SDN-based Application-Network Collaboration for Low-Latency Networked Music Performance Systems

Emmanouil Lakiotakis

Thesis submitted in partial fulfillment of the requirements for the

Master of Science degree in Computer Science

University of Crete
School of Sciences and Engineering
Computer Science Department
University Campus, Voutes, Heraklion, GR-70013, Greece

Thesis Advisor: Assistant Prof. *Xenofontas Dimitropoulos*

This work was partially supported by **Institute of Computer Science, Foundation of Research and Technology Hellas**

UNIVERSITY OF CRETE
COMPUTER SCIENCE DEPARTMENT

SDN-based Application-Network Collaboration for Low-Latency Networked Music Performance Systems

Thesis submitted by
Emmanouil Lakiotakis
in partial fulfillment of the requirements for the
Master of Science degree in Computer Science

THESIS APPROVAL

Author: _____
Emmanouil Lakiotakis

Committee approvals: _____
Xenofontas Dimitropoulos
Assistant Professor, Thesis Supervisor

_____
Panagiotis Tsakalides
Professor, Committee Member

_____
Athanassios Mouchtaris
Associate Professor, Committee Member

Departmental approval: _____
Antonios Argyros
Professor, Director of Graduate Studies

Heraklion, February 2017

Abstract

A class of network services that attracts the interest of the research community is ultra delay-sensitive applications. In this type of services, end-to-end delay is acceptable up to a threshold which is at the level of milliseconds. A representative example of ultra delay-sensitive services is Networked Music Performance (NMP) systems. An NMP involves musicians that are located in different places, who perform music while staying synchronized via the Internet. The maximum end-to-end delay in NMP is called Ensemble Performance Threshold (EPT) and should be less than 25 milliseconds. Due to this constraint, NMP systems require ultra-low delay solutions for audio coding, transmission via the network, relaying and decoding, each one a very challenging task on its own. There are two types of approaches in related work referring to NMP systems. From the perspective of audio, researchers experiment on low-delay encoders and transmission patterns to reduce the required bandwidth and processing delay of audio transmission, but they do not bring network performance into play, treating it as a "black box". On the other hand, network researchers try to find optimal ways for reducing network delay, which contributes to reduced end-to-end delay for NMPs.

In this master thesis, the proposed approach introduces an integration of dynamic audio and network modification to satisfy the EPT constraint. The basic idea is that the major components participating in a NMP system, application and network, communicate and interact as the live network music performance event takes place for improved performance. When network delay increases, the network tries to absorb this increase by modifying routing behavior. If network delay exceeds a maximum acceptable threshold, the network reacts by informing the application to change the audio processing pattern to overcome delay increase. This interaction enables end-to-end-delay below the EPT value when otherwise would not be possible. Our implementation incorporates Software Defined Networking, because it increases the flexibility and dynamic adaptation to rapid changes. Also, the implementation exploits the ability of SDN's central entity called SDN controller that has a global view of the whole network and takes optimal routing decisions. We designed and developed a full prototype of the proposed NMP system, which was successfully evaluated in an emulation environment. The results show that delay improvement is feasible up to 59%.

Περίληψη

Μια κατηγορία δικτυακών υπηρεσιών που ελκύει το ενδιαφέρον της ερευνητικής κοινότητας είναι οι εφαρμογές με υπερβολική ευαισθησία στην καθυστέρηση. Σε αυτή την κατηγορία, η καθυστέρηση από άκρο σε άκρο είναι αποδεκτή μέχρι ένα κατώφλι της τάξης των χιλιοστών του δευτερολέπτου. Ένα αντιπροσωπευτικό παράδειγμα εφαρμογών υπερβολικά ευαίσθητων στην καθυστέρηση είναι τα συστήματα Δικτυακών Μουσικών Παραστάσεων. Μια Δικτυακή Μουσική Παράσταση περιγράφει την διαδικασία όπου μουσικοί από διαφορετικά σημεία του πλανήτη παρουσιάζουν συγχρονισμένα μέσω του Διαδικτύου. Η μέγιστη επιτρεπτή καθυστέρηση από άκρο σε άκρο σε αυτά τα συστήματα ονομάζεται Συνολικό Κατώφλι Απόδοσης και εκτιμάται λιγότερη από 25 χιλιοστά του δευτερολέπτου. Λόγω του παραπάνω περιορισμού, τα συστήματα Δικτυακών Μουσικών Παραστάσεων απαιτούν λύσεις χαμηλής καθυστέρησης σχετικά με την κωδικοποίηση, τη μετάδοση μέσω δικτύου και την αποκωδικοποίηση του ήχου, καθένα από τα οποία είναι μια ενδιαφέρουσα πρόκληση από μόνο του. Υπάρχουν δύο τύποι προσέγγισης στην σχετική βιβλιογραφία αναφορικά με τα συστήματα Δικτυακών Μουσικών Παραστάσεων. Από την ηχητική οπτική, οι ερευνητές πειραματίζονται σε κωδικοποιητές χαμηλής καθυστέρησης και πρότυπα μετάδοσης του ήχου με σκοπό να μειώσουν το απαιτούμενο εύρος ζώνης και την καθυστέρηση επεξεργασίας του ήχου αγνοώντας την συμπεριφορά του δικτύου, θεωρώντας την ως αδιαπραγμάτευτη. Από τη άλλη πλευρά, ερευνητές του χώρου των δικτύων επιχειρούν να επινοήσουν βέλτιστους τρόπους δρομολόγησης για την μείωση της καθυστέρησης δικτύου, συντελώντας με τον τρόπο στην εν γένει μείωση της καθυστέρησης από άκρο σε άκρο για τα συστήματα Δικτυακών Μουσικών Παραστάσεων.

Στην παρούσα μεταπτυχιακή εργασία, η προτεινόμενη προσέγγιση εισάγει την ενσωμάτωση των δύο παραπάνω οπτικών δυναμικής παραμετροποίησης τόσο στον τομέα του ήχου αλλά και στον τομέα του δικτύου ώστε να ικανοποιηθεί ο περιορισμός της ανώτατης επιτρεπτής χρονικής καθυστέρησης. Η βασική ιδέα που εισάγεται είναι η επικοινωνία και αλληλεπίδραση των βασικών στοιχείων που απαρτίζουν μια Δικτυακή Μουσική Παράσταση, η εφαρμογή και το δίκτυο, καθώς εξελίσσεται η παράσταση με σκοπό βελτίωση της απόδοσης του συστήματος. Αρχικά, καθώς η καθυστέρηση δικτύου αυξάνεται, το δίκτυο επιχειρεί να απορροφήσει την αύξηση μεταβάλλοντας την συμπεριφορά δρομολόγησής του. Σε περίπτωση που η καθυστέρηση δικτύου ξεπεράσει ένα μέγιστο επιτρεπτό όριο, το δίκτυο αλληλεπιδρά με την εφαρμογή ώστε να μεταβάλλει τον τρόπο επεξεργασίας του ήχου, με απώτερο σκοπό να αντιμετωπιστεί η αύξηση της καθυστέρησης. Η παραπάνω αλληλεπίδραση επιτρέπει τιμές καθυστέρησης από άκρο σε άκρο χαμηλότερες του κατωφλίου όταν σε διαφορετική περίπτωση δεν θα ήταν εφικτό. Η προτεινόμενη αρχιτεκτονική υλοποίησης ενσωματώνει την Καθοριζόμενη από Λογισμικό Δικτύωση, καθώς αυξάνει την ευελιξία και την δυνατότητα προσαρμογής σε ξαφνικές μεταβολές του δικτύου. Επίσης, η υλοποίηση κάνει χρήση της ικανότητας της κεντρικής οντότητας σε κάθε Δίκτυο Καθοριζόμενο από Λογισμικό, του ελεγκτή, ο οποίος έχει καθολική εικόνα του δικτύου και μπορεί να λαμβάνει βέλτιστες αποφάσεις δρομολόγησης. Μέσα από την παρούσα μεταπτυχιακή εργασία επιχειρούμε να αναπτύξουμε και να παρουσιάσουμε ένα πρωτότυπο σύστημα Δικτυα-

κών Μουσικών Παραστάσεων το οποίο αξιολογήθηκε σε περιβάλλον εξομοίωσης. Τα αποτελέσματα έδειξαν μείωση της καθυστέρησης από άκρο σε άκρο μέχρι 59%.

