop routing allows each LSR to independently choose the next hop for each FEC. This is the usual mode today in existing IP networks. On the other hand, explicit routes can be determined by specific LSRs via source routing or by the network operator manually. Explicit routing may be useful for a number of purposes, such as policy routing or traffic engineering [42].

The main advantages of MPLS over IP can be summarized as follows:

- MPLS's labeling mechanism avoids complex routing table lookups and provides ultra-fast switching which is suitable for delay sensitive applications.

- MPLS is considered as a layer 2.5 protocol, lying between link layer (L2) and network layer (L3). So, it can encapsulate various network layer protocols such as IP, ATM, SONET [43] and inter-operate with both packet-switched and circuit switched networks.

In order to balance the network load, MPLS based traffic engineering, MPLS-TE, [44, 45] is proposed for selecting most efficient paths across an MPLS network based on bandwidth and administrative policies. In MPLS-TE, each LSR maintains a traffic engineering (TE) state database with up-to-date network topology where any change in the network is flooded to distribute over the network. Based on network state information, constrained based routing is employed. The main advantage of implementing MPLS-TE is that it provides a combination of ATM's TE capabilities along with the class of service differentiation of IP. Moreover, [46] defines flexible solution for support of DiffServ over MPLS networks. Although this architecture is completely different from IntServ, it uses RSVP's traffic engineering extension RSVP-TE [47] as signalling protocol. Its mechanism is very similar to DiffServ over IP discussed in Section 2.1.2.

2.2 The Proposed QoS Architecture

In this section we present our proposed QoS architecture which is based on OpenFlow. We first discuss OpenFlow and its beneficial functionalities for the proposed architecture in Section 2.2.1 and then we present our controller design providing QoS in Section 2.2.2.

2.2.1 OpenFlow and Its Benefits to the Proposed QoS Architecture

In the current Internet architecture, it is not possible to change network routing on a per-flow basis. When a packet arrives at a router, it checks the packet's source and destination address pair with the entries of the routing table, and forwards it according to predefined rules (e.g. routing protocol) configured by the network operator. OpenFlow offers a new paradigm to mainly remedy this deficiency by allowing us to define different routing rules associated with data flows so that the partitions of the network's layout and traffic flows can be instantly modified. The OpenFlow controller is the key network element where routing decisions are made. Thus, different algorithms and rules in the controller associated with different data flows may yield different routing choices. The controller provides access to flow tables, and the rules that tell the network forwarders how to direct traffic flows. Fig.2.1 illustrates the OpenFlow's architecture where the controller makes per-flow decisions based on network feedback coming from the forwarders and instantly modifies the forwarders' flow tables accordingly.

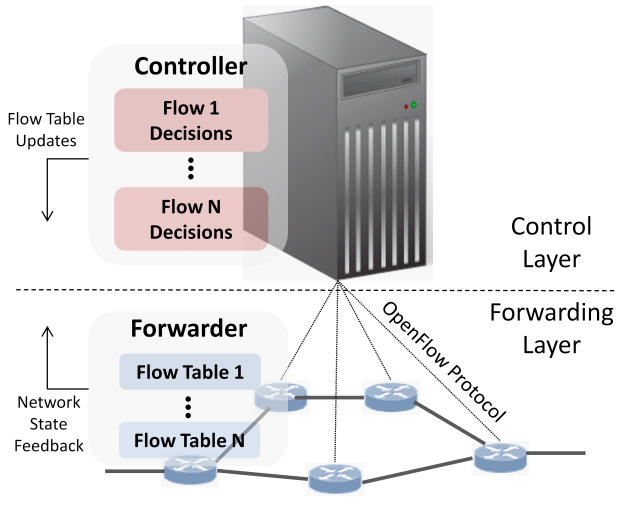


Figure 2.1: OpenFlow flow-based routing architecture

In OpenFlow, network devices store the flow entries and their associated rules in flow tables which are processed as a pipeline shown in Fig.2.2. The goal of

the pipelined processing is to reduce the packet processing time. An OpenFlow switch/router is required to have at least one flow table, and can optionally have more flow tables. When a packet arrives to an OpenFlow switch/router, it checks whether there exists a matching flow entry in flow table with index 0 or not. If a matching flow entry is found, the instruction set included in that flow entry is executed. Those instructions may explicitly direct the packet to another flow table by updating its associated action set, where the same process is repeated again. If the matching flow entry does not direct packets to another flow table, pipeline processing stops and the packet is processed with its associated action set and usually forwarded. If the packet does not match a flow entry in the table, the behaviour depends on the flow table configuration. The default action may be to send the packet to the controller and ask what to do via `PACKET_IN` message.

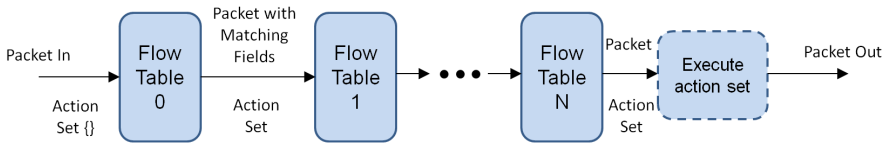


Figure 2.2: Flow tables and their pipelined processing

Using the OpenFlow protocol's flow modification messages (`OFPFC`), the controller can add, update, and delete flow entries in flow tables, both reactively (in response to packets, i.e. `PACKET_IN` messages) and proactively. A flow table contains a set of flow entries where each flow entry consists of match fields, counters, and a set of instructions to be applied on matching packets as illustrated in Fig.2.3 [31]. In each flow entry;

- *Match fields* – keep flow definitions to match against packets.
- *Counters* – hold the per-flow statistics (i.e. packet count, byte count)
- *Instructions* – keep actions associated to matching packets.

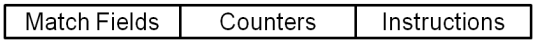


Figure 2.3: Main components of a flow entry in a flow table

In OpenFlow, we can define flows in many ways. Flows can contain same type or different types of packets. For example, packets with the TCP port number 80 (reserved for HTTP) can be a flow definition, or packets having RTP header may indicate a flow which carries voice, video or both. In essence, it is possible to set flows (i.e. matching fields) as a combination of header fields as illustrated in Fig.2.4, but the network operator should also take into account the processing power limitations of the network devices (routers or switches). In order to avoid complex flow table lookups, flow definitions should be cleverly set and if possible aggregated. Multimedia flow definitions may be determined by using the following packet header fields or values:

- Traffic class header field in MPLS,
- TOS (Type of Service) field of IPv4,
- Traffic class field in IPv6,
- If multimedia server is known, source IP address,
- Transport source and/or destination port numbers.

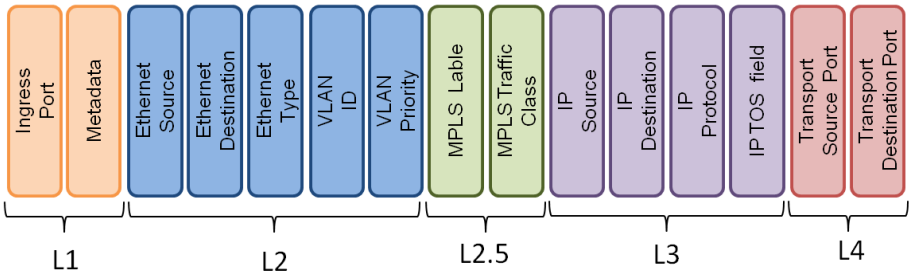


Figure 2.4: Flow identification fields in OpenFlow

It is desirable to define flows according to lower layer (L2, L3) packet headers since the packet parsing complexity is lower compared to processing up to upper layers (L4). Therefore, we propose to define multimedia flows using fields in MPLS which is considered in between data link and network layer (L2.5), and provides ultra-fast switching capability. But, in some cases upper layer header fields may also be required for better packet type discrimination, and OpenFlow allows the flexibility of defining flows using upper layer (L4) header fields. Besides, flow definitions may not rely on current IP. Any addressing scheme with service level information can be used to define multimedia type flows.

Other than providing flexible flow (class) definitions, OpenFlow eases many other QoS related problems existing in current architectures such as,

- *Network resource monitoring and end-to-end QoS support*: OpenFlow's centralized control provides complete network resource visibility and instant management over network devices to seamlessly adapt end-to-end network behaviour. Therefore, any QoS mechanism/architecture deployed on top of OpenFlow will have end-to-end QoS support.

- *Application-layer aware QoS*: By centralizing network control and making state information available up to application layer, an OpenFlow-based QoS architecture can better adapt dynamic user needs.

- *Differential services*: More granular network control with wide ranging policies at session, user, device, application levels will allow service providers to apply differential pricing strategies.

- *Virtualization*: OpenFlow enables to virtually slice a network for creating special purpose networks, e.g. file transfer network, delay-sensitive multimedia network.

- *Packet type discrimination*: The packet marking and the packet classification functions, found in DiffServ, will not be required in an OpenFlow-based QoS

architecture. The controller can easily classify and mark the packets by getting `PACKET_IN` messages from the forwarders and pushing new flow definitions to the forwarders using `OFPFC` messages. If a forwarder encounters an unknown packet type, a `PACKET_IN` message is sent to the OpenFlow controller. The controller parses the packet and informs the forwarder about what to do. If the packet is a valid type, then the controller sets necessary flow tables' entries using flow modification messages.

- *QoS routing*: To find the QoS guaranteed routes, it is essential to collect up-to-date global network state information, such as delay, bandwidth, and packet loss rate for each link. The performance of any routing algorithm is directly related to how precise the network state information is. Over large networks, collecting the network state globally may be challenging due to the scale of the network. The problem becomes even more difficult in the Internet because of its completely distributed (hop-by-hop) architecture. OpenFlow eases this task by employing a centralized controller. Instead of sharing the state information with all other routers, OpenFlow forwarders directly send their local state information to the controller. Then, the controller collects the forwarders' state information and computes the best feasible routes accordingly.

2.2.2 OpenFlow Controller Design Providing QoS

The proposed controller, depicted in Fig.2.5, offers various interfaces and functions, some of which have been part of a router in the classical Internet model.

The main interfaces of the controller are:

- **Controller-Forwarder interface**: This interface is defined by OpenFlow. The controller attaches to forwarders with a secure channel using the OpenFlow protocol to share necessary information. The controller is responsible to send flow tables associated with data flows, to request network state information from forwarders for discovering the network topology, and to monitor the network.

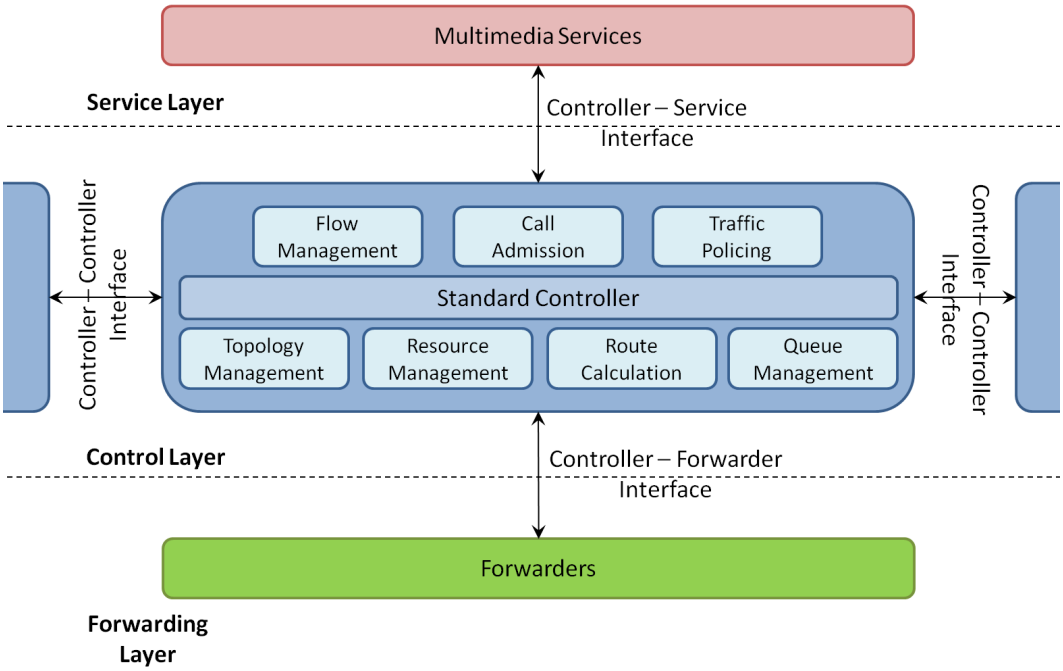


Figure 2.5: The proposed OpenFlow controller and interfaces

- **Controller-Controller interface**: The single controller architecture does not scale well when the network is large. As the number of the OpenFlow nodes increases, multiple controllers are required. This interface allows controllers to share the necessary information to cooperatively manage the whole network. This interface will be further discussed in Chapter 4.

- **Controller-Service interface**: The controller provides an open, secure interface for service providers to set flow definitions for new data partitions and even to define new routing rules associated with these partitions. It also provides a real-time interface to signal the controller when a new application starts a data flow.

The controller manages several key functions:

- *Topology Management*: This function is responsible for discovering and maintaining network connectivity through data received from forwarders.

- *Resource Management*: This function is responsible for determining the availability and packet forwarding performance of forwarders to aid the route calculation and/or queue management. This requires collecting the up-to-date network state from the forwarders on a synchronous or asynchronous basis and mapping the collected information based on a specified metric. When a forwarding decision is made for a flow, the associated flow tables are instantly modified accordingly.

- *Route Calculation*: This function is responsible for calculating and determining routes (e.g. shortest path and QoS routes) for different types of flows. Several routing algorithms can run in parallel to meet the performance requirements and objectives of different flows. This function interoperates with both topology management and resource management functions where the network topology and the network state information are input to this function.

- *Queue Management*: This function provides QoS support based on prioritization of queues. One (or more) queues can be attached to a forwarder's physical port, and flows are mapped to pre-configured queues.

- *Flow Management*: This function is responsible for collecting the flow definitions received from the service provider through the controller-service interface, and may allow efficient flow management by aggregating flow definitions.

- *Call Admission*: This function denies/blocks a request when the requested QoS parameters cannot be satisfied (e.g. there may be no feasible route), and informs the controller to take necessary actions.

- *Traffic Policing*: This function is responsible for determining whether data flows agree with their requested QoS parameters, and applying the policy rules when they do not (e.g. pre-empting traffic or selective packet dropping).

When a video service requests a QoS option from the network, it is initially received by the *call admission* function of the controller and this function determines if the requested service can be delivered based on other reservations being made. If the reservation is accepted, the *flow management* function matches the data flow with predefined multimedia flows definitions for the service starting. The data flow may start when the controller is signalled with a `PACKET_IN` message [31], at which time the *route calculation* function computes the exact route and uploads new flow table entries to appropriate forwarders. If the *resource management* function detects congestion in the network, it reactivates the *route calculation* function to determine a new route for the data flow. To mitigate network congestion, the service provider also has another option of using *queue management* function that allows to prioritize flows. It is important to note that the *traffic policing* function must also be implemented to make sure the end points conform to their Service Level Agreements (SLAs) stated in their QoS contract.

We can set up different rules for traffic coming from or going to a certain destination (e.g. from a specific server to a client), or of certain type (e.g. video) or protocol (e.g. RTP). The corresponding flow tables are dynamically uploaded to forwarders by the controller. The *route calculation* function determines the QoS routing by optimizing a different cost function other than the hop count. Routes that have larger capacity (even with longer path lengths) may be more preferable to shorter routes that cause packet loss. The QoS flows can be dynamically rerouted based on performance indicators (such as packet loss) collected on the flows' path.