Acknowledgements

First of all, I would like to thank my supervisor Prof. Xenofontas Dimitropoulos for showing me his trust and belief during my studies as a researcher and as a teaching assistant in his courses and giving me the opportunity to work with him and his great advice and support.

I would also like to thank the members of my dissertation committee, Professors Panagiotis Tsakalides and Athanasios Mouchtaris for their valuable suggestions, questions and advises for this work.

I am also really grateful to Christos Liaskos, Vasileios Kotronis and Nikos Stefanakis for the continuous support, ideas and valuable help and contribution to this work.

I would like to acknowledge the Institute of Computer Science (FORTH-ICS) for the sponsorship and especially Telecommunications and Networks Laboratory and Signal Processing Laboratory for providing all the necessary equipment during this work. Moreover I would like to thank GRNET for providing equipment for this research work.

I would also like to thank all my colleagues at the Telecommunications and Network Lab and especially all the INSPIRE group members for their support during these years. I would like to thank Petros, Gabriel, Michalis, Alex and Dimitris for the helpful discussions, the encouragement and their useful advice.

Last, but definitely not least, I would like to devote this work to my father and my grandfather that they would be proud of me if they were with us. Also this work is totally devoted to my favorite grandmother and all my family for their support and belief during these years. Thank you all.

The project leading to this application has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 338402.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In this master thesis we examine a prototype of cooperation between application and network for better performance in delay-sensitive network applications. A large number of the applications used in daily life by humans require Internet connectivity. In this category belong instant messaging services, social network services, world wide web browsing, file transfer, multimedia streaming, financial transactions etc. In all above cases, user occupies applications or the web browsers to send or receive data to destinations or from sources that are located hundreds of kilometers far away. Among the above categories, there are subcategories due to the requirements that a service contains. For instance, multimedia streaming services require low delay connectivity and file transfer services require faultless transmission but they don't have low delay as priority. Due to the nature of each service type, there are different service categories formed. In the current master thesis, we conduct a study of the interaction between application and network in ultra-low delay sensitive applications focused on Networked Music Performance systems.

Ultra-low delay sensitive applications form a subset of real time applications. As the name indicates, real time applications refer to applications that require human interaction in real time. This means apart from the fact that people participating in the service might be in different places around the word, distance should not affect service performance and they should interact as being in the same place. It is obvious that in real time applications, end to end delay is a critical metric and it should be kept in low levels. In case of ultra-low delay sensitive applications, end-to-end delay should be minimized or eliminated. An interesting type of ultra-low delay sensitive applications is called Networked Music Performance services (NMP). The term NMP was initiated by John Lazzaro from Berkeley University in 2001 and since then the term is globally used for describing real time distant musical interaction using the Internet [1].

Compared with other multimedia real time applications, Networked Music Performance category has one key feature. As described above, an NMP event describes the process

where musicians located in different places around the world perform together via the Internet. This process which refers to proper interaction between the musicians has very low delay tolerance [2]. More specifically, in NMP services the maximum affordable delay between the transmitted and the finally played signal should be up to 25 ms. This constraint is called Ensemble Performance Threshold (EPT) in audio community [3]. Following the above restriction, NMP is applicable only in case that the physical distance between audio source and destination does not result in latency over 25 ms and a stable Internet connection is offered [4]. In the current master thesis, a study of the NMP systems takes place, considering the state of the art about this topic, parameters that affect their performance and a possible architecture to overcome problems that happen during the process.

1.2 Organization

The rest of the thesis is organized as follows. In Chapter 2 we introduce all the relevant background related to NMP systems as well as Software Defined Networking, OpenFlow and development platforms that we used in this master thesis. A detailed description about related work and similar projects is in Chapter 3. In Chapter 4 we describe the methodology that we used in our approach and examine some use cases that our system covers. After this, in Chapter 5 we present the results from the performance evaluation of the system, in Chapter 6 our approach is compared with other approaches and finally in Chapter 7 we analyze the conclusions that came from our approach, constraints that we came up with during implementation and possible future extension of our system.

Chapter 2

Background

In previous chapter a short introduction about Networked Music Performance systems took place. In this section, there is a detailed analysis of the performance of NMP systems, the structural modules that are required for a Network Music Performance event and the main constraints and problems that rise during the process. Additionally, there is a description about the different implementations in NMP area.

Comparing NMP systems with other teleconference applications, there are many obvious causes that result in end-to-end delay. These can be grouped in two levels: the first level contains the delays that are related to the audio context. In audio level, delay is caused by audio capturing from the audio hardware, audio coding in transmitter's side and decoding in receiver's side for audio quality modification and audio signal fragmentation into audio packets called frame sizes that will be transmitted. The second level contains the delay caused by transmission via the network and queuing during transmission.

About network delay, given a direct path between 2 peers, the propagation delay using an optic fiber is limited by $0.7 * c$, where c denotes the speed of light. This results in 5 ms network delay increase every 1000 km [4]. Figure 2.1 shows the network average delay distribution around the world.

In practice, European backbone structure does not contain direct links between peers. Transmitted packets should pass from hundreds central routers and switches in order to reach final destination, depending on the routing strategy. In each switch, packets are delayed for a few milliseconds but in routers waiting period is longer. This delay is created by queuing that is applied during transmission. In an ideal scenario, routers should forward the packets that they receive instantly but in cases of bandwidth overload this is not feasible. This explains the jitter that is appears and affects transmission. Delay because of queuing confirms that the finally evaluated network delay is higher than the real propagation delay caused by the physical distance between peers.

Apart from the delay caused by routing policies, delay is also caused by limitations in

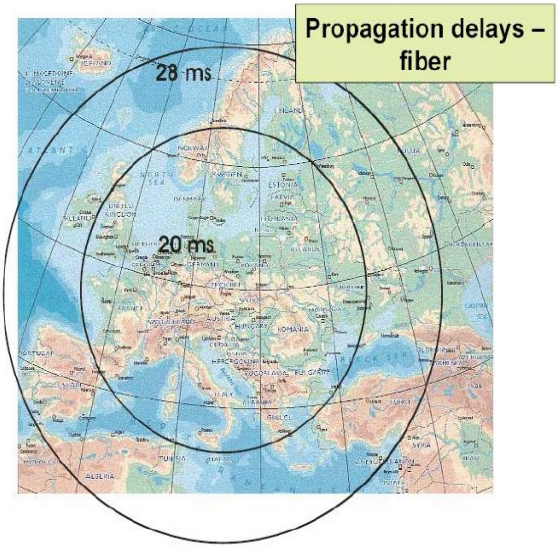


Figure 2.1: Average propagation delay diameter [4]

bandwidth offered to users by Internet Service Providers (ISPs). For conventional Internet connections such as DSL, following the constraint of the EPT makes NMP impossible for many reasons. First of all, even a small ICMP (Internet Control Message Protocol) packet has response time beyond 50 ms which is twice the value of EPT as mentioned above. Secondly, the majority of sound-cards work with quality 48 kHz/16 Bit and for upstream demands in range between 128 to 512 kBit/s, which prevents the audio signal to be sent out over these links (48 kHz * 16 Bits * 1 channel = 768 kBit/s).

Using audio compression techniques would be an important solution towards reducing bit-rate to required levels but conventional audio coders increase latency due to encoding/decoding process and this is not acceptable in NMPs. For instance, standard coders like MP3 or AAC have a delay of about 100 ms or more. Even AAC-Low Delay encoder still increases about 20 ms using 48 kHz sampling rate [5]. This automatically prevents community from using conventional audio encoding/decoding methods.

Apart from traditional encoding/decoding methods, audio research community has developed a small number of tools that offer ultra-low delay coding by reducing used frame size. The most popular among them are the Advanced Audio Coding-Low Delay (AAC-LD) algorithm [6], the Ultra Low Delay (ULD) audio coding algorithm [7] and the Constrained Energy Lapped Transform (CELT) codec, recently merged into the Opus Codec [8]. CELT offers ultra-low delay coding and feasible bit-rate requirements. Also, it is open source software and this makes it attractive to research audio community. Opus encoder supports discrete sampling rates which are 8, 12, 16, 24 and 48 kHz and the algorithmic delay varies in range between 2.5 and 20 ms. Bit-rate levels are achievable between 6 to

510 kbps. For instance, using frame size values equal to 20 ms and stereo music encoding it is recommended to use bit rates in the range of 64 to 128 kbps.

The overall delay using uncompressed audio for the signal that inserts the sound-card output of the transmitter till audio signal reaches receiver's output is called mouth-to-ear delay. From the above description, mouth-to-ear delay results from the following equation:

$$d_{mouth-to-ear} = d_{sound-trans} + d_{proc-trans} + d_{network} + d_{sound-rec} + d_{proc-rec} \quad (2.1)$$

where $d_{mouth-to-ear}$ denotes mouth-to-ear latency, $d_{sound-trans}$ is the delay inserted by the transmitter's sound-card, $d_{network}$ is the delay added due to transmission through the network and $d_{sound-rec}$ is the delay inserted by the receiver's sound-card.$d_{proc-trans}$ and $d_{proc-rec}$ describe the delay inserted due to audio processing and encoding/decoding in transmitter/receiver side. In this master thesis uncompressed audio is transmitted so equation 2.1 is transformed to Equation 2.2. Figure 2.2 depicts graphically mouth-to-ear delay meaning.

$$d_{mouth-to-ear} = d_{sound-trans} + d_{network} + d_{sound-rec} \quad (2.2)$$

In cases that transmitter-receiver use sound-cards with similar specifications regarding to reading/recording processes, Equation 2.2 is transformed to:

$$d_{mouth-to-ear} = 2 * d_{sound} + d_{network} \quad (2.3)$$

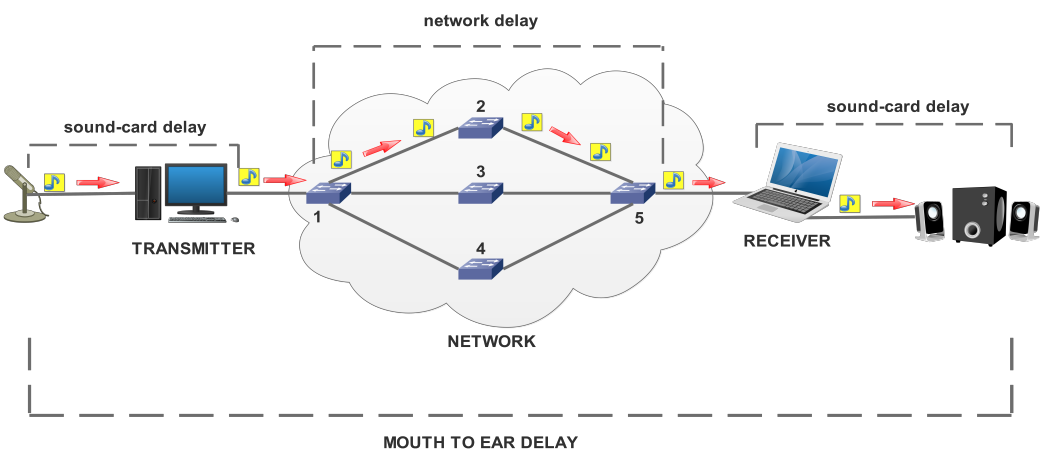


Figure 2.2: End-to-end delay in NMP systems

The latency inserted by the sound-card is called sound-card blocking delay [9]. The below matrix contains values of blocking delay as a function of frame size and sample rate.

Frame Size (samples)	**Sample Rate (Hz)**	**Blocking Delay (ms)**
2048	48000	42.6
1024	48000	21.3
512	48000	10.6
256	48000	5.3
128	48000	2.6
64	48000	1.3

Table 2.1: Example sound-card latency

2.1 Overview

Networked Music Performance systems consist of two fundamental components: application and network. As it is mentioned in previous sections, end-to-end delay is the summary of network delay and audio delay in each pair of transmitter and receiver. In order to minimize end to end delay, a good solution would be to minimize or reduce each of structural components. This strategy is followed in the implementation phase of the current master thesis. In this section, an introduction in tools that we used in the implemented architecture takes place.

In network field, due to the increasing number of applications, there is a great demand for the available network resources. Each application targets to satisfy user's demands by providing best-effort services. In order to quantify the level of user's satisfaction, a new term has arisen. This term is called Quality of Service (QoS). In general, QoS describes the performance of network connection. This could be for instance in a Skype call, the quality of the sound, distortion during the call, echo effect, picture quality [10]. There are two metrics for QoS evaluation: Mean Opinion Score and Quality of Experience (QoE).

The importance of QoS as a metric increases the need for QoS management mechanisms. QoS management describes the ability of network to keep network parameters compatible with application requirements. Throughput is one of the most important parameters that define QoS. Throughput describes the fraction of available bandwidth for each application. Another critical set of important parameters are end-to-end delay, which is the time needed for a successful packet transmission between source and destination and jitter which equals to standard deviation of delay. Finally, error rate is also a critical parameter for QoS management which describes the fraction of missed or damaged packets through the connection.

Each application has different requirements due to it's functionality. Ideally, connectivity that satisfies required QoS should be given to each application. This is a hard process because of the restricted availability of network resources. There is a lot of research based on resource allocation for guaranteed QoS.

Integrated Services (IntServ) architecture specifies discrete roles during transmission process (sender, receiver and intermediate routers) and each one is able to request and receive resource reservations for guaranteed QoS [11]. IntServ occupies two different protocols: one for flow specification for traffic pattern export and a Resource Reservation Protocol (RSVP) for reservation message exchanges among network elements for resource reservation by applications. This architecture can be deployed in small-scale networks and provide high quality QoS [12]. In case of large-scale networks with increased number of flows, resource consumption of each router increases and reaches network's performance so it reduces performance of whole network. For this reason, IntServ is applicable only is small-scale networks.

Another method for guaranteed QoS is called Differentiated Services (Diffserv) [13]. DiffServ forms classes for traffic classification based on the service type and the required quality of service. According to DiffServ approach, domains of DiffServ services are formed and QoS is guaranteed only in one DiffServ domain. DiffServ model is easier to be implemented but on the other hand it only guarantees QoS along a DiffServ domain and does not offer end-to-end guaranteed QoS.

A similar approach to DiffServ is called Multiprotocol Label Switching (MPLS). Is is used for high performance telecommunication networks. The basic concept is that in this protocol traffic from one node to the next is forwarded according to short path labels rather than conventional look-up in routing tables. As it's name indicates, MPLS can encapsulate packets of various network protocols. It supports a wide variety of services but on the other hand it lacks of real time adaptability. For this reason, MPLS is not applicable in cases of bursty traffic with rapid changes [14], [11].

For the reasons described previously, guaranteed end-to-end QoS is often solved by using separate physical infrastructure or via dedicated networks for each different type of traffic. This is not an optimal solution because it increases the overall communication cost and new assigned hardware is underutilized for a huge fraction of time. Also, routing protocols need to become more adaptive to sudden changes in the network. For instance, let's assume the scenario in Figure 2.3. A node (sender) wants to send audio to another node (receiver) into an AS-1. P_1, P_2, P_3, P_4, P_5 are the intermediate routers. If all routers use shortest path as criterion for path selection, path P_1 - P_2 -P_5 will be selected as shortest path in terms of number of hops, despite the fact that if path P_1 - P_2 -P_5 is congested and path P_1 - P_3 - P_4 - P_5 is not , which means in terms of delay that d_1 + d_2 > d_3 + d_4 +d_5. This indicates that path P_1 - P_2 -P_5 is not optimal path for transmission.