When a QoS traffic is placed on a route, more packet losses may be observed on other types of traffic on the shared route. Therefore, any performance optimization process which cares about QoS flows must also consider the impact on other types of traffic. In order to minimize the adverse effects of QoS provisioning on other flows, we only propose to employ dynamic routing and to do not exploit resource reservation [3] and/or priority queuing [4, 48] mechanisms. On the other hand, the service provider may also want to have an option to set priorities to different flows. In this case,

we propose to employ both priority queuing and dynamic QoS routing, so that the dynamic routing should be triggered when the QoS requirements are not met by the forwarders' queues along the path. Hence, assuming the flow-queue mapping is static we define flow types based on their QoS routing precedences as follows:

- QoS level-1: which will be dynamically rerouted first with highest priority
- QoS level-k ($2 \leq k \leq n$): which will be dynamically rerouted after the routes of QoS level-1, ..., $(k-1)$ traffics are fixed
- Best-effort: with no dynamic rerouting (will follow shortest path)

where n is the number of QoS levels which is determined based on the application requirements. Note that the flow priorities decrease in ascending order of QoS-levels. The controller generates and sends $(n+1)$ sets of flow tables to the forwarders distinguishing the level-1,...,n QoS flows and the best-effort flow routing for the flows between the same ingress and egress points, as illustrated in Fig.2.6. We discuss the proposed optimization framework for routing of these n QoS-levels in Section 3.2 of Chapter 3.

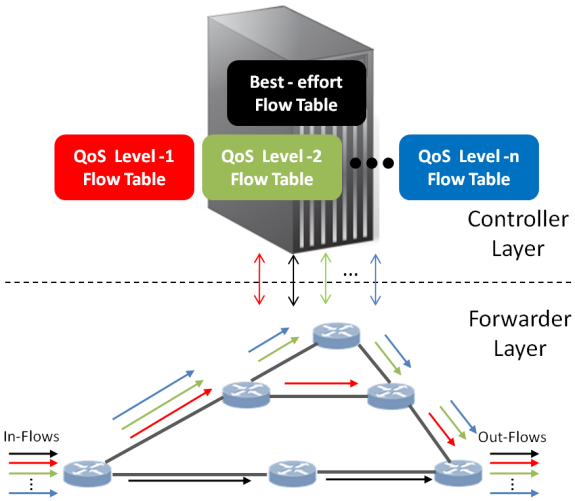


Figure 2.6: OpenFlow controller and forwarder interaction with n QoS-levels

Chapter 3

OPTIMIZATION OF QOS ROUTING

The term *QoS routing* can be defined as selecting network routes with sufficient resources which are determined according to requested QoS parameters. For multimedia applications such QoS parameters are bandwidth, end-to-end delay, delay variation (jitter), and reliability. In order to guarantee QoS, determining routes that satisfy the QoS requirements is essential and therefore; performance of any QoS architecture highly depends on its QoS routing implementation. This implementation should cover following two major building blocks:

- *A routing protocol* that collects up-to-date network state information including necessary QoS parameters from the network.

- *A routing algorithm* that is fast and closely approaches to the globally optimal route satisfying QoS requirements.

where the *routing protocol* is replaced by the OpenFlow protocol in the proposed QoS architecture (see Section 2.2 of Chapter 2). The OpenFlow controller is the network unit that keeps the complete network state and the routing algorithm(s) calculating QoS routes.

In this chapter, we provide a comprehensive study on QoS routing. Section 3.1 presents a review of QoS routing problems and algorithms. In Section 3.2, we propose an optimization framework on QoS routing for our OpenFlow-based QoS architecture. Then, we apply the proposed framework to scalable video streaming and give simulation results in Section 3.3.

3.1 Review of QoS Routing

In the literature many unicast/multicast QoS routing algorithms have been proposed. The purpose of unicast QoS routing is to find a QoS optimized route between a source and a destination node pair. On the other hand, the multicast QoS routing algorithms try to find a QoS optimized tree between a source node and multiple destination nodes. In this section, we restrict our attention to unicast QoS routing and multicast QoS routing is not in our scope. For detailed discussion of multicast QoS routing algorithms, we refer to references [49–51].

The problem of QoS routing can be posed as constrained-based path selection problems where the requested QoS parameters determine the constraints. Accordingly, we define three types of problems:

1. *Multi-Constrained Path (MCP) problem*: The network is represented as a directed simple graph $G(N, A)$, where N is the set of nodes and A is the set of all links. Each link, $(i, j) \in A$, is specified by a link weight vector with m additive weights $w_{ij}^k \geq 0$, $k = 1, ..., m$. Given m constraints D^k, $k = 1, ..., m$, the MCP problem is finding a path (route) $r \in R_{st}$ where R_{st} is the set of all paths from the source node s to the destination node t such that,

$$r^* = \left\{ r \mid f_{w_k}(r) \leq D^k \right\} \text{ for } k = 1, ..., m \tag{3.1}$$

where $f_{w_k}(r) = \sum_{(i,j) \in r} w_{ij}^k$ and r^* is the set of all feasible paths that satisfy m constraints.

2. *Multi-Constrained Optimal Path (MCOP) problem*: Given a network is represented as a directed simple graph $G(N, A)$, where N is the set of nodes and A is the set of all links. Each link, $(i, j) \in A$, is specified by a link weight vector with m additive weights $w_{ij}^k \geq 0$, $k = 1, ..., m$. Given m constraints D^k, $k = 1, ..., m$, the MCP problem is finding a path (route) $r \in R_{st}$ where R_{st} is the set of all

paths from the source node s to the destination node t such that,

$$r^* = \arg\min_r \left\{ f_{\mathbf{w}}(r) \mid f_{w_k}(r) \leq D^k \right\} \text{ for } k = 1, ..., m \tag{3.2}$$

where $f_{w_k}(r) = \sum_{(i,j)\in r} w_{ij}^k$, $f_{\mathbf{w}}(r)$ is any function of weights (weight vector) and r^* is the best feasible path that minimizes the function $f_{\mathbf{w}}(r)$ and satisfies m constraints.

3. *Constrained Shortest Path (CSP) problem*: Given a network is represented as a directed simple graph $G(N, A)$, where N is the set of nodes and A is the set of all links. Each link is specified. Each link, $(i, j) \in A$, is specified by two nonnegative additive weights, w_{ij}^1 and w_{ij}^2. The CSP problem is finding a path (route) $r \in R_{st}$ where R_{st} is the set of all paths from the source node s to the destination node t such that,

$$r^* = \arg\min_r \{ f_{w_1}(r) \mid f_{w_2}(r) \leq D_{max} \} \tag{3.3}$$

where $f_{w_1}(r) = \sum_{(i,j)\in r} w_{ij}^1$, $f_{w_2}(r) = \sum_{(i,j)\in r} w_{ij}^2$ and r^* is the best feasible path that minimizes the function $f_{w_1}(r)$ and satisfies the constraint D_{max}.

In the optimization problems above, all weights are additive which means the QoS measure of a path is the sum of individual QoS weights of the link defining the path. Fortunately, for multimedia most of the QoS parameters are additive such as delay, jitter and the logarithm of packet loss (i.e. originally packet loss is multiplicative). However, some of the QoS parameters such as bandwidth, buffer space and policy flags are not additive. Therefore, a path's QoS measure may be the minimum/maximum of the QoS link weights along that path. The constraints on min./max. QoS measures can be easily resolved by removing the the links that do not satisfy the requested min./max. QoS constraints.

The MCP, MCOP and CSP problems are all known to be NP-complete problems [52,53], so there are heuristic and approximation algorithms proposed in the literature.

Kuipers et al. [54] present an overview of constraint-based path selection algorithms for QoS routing. Chen and Nahrstedt [49] provide an overview of QoS routing over next generation high-speed networks. They present different QoS routing problems, their challenges, the QoS routing strategies and an evaluation of existing routing algorithms. Masip-Bruin et al. [55] present an accurate description of the state-of-the-art and outline research challenges in QoS routing. Wang and Crowcroft [56] propose multi-constrained path computing algorithms and also discuss cost metric selections for QoS routing. Xue et al. [57] discuss various approximation algorithms for MCP problems and propose an improved approximation scheme. Juttner et al. [58] propose a method for delay-constrained least cost QoS routing, which formulates aggregated costs and finds the optimal Lagrange multiplier using the LARAC algorithm. This method runs in polynomial time and produces a theoretical lower bound along with the solution. More recently, Chen et al. [59] presents two approximation algorithms for the CSP problem, which are alternatives to the LARAC algorithm. These two algorithms achieve smaller average path cost than LARAC, but they run slower. There are also several overview works in the literature related to QoS routing. However, there is no technical problem formulation or solution methodology in their paper. Note that references [56–59] only consider route calculation task of QoS routing and do not address how to collect information about the state of the network and the available network resources. Moreover, references [49, 54, 55] focus their attention on resource reservation and Intserv-based architectures, and do not discuss either dynamic QoS routing.

3.2 The Optimization Framework for QoS Routing

This section presents an optimization framework to design an OpenFlow controller which provides QoS guarantees to designated streams, called QoS flows; that is, to optimize routing dynamically to ensure delivery of QoS flows within specified constraints. To achieve this, we present a Constrained Shortest Path (CSP) routing framework, instead of the classical Shortest Path (SP) routing. In Section 3.2.1, we

pose dynamic rerouting as a CSP problem, and discuss cost metric and constraint selections to support n QoS levels. Then, in Section 3.2.2, we apply the LARAC algorithm [58] to solve the CSP problem for computing the optimized QoS routes.

3.2.1 Optimization of Dynamic QoS Routing as a Constrained Shortest Path Problem

We pose the dynamic QoS routing as a Constrained Shortest Path (CSP) problem. For the CSP problem, it is crucial to select a cost metric and constraints where they both characterize the network conditions and support QoS requirements. In multimedia applications, the typical QoS indicators are packet loss, delay and delay variation (jitter); therefore we need to determine the cost metric and the constraint accordingly. Obviously, all applications require that the packet loss is minimized for better QoS. However, some QoS indicators may differ depending on the type of the application:

- *Interactive multimedia applications* have strict end-to-end delay requirements (e.g. 150-200 ms for video conferencing). So, the CSP problem constraint should be based on the total delay.

- *Video streaming applications* require steady network conditions for continuous video playout; however, the initial start-up delay may vary from user to user. This implies that the delay variation is required to be bounded, so the CSP problem constraint should be based on the delay variation.

Since in this thesis our focus is on video streaming, we will employ delay variation as the constraint in our problem formulation. Note that, it is straightforward to modify the proposed problem formulation for interactive multimedia applications by using the total delay as a constraint instead of delay variation.

In our formulation, a network is represented as a directed simple graph $G(N, A)$, where N is the set of nodes and A is the set of all arcs (also called links), so that arc (i, j) is an ordered pair, which is outgoing from node i and incoming to node j. Let

R_{st} be the set of all routes (subsets of A) from source node s to destination node t. For any route $r \in R_{st}$ we define cost $f_C(r)$ and delay variation $f_D(r)$ measures as,

$$f_C(r) = \sum_{(i,j)\in r} c_{ij} \tag{3.4}$$

$$f_D(r) = \sum_{(i,j)\in r} d_{ij} \tag{3.5}$$

where c_{ij} and d_{ij} are cost and delay variation coefficients for the arc (i, j), respectively. The CSP problem can then be formally stated as finding

$$r^* = \arg\min_r \{ f_C(r) \mid r \in R_{st}, f_D(r) \le D_{max} \} \tag{3.6}$$

that is, finding a route r which minimizes the cost function $f_C(r)$ subject to the delay variation $f_D(r)$ to be less than or equal to a specified value D_{max}. This CSP problem (3.6) falls into the general category of integer linear program [60] which can be represented as follows:

$$\begin{aligned} &\text{minimize} \sum_{(i,j)\in A} c_{ij} x_{ij} \\ &\text{subject to} \sum_{(i,j)\in A} d_{ij} x_{ij} \le D_{max} \\ &\sum_{\{j|(i,j)\in A\}} x_{ij} - \sum_{\{j|(j,i)\in A\}} x_{ji} = \begin{cases} 1, & \text{if } i = s \\ -1, & \text{if } i = t \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{3.7}$$

where x_{ij}, $\forall (i, j) \in A$, is the flow variable, such that

$$x_{ij} = \begin{cases} 1, & \text{if } (i,j) \in r^* \\ 0, & \text{otherwise} \end{cases} \tag{3.8}$$

where 1 and -1, in the equality constraint, represent flow divergence on node i, c_{ij} is the cost coefficient for the arc/link (i, j), d_{ij} is the delay variation coefficient for the arc/link (i, j), D_{max} is the maximum tolerable delay variation, s is the source node, and t is the destination node. We select the cost metric as the weighted sum of packet loss measure and delay variation as follows,

$$c_{ij} = (1 - \beta)d_{ij} + \beta p_{ij} \text{ for } 0 \leq \beta \leq 1, \forall (i, j) \in A \tag{3.9}$$

where p_{ij} denotes the packet loss measure for the traffic on link (i, j), and β is the scale factor. The formula for p_{ij} is as follows,

$$p_{ij} = \begin{cases} \frac{Q^t_{ij}+T_{ij}-B_{ij}}{Q^t_{ij}+T_{ij}}, & B_{ij} < Q^t_{ij} + T_{ij} \\ 0, & B_{ij} \geq Q^t_{ij} + T_{ij} \end{cases} \tag{3.10}$$

where B_{ij} is the bandwidth of the link (i, j), T_{ij} is the amount of best-effort traffic observed on the link (i, j) and Q^t_{ij} is the total amount of QoS traffic (i.e. sum of individual QoS level traffics: $Q^t_{ij} = Q^1_{ij} + Q^2_{ij} + ... + Q^n_{ij}$) on the link (i, j). It is crucial that forwarders return accurate (up-to-date) estimates of p_{ij} and d_{ij} to determine precise routes. In the proposed controller architecture (Section 2.2 of Chapter 2), the *routing management* function collects data from forwarders (i.e., proper estimates of p_{ij} and d_{ij}) and passes them to the *route calculation* function. At the forwarder level, necessary parameters are estimated as follows:

- *Packet loss measure* (p_{ij}) is calculated using Eqn.3.10 where B_{ij}, $Q^1_{ij}, ..., Q^n_{ij}$ and T_{ij} are required parameters for the calculation. OpenFlow protocol enables us to monitor the per-flow traffic amounts (i.e., Q^1_{ij},...,Q^n_{ij} and T_{ij}) on each link. This is done by per-flow counters maintained in the forwarders. The controller can collect the per-flow statistics whenever it requests [31]. The link bandwidth, B_{ij}, is assumed to be known by experimenting or setting manually during the network setup.

- *Delay* is obtained by averaging the observed delay using time stamping (e.g. RTP [61])

- *Delay variation* (d_{ij}) is measured as the first derivative (rate of change) of the delay.

The weight β determines the relative importance of the delay variation and the packet losses depending on network and traffic characteristics. For large β, route selection would be more sensitive to packet losses on the QoS route. Vice versa, for small β route selection would be more sensitive to delay variation.