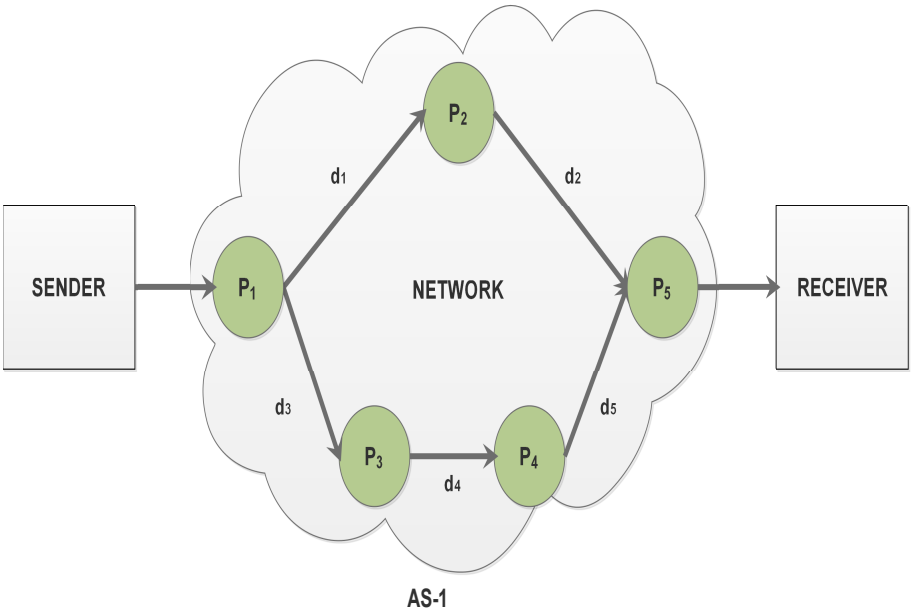


Figure 2.3: Simple Network Topology with a Sender and a Receiver

Due to the lack of dynamic network adaptability to rapid changes and failures, service providers should have their own policies for dynamic reconfiguration. For this reason, existing Internet architecture should change in order to support current and future user requirements. Also, current Internet is static and this prevents researchers from inserting new technologies and innovations. Thus, to activate real time adaptability in Internet architecture, since redesigning from scratch is not feasible due to the high cost, researchers have found alternative ways for guaranteed end-to-end QoS. In more details, current Internet architecture can be described as a superimposition of large scale networks. These networks belong to Internet Providers. Each packet, while traveling in the Internet towards a destination crosses networks of different owners. Network providers make Service Level Agreement (SLA) describing which services are provided and the cost for these service [15]. Using SLAs ensures that network providers cooperate in a coordinated way.

Because of limited network resources, a new approach for efficient resource allocation and guaranteed end-to-end QoS is application-aware networking [16]. This means that network should be flexible to adjust to QoE apllications' demands that use it [16]. This can be achievable in case that there is an entity that has a full view of the network performance and can allocate the available resources to the demanding applications. This flexibility is inserted by a new technology in Computer Networks domain called Software Defined Networking (SDN).

2.2 Software Defined Networks

Software Defined Networking is a new approach in networks field. It minimizes the rigidity and static profile, as described previously, in conventional networks. It's main target is increasing the degree of network flexibility and adaptability to each user demands. SDN's importance is enlarged by the comparison with traditional network architecture.

In conventional networks, all devices participating in transmission process have similar features refering to design and functionality [17]. The major concept is that there is hardware specialized in packet processing that composes the data plane and the hardware is manipulated by an operating system (usually Linux) that collects information from the hardware and executes software application, which is called data plane. The software application contains thousands of lines that define the network behavior and follow the rules defined by each protocol. All protocols are described in the corresponding RFCs or vendor instruction manuals. The major disadvantage of this process is that the code is not accessible by network administrator. The administrator is limited to modify network behavior via low level commands using command line interfaces (CLI).

Additionally, each node in a path can be considered as an autonomous system which takes the decision about the next hop that the packet follows towards final destination. Some protocols, like OSPF or BGP allow neighboring nodes to exchange information to decide next hop considering traffic load in the network. Information exchange helps in next hop decision but these models lack of an entity that has a global view of the network. In case of changing path, in conventional networks, administrator modifies basic parameters to reach requested performance. Each rerouting action in the network requires individual custom configuration by administrator directly in each of the participating forwarding devices, which makes rerouting process heavy and time wasting. Also, applied policies do not change dynamically and in case of rapid network changes, administrator has to separate them individually in order to avoid traffic congestion. It's obvious that current network's rigidity is a huge obstacle in real time network adaptability.

During recent years, new needs and technological trends have arisen. For instance, multi-CPU, multi-GPU and touch-screen support are some of the latest technological achievements. Also, in networks domain new features such as VLAN, IPV6, QoS have appeared in human life. This means that network has to update it's architecture by embedding new changes. However, in every operating system where software is separated from hardware, continuous update is feasible or even modification or re-installation of a software. In networks area, experimenting on new ideas and incorporating them into new devices is a tough process which requires many years' research. Also, because some devices are compatible only with other devices from the same vendor, this creates a kind of dependence on vector specific implementation.

The limitation for researchers to innovate in real hardware and the required time, effort and cost resulted in establishment of Internet Engineering Task Force (IETF). To reduce

the required cost for new ideas and make them applicable to real infrastructure, fundamental network devices should be re-programmable and re-usable. The growth of traffic over the Internet increases the demand for better network supervision. Also, to avoid cases of traffic congestion and link failure, it is necessary to improve network management functions such as path management, traffic prediction, recovery methods etc. Also, the increasing trend of link rates indicates that transmission process is based on hardware, separating control to software application. These processes require huge amount of computing resources and due to this they are executed on servers rather than simple network devices.

An approach towards reducing cost and rigidity of current network architecture is given through Network Function Virtualisation (NFV) [18]. NFV describes the architecture that IT virtualization methods are used for virtualizing classes of network node functionalities and by connecting them, communication services are created. These services can be used by researchers for experimental purposes instead of using real infrastructure that is expensive. Moreover, NFV connection can result in providing to users end-to-end connectivity along interconnected VNFs. This is called as Virtual Network Function as-a-Service (VNFaaS) [19]. Since NFVs can have programmable behavior via software, they can be used for providing certain end-to-end QoS.

The emergence for increased flexibility and network programmability that are offered by NFV and Software Defined Networking (SDN) has been embraced at large in industrial and research domain promising agility and optimization in network management. SDN is a powerful concept to increase flexibility and adaptability in today's communication networks [20], [21]. Incorporating SDN in QoS mechanisms enhances current application-aware resource allocation strategy. Using SDN for resource management benefits applications from fast and sudden change in network resource allocation which is called dynamic QoS management.

Software Defined Networking inserts major changes in current network architecture. First, it proposes decoupling between control plane and data plane [17]. This allows each of the two separated fields to be developed independently enabling evolution and innovation in both planes. A comparison between conventional networks and software defined networks is depicted in Figure 2.4. Software Defined Networking approach trends to be one of the most popular future Internet technologies. As described previously, control plane logic is removed from forwarding devices (switches and routers) that participate in the network and is incorporated into an external entity that instructs them and is called SDN controller. Instead of controller, for large scale networks, a cluster of SDN controllers can be used for better network management [22]. Single controller architecture is simpler and cheaper but centralized approach raises problem regarding to scalability. As the number of switches increases, relying on a single controller is not secure for many reasons: first, instructions from SDN controller are passed to switches as messages. The amount of control messages increases with the number of switches. Additionally, if the network diameter increases extremely, some distant switches will have longer setup delay com-

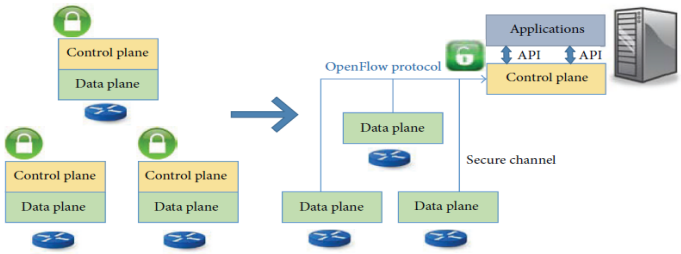


Figure 2.4: Comparison between traditional networks and SDN [17]

pared with the nearest switches. Finally, controller's processing power bounds Software Defined Networking performance. Large number of switches can cause great setup times, treating networks secure performance [23]. SDN controller can be treated as Network operating system (NOX) that allows management scripts to be executed over high level abstractions for defining network behavior [24].

The SDN controller has similar role as the maestro in a band: it orchestrates switches like the maestro that orchestrates the musicians in a concert. The relation between switches and controller in the network is shown in Figure 2.5. Software Defined Networking is

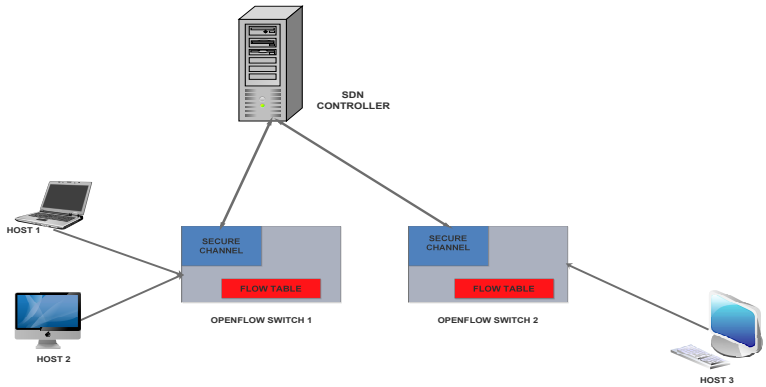


Figure 2.5: Basic components in SDN

like a software application assigned with Network administrator duties. For programming an SDN controller, a program designer with network background is needed. It replaces physical presence of network administrator (Figure 2.6) because once it is programmed, it operates continuously and in case of functionality change, it is easy to adopt because of re-programmability. This makes SDN controller more flexible since it can adjust to future demands easy and fast, without need of infrastructure update. For message exchange between SDN controller and switches, OpenFlow protocol is used. This protocol will be analyzed in the next session.

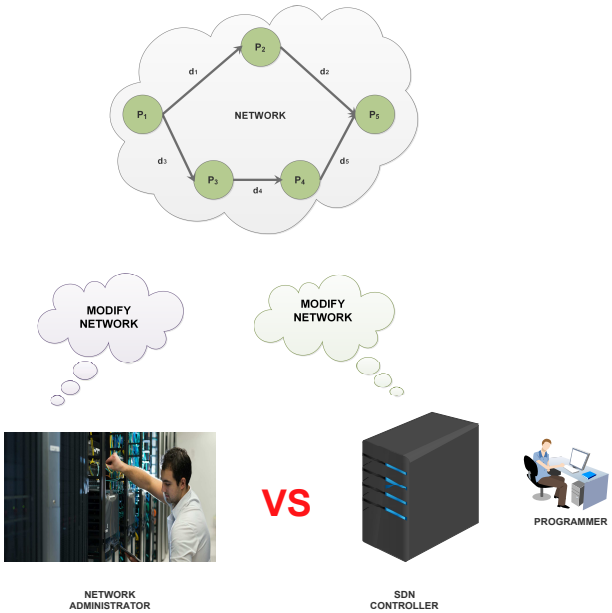


Figure 2.6: Human vs 'software' Network Administrator

2.3 OpenFlow

Traffic engineering (TE) is an important process to improve networks performance by real-time analyzing, predicting and manipulating data flows. It has been used in past years in data networks such as ATM or IP/MPLS. As mentioned before, these architectures are not suitable for next generation networks for two main reasons: first, current Internet applications require the underlying architecture to dynamically adjust to network changes and to be scalable to large amount of traffic. The required functionality needs to group different types of traffic from different applications and provide certain QoS service in a short period of time (in a few milliseconds). Also, the extended growth in cloud computing technology and the demand for data centers that receive massive amount of traffic requires more flexible traffic engineering approaches for better system performance [25].

OpenFlow is a configuration protocol that supports Software Define Networking. It is originated from research at Stanford University and the University of California [26]. Architectures that use OpenFlow can be adopted by researchers to experiment and test novel ideas and innovations. It offers a variety of capabilities for research purposes such as software-based traffic analysis, dynamic installation of new flow rules and flow abstraction [27]. This makes network configuration easier and increases security level in complex architectures such as data centers.

Related to Network Operating system that was described previously, users can create applications using high-level of abstraction of available resources and hardware. In SDN field, many researchers classify available network resources in northbound and south-

bound classes. Southbound interface offers abstraction of the programmable switch and allows connection with controller software. On the other hand, northbound interfaces allow applications and high-level policies to be applied to network and passed to the Network Operating System. The separation between the two levels is shown in Figure 2.7. OpenFlow is widely used in Software Defined Networking domain. All switches partic-

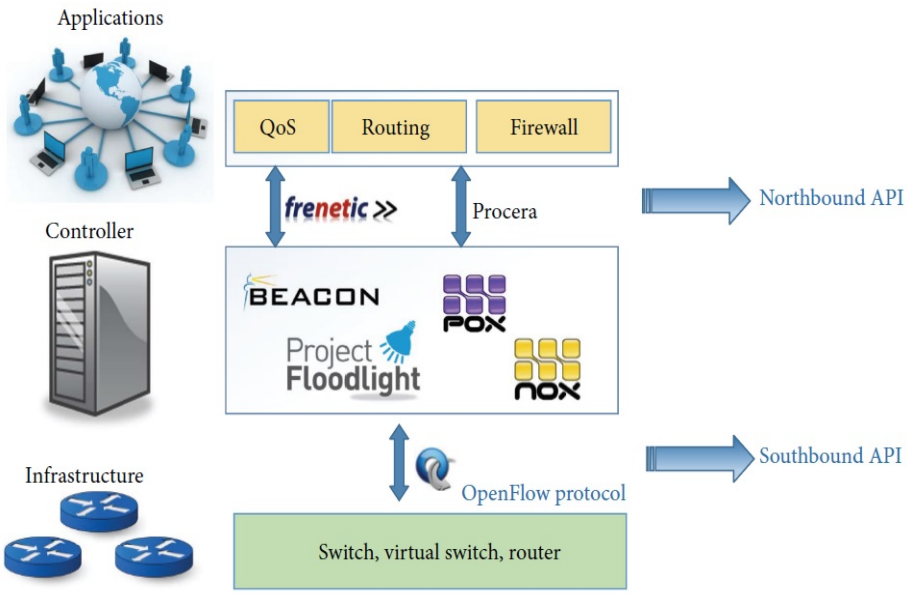


Figure 2.7: Separation between northbound and southbound APIs [17]

ipating support OpenFlow protocol. OpenFlow is used for communication between the data plane that switches contain and control plane that is located in the controller. This communication takes place via s secure channel that the controller uses to send a OpenFlow messages to the switches and receive from them.

Initially, all network switches have empty routing tables. These tables have various fields such as IP of inbound, outbound traffic, port numbers. These fields are called matching fields. In more details, each flow table can be decomposed into three fundamental modules: packet header, actions and statistics. Packet header is the pattern that is applied to match the required type of traffic and apply certain policies [28]. It selects the packets that the switch will process. The fields used in packet header can refer to layer 2, 3, or 4 from TCP/IP stack. The number and the name of the fields that are supported in packet headers are described by the RFC of the OpenFlow version. For instance, in OpenFlow 1.0 version 12 different fields were supported and in OpenFlow 1.3 version supports 40 headers including IPV6.

If the header of the incoming packet matches the existing entry in the flow table, the switch applies the corresponding actions. These actions can be either forwarding all similar packets through a particular port, or re-writing some header fields in the packet or

dropping the packet. In case that the incoming packet does not match to any of the existing packet headers in the flow table, a PacketIn message is sent from the switch to the controller. Via PacketIn message, the switch indicates that it does not have a pattern how to process this type of packets and requests further instruction from the controller. On the other hand, the controller, based on the strategy that is installed when it was programmed decides the way that this and all coming similar packets have to be processed and replies back by sending FlowMod message. In other words, FlowMod message is destined from controller to the switch/switches in the datapath and inserts a new flow rule into the routing tables. SDN controller can also send a packet though a specific port without inserting a flow rule in the routing table. This is feasible via PacketOut messages that are sent to the switches. Finally, each switch keeps statistics for the traffic that passed from it and they are available to the controller on demand. The major components of OpenFlow process are shown in Figure 2.8. In the figure are also depicted some well-known SDN controllers such as POX, NOX, Floodlight and Beacon. In the next session, there is a short intro into POX, which is the SDN controller that we used in the master thesis implementation.