In order to find n QoS routes for each flow, defined in Section 2.2.2 of Chapter 2, we solve the proposed CSP problem (using the method in Section 3.2.2) successively n times; we first find the QoS level-1 route (with highest priority) and then find the QoS level-2 route by fixing the QoS level-1 route and modifying the cost parameters accordingly. After first two QoS level flows are set, necessary cost parameters are modified to calculate QoS level-3 flow's route. This procedure continues up to reach QoS level-n route. For the QoS level-1 route, r_1, we directly use the estimated packet loss measure (p_{ij}) and delay variation (d_{ij}) parameters to calculate cost coefficients (c_{ij} in Eqn.3.9) and the CSP problem is solved accordingly to get the optimal route, r_1^*. Then, the QoS level-1 flows (Q_{ij}^1) are rerouted to the optimal route, r_1^*. The QoS level-2 route, r_2, is found after the route of the QoS level-1 flows is fixed. In order to do this, we update the packet loss measure, denoted as $p_{ij}^{(1)}$, by removing the observed Q_{ij}^1 traffic from its old route, r_{old}^1, and placing it to the optimal QoS level-1 route, r_1^*, which can be formulated as follows,

$$p_{ij}^{(1)} = \begin{cases} \frac{Q_{avg}^1+Q_{ij}^t+T_{ij}-B_{ij}}{Q_{avg}^1+Q_{ij}^t+T_{ij}}, & B_{ij} < Q_{avg}^1 + Q_{ij}^t + T_{ij}, (i,j) \in r_1^* \\ 0, & B_{ij} \geq Q_{avg}^1 + Q_{ij}^t + T_{ij}, (i,j) \in r_1^* \\ \frac{Q_{ij}^t-Q_{ij}^1+T_{ij}-B_{ij}}{Q_{ij}^t-Q_{ij}^1+T_{ij}}, & B_{ij} < Q_{ij}^t - Q_{ij}^1 + T_{ij}, (i,j) \in r_{old}^1 \\ 0, & B_{ij} \geq Q_{ij}^t - Q_{ij}^1 + T_{ij}, (i,j) \in r_{old}^1 \\ p_{ij}, & \text{otherwise} \end{cases} \tag{3.11}$$

where Q^1_{avg} is the average QoS level-1 traffic on the route r^1_{old}. Then, we recalculate the cost coefficients (c_{ij} in Eqn.3.9) using updated packet loss measure ($p^{(1)}_{ij}$) as p_{ij} and estimated delay variation (d_{ij}). We solve the CSP problem using the new cost coefficients to get the route, r^*_2, for QoS level-2 (Q^2_{ij}) flows and these flows are rerouted to the route r^*_2. The remaining $n-2$ QoS level routes are calculated by following the same procedure. The only difference is in the formulation of the packet loss measure update which is generalized as follows, assuming QoS level-k route is calculated,

$$p^{(k)}_{ij} = \begin{cases} \frac{Q^k_{avg}+Q^t_{ij}(k-1)+T_{ij}-B_{ij}}{Q^k_{avg}+Q^t_{ij}(k-1)+T_{ij}}, & B_{ij} < Q^k_{avg} + Q^t_{ij}(k-1) + T_{ij}, (i,j) \in r^*_k \\ 0, & B_{ij} \geq Q^k_{avg} + Q^t_{ij}(k-1) + T_{ij}, (i,j) \in r^*_k \\ \frac{Q^t_{ij}(k-1)-Q^k_{ij}+T_{ij}-B_{ij}}{Q^t_{ij}(k-1)-Q^k_{ij}+T_{ij}}, & B_{ij} < Q^t_{ij}(k-1) - Q^k_{ij} + T_{ij}, (i,j) \in r^k_{old} \\ 0, & B_{ij} \geq Q^t_{ij}(k-1) - Q^k_{ij} + T_{ij}, (i,j) \in r^k_{old} \\ p^{(k-1)}_{ij}, & \text{otherwise} \end{cases} \tag{3.12}$$

where r^*_k is the optimal QoS route for level-k flow, $p^{(k-1)}_{ij}$ is the packet loss measure update from previous step, Q^k_{ij} is the amount of QoS level-k traffic, Q^k_{avg} is the average QoS level-k traffic on its old route, r^k_{old}, and $Q^t_{ij}(k-1)$ is the total amount of QoS traffic after calculating and rerouting $k-1$ QoS routes. The updated packet loss measure, $p^{(k)}_{ij}$, is used as p_{ij} in Eqn.3.9, then QoS level-$(k+1)$ route (r^*_{k+1}) is calculated by solving the CSP problem according to new cost parameters.

3.2.2 Solution to the Constrained Shortest Path Problem

The solution of the CSP problem stated in (3.6) will give the minimum cost route satisfying a pre-specified maximum delay variation from source to destination. However, the CSP problem is known to be NP-complete [56], so there are heuristic and approximation algorithms as discussed in Section 3.1 . Here, we propose to use the Lagrangian Relaxation Based Aggregated Cost (LARAC) algorithm since it is a polynomial-time algorithm that efficiently finds a good route without deviating from the optimal solution [58], [62].

The LARAC algorithm solves the Relaxed CSP (RCSP) problem which is obtained by replacing (relaxing) the condition $x_{ij} = 0$ or 1 (see Eqn.3.8) with $x_{ij} \geq 0$. Moreover, the LARAC algorithm exploits the dual of the RCSP problem, which is defined as a maximization problem,

$$\begin{aligned} &\text{maximize} \quad L(\lambda) \\ &\text{subject to} \quad \lambda \geq 0 \end{aligned} \tag{3.13}$$

where the Lagrangian dual function, $L(\lambda)$, is defined as,

$$\begin{aligned} L(\lambda) &= \min\{f_\lambda(r) - \lambda D_{max} \mid r \in R_{st}\} \\ &= \{f_\lambda(r) \mid r \in R_{st}\} - \lambda D_{max} \end{aligned} \tag{3.14}$$

and

$$f_\lambda(r) = \sum_{(i,j)\in r} (c_{ij} + \lambda d_{ij}) \tag{3.15}$$

denotes the aggregated cost of a route $r \in R_{st}$ with the Lagrange multiplier $\lambda \geq 0$.

Note that, the minimization term in Eqn.3.14, $\{f_\lambda(r) \mid r \in R_{st}\}$, can be easily solved by using Dijkstra's shortest path algorithm [63] on aggregated arc costs, for a given λ. The Lagrangian dual function (3.14) gives the lower bound on optimal value of the original RCSP problem for each $\lambda \geq 0$. By maximizing the Lagrangian dual function, $L(\lambda)$, we can find the best lower bound for the optimal solution [64]. This leads us to the solution of dual RCSP problem in (3.13). In order to solve the dual problem given by (3.13), it is crucial to search for the optimal λ and to determine the criteria for stopping the search. The LARAC algorithm, presented in Fig.3.1, provides an efficient search procedure for λ.

In the LARAC algorithm of Fig.3.1, $\mathbf{G}$ denotes the network representation as a directed simple graph, $\mathbf{c}$ is the vector of cost coefficients, c_{ij}, for all links, $\mathbf{d}$ is the vector of delay coefficients, d_{ij}, for all links, $\mathbf{c}_\lambda$ is the vector of aggregated link costs coefficients, $c_{ij} + \lambda d_{ij}$, for all links, D_{max} is the maximum tolerable delay, s is the source node, and t is the destination node. Furthermore, $\mathit{Dijkstra}(\mathbf{G}, s, t, \mathbf{c})$,

```
procedure LARAC(G, s, t, c, d, D_max)
    r_C ← Dijkstra(G, s, t, c)
    if f_D(r_C) ≤ D_max then return r_C
    else r_D ← Dijkstra(G, s, t, d)
        if f_D(r_D) > D_max then return "no feasible solution"
        else
            while true do
                λ ← (f_C(r_C) − f_C(r_D)) / (f_D(r_D) − f_D(r_C))
                r ← Dijkstra(G, s, t, c_λ)
                if f_λ(r) = f_λ(r_C) then return r_D
                else if f_D(r_C) ≤ D_max then r_D ← r
                else r_C ← r
                end if
            end while
        end if
    end if
end procedure
```

Figure 3.1: LARAC Algorithm

Dijkstra$(\mathbf{G}, s, t, \mathbf{d})$ and *Dijkstra*$(\mathbf{G}, s, t, \mathbf{c}_\lambda)$ procedures calculate shortest paths using link costs, link delays and aggregated link costs, respectively. The functions $f_C(r)$, $f_D(r)$ (see Eqns.3.4, 3.5) and $f_\lambda(r)$ (see Eqn.3.15) give the cost, delay and aggregated cost of route, respectively.

In the first step, LARAC finds shortest path (route) r_C using link costs. If r_C satisfies the delay constraint, then it is the optimal route. Otherwise, the algorithm checks to determine if a feasible solution exists. To this effect, the algorithm calculates the shortest path (route) r_D using link delays. If the minimum delay route r_D does not satisfy the delay constraint, then a feasible solution does not exist, and the algorithm stops. Otherwise, the algorithm finds the optimal solution by iteratively searching for optimal λ (see while block in Fig.3.1). In [58], the authors provide an algebraic approach to efficiently search for λ and to determine the stopping criterion. The corresponding proofs are provided in [62].

It is shown that LARAC is a polynomial-time algorithm that efficiently finds a good route without deviating from the optimal solution [58], [62] in $O([n + mlogm]^2)$

time [65], where n and m are the number of nodes and links, respectively. LARAC also provides a lower bound for the theoretical optimal solution, which leads us to evaluate the quality of the result. By further relaxing the optimality of routes, it provides some flexibility to control the tradeoff between optimality and runtime of the algorithm. Therefore, the LARAC algorithm is well suited for use in real-time dynamic QoS routing.

3.3 Application of the Proposed Framework to Scalable Video Streaming

In this section, we present methods for QoS-enabled streaming of videos, which are scalable encoded in one base layer and one or more enhancement layers. Naturally, there will be an extra cost for requesting QoS from the service provider. If the video is encoded at a single layer, such as using H264/MPEG-4 AVC [19], the entire video needs to be served with either QoS or best-effort. While serving the entire video with QoS clearly provides the highest quality, it comes at a premium cost. On the opposite end, streaming entire video with best-effort cannot provide any quality guarantee. Alternatively, scalable video coding, such as MPEG-4 SVC [20, 26], encodes video in a base layer and one or more enhancement layers and thereby provides ability to offer different grades of service at a trade-off point between reasonable guaranteed quality and reasonable cost. In order to guarantee a reasonable quality, it is sufficient to stream the base layer video without any packet loss or delay variation, while enhancement layers can be served with best-effort – or when capacity is available as a QoS stream. Note that all enhancement layers are decoded with reference to the base layer; hence, it is crucial that the base layer is streamed without any packet loss or delay variation to guarantee continuous video playback at the receiving peer, while the enhancement layers may be regarded as discardable, since loss of an enhancement layer packet typically causes only small variation in the received video quality. In the following, we propose two approaches to map the base and enhancement layers of video to level-1 and level-2 QoS, defined in Section 2.2.2 of Chapter 2 where we set

the number of QoS flows to 2 ($n = 2$).

Approach-1: The video is MPEG-4 SVC (scalable) encoded, where the base layer packets are sent as level-1 QoS flow, which ensures rerouting to avoid packet losses and minimize delay variation, and the rest of the video (enhancement layer packets) remain as best-effort flows. This approach implies the controller generates two flow tables; one for the QoS flow (level-1); and one for the rest of the traffic (best-effort).

Approach-2: The video is MPEG-4 SVC (scalable) encoded, where the base layer packets are sent as level-1 QoS flow, while the rest of the video (enhancement layer packets) are sent as level-2 QoS flow, which provides lower priority rerouting after the routes of level-1 traffic are determined and fixed. This approach implies that the controller generates three flow tables; one for level-1 QoS flow; one for level-2 QoS flow, and one for the best-effort flow for cross traffic. We assume that level-1 QoS will be the most expensive and best-effort will be the least expensive option, and investigate whether rerouting of SVC enhancement layer in *Approach-2* (more expensive) results in noticeable video quality increase compared to *Approach-1*.

We will also evaluate the performances of the proposed *two approaches* against *three benchmarks*:

Benchmark-1: The video is H.264/AVC (not scalable) encoded, and all packets are sent using level-1 QoS support. This approach is used as a benchmark to compare the efficiency of layered coding (MPEG-4 SVC), and represents the highest quality but also the highest cost solution.

Benchmark-2: The video H.264/AVC (not scalable) encoded, and all packets are sent using best-effort flows. This approach represents the least quality but also the least cost solution.

Benchmark-3: The video is SVC encoded, and all packets are sent using best-effort flows.

We have simulated a comprehensive test network to evaluate the performance of several variants of the proposed QoS routing architecture for Scalable Video Streaming using an open-source network optimization tool, Library for Efficient Modeling and

Optimization in Networks (LEMON) [66]. LEMON is a C++ template library which solves combinatorial optimization problems with network graphs.

In order to generate a realistic large scale network, we employ the Transit-Stub (TS) network model [67], implemented by the Georgia Tech. Internetwork Topology Modeling Tool (GT-ITM) [68]. The TS model uses two types of routing domains, called transit and stub domains, modeling the typical Internet backbone and local networks attached to the backbone, respectively. By interconnecting these two types of domains, GT-ITM generates a connected network according to given parameters. Fig.3.2 illustrates a small cross-section of a network using the Transit-Stub model. With GT-ITM, we generate two simulation networks with 100 nodes (5 Transit domains) and 300 nodes (15 Transit domains), respectively. These networks are input to LEMON. The two nodes that represent the video streaming server and client are selected to be in different transit domains.

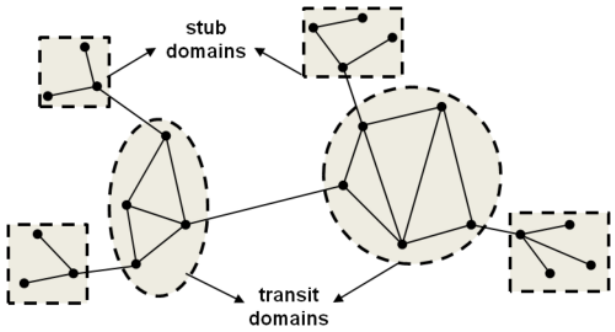


Figure 3.2: Transit-Stub model

We use the MPEG test video "Train" which is 704 × 576 pixels and 241 frames. We loop original sequence 4 times to generate 900 frames lasting about 30 seconds. This video is scalable encoded using the SVC reference software, Joint Scalable Video Mode (JSVM 9.19), to obtain an SVC base layer stream and an enhancement layer stream where the Group of Picture (GoP) size is set to 16. We set all link capacities to be 15 Mbps and the cross traffic on each link is modeled as an independent Poisson random process which is a good model for bursty nature of the Internet in sub-second

small time scales [69]. The link delays are modeled as Γ-distributed random variables with means 10 ms, 15 ms and 20 ms where we randomly assign these random variables to each link. The maximum tolerable delay variation, D_{max}, is set to 200 ms.

The proposed controller calculates new routes for the video stream by solving the CSP problem posed in Section 3.2. To solve this optimization problem, the controller uses the LARAC algorithm (see Section 3.2.2) implemented in LEMON. Dynamic routing is performed at each second which corresponds to 2 GoPs of SVC video approximately. In a nutshell, for each time interval (1 second) the proposed controller method executes following steps:

- detects which video packets are lost on the previous route and calculates the packet loss probability p_{ij} and delay variance d_{ij};
- calculates the cost coefficients c_{ij} using p_{ij} and d_{ij} from previous step for given β value (see Eqn.3.9);
- calculates necessary routes (QoS level-1 and level-2) by solving the CSP problem stated in (3.6) using the LARAC algorithm (see Fig.3.1);
- reroutes the SVC video layers according to *approaches* (1 and 2) or *benchmarks* (1, 2 and 3).

By matching the lost packets with the Network Access Layer (NAL) units of the SVC video stream, we find which NAL units are lost in the received video stream and erase them. Then, the manipulated video is decoded and the Peak Signal to Noise Ratio (PSNR) is measured.

We execute 18 different test scenarios as combinations of the following simulation parameters.

- *Network size*: We simulate two different networks with
  1. 100 nodes;

2. 300 nodes.

- *Congestion Level*: We model three congestion levels for shortest path links with

  1. Poisson random processes with mean 13 Mbps;
  2. Poisson random processes with mean 14 Mbps;
  3. Poisson random processes with mean 15 Mbps.

  while other links have less congestion (half of the congestion on the average).

- *Video encoding*: We encode the "Train" video in 3 different configurations (see Fig.3.3)

  1. *Train1*: SVC base and enhancement layers are encoded at 0.5Mbps (31.61 dB) and 1.8Mbps (39.59 dB), respectively;
  2. *Train2*: SVC base and enhancement layers are encoded at 1.0Mbps (35.32 dB) and 1.3Mbps (39.85 dB), respectively;
  3. *Train3*: The video is encoded as single layer (non-scalable) at 2.3Mbps (41.30 dB).

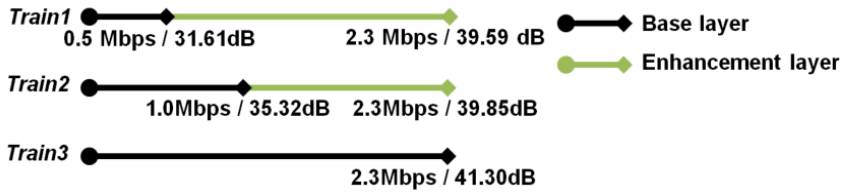


Figure 3.3: Rate and quality measures for three different encoding configurations

We perform Monte-Carlo simulations that is, we repeat each test scenario 25 times and evaluate the corresponding average PSNR values. For each test scenario, we compare the performance of the traditional best-effort Internet routing and the proposed dynamic QoS routing approaches. The results are depicted in Fig.3.4 in

terms of received video quality by calculating the average PSNR values within each GoP, and we observe that the proposed QoS routing approaches (*Approach-1* and *Approach-2*) significantly outperform the traditional Internet routing (*Benchmark-2* and *Benchmark-3*). In addition, Table 3.1 summarizes the performances of the proposed approaches and benchmarks in terms of the overall average PSNR values. All improvement values are given with respect to *Benchmark-2*.

Inspection of experimental results yields the following observations:

1. The proposed optimization scheme for dynamic rerouting of QoS streams over OpenFlow networks improves the quality of scalable video streaming significantly while causing minimum disturbance on best-effort traffic.

2. Dynamically rerouting the base layer only with QoS (*Approach-1*) attains in the range of 6-12 dB overall PSNR improvement which is significant over sending the entire video stream with best-effort.

3. Dynamically rerouting enhancement layers also (*Approach-2*) provides only marginal improvement (max. 2 dB in overall PSNR) at higher congestion levels given that the base layer bitrate is selected high enough to provide acceptable video quality.

4. As base layer video coding rate increases, both approaches (1 and 2) get closer to rerouting entire non-scalable coded video stream with QoS (*Benchmark-1*), as expected.

Therefore, we conclude that in streaming scalable video over an OpenFlow network, dynamic rerouting of base layer video only (*Approach-1*) is sufficient to attain significant quality improvement over streaming scalable or non-scalable video with best-effort if the network congestion level is low or base layer's bitrate is high enough to achieve high video quality. Further dynamic rerouting of enhancement layer (*Approach-2*) pays off the cost of forwarding along QoS level-2 route if the network congestion level is high and base layer's bitrate is low compared to overall video

bitrate. Moreover, since the proposed QoS provisioning scheme is only based on dynamic routing and built on top of OpenFlow's flow reservation paradigm, it has no adverse effect on any other type of flows.

Table 3.1: Performance Comparison of Proposed Approaches and Benchmarks

Network	**Video rate (Mbps)**			**Base=0.5, Enh.=1.8**			**Base=1.0, Enh.=1.3**			**Single=2.3**	
	Method			**A1**	**A2**	**B3**	**A1**	**A2**	**B3**	**B1**	**B2**
100 (nodes)	Congestion Level	*13*	PLR	8%	5%	13%	7%	4%	12%	2%	13%
			PSNR (dB)	34,30	35,04	29,29	34,30	36,44	29,43	39,53	28,59
			Improv. (dB)	5,71	6,45	0,70	5,71	7,85	0,84	10,94	0
		14	PLR	11%	5%	18%	10%	4%	18%	2%	18%
			PSNR (dB)	33,79	35,06	26,51	35,76	36,57	26,44	39,60	25,62
			Impr. (dB)	8,17	9,44	0,89	10,14	10,95	0,82	13,98	0
		15	PLR	12%	4%	24%	14%	4%	23%	2%	22%
			PSNR (dB)	33,72	35,36	24,17	35,90	36,81	24,14	39,72	23,87
			Improv. (dB)	9,85	11,49	0,30	12,03	12,94	0,27	15,85	0
300 (nodes)	Congestion Level	*13*	PLR	14%	8%	20%	14%	9%	19%	4%	20%
			PSNR (dB)	31,75	32,40	25,20	32,87	33,49	25,65	38,77	24,32
			Improv. (dB)	7,43	8,08	0,88	8,55	9,16	1,33	14,45	0
		14	PLR	17%	8%	28%	18%	8%	28%	4%	29%
			PSNR (dB)	31,59	32,99	23,17	33,27	33,91	22,89	38,14	22,38
			Improv. (dB)	9,21	10,61	0,79	10,89	11,53	0,51	15,75	0
		15	PLR	20%	8%	37%	20%	7%	35%	4%	36%
			PSNR (dB)	31,40	33,62	21,64	33,20	34,25	21,65	37,93	21,27
			Improv. (dB)	10,12	12,35	0,37	11,93	12,98	0,37	16,65	0

Abbreviations: A1 = *Approach-1*, A2 = *Approach-2*, B1 = *Benchmark-1*, B2 = *Benchmark-2*, B3 = *Benchmark-3*,
PLR = Packet Loss Rate,
Improv. = Improvement values with respect to *Benchmark-2*.

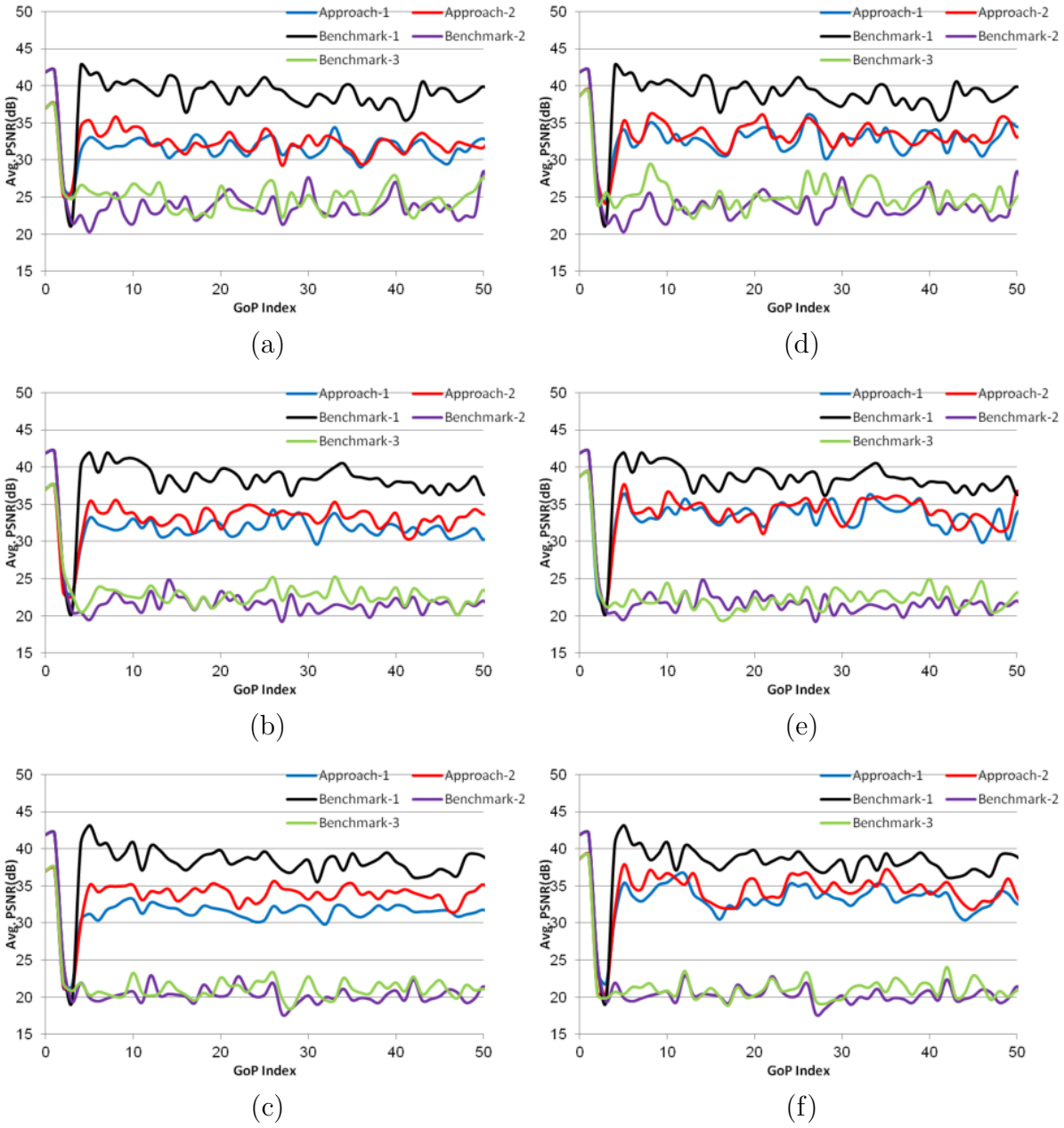


Figure 3.4: Comparison of the proposed approaches and benchmarks obtained over the network with 300 nodes by streaming *Train1* under congestion levels: (a)13, (b)14, (c)15 and *Train2* under congestion levels: (d)13, (e)14, (f)15.

Chapter 4

DISTRIBUTED QOS ARCHITECTURE FOR MULTI-DOMAIN OPENFLOW NETWORKS

This chapter proposes a Quality of Service (QoS) optimized routing architecture for video streaming over large-scale multi-domain OpenFlow networks managed by a distributed control plane, where each controller performs optimal routing within its domain and shares summarized intra-domain routing data with other controllers to reduce problem dimensionality for calculating inter-domain routing. We apply the proposed architecture to streaming of scalable (layered) videos, where the base layer routes are dynamically optimized to fulfill a required QoS level, while enhancement layers follow traditional shortest path. As we have showed in Chapter 3, rerouting the base layer only is sufficient in most of the cases.

The current OpenFlow [31] only supports networks with a single controller which is not scalable. FlowVisor [70] provides an interface for virtual multiple controllers but it is for managing multiple network slices within the same network domain. As the size/number of OpenFlow networks increase, the single controller architecture is not scalable to manage the whole network because of two main reasons:

- a single controller may not be able to update flow tables of all forwarders in time due to limited processing power and latency introduced by physically distant forwarders;

- there would be a large volume of traffic towards the controller due to messaging between controller and all forwarders.

Therefore, it is essential to implement a distributed control plane supporting multiple controllers. In the literature, there are distributed control plane designs such as Onix

[71] and HyperFlow [72], but none provides an overall network-wide QoS architecture.

In Chapter 3, we have proposed dynamic QoS routing for scalable video streaming over OpenFlow networks, but we have assumed that a single controller has full access to all link state information (not feasible for large networks) to determine the globally optimum routes. This chapter extends this to large-scale OpenFlow networks managed by a distributed control plane in which each controller is responsible for its dedicated intra-domain QoS routing and exchange messages with other controllers to help inter-domain QoS routing decisions. In the remainder of this chapter, Section 4.1 provides a review of current inter-domain routing over Internet. In Section 4.2, the distributed QoS architecture and the controller-controller interface is presented. Section 4.3 offers two control plane designs. Section 4.4 defines the proposed distributed dynamic QoS routing problem and introduces the proposed solution. Application of the distributed routing framework to the scalable video streaming and the simulation results are given in Section 4.5.

4.1 Review of Inter-Domain Routing

There is a common misconception that the general view of Internet topology can be considered as a cloud of routers connecting end-hosts, as shown in Fig.4.1(a), where the routers cooperatively make global routing decisions using routing protocols that exchange shortest path or similar information. Unfortunately, the Internet routing infrastructure is not that simple. The Internet service is provided by a large number of commercial enterprises so called Internet Service Providers (ISPs) which are in competition with each other. However, for the global connectivity cooperation is preferred over competition. Therefore, ISPs have to set their routing policies based on some agreements. The more accurate view of the Internet topology is illustrated in Fig.4.1(b).

The current Internet routing architecture is divided into domains called Autonomous Systems (ASes) where each AS is owned by a single commercial entity. In essence, the Internet routing can be categorized into two classes:

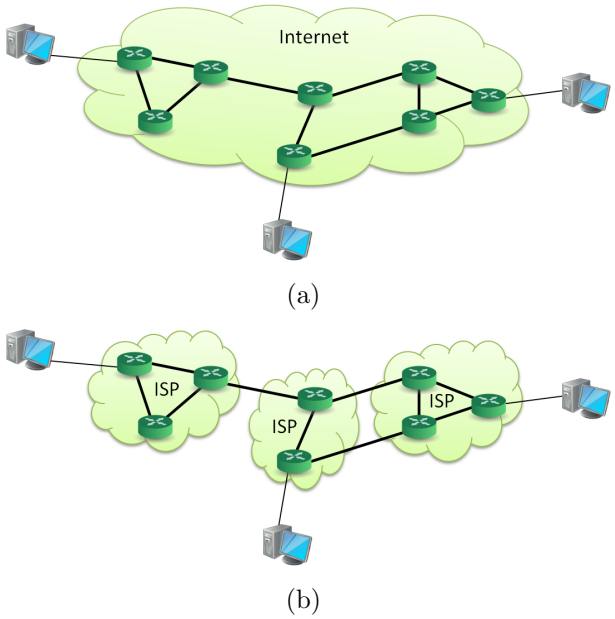


Figure 4.1: Internet topology abstraction (a) as a cloud of routers , (b) as a collection of a number of commercial entities

- *Intra-AS Routing* relies on Interior Gateway Protocols (IGPs) including protocols such as RIP [73], OSPF [74], IS-IS [75] and Cisco's EIGRP.

- *Inter-AS Routing* relies on Exterior Gateway Protocols (EGPs) such as Border Gateway Protocol, Version 4 (BGP4), simply called BGP, [76].

IGPs are concerned with *optimizing a path metric* which requires the complete network state information and therefore; IGPs are not scalable to wide-area networks. On the other hand, an EGP, e.g. BGP, is concerned with *reachability*, and implements *AS-specific routing policies* in a scalable manner. In the Internet, each AS can run a different IGP protocol, but the routing between ASes is typically based on BGP, as illustrated in Fig.4.2. BGP provides each AS

- to obtain subnet reachability information, i.e. IP-address prefixes, from its neighbour ASes,

- to advertise the reachability information to all routers within the AS,

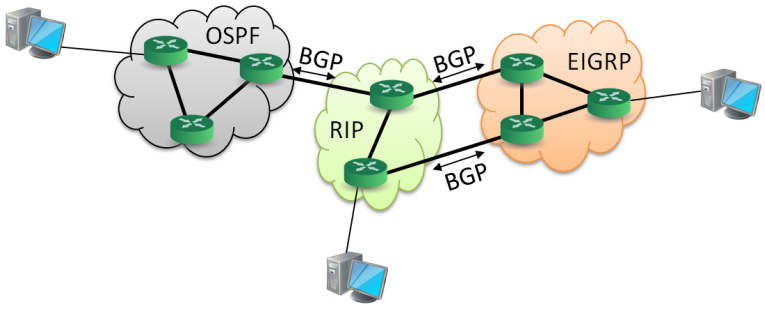


Figure 4.2: IGPs and EGPs in the Internet

- to determine routes to outer subnets based on reachability information and the AS's policies.