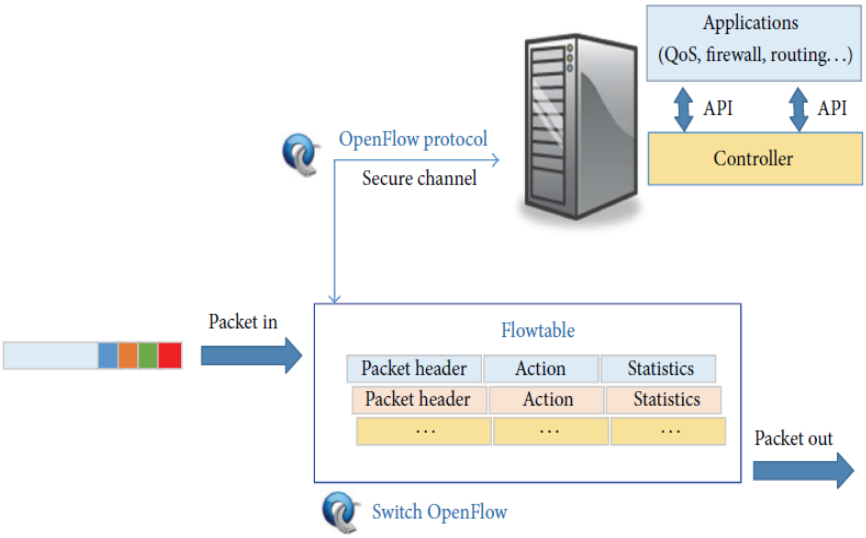


Figure 2.8: OpenFlow architecture [17]

2.4 POX

Software Defined Networking combined with OpenFlow targets to move network's intelligence from modular forwarding devices such as switches, routers to a centralized entity called controller. Forwarding decisions are taken in controller side and they are passed to the switches via OpenFlow messages through a secure channel. This allows having a global view of the whole network improving network utilization. It also explains the

decision of huge brand names like Microsoft and Google to incorporate SDN into their data centers.

An SDN controller is equivalent to the operating system in computers. Current state of SDN controllers can me compared to early types of operating systems. Controller applications are simple application scripts with hundreds of lines code that interact with forwarding devices and provide a programmable interface, making accessible to users. So a tradition Network administrator in SDN field can be replaced by a program designer with computer network background.

There are various SDN controllers. They can be classified in groups due to the programming language that they are written, their architecture and whether they are open source or not. For instance, NOX is a multi-threaded C++ based controller written on the top of Boost library [29]. Beacon has multi-threaded architecture which is Java-based and relies on OSGi and Spring frameworks [30]. Floodlight is multi-threaded Java-based controller using Netty framework [31]. MUL is multi-threaded C-based controller written on the top of libevent and glib [32]. Maestro is also multi-threaded Java-based controller [33]. Ryu is a Python-based controller that uses wrapper of libevent [34]. Finally, Open Network Operating System (ONOS) is a Java-based SDN controller that adopts distributed architecture for high scalability [35].

POX belongs to open source SDN controllers. It is widely used for rapid development and prototyping in research and industrial field related with Software Defined Networking [36]. POX is written in Python and this makes it favorable in research community. Due to this, POX is available in every simple computer that has Python installed on it [37]. POX contains reusable components for path selection, dump-forwarding, learning switch functionality, load balancing, firewall and topology discovery. It also supports GUI and visualization tools [38]. POX accepts arguments in order to modify performance and is applicable on real or experimental testbeds and also in emulation environments such as Mininet, a tool that will be analyzed in next section [39], [37], [40]. It is worthwhile to mention that both Python-based controller (POX and RYU) are single-threaded which means that they show no scalability across CPU cores. Moreover, their performance does not depend on the number of participating in the network switches since they are single-threaded [41], [42], [43]. For the implementation, POX is selected as SDN controller.

2.5 Mininet

As described in 2.2, SDN is a new trend in computer networks. For this reason, there is not a wide market for devices that support OpenFlow technology. Moreover, existing OpenFlow-supported devices are very expensive and this prevents companies from incorporating SDN in daily life. Also, the high cost for SDN-supporting devices does not allow researchers to make experiments with large number of switches for testing new ideas [38]. So, high cost of SDN devices is an obstacle for regulating it into Computer Networks field.

To overcome these limitations, virtualized approach has been used. In more details, for testing and research purposes, emulation environments have been invented. On of them is called Mininet. Minitet allows emulating different network components such as hosts, layer-2 switches, layer-3 routers and links. It uses a single Unix kernel and it can emulate a whole network into a single machine. The created virtual components are real-world elements although they are created through software [44]. One major advantage of Mininet

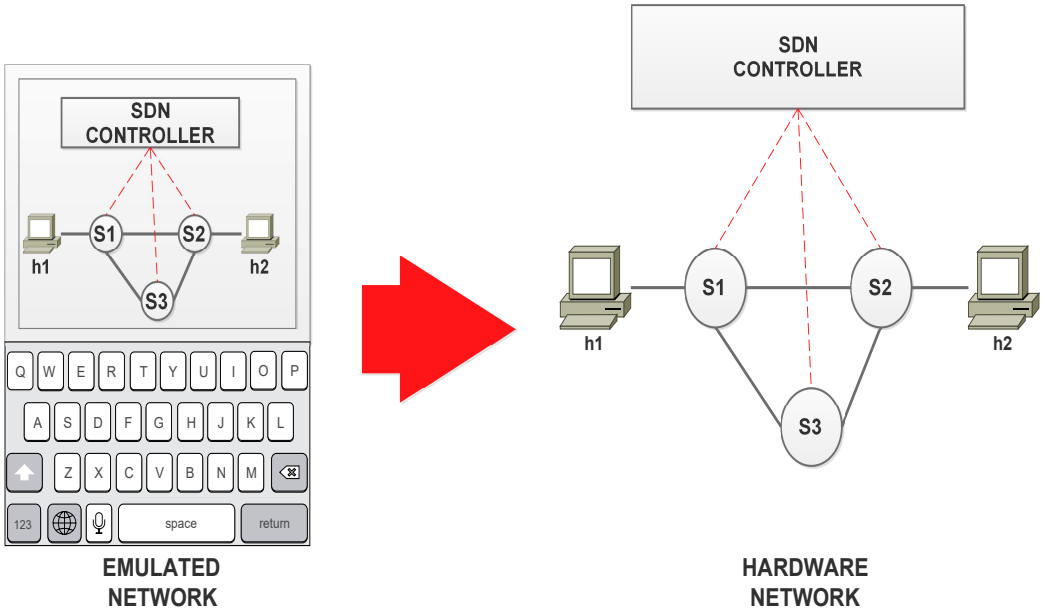


Figure 2.9: Network emulation with Mininet

is that it supports SDN technology. Mininet's VM contains various of SDN controllers like POX and Pyretic to emulate real world experiments. Initially, Mininet was created by a research team at Stanford University to teach fundamental technologies [45]. Now Mininet has evolved to a powerful network tool that supports Software Defined Networks consisting of an OpenFlow controller and an Ethernet network with OpenFlow supporting Ethernet switches that form a datapath. Mininet is open source network emulation software and it allows manipulation also by using Mininet Python API.

Mininet topologies are created by connecting switches, hosts and controller with links. These links act as real wired connections between two (virtual) interfaces. Packets are sent via the link from one interface towards the destination interface and each interface appears as fully functional Ethernet port visible to operating system. Virtual links can be attached to virtual switches like Linux bridge or OpenFlow supporting switches. Hosts are described in network namespaces providing access to interfaces, ports and routing tables. Switches act as real Ethernet switches by forwarding packets and both user-space and kernel-space switches are supported. Finally, the SDN controller can run either in the same or in a remote computer since there is IP-connectivity between the machine that

hosts Mininet and the machine that hosts the remote controller [46], [47].

Mininet enables emuting large topologies (up to 1000 hosts) because it virtualizes less and shares more. This means that file system, used ID space, process ID space, kernel device drivers and libraries are shared among the hosts participating in the virtualized network and managed by the operating system. The major limitation of Mininet is the lack of accuracy in high workload cases. Processing resources are multiplexed during emulation time by the default Linux scheduler, which does not guarantee that a host sends a packet instantly or that all participating switches act synchronized with the same rate. In Mininet, delay or bandwidth limitations can be applied on each link. This is feasible via the Linux traffic control program (tc) [48].

As a conclusion, Mininet is a useful tool for SDN innovating purposes. It provides flexibility to create custom complex or simple topologies using programming languages. It also guarantees applicability of Mininet tested applications that can perform in the same way if they are applied to real hardware. Also, it provides scalability by allowing emulating large scale networks in only one machine. Finally, Mininet is share-able which means that created prototypes can be included in virtual machines accessible to other collaborators for experimental and prototyping purposes [44]. In the implementation phase of the master thesis, Mininet is used for emulating the required network infrastructure.