In BGP, the pairs of routers exchange routing information, e.g. reachability, over TCP connections so called BGP sessions. There are two types of BGP sessions (see Fig.4.3):

- To share inter-AS routing information, *external BGP (eBGP) sessions* are established between the border routers of different ASes.

- To inform intra-AS routers about the reachability over neighbour ASes, each border router in an AS establishes *internal BGP (iBGP) sessions* to the routers within AS.

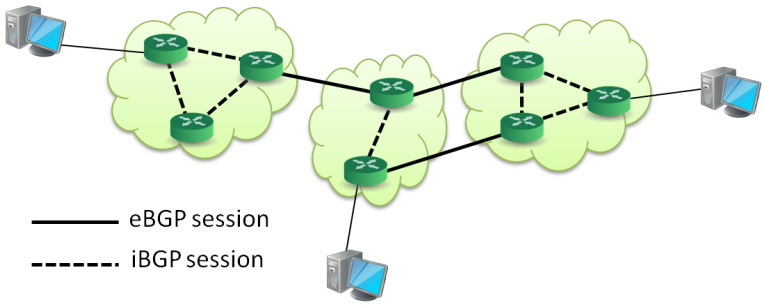


Figure 4.3: Establishment of iBGP and eBGP TCP sessions in BGP

BGP is a critical protocol that glues the whole thing in the Internet. Its mechanism is extremely complex, so we only discuss some of its basics. For further details the reader is referred to [76, 77].

The Internet is composed of many different types of ASes, such as ISPs, universities and companies, and naturally, they have different business relations among them. The BGP routing policies are determined based on these relations where there are two common forms of AS to AS interconnection:

- *Transit* is the provider-costumer relation where financial agreement is involved. The provider charges its costumers for Internet access, in return for forwarding the costumers' packets to their destinations.

- *Peering* is the relation where two ASes have mutual gains. Typically, ISPs make peering agreements on providing access to a subset of each others' routing tables. Like transit, peering is a business agreement but it usually does not involve financial settlement.

BGP determines the routes according to some set of *routing attributes*. We can summarize some of the important attributes as follows:

- *Local Preference* (`LOCAL PREF`): This attribute keeps the transit and peering information between ASes. `LOCAL PREF` is the first criteria used for selecting BGP routes.

- *AS path* (`ASPATH`): This attribute is the second important criteria for BGP route selection. (`ASPATH`) is a vector that lists all the ASes that the route advertisement has traversed, and BGP selects the route with the shortest `ASPATH` length among the routes having same `LOCAL PREF`. Note that, the route with the shortest `ASPATH` length may not be the global shortest path.

- *Multiple exit discriminator* (`MED`): This attribute is used for comparing two or more routes from the same neighbouring AS. The neighbouring AS can set the

`MED` value in order to determine which route it prefers to receive packets. BGP prefer the route with the lowest `MED` among the routes having the same `LOCAL PREF`s and `ASPATH` lengths.

Even though the *Next Hop* (`NEXT HOP`) attribute does not contribute to the route decision process, it is an integral part of packet forwarding. (`NEXT HOP`) keeps the IP address of the next hop router (border router of the next hop AS) along the path to the destination.

In the context of QoS, the default BGP4 does not have any routing attribute that reflects QoS-related parameters. Although there are some QoS extensions of BGP proposed [78,79], due to the hop-by-hop nature of the BGP, it is still hard predict the end-to-end behaviour of the QoS-related parameters. We further criticise this issue and present an OpenFlow-based solution in the next section of 4.2.

4.2 The Proposed Distributed QoS Architecture

In order to ensure optimal end-to-end QoS, collecting up-to-date global network state information, such as delay, bandwidth, and packet loss rate for each link, is essential. Yet, over a large-scale network, this is a difficult task because of dimensionality. The problem becomes even more difficult because of the distributed (hop-by-hop) architecture of the current Internet. The current Internet's state-of-the-art inter-domain routing protocols such as BGP4 and its QoS extensions [78, 79] are hop-by-hop, and therefore not suitable for optimizing end-to-end QoS. OpenFlow eases this latter point by employing a centralized controller (see Section 2.2 and Fig.2.1 in Chapter 2). Instead of sharing the state information with all other routers, OpenFlow forwarders directly send their local state information to the controller using the OpenFlow protocol. Controller processes each forwarder's state information and recomputes the best feasible routes using up-to-date global network state information.

However, the single controller solution in the current OpenFlow specification is not scalable to large scale multi-domain networks. Therefore, there is need for a distributed control plane with multiple controllers so that each controller is responsible

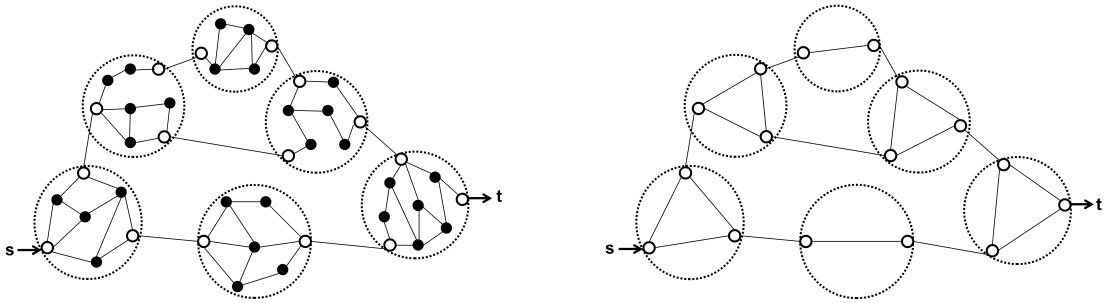


Figure 4.4: A sample multi-domain OpenFlow network: (a) complete network view, (b) aggregated version of the network

for a part (domain) of the network. In addition, there is also need to implement a controller-to-controller interface that allows a logically centralized control plane managing the overall OpenFlow network.

We propose a new simplified (aggregated) architecture for QoS routing over multi-domain OpenFlow networks. Fig.4.4(a) illustrates a sample OpenFlow network with multiple domains. The filled and unfilled dots stand for *forwarders (nodes)* and *border forwarders (border nodes)* respectively. There are two types of *links* which are inter-domain, and intra-domain links. In order to reduce problem size, we propose to aggregate the original network by replacing the intra-domain links by a set of completely meshed *virtual links* between border forwarders that are also the end points of inter-domain links as shown in Fig.4.4(b). The controller-controller interface allows controllers to share necessary (possibly aggregated) routing information among them and to help the inter-domain routing decision. Thus, although the controllers are physically distributed, they form a logically centralized control plane. To support end-to-end QoS, the controllers also shares necessary QoS parameters through controller-controller interface.

The proposed controller–controller interface is based on the following premises:

- Each network domain's size and each controller address (IP) are determined by the network administrator, so they are known beforehand.

- Each domain is managed by a single controller which is responsible for intra-domain routing and advertising its domain's state information to other controllers.

- Inter-domain routing is calculated over an aggregated version of the real network by a logically centralized control plane.

- Before finding the inter-domain route, necessary cost parameters of each virtual link summarizing the network state information has to be calculated, as discussed in Section 4.4.

- After an inter-domain route is found, each controller optimizes its intra-domain routing by replacing the virtual links with actual links.

- Both intra and inter domain QoS routes are found by solving the optimization problems stated in Section 4.4.

The controller–controller interface has following features:

- It opens a semi-permanent TCP connection between controllers to share inter-domain routing information (e.g. link up/down status, QoS parameters).

- In the case of drastic events such as network failure or congestion, the interface informs other controllers actively.

- It periodically collects network topology/state information, distributes and keep them in sync.

The key step that allows scalability is the proposed aggregation of the intra-domain network information. Obviously, network aggregation introduces some imprecision on the global network state information, but this is tolerable and necessary to obtain a scalable routing solution. We implicitly evaluate the effect of topology aggregation in Section 4.5.

4.3 Distributed Control Plane Designs

In the following, we present two design options for control plane where the controllers communicate with each other through the controller–controller interface.

4.3.1 Fully Distributed Control Plane

Fig.4.5 illustrates the completely distributed control plane where each controller

- is responsible for both intra-domain and inter-domain routing,
- has a designated domain whose complete network topology view is only accessible to that specific controller,
- uses controller–controller interface to advertise the aggregated routing information of the designated domain to the other controllers.
- uses controller–controller interface to get aggregated routing information of all other domains, and based on this knowledge the inter-domain route is determined.

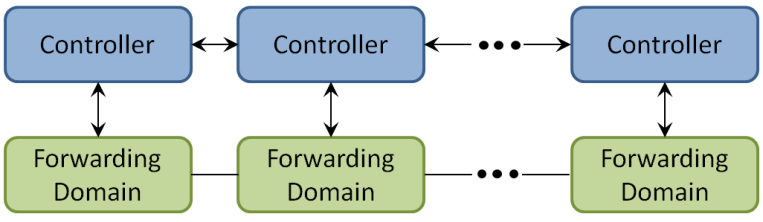


Figure 4.5: Fully distributed control plane design

4.3.2 Hierarchically Distributed Control Plane

Fig.4.6 illustrates the hierarchically distributed control plane where each controller

- is only responsible for the intra-domain routing,

- has a designated domain whose complete network topology view is only accessible to that specific controller,
- uses controller–controller interface to advertise the aggregated routing information of the designated domain to the super controller,
- uses controller–controller interface to get inter-domain route(s) determined by the super controller,

and the super controller

- is only responsible for the inter-domain routing,
- uses controller–controller interface to get aggregated routing information of all controllers, and based on this knowledge the inter-domain route is determined,
- uses controller–controller interface to push inter-domain routing decisions to all controllers.

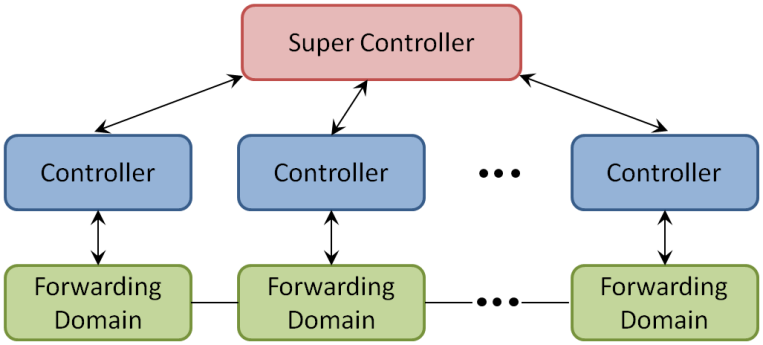


Figure 4.6: Hierarchically distributed control plane design

4.4 Distributed Optimization of QoS Routing

In this section, we pose the general QoS routing problem as a Constrained Shortest Path (CSP) problem and extend it for the proposed QoS routing architecture discussed in Section 3.2 of Chapter 3. For the CSP problem, it is crucial to select a

cost metric and constraints where they both characterize the network conditions and support QoS requirements. Since our focus is video streaming, we choose our QoS indicators as packet loss and delay variation (jitter).

In our formulation, the global network, aggregated network and the global network without inter-domain links (i.e., union of domains) are represented as directed simple graphs $G_g(N_g, A_g)$, $G_a(N_a, A_a)$, $G_d(N_d, A_d)$, respectively. N_g, N_a, N_d are the set of nodes and A_g, A_a, A_d are the set of arcs (links) in each graph. The set of virtual links is defined as $A_v \subset A_a$. Note that, $N_g = N_d \supset N_a$. We define the arc (i, j) as an ordered pair, which is outgoing from node i and incoming to node j and $R(s, t)$ (subset of set of arcs) denotes the set of routes from source node s to destination node t. For any route $r \in R(s, t)$ we define cost f_C and delay variation f_D measures as,

$$f_C(r) = \sum_{(i,j)\in r} c_{ij}, \;\; f_D(r) = \sum_{(i,j)\in r} d_{ij} \tag{4.1}$$

where c_{ij} and d_{ij} are cost and delay variation coefficients for the arc (i, j), respectively. With a slight change of notation in Section 3.2 of Chapter 3, the CSP problem is stated as,

$$r^* = \arg\min_r \{f_C(r) \mid r \in R(s, t), f_D(r) \leq D_{max}\} \tag{4.2}$$

that is, finding a route r which minimizes the cost function $f_C(r)$ subject to the delay variation $f_D(r)$ to be less than or equal to a specified value D_{max}. In our case, we choose the cost metric as the weighted sum of packet loss measure and delay variation as follows,

$$c_{ij} = (1 - \beta)d_{ij} + \beta p_{ij} \text{ for } 0 \leq \beta \leq 1, \forall (i, j) \in A_g \tag{4.3}$$

where p_{ij} denotes the packet loss measure for the traffic on link (i, j), β is the scale factor. The parameters p_{ij} and d_{ij} are nothing but the network state information that we discussed in Section 2. So, it is crucial that forwarders return up-to-date estimates in order to find the precise QoS route. OpenFlow enables us to monitor the traffic statistics on a per-flow basis and the controller can collect these statistics whenever

it requests [31].

In our proposed QoS routing framework, solution to intra-domain routing is straightforward. Since, each controller has full access to all physical links and their state information, it directly solves the CSP problem using LARAC algorithm (see Section 3.2 of Chapter 3) for given source and destination, then the QoS flows, SVC base layer packets in our case, are forwarded accordingly. On the other hand, inter-domain routing is not that trivial, because state information is not readily available for the virtual links in the aggregated network. So, the QoS indicating network state parameters (i.e. c_{ij} and d_{ij}) of each virtual link has to be set cleverly so that it summarizes the network state inside of each domain, which is not directly seen by the control plane. Then, inter-domain routing with QoS becomes feasible.

For the distributed problem formulation, we modify the CSP problem and define the CSP problem instance as follows,

$$P(G,(i,j)) = \arg\min_{r}\{f_C(r) \mid r \in R(i,j) \subseteq A, f_D(r) \leq D_{max}\} \tag{4.4}$$

where G and (i,j) are the arguments of the problem instance. G represents the network and (i,j) is the ordered pair where i and j stand for source and destination nodes. A is the set of all arcs in G and $R(i,j)$ is the set of all paths from node i to j. For example, $P(G_g,(s,t))$ is equal to the problem stated in (4.2). We propose two methods to select required parameters for virtual links in the aggregated network:

- *Method-1*: For every virtual link $(i,j) \in A_v$, the controller finds the best feasible path r^*_{ij} between border node pair (i,j) within the domain by solving the problem instance $P(G_d,(i,j))$. Then, the total cost and the delay variation of r^*_{ij} are assigned to the corresponding parameters of the virtual link between border node pair (i,j) that is $c_{ij} = f_C(r^*_{ij})$ and $d_{ij} = f_D(r^*_{ij})$.

- *Method-2*: For every virtual link $(i,j) \in A_v$,the controller finds k-disjoint best feasible paths $r^*_1, r^*_2, \ldots, r^*_k$ between border node pair (i,j) within the domain by solving CSP problem k times. Then, the average costs and delay variation

of paths $r_1^*, r_2^*, \ldots, r_k^*$ are assigned to the corresponding parameters of virtual link.

After setting the cost and delay variation parameters of virtual links using one of the methods above, it is now possible to calculate QoS routes. We formulate the QoS routing problem in two steps given in (4.5) and (4.6) in terms of the CSP problem instances stated in (4.4),

$$\textit{First Step:} \qquad r_a^* = P(G_a, (s,t)) \tag{4.5}$$

$$\textit{Second Step:} \quad r^* = \bigcup_{l=1}^{L} P(G_g, r_a^*(l)) \tag{4.6}$$

where the first step formulates the inter-domain QoS routing between source (s) and destination (t) over the aggregated network. The route r_a^* denotes the best feasible inter-domain route. The second step uses the result from the first step and formulates the end-to-end QoS routing. The route r^* denotes the complete QoS route where $r_a^*(l)$ is the l^{th} arc (ordered pair) of the route, r_a^*, and L is the number of arcs in r_a^*. Note that, each problem instance above can be solved using LARAC algorithm [58].

4.5 Application of the Distributed Optimization Framework to Scalable Video Streaming

In order to simulate the proposed QoS routing optimization framework we implemented a simulator by using the network optimization library LEMON [66] which has efficient optimization algorithms (including LARAC) for combinatorial optimization problems with graphs and networks.

The network topology we used in our simulations has 6 domains connected as shown in Fig.4.4. Each domain has 30 nodes, which is randomly designed using GT-ITM tool [68]. Hence, the overall network size is 180 nodes. The border nodes are also selected randomly. We set all intra-domain link capacities as 150 Mbps and inter-domain link capacities as 1 Gbps. The cross traffic (congestion) on each link is modeled as an independent Poisson random process which is a good model for bursty

nature of the Internet. Also, during the simulation runtime the statistics of each link may change depending on the state of the domain where it belongs. The state of each domain is modeled as a two state Markov chain which decides whether the domain is in good or bad state. The link delays are modeled as Γ-distributed random variables with means 10 ms, 15 ms and 20 ms where we randomly assign these random variables to each link. The maximum tolerable delay variation, D_{max}, is set to 250 ms.

Throughout the simulations, we used MPEG test sequence *Train* and the animation video *Big Buck Bunny* (*BBB*) with resolutions 704×576 and 1280×720, respectively. We loop both videos to obtain 900 frames lasting about 30 sec. We encode them using SVC reference software JSVM 9.19 to obtain a base and an enhancement layer (see Table 4.1).

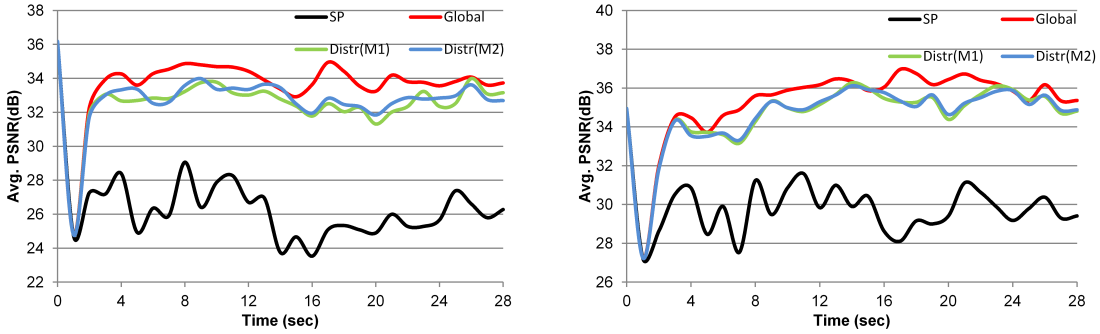
Table 4.1: Rate-Distortion values of the encoded sequences

Video	*Total Rate*	*Full PSNR*	*Base Rate*	*Base PSNR*
Train	1.3Mbps	36.89dB	0.7Mbps	33.60dB
BBB	1.2Mbps	37.67dB	0.4Mbps	33.92dB

The simulator generates QoS routes only for the SVC base layer packets while enhancement layer packets remain on their traditional shortest path. Dynamic routing is also enabled and rerouting occurs when at least one domain goes into a bad state from which SVC base layer packets are passing through. The simulator calculates the QoS routes by following exactly the same procedure that we discussed above, that is, it first updates the virtual link parameters in the aggregated network by using *Method-1* and *Method-2*, then solves the CSP instance stated in (4.5) to determine inter-domain route and finally, finds the global route from source to destination by solving CSP instances for each domain and combining the results as in (4.6).

The simulator provides us a trace driven simulation environment so that we can track which specific video packets are lost. By matching those lost packets with the Network Access Layer (NAL) units of the SVC video stream, we detect and erase the NAL units that are lost. Then, the manipulated stream is decoded and the PSNR values are measured. For each QoS routing scenario, we repeat our simula-

tions 50 times and the average PSNR values are calculated. The simulation results are shown in Fig.4.7 and we observe that the proposed distributed approaches using aggregation *Method-1*(Distr(M1)) and *Method-2* (Distr(M2)) closely approach to the globally optimum QoS routing(Global) and significantly outperforms traditional shortest path(SP). In comparison of network aggregation methods, *Method-2* performs slightly better than *Method-1* on the average. This is because, *Method-2* provides intra-domain summarization based on multiple candidates of QoS routes while *Method-1* is based on single but best QoS route which may not exist after its calculation.



(a) (b)

Figure 4.7: Simulation results: (a) Train, (b) Big Buck Bunny

The proposed network aggregation method significantly reduces the problem size down to the order of number of border nodes. Comparing the link summarization methods, we observed that *Method-2* is slightly better (less than 0.5dB) and provides more stable intra-domain summarization than *Method-1*. We show that the proposed distributed optimization of QoS routing closely approaches the non-scalable globally optimum solution and the discrepancy between them in terms of end-user video quality of experience is less than 1dB on the average.

Chapter 5

OPENFLOW TEST NETWORK AND CONTROLLER IMPLEMENTATION

5.1 Test Network

We deployed the OpenFlow test network composed of three OpenFlow enabled Pronto 3290 switches, one controller and 3 host computers. As shown in Fig.5.1, the switches are connected in a triangular shape to ensure path diversity. The video streaming server and the client are connected to different switches, while the traffic loader is connected to same switch that the server connects, generating cross-traffic into the network. Each switch initiates a secure connection to the controller using the OpenFlow protocol (see dashed lines in Fig.5.1). The controller runs our OpenFlow controller implementation which is described in detail in Section 5.2.

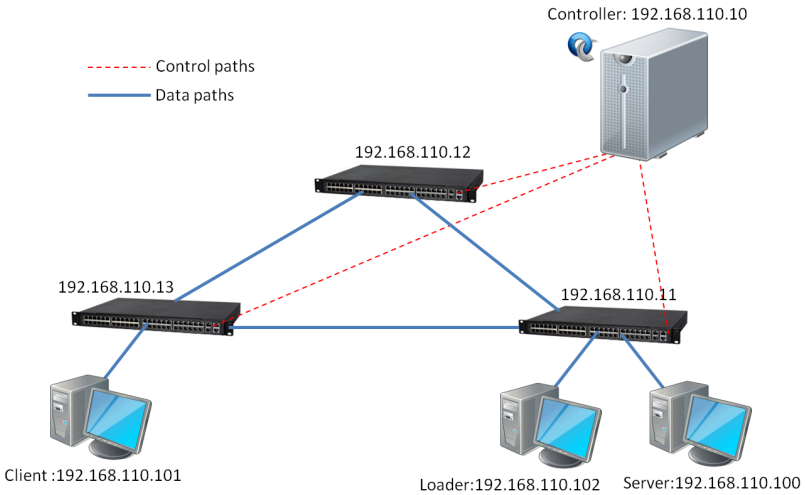


Figure 5.1: OpenFlow Testbed in our campus

5.2 Controller Implementation: OpenQoS

We implement our controller, OpenQoS, over a standard OpenFlow controller, Floodlight [80]. There are also several standard controller alternatives such as NOX [81], Beacon [82], Maestro [83] to implement a controller, but currently Floodlight is the most stable one. Floodlight is an open source controller written in Java. It provides a modular programming environment so that we can easily add new modules on top and decide which existing modules to be run.

In our implementation of OpenFlow controller (OpenQoS), we add two major modules to enable *route calculation* and *resource management* functions discussed in Section 2.2 of Chapter 2. The *topology management* function has already been implemented in Floodlight and we directly used that module. These functions are essential building blocks of our controller design which makes dynamic QoS routing possible. Yet, the our implementation is still incomplete. First, controller-to-controller, controller-to-service interfaces must be deployed, and then the functions using those interfaces (flow management, call admission, traffic policing) must be implemented. Since we concentrate on QoS routing in this thesis, we left them as open issues due to limited time.

5.2.1 Route Calculation

In Floodlight, route calculation is done when a `PACKET_IN` message arrives to the controller. It calculates the shortest path route and pushes flow definitions to the switches along that path accordingly. In addition to that, OpenQoS first checks if it is a multimedia packet or not, based on pre-defined flow definitions. Then, the route calculation module calculates two paths between the source and destination pair of the incoming packets. One path is the QoS optimized path and the other is the shortest path. A QoS route is found by solving a CSP problem as stated in (3.6) in Section 3.2 of Chapter 3:

$$r^* = \arg\min_r \{ f_C(r) \mid r \in R_{st}, f_D(r) \leq D_{max} \} \tag{5.1}$$

that is, finding a route r which minimizes the cost function $f_C(r)$ subject to the delay parameter $f_D(r)$ to be less than or equal to a specified value D_{max}. We select the cost metric as the sum of congestion and delay measures,

$$c_{ij} = g_{ij} + d_{ij} \ ,\forall(i,j) \in A \tag{5.2}$$

where g_{ij} denotes the congestion measure (similar to the packet loss measure in Section 3.2) for the link (i,j) and d_{ij} is the delay parameter for each link (i,j). We discuss how we choose g_{ij} and d_{ij} in the next section (Section 5.2.2). As we have already mentioned in Chapter 3, we propose to use LARAC algorithm to solve CSP problem stated in (5.1). So, in OpenQoS, we implement the LARAC for finding QoS optimized routes. Currently, we employ QoS routing on multimedia packets only, but OpenQoS can be easily modified to add new routing policies to new type of services.

5.2.2 Resource Management

The resource management module provides one of the key functions in the OpenQoS controller. It collects the up-to-date network state information such as link speed, available bandwidth and packet drop counts from the forwarders. The controller requests various statistics from forwarders by sending `FEATURE_REQUEST` messages, and in return forwarders send `FEATURE_REPLY` messages containing requested statistics. These messaging mechanisms are described in detail in OpenFlow specification v1.0 [84].

In order to support dynamic QoS, it is essential to keep the network state information up-to-date. The performance of the route calculation depends on the accuracy of the collected data. So, OpenQoS controller periodically collects available bandwidth for each link. The period is set to 1s since in the literature it has been shown that the Internet traffic behaves like independent Poisson distribution in sub-second time scales [69]. After receiving the available bandwidth measures from the forwarders, the resource management module

- detects whether there is a congestion event in any of the links.

- determines link cost parameters to be used in the optimization problem stated in (5.1).

Each link can be in two states: congested or non-congested. In practice, a link is assumed to be congested if the utilization of that link exceeds 75% - 85% . In our setup, we consider that a link is congested if that link is 70% bandwidth utilized. The link costs are determined by using the exact same formula in (5.2), where the congestion measure is found as,

$$g_{ij} = \begin{cases} \frac{T_{ij} - 0.7 \times B_{ij}}{T_{ij}}, & 0.7 \times B_{ij} < T_{ij} \\ 0, & 0.7 \times B_{ij} \geq T_{ij} \end{cases} \tag{5.3}$$

where T_{ij} is the total measured traffic amount in bps and B_{ij} is the maximum achievable bandwidth in bps on link (i, j). Note that, in (5.3), the non-congested links have 0 congestion measure value. The delay parameter d_{ij} in (5.2) is set to 1 which simply corresponds to hop-count. This is because the current OpenFlow switch implementations do not have any support on collecting delay related statistics (total delay, jitter).

In order to add an event based dynamicity to the QoS routing, the route manager signals forwarders when QoS routes need to be rerouted. This signalling can be achieved by deleting a specific flow entry. After a QoS flow entry is deleted, the forwarders cannot match newly coming packets, therefore they ask the controller to define new flow entries which causes multimedia packets to be rerouted. The flow deletion is triggered in two cases:

1. If a link previously non-congested is now congested, we delete the flow entries matching multimedia (QoS) packets in the flow tables of the forwarders.

2. If a link previously congested is non-congested in the last 3 periods, we again delete the flows accordingly. We require 3 periods of non-congested state to

ensure there are no fluctuations in the traffic rate on the links.

5.3 Test Results

To demonstrate the performance of our OpenQoS implementation, we built a video streaming environment over a real OpenFlow test network shown in Fig.5.1. Throughout the tests, we used a well-known test sequence "*in to tree*" having 500 frames with the resolution of 1280×720. We looped the raw video sequence reversely once to have 1000 frames lasting about 40s. We then encoded the looped sequence in H.264 format using the *ffmpeg* encoder (v.0.7.3) [85] at three different average bit-rates to have

- *Stream 1* at 1800 kbps (32.55dB),
- *Stream 2* at 900 kbps (30.57dB),
- *Stream 3* at 450 kbps (28.75dB).

These three H.264 video streams are used in two test scenarios presented in the next subsections.

5.3.1 Streaming over UDP

We created a scenario where two copies of the *Stream 1* are sent from the server residing at 192.168.110.100 to the client with the IP address 192.168.110.101 (see Fig.5.1). The server uses VLC media player [86] to stream videos using RTP/UDP. One copy of the video is sent to the destination port 5004 while the other copy is sent to port 5005. To show the performance difference in terms of QoS, we matched the multimedia flows (QoS flows) to the transport port number, 5004. Thus, the video packets destined to port 5004 are identified as being part of a multimedia flow by the OpenQoS controller and routed accordingly, while the other video (destined to port 5005) is considered as a data flow which has no QoS support (i.e. best-effort). In each test, 10 second long cross-traffic is sent from the loader (192.168.110.102) to the client once at a random time. The client runs two VLC player sessions, listening

RTP/UDP packets at ports 5004 and 5005, to save the received videos. We expect to see distortions in the video received on port 5005 during the cross-traffic while the other video received on port 5004 will be rerouted and affected little or not at all in terms of video quality.

We decode the received videos using *ffmpeg* and measure their qualities using the peak signal to noise ratio (PSNR) values with respect to the original raw video. The results are given in Figs.5.2 and 5.3 which are in terms of received video quality (PSNR) versus time. The vertical dashed lines mark the start and end times of the cross-traffic.

The best case result is shown in Fig.5.2 where the video with QoS support (w/ QoS) is not affected from the cross traffic and approaches full video quality, while the video without QoS support (w/o QoS) has significant amount of quality loss. However, in Fig.5.3, the video with QoS also suffers, it is recovered in less than 1s. After repeating the scenario 20 times, we observed that the average loss recovery period is 0.76s. Most of the time the user watching the video is not disturbed from the quality loss in such a small interval even if we use UDP which does not guarantee reliable delivery at all.