Chapter 3

Related Work

Due to the exponential growth of Internet connectivity, many researchers are focused on investigating patterns for network traffic shaping. The extended Internet use by most of today applications creates large amount of traffic that have to be transmitted via computer networks. This traffic stems from file transfer, Voice over IP services, multimedia services, social networking, web-browsing etc. Low Internet cost combined with high performance have introduced Internet as an entertainment tool in human daily life. This results in the huge traffic increase but also great demand for high quality services. The metric for these services is QoS as described in 2.1, which reflects the degree of user satisfaction for the provided service. Parameters that seem to be crucial for users are end-to-end delay, jitter, packet loss and error rate. Service providers are interested in providing high QoS and they are also interested for user's feedback to improve QoS. QoE and Mean Opinion Score (MOS) are two major metrics for QoS evaluation.

In addition to Internet growth, new technologies have arisen to support Internet use. Content Delivery Networks and Real Time applications is a recent decade's trend. They exploit computer networking for multimedia delivery (e.g audio, video) or communication between people located in different places around the world. A subcategory of real time applications contains Network Music Performance services (NMPs) through which musicians in different places perform synchronized in a live music event. Among real-time multimedia services, NMPs have the most stringent delay constraints, since musicians' cooperation has a very low delay tolerance. In NMP, as described in 2.1, affordable end-to-end delay is about 25 ms (EPT). Network Music Performance services is a representative example of high QoS, that community has approached with multiple ways in order to improve provided QoS. These approaches are related with the two modular components that are combined in NMPs, which is Audio processing and Networking.

3.1 The audio-centric perspective

From audio context perspective, many research topics examine the way that signal is transmitted towards the receiver. In more details, as described in [49], each transmitter sends one or more audio streams. All receivers are not interested in receiving all transmitted streams. For this reason, they apply filters to the transmitted signal in order to receive only audio streams that are interested in. To find which flows they are interested, each participant has formed an audio profile, according to which stream flows that match profile's criteria are only received. This processing stage for matching audio streams to profile increases complexity and end-to-end delay. Based on the number of streams that are transmitted and received, transmission type can be either multicast where each audio flow is transmitted to all participants. Multicast does not require processing stage of matching streams to interested participants so it does not increase delay but multicast method causes network underutilization since all participants receive the same amount of audio traffic without taking into account if they are interested or not. On the other hand, selective transmission to only interested participants requires separation among participants in order to find the interested according to their profiles and this results in delay increase but in this way participants avoid receiving streams they are not interested in so enables better bandwidth allocation and optimal performance. This transmission type is called unicast.

For selective transmission of audio streams a usual solution is called Selecting Forwarding Unit (SFU) [49]. This component is used for deciding the audio streams each participant is interested in. For this reason, all clients can send requests to SFU in order to declare the interested audio streams and SFU records all these requests and simply forwards the flows that match participants preferences [1], [50], [9], [51]. Also, many researchers evolve the functionality of SFU. For example, because audio flows have to be mixed in order to reach the destination participant, SFU is replaced by another entity called Multipoint Conferencing Unit (MCU) [52], [2]. The role of MCU is that it receives all audio streams from transmitting participants, it mixes them and transmits the audio result to the receiving participants [52]. Apart from MCU functionality, a common trend in NMP projects is that Session Initiation Protocol (SIP) [53] is used to support control messages among transmitter and receiver side. SIP is a protocol widely used in parallel with Real Time Protocol (RTP) for initial handshaking and dynamic transmission modifications during runtime [54], [55], [56], [57], [58]. For instance, in Figure 3.1, all participating clients that can be either musicians or audience, use SIP server in order to confirm audio transmission parameters and Media Relay Server acts as SFU and collects all music flows from all participating nodes and forwards them to each other without inserting processing overhead in terms of time.

In NNP application field, there are many industrial projects for real time audio delivery. These applications can be grouped in three categories [9]:

- Realistic Jam Approach (RJA)

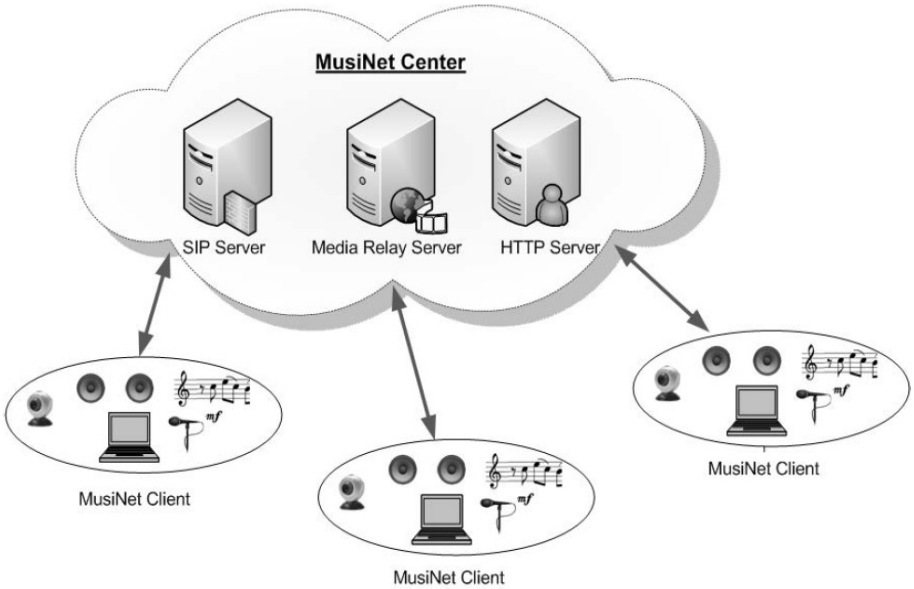


Figure 3.1: Musicnet approach using SIP and SFU [52]

- Latency Accepting Approach (LAA)
- Remote Recording Approach (RRA)

The first category of 3.1, Realistic Jam Approach contains applications where real-time live music interactions are critical so the final music result should be similar with the case musicians would be performing in the same room. This means that delay due to the distance between them should be minimized. RJA approach targets to send and receive data as quickly as possible, given the assumption that network connectivity is also good. In RJA approach belong projects as Soundjack and eJamming.

Soundjack [50] is an application developed by Alexander Carôt at the ISNM in Lübeck, Germany and inspired by the SoundWIRE project. Soundjack offers direct access to the sound-card buffer and uses UDP/IP protocol for audio transmission. It does not require any relay server because it is based on peer-to-peer connectivity. The quality of the audio stream depends on the sound-card's quality and the network's quality. In order to avoid jitter, soundjack allows audio buffer increase and it also provides GUI interface that gives useful operational information to the user.

Another application that belongs to RJA is called ejamming [59]. It is similar to sounjack but it uses MIDI encoding which reduces required bandwidth. MIDI transmission is user-enabled so the system does not have to process continuously data which makes it vulnerable to jitter case. In case of jitter, packet is played delayed. User can also define the maximum affordable jitter value over which signal is not played out. This value is

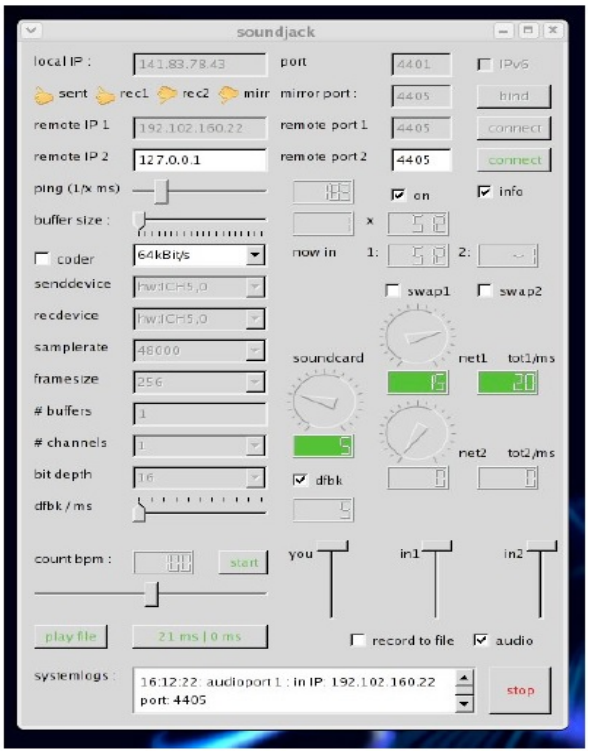


Figure 3.2: Soundjack GUI interface [9]

called Late-Note Tolerance. Also, eJamming inserts the approach of delayed feedback [60], where which delays the sound of its own instrument in order to be synchronized with the incoming sound from external participants.