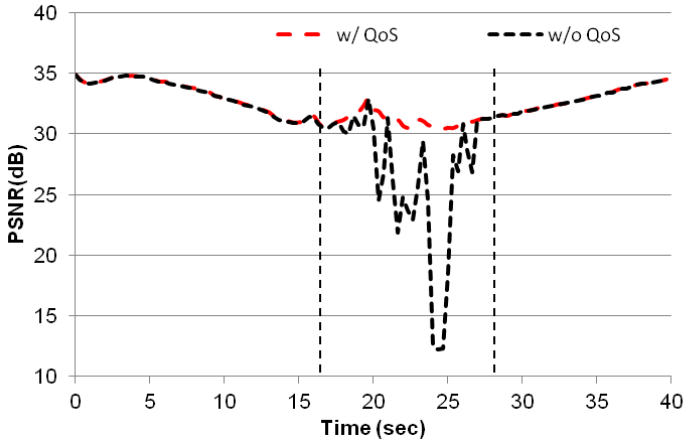


Figure 5.2: Best case result of UDP streaming

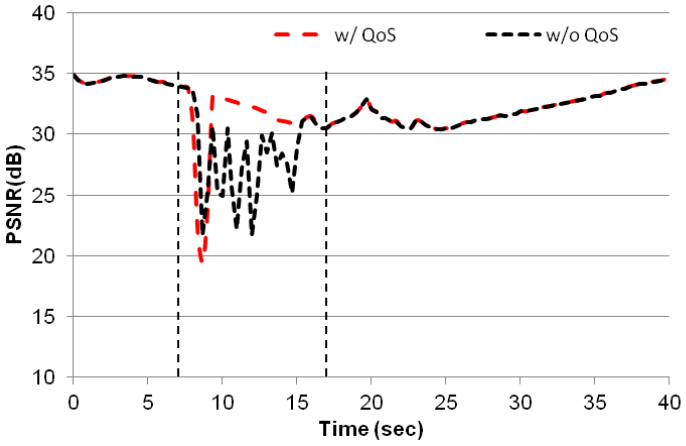


Figure 5.3: One case result of UDP streaming

5.3.2 HTTP-based Adaptive Streaming

We built a test scenario similar to the one discussed in Section 5.3.1 where TCP is employed as a transport protocol instead of UDP. The server sends

- *Stream 1* with QoS support,
- a video without QoS support chosen adaptively among *Stream 1, 2* and *3*.

For adaptive video streaming we used *Adobe Flash Media Server 4.5* [87]. At the server side, each video stream (*Stream 1, 2* and *3*) is fragmented into 4 second long sub-streams and an associated *m3u8* playlist is created. At the client side, the VLC player first downloads the *m3u8* playlist and then selects an appropriate sub-stream rate-adaptively. While the loader (see Fig.5.1) applies 10 second cross-traffic, the the video with QoS (i.e. *Stream 1*) is rerouted, and the video without QoS is rate adapted. Fig.5.4 illustrates the quality difference between the rate adaptation (w/o QoS) and the QoS rerouting (w/ QoS) of a sample test. We repeat the same test scenario over 30 times, and we do not observe any quality loss in the video with QoS.

Figure 5.4: Adaptive HTTP streaming result

5.3.3 Interpretation of Test Results

OpenQoS is a novel approach to stream video over OpenFlow networks with QoS. It is different from the current QoS mechanisms since we propose dynamic QoS routing to fulfill end-to-end QoS support which is possible with OpenFlow's centralized control capabilities over the network. Unlike other QoS architectures, OpenQoS minimizes the adverse effects (such as packet loss and latency) on other types of flows. Inspection of our experimental results yields the following observations:

- OpenQoS working along with TCP outperforms the state-of-the-art, HTTP-based multi-bitrate adaptive streaming, under network congestion.

- OpenQoS can guarantee seamless video delivery with little or no disturbance experienced by the end users even if an unreliable transport protocol, such as UDP, is used.

- If a reliable transport protocol, such as TCP, is used, OpenQoS can guarantee full video quality.

Chapter 6

FUTURE DIRECTIONS AND APPLICATION AREAS

In this chapter, we present some possible application areas of the proposed OpenFlow based architecture in the following sections.

6.1 Koc-Ozyegin-Argela OpenFlow Test Network

We have initiated a multi-domain OpenFlow test network deployment for the first time in Turkey. The Fig.6.1 depicts the OpenFlow network deployed over three campuses, Koç University, Özyeğin University and Argela company. Currently, the communication between the domains are provided via VPN connections, and we can efficiently control the overall network from a single controller at Koç University.

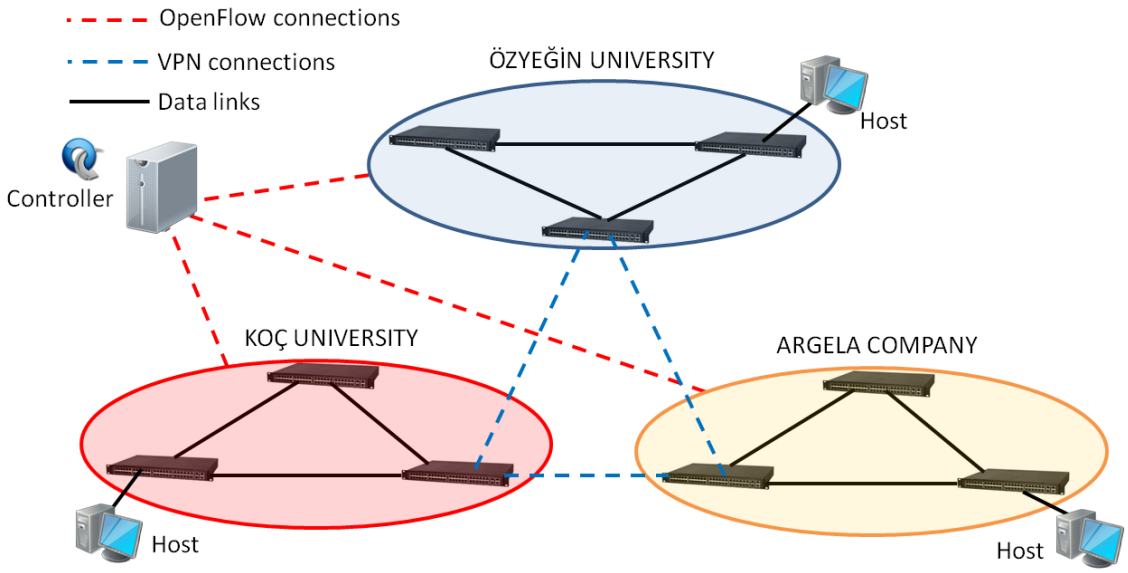


Figure 6.1: First OpenFlow test network deployed over three campuses

As future directions,

- the realizations of distributed QoS architecture, distributed control plane and controller-controller interface discussed in Chapter 4 can be implemented,

- for better forwarding performance VPN connections between domains can be converted to dedicated lines by requesting from the service provider.

6.2 Load Balancing in Content Distribution Networks (CDNs)

Load balancing is a networking methodology that distributes the workload across the network elements. The purpose of load balancing is providing a service from multiple servers by choosing an appropriate server. Therefore, it is essential for networking technologies such as content delivery networks (CDN), domain name systems (DNS) and newly emerging cloud services. It is usually implemented by a load balancing switch (i.e. load balancer) which forwards a request coming from a client to one of the servers which, in general, replies to the load balancer, as illustrated in Fig.6.2. This operation is done without the client which is unaware of the presence of load balancer and backend servers. A load balancer selects a server by using a variety of scheduling algorithms which may consider factors such as servers' reported load, servers' up/down frequencies, location of the servers (i.e. propagation delay), type of the requested content and the amount of traffic assigned to a server.

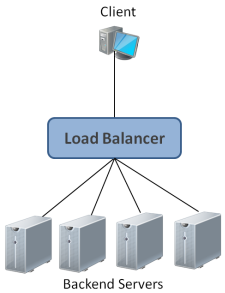
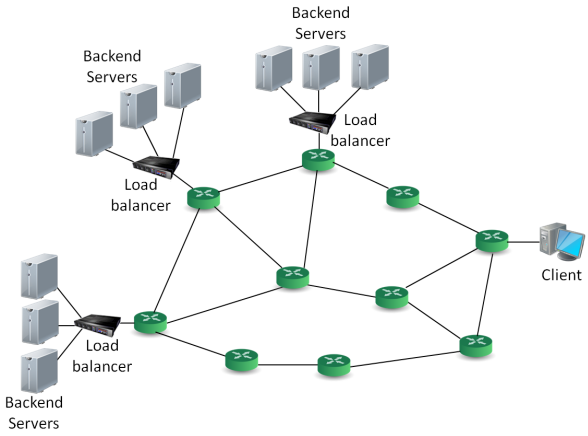
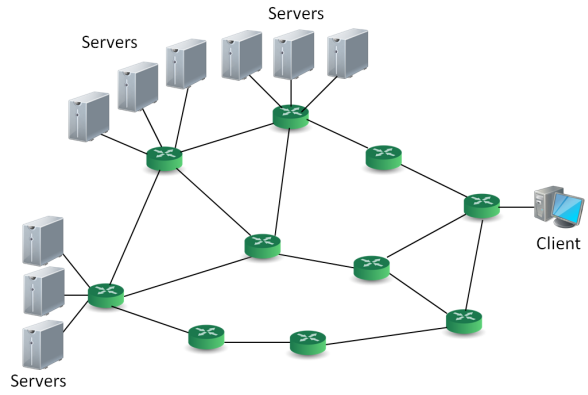


Figure 6.2: Load Balancer



(a)



(b)

Figure 6.3: Load balancing (a) over the Internet is limited to server selection, (b) over OpenFlow allows the joint selection of servers and routes

Content delivery networks are distributed system of servers that serve contents such as web objects, documents and multimedia to end-users with high availability and high performances. Especially, most of the current state-of-the-art multimedia streaming applications (e.g. live and on-demand streaming) over the Internet rely on CDNs, and load balancing is the integral part of the CDN. However, in the Internet, only server-based load balancing is possible. We can overcome this deficiency by using OpenFlow. In OpenFlow load balancing can be considered as a network primitive which does not require additional equipment that implements load balancing functions. Also, OpenFlow (Aster*x controller [9]) enables joint optimization of server

and route selection which is not possible in the Internet, as illustrated in Fig.6.3.

6.3 Multiple Description Coding

Multiple description coding (MDC) is a coding technique [21, 88, 89] that encodes a source into multiple bitstreams (descriptions) supporting multiple quality levels of decoding. The packets of each description are independently decodable, i.e. any description can be used, however, the quality improves with the number of descriptions received. The main objective of MDC is to provide error resilience to multimedia delivery. Since an arbitrary subset of descriptions can be used to decode the original stream, in the case of network congestion or packet loss which are common in best-effort networks (e.g. the Internet) MDC is more robust than single description (SD) coding. However, the loss of compression efficiency, the transmission overhead and high encoder/decoder complexity are the major drawbacks of MDC.

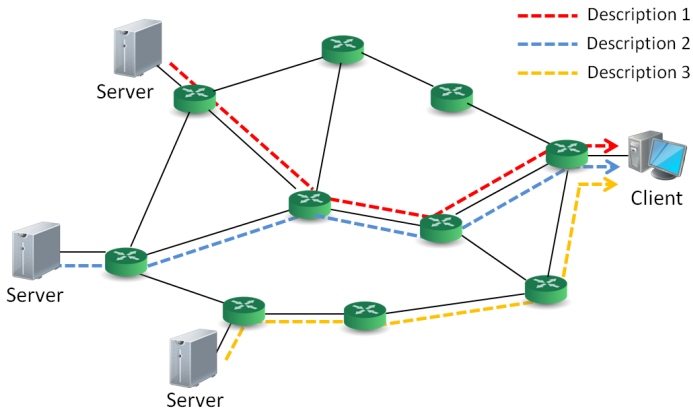
The idea of MDC is different than scalable (layered) coding. In scalable coding, a base and one or more enhancement layers are generated where the base layer is necessary to decode the multimedia stream and enhancement layers are applied to improve stream quality. Each enhancement layer cannot be decoded independently, since it depends on subordinate quality layer (base layer or previous enhancement layer).

To gain robustness to the loss of descriptions and to reduce communication overhead, MDC must embrace channel and/or path diversity. In wireless networks, both channel and path diversity can be exploited, but in the Internet we can only employ path (route) diversity, since all packet transmissions are delivered on the same channel. In the Internet, each description should be sent over different routes; because, in general, average route behaviour provides better performance than the behaviour of any individual random route. For example, the probability that all of the multiple routes are simultaneously congested is much less than the probability that a single route is congested. However, current Internet determines a single route (or single multicast tree) for source and destination pairs, so MDC cannot be applied when

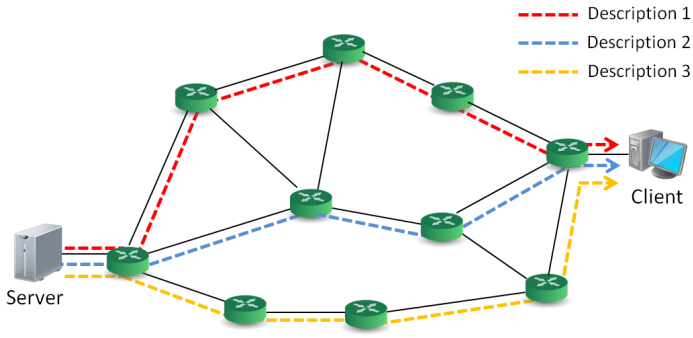
there is a single multimedia source (server). In order to take advantage of MDC in the Internet, different descriptions have to be distributed over different sources to enable multi-path diversity. Hence, current MDC-based multimedia delivery proposals are limited to peer-to-peer (P2P) and content distribution networks (CDNs) (see Fig.6.4(a)). On the other hand, OpenFlow removes this limitation with its per-flow routing capability. In OpenFlow, each MDC description can be defined as a different flow and therefore, descriptions can be placed on disjoint or partially disjoint routes even if there is a single multimedia server (see Fig.6.4(b)). The routes of each description can be found using k-disjoint shortest path or can be further optimized by using constrained based disjoint routing algorithms [49, 65, 90]. Unlike current MDC-based multimedia systems, MDC streaming over OpenFlow does not require distributing descriptions among servers placed around the network, as illustrated in Fig.6.4.

6.4 Enabling Cross Layer Design in the Internet and OpenFlow Wireless

In the literature there are cross-layer designs for QoS routing over wireless networks (e.g. ad-hoc and sensor networks) [91–93], but they cannot be implemented on wired networks. The Internet is a closed environment where researchers cannot easily experiment their ideas related to the core network such as routing. This is because, current Internet router vendors provide a hardware and associated software which is not open to its users. OpenFlow removes the boundaries of the traditional Internet; it provides completely open and programmable networking environment to the operators, enterprises, independent software vendors and users. It also allows researchers to develop their ideas similar to the cross-layer approaches as in wireless networks. Even though our focus is on wired networks in this thesis, there is an initial implementation of wireless extension of OpenFlow (OpenRoads) [94] on which our framework can be implemented with a little effort.



(a)



(b)

Figure 6.4: Streaming three MDC descriptions to a client (a) over the Internet from multiple servers, (b) over OpenFlow from a single server

Chapter 7

CONCLUSIONS

In this thesis, we have presented a novel Quality of Service (QoS) architecture and an associated optimization framework for QoS routing. We have also presented an extension of the proposed architecture that is scalable to large-scale networks. The main contributions of this thesis can be summarized as follows:

- We have proposed a QoS architecture that fulfills end-to-end QoS by dynamically optimizing QoS routes. This is the first OpenFlow-based QoS architecture in the literature.

- On top of our architecture we have built an optimization framework for QoS routing; we have designed an OpenFlow controller that enables end-to-end QoS.

- We have posed the dynamic QoS routing problem as a Constrained Shortest Path (CSP) and have proposed a novel QoS metric selection. The CSP problem has well established solutions in the literature and we have proposed to use a Lagrangian relaxation based algorithm, called LARAC, to solve the problem.

- Since the proposed QoS architecture does not scale to large-networks, we have presented a distributed extension of the architecture where we have designed a distributed control plane to manage large-scale multi-domain OpenFlow networks. Then, we have redefined the CSP problem for the distributed architecture.

- We have applied our approaches to scalable (layered) video streaming that allows different grades of video streaming service to service providers.

- We have deployed an OpenFlow test network which is the first OpenFlow network in our country.

- We have developed a controller software, called OpenQoS, that implements the proposed dynamic routing framework and manages the OpenFlow test network we deployed.

BIBLIOGRAPHY

[1] J. H. Saltzer, D. P. Reed, and D. Clark, "End-to-End Arguments in System Design," *ACM Transactions on Computer Systems*, vol. 2, no. 4, Nov. 1984.

[2] D. Clark, "The design philosophy of the DARPA internet protocols," in *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*. ACM, 1988, pp. 106–114.

[3] R. Braden, D. Clark, and S. Shenker, "Integrated services in the internet architecture: an overview," RFC 1633, Internet Engineering Task Force, June 1994.

[4] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," RFC 2475, Internet Engineering Task Force, Dec. 1998.

[5] E. Rosen and Y. Rekhter, "BGP/MPLS VPNs," RFC 2547, Internet Engineering Task Force, 1999.

[6] Open Networking Foundation. [Online]. Available: http://opennetworking.org

[7] Open Networking Foundation (ONF), "Software defined networking: the new norm for networks," 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/openflow/wp-sdn-newnorm.pdf

[8] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[9] OpenFlow Consortium. [Online]. Available: http://openflowswitch.org

[10] "Internet protocol," RFC 791, Internet Engineering Task Force, September 1981.

[11] Z. He, J. Cai, and C. W. Chen, "Joint source channel rate-distortion analysis for adaptive mode selection and rate control in wireless video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 6, pp. 511 –523, jun 2002.

[12] Q. Zhang, W. Zhu, and Y. Zhang, "End-to-end qos for video delivery over wireless internet," *Proceedings of the IEEE*, vol. 93, no. 1, pp. 123 –134, jan. 2005.

[13] R. Rejaie, M. Handley, and D. Estrin, "Layered quality adaptation for internet video streaming," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 12, pp. 2530 –2543, dec 2000.

[14] Z. Miao and A. Ortega, "Proxy caching for efficient video services over the internet," in *9th International Packet Video Workshop (PVW '99)*, 1999.

[15] K. Stuhlmuller, M. Link, B. Girod, and U. Horn, "Scalable internet video streaming with unequal error protection," in *Proc. of Packet Video Workshop*, 1999.

[16] H. Radha, M. van der Schaar, and Y. Chen, "The mpeg-4 fine-grained scalable video coding method for multimedia streaming over ip," *IEEE Transactions on Multimedia*, vol. 3, no. 1, pp. 53 –68, mar 2001.

[17] M. Kalman, E. Steinbach, and B. Girod, "Adaptive media playout for low-delay video streaming over error-prone channels," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 6, pp. 841 – 851, june 2004.

[18] R. Puri, K. Ramchandran, K. Lee, and V. Bharghavan, "Forward error correction (fec) codes based multiple description coding for internet video streaming and

multicast," *Signal Processing: Image Communication*, vol. 16, no. 8, pp. 745 – 762, 2001.

[19] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h.264/avc video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560 –576, july 2003.

[20] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the h.264/avc standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1103 –1120, sept. 2007.

[21] V. Goyal, "Multiple description coding: compression meets the network," *IEEE Signal Processing Magazine*, vol. 18, no. 5, pp. 74 –93, sep 2001.

[22] A. Ortega and K. Ramchandran, "Rate-distortion methods for image and video compression," *IEEE Signal Processing Magazine*, vol. 15, no. 6, pp. 23 –50, nov 1998.

[23] T. Wiegand, H. Schwarz, A. Joch, F. Kossentini, and G. Sullivan, "Rate-constrained coder control and comparison of video coding standards," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 688 – 703, july 2003.

[24] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the h.264/avc video compression standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 620 – 636, july 2003.

[25] D. Wu, Y. Hou, W. Zhu, Y.-Q. Zhang, and J. Peha, "Streaming video over the internet: approaches and directions," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 3, pp. 282–300, mar 2001.

[26] H. Sun, A. Vetro, and J. Xin, "An overview of scalable video streaming," *Wireless Communications and Mobile Computing*, vol. 7, pp. 159–172, Feb. 2007.

[27] B. Girod, M. Kalman, Y. J. Liang, and R. Zhang, "Advances in channel-adaptive video streaming," *Wireless Communications and Mobile Computing*, vol. 2, no. 6, pp. 573–584, 2002.

[28] B. Li and J. Liu, "Multirate video multicast over the internet: an overview," *IEEE Network*, vol. 17, no. 1, pp. 24 – 29, jan/feb 2003.

[29] H. E. Egilmez, B. Gorkemli, A. M. Tekalp, and S. Civanlar, "Scalable video streaming over OpenFlow networks: an optimization framework for QoS routing," in *Proc. IEEE International Conference on Image Processing (ICIP)*, Sept. 2011, pp. 2241–2244.

[30] H. E. Egilmez, S. Civanlar, and A. M. Tekalp, "An optimization framework for qos-enabled adaptive video streaming over OpenFlow networks," unpublished.

[31] OpenFlow Switch Specification v1.1.0. [Online]. Available: http://www.openflow.org/wp/documents/

[32] H. E. Egilmez, S. Civanlar, and A. M. Tekalp, "A distributed QoS routing architecture for scalable video streaming over multi-domain OpenFlow networks," to appear, 2012.

[33] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp, "OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end quality of service over software-defined networks," unpublished.

[34] Cisco, "Cisco visual networking index: Forecast and methodology, 20102015," 2011. [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/

[35] S. Shenker, C. Partridge, and R. Guerin, "Specification of Guaranteed Quality of Service," RFC 2212, Internet Engineering Task Force, September 1997.

[36] J. Wroclawski, "Specification of the Controlled-Load Network Element Service," RFC 2211, Internet Engineering Task Force, September 1997.

[37] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification," RFC 2205, Internet Engineering Task Force, September 1997.

[38] M. de Prycker, *Asynchronous transfer mode: solution for broadband ISDN*. Upper Saddle River, NJ, USA: Ellis Horwood, 1991.

[39] V. Jacobson, K. Nichols, and K. Poduri, "An Expedited Forwarding PHB," RFC 2598, Internet Engineering Task Force, June 1999, obsoleted by RFC 3246.

[40] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, "Assured Forwarding PHB Group," RFC 2597, Internet Engineering Task Force, June 1999.

[41] K. Nichols, S. Blake, F. Baker, and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," RFC 2474, Internet Engineering Task Force, Dec. 1998.

[42] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture," RFC 3031, Internet Engineering Task Force, January 2001.

[43] R. Ballart and Y.-C. Ching, "Sonet: now it's the standard optical network," *Communications Magazine, IEEE*, vol. 27, no. 3, pp. 8 –15, march 1989.

[44] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus, "Requirements for Traffic Engineering Over MPLS," RFC 2702 (Informational), Internet Engineering Task Force, September 1999.

[45] C. Srinivasan, A. Viswanathan, and T. Nadeau, "Multiprotocol Label Switching (MPLS) Traffic Engineering (TE) Management Information Base (MIB)," RFC 3812, Internet Engineering Task Force, June 2004.

[46] F. L. Faucheur, L. Wu, B. Davie, S. Davari, P. Vaananen, R. Krishnan, P. Cheval, and J. Heinanen, "Multi-Protocol Label Switching (MPLS) Support of Differentiated Services," RFC 3270, Internet Engineering Task Force, May 2002.

[47] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow, "RSVP-TE: Extensions to RSVP for LSP Tunnels," RFC 3209, Internet Engineering Task Force, December 2001.

[48] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S.-J. Lee, and P. Yalagandula, "Automated and scalable qos control for network convergence," in *Proc. INM/WREN'10*, 2010, pp. 1–1.

[49] S. Chen and K. Nahrstedt, "An overview of quality of service routing for next-generation high-speed networks: problems and solutions," *IEEE Network*, vol. 12, no. 6, pp. 64–79, Nov/Dec 1998.

[50] H. F. Salama, D. S. Reeves, and Y. Viniotis, "Evaluation of multicast routing algorithms for real-time communication on high-speed networks," *IEEE Journal on Selected Areas in Communications*, vol. 15, pp. 332–345, 1997.

[51] B. Wang and J. C. Hou, "Multicast routing and its qos extension: Problems, algorithms, and protocols," *IEEE Network*, vol. 14, pp. 22–36, 2000.

[52] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

[53] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows: theory, algorithms, and applications*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993.

[54] F. Kuipers, P. Van Mieghem, T. Korkmaz, and M. Krunz, "An overview of constraint-based path selection algorithms for QoS routing," *IEEE Communications Magazine*, vol. 40, no. 12, pp. 50–55, Dec 2002.

[55] X. Masip-Bruin, M. Yannuzzi, J. Domingo-Pascual, A. Fonte, M. Curado, E. Monteiro, F. Kuipers, P. V. Mieghem, S. Avallone, G. Ventre, P. Aranda-Gutirrez, M. Hollick, R. Steinmetz, L. Iannone, and K. Salamatian, "Research challenges in QoS routing," *Computer Communications*, vol. 29, no. 5, pp. 563–581, 2006.

[56] Z. Wang and J. Crowcroft, "Quality-of-service routing for supporting multimedia applications," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, pp. 1228–1234, Sep. 1996.

[57] G. Xue, W. Zhang, J. Tang, and K. Thulasiraman, "Polynomial time approximation algorithms for multi-constrained QoS routing," *IEEE/ACM Transactions on Networking*, vol. 16, no. 3, pp. 656–669, June 2008.

[58] A. Juttner, B. Szviatovski, I. Mecs, and Z. Rajko, "Lagrange relaxation based method for the QoS routing problem," in *Proc. IEEE INFOCOM*, vol. 2, Apr. 2001, pp. 859–868.

[59] S. Chen, M. Song, and S. Sahni, "Two techniques for fast computation of constrained shortest paths," *IEEE/ACM Transactions on Networking*, vol. 16, no. 1, pp. 105–115, Feb. 2008.

[60] D. P. Bertsekas, *Network Optimization: Continuous and Discrete Models*. Belmont Mass., USA: Athena Scientific, 1998.

[61] H. Schulzrinne, S. L. Casner, R. Frederick, and V. Jacobson, "RTP: a transport protocol for real-time applications," RFC 3550, Internet Engineering Task Force, Jul. 2003.

[62] Y. Xiao, K. Thulasiraman, G. Xue, and A. Juttner, "The constrained shortest path problem: algorithmic approaches and an algebraic study with generalization," *AKCE J. Graphs. Combin*, vol. 2, no. 2, pp. 63–86, 2005.

[63] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[64] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.

[65] Y. Xiao, K. Thulasiraman, and G. Xue, "Gen-larac: a generalized approach to the constrained shortest path problem under multiple additive constraints," in *Proc. ISAAC*, 2005, pp. 92–105.

[66] LEMON, Library for Efficient Modeling and Optimization in Networks. [Online]. Available: http://lemon.cs.elte.hu

[67] K. Calvert, M. Doar, and E. Zegura, "Modeling internet topology," *IEEE Communications Magazine*, vol. 35, no. 6, pp. 160–163, June 1997.

[68] GT-ITM, Georgia Tech Internetwork Topology Models. [Online]. Available: http://www.cc.gatech.edu/projects/gtitm/

[69] V. Frost and B. Melamed, "Traffic modeling for telecommunications networks," *IEEE Communications Magazine*, vol. 32, no. 3, pp. 70–81, Mar. 1994.

[70] R. Sherwood, G. Gibb, K. K. Yap, M. Casado, N. Mckeown, and G. Parulkar, "Can the production network be the testbed," in *OSDI'10*, 2010.

[71] T. Koponen and et.al., "Onix: a distributed control platform for large-scale production networks," in *OSDI'10*, 2010, pp. 1–6.

[72] A. Tootoonchian and Y. Ganjali, "Hyperflow: a distributed control plane for OpenFlow," ser. INM/WREN'10, 2010, pp. 3–3.

[73] C. L. Hedrick, "Routing information protocol," RFC 1058, Internet Engineering Task Force, United States, 1988.

[74] J. Moy, "OSPF Version 2," RFC 1583, Internet Engineering Task Force, March 1994.

[75] D. Oran, "OSI IS-IS Intra-domain Routing Protocol," RFC 1142, Internet Engineering Task Force, February 1990.

[76] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," RFC 4271, Internet Engineering Task Force, January 2006.

[77] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 5th ed. USA: Addison-Wesley Publishing Company, 2009.

[78] L. Xiao, K.-S. Lui, J. Wang, and K. Nahrsted, "Qos extension to bgp," in *Proceedings of the 10th IEEE International Conference on Network Protocols*, ser. ICNP'02. IEEE Computer Society, 2002, pp. 100–109.

[79] T. Zhang, Y. Cui, Y. Zhao, L. Fu, and T. Korkmaz, "Scalable bgp qos extension with multiple metrics," in *Proceedings of the International conference on Networking and Services*, ser. ICNS'06. IEEE Computer Society, 2006.

[80] Floodlight. [Online]. Available: http://floodlight.openflowhub.org

[81] Nox. [Online]. Available: http://noxrepo.org

[82] Beacon. [Online]. Available: http://openflow.stanford.edu/display/Beacon/

[83] Z. Cai, A. L. Cox, and T. S. Eugene Ng, "Maestro: balancing fairness, latency and throughput in the OpenFlow control plane," Rice University Technical Report TR11-07, 2011.

[84] Openflow switch specification v1.0. [Online]. Available: http://openflow.org/wp/documents/

[85] ffmpeg. [Online]. Available: http://ffmpeg.org

[86] VLC media player. [Online]. Available: http://videolan.org/vlc

[87] Flash Media Streaming Server 4.5. [Online]. Available: http://www.adobe.com/products/flash-media-streaming.html

[88] Y. Wang, A. Reibman, and S. Lin, "Multiple description coding for video delivery," *Proceedings of the IEEE*, vol. 93, no. 1, pp. 57 –70, Jan. 2005.

[89] J. Apostolopoulos, "Reliable video communication over lossy packet networks using multiple state encoding and path diversity," in *Visual Communications and Image Processing (VCIP)*, 2001, pp. 392–409.

[90] D. Sidhu, R. Nair, and S. Abdallah, "Finding disjoint paths in networks," *SIGCOMM Comput. Commun. Rev.*, vol. 21, no. 4, pp. 43–51, 1991.

[91] S. Misra, M. Reisslein, and G. Xue, "A survey of multimedia streaming in wireless sensor networks," *IEEE Communications Surveys Tutorials*, vol. 10, no. 4, pp. 18 –39, quarter 2008.

[92] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," *Wirel. Netw.*, vol. 11, no. 4, pp. 419–434, Jul. 2005.

[93] Q. Zhang and Y.-Q. Zhang, "Cross-layer design for qos support in multihop wireless networks," *Proceedings of the IEEE*, vol. 96, no. 1, pp. 64 –76, jan. 2008.

[94] K.-K. Yap, M. Kobayashi, R. Sherwood, T.-Y. Huang, M. Chan, N. Handigol, and N. McKeown, "Openroads: empowering research in mobile networks," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 125–126, Jan. 2010.