The second category described in 3.1 is called Latency Accepting Approach (LAA). According to this approach, network is regarded as a decentralized and space independent medium and network delays at 200 ms or more are affordable. Thus, these delay values violate the EPT constraint so these applications are not regarded as ultra-low delay sensitive. Well-known LAA projects are Ninjam and Quintet.net.

Ninjam [61] accepts the assumption that network latency prevents participating musicians from real time synchronization. In practice, musicians play asynchronously to the music the other participants have played at least one measure before. This assumption is called faketime. Ninjam requires a central entity supporting SIP signaling in comparison to Soundjack and eJamming and act as SFU and also applying delay cancellation mechanisms.

Quintet.net is a network performance application developed by composer and computer musician Georg Hajdu [62]. It supports performers from up to five different places. It

requires central server collecting and re transmitting audio streams to the clients. The server is coordinated by another fundamental entity called conductor which can change dynamically transmitting audio parameters and inform participating musicians for possible problem encountered. It also supports video context. The assumption of certain network delays requires an adaptation during mixing in server sides for better acoustic result.

The last category in 3.1 is called Remote Recording Approach (RRA). This approach uses Internet as a medium for remote recording sessions. The audio signal when sent is "time stamped" and this allows delay cancellation or absorption when receiving. RRA application does not support real time human to human interaction. In this category belong Digital Musician Link (DML) and VSTunnel.

Digital Musician Link (DML) [63] allows two users to agree on a session. The two DML users have discrete roles: one is playback/recorder (A) and one as the performer. Client A starts the process by assigning node B a track in his production and start recording process. On the other hand, node B receives the mixed stream that node A sent and plays his track to the session as being in the same studio with node A. Node B records a time-stamp for each recorded sample which send it to node A who sorts the received data and put it into the assigned track. The playback of node A does not start until node B data has finished transmitting so both nodes act as being in the same environment. Fot DML participation, each user should pass authentication process. DML provides several levels of provided service based on the fee paid. Figure 3.3 shows a screenshot of Nuendo session with the DML support.

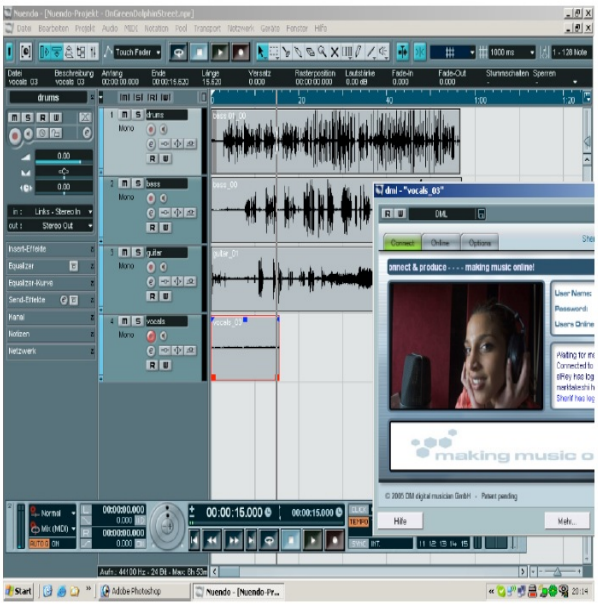


Figure 3.3: Nuendo session with DML-Plugin [9]

VSTunnel Plug-In [64] allows user to initiate session between them or join existing sessions. Each created session can be either public or private. Each user can be preview all public sessions and by clicking on it can join the session. In case of local audio changes, application informs the other participants. Each one receives a mix of all transmitted audio streams. Additionally, when receiving audio each participant can modify the quality or the volume of the received signal. VST also allows instant messaging with chat among users.

Apart from the applications mentioned previously, there are more recent applications like Jacktrip [65], Distributed Imersive Performance [66], Diamouses [67] etc. All these applications transmit uncompressed audio streams which require excessive amount of bandwidth. For this reason, in order to support these technologies leased line or high quality Internet connectivity is the only solution for high performance music events.

Except from tight analysis of a NMP system as an audio processing application, in 2.1 NMP systems are described as high QoS demanding applications due to the strict requirements they contain about end-to-end delay and jitter. For this reason, NMP study should also cover QoS area and especially QoS supported by Software Defined Networking. In 2.2, we describe that SDN is a evolutionary trend that tries to make networks more flexible and robust to sudden network changes. SDN's administrator, the controller, can have a global view of the network and satisfy the application requirements with particular routing decisions. So, network can interact simultaneously with each application in order to provide better QoS services. This creates a kind of QoS-aware networking, a subject that many researches have approached in many different ways.

3.2 The network-centric perspective

One approach that is globally met in many research studies for high QoS services via SDN is matching traffic patterns to incoming traffic in order to split it into multiple flows destined to different queues [68], [69], [70],[71], [72], [73]. The main concept of all these approaches is that there is a kind of prioritization between users and for separation between flows with different priority, queues with different parameters are used. High priority flows are directed to queues that apply fast forwarding and this results to low end-to-end delay and low priority flows are sent via larger delay queues. The separation between flows can take place according to the kind of the flows, the importance of the transmitted data each flow carries or the fees paid to the service providers for better quality services defined by corresponding SLAs, as described in 2.1.

Traffic classification is a usual method in QoS-centric research in computer networks area. Researchers try to find discrete features in order to classify traffic and apply different routing policies to each kind of traffic [68], [74]. Also, in [22] Type Of Service

(TOS) matching is introduced. Traffic is grouped in two categories: business traffic and best-effort traffic. If TOS field is enabled in a packet, this denotes business traffic and packet is forwarded through high priority queue. Otherwise, traffic is regarded as best-effort and packet is forwarded to second priority queue. In [75], also a similar approach is proposed. Incoming traffic is grouped into data flows and multimedia flows. The second category flows are dynamically assigned on QoS guaranteed paths and data flows remain on traditional shortest paths. Criteria for traffic classification are headers used in MPLS, TOS field in IPV4, Traffic class field in IPV6, the source IP in case of a well-known multimedia server or matching pair of Transport port numbers.

Another important feature in QoS-aware Software Defined Networks is the methodology that is followed to classify the quality for each path of the network [76]. For instance, in [54] the proposed method initially selects the best delivery nodes. The SDN controller receives measurements from these nodes about delay and jitter. Then the controller installs flow-rules for path between source IP and best delivery node using MPLS approach. Finally end-point node evaluates the QoE of the service and informs the SDN controller. In this work also Constrained Shortest Path First algorithm is proposed over MPLS in order to find the best routing path for each streaming flow.

In [16], SDN controller collects application state information periodically and based on this feedback it manipulates network resource management in order to optimize user experience. Another interesting approach is introduced in [77], where they try to map QoS metrics to QoE levels. For this reason, there is a QoE-server that requests periodically QoS metrics from Mobile Network Operators (MNO) for QoE monitoring. These requests are translated into flow rules for the MNOs. The MNO asks the appropriate nodes and informs the controller to modify policies for improved QoS. Service collaboration with Network Providers is also proposed in [15] and [68] where negotiation between controller and Internet Service Providers takes place for QoS optimization. Feedback collected in distributed way and information transmission to SDN controller is also proposed in [78] and in [19]. QoS-related information is also collected from the controller in Real-Time Online Interactive approaches introduced in [79], where boundaries of QoS metrics are examined from the controller.

In [14], incoming traffic is classified too. The main difference is the way that available paths are assigned by optimizing Constrained Shortest Path Problem with suitable cost function selection. On the contrary, in [80], Mixed Integer Program Algorithm is used for path selection. Research described in [10] introduces an approach that a certain flow can split in various sub-flows. They are sent separately and they are multiplexed into a Single Application flow at destination side that acts as a Multi Path Agent(MPA). The goal of this separation is to maximize the aggregated QoS for the whole network [81]. Rerouting process is feasible through over-writing IP headers, where a gateway component changes the headers of packets [82].

Session Initiation Protocol (SIP) is commonly used in many QoS-aware projects. In [83],

audio codec adaptation over Software Defined Networks is introduced. This project contains a SIP module that records SIP messages exchanged by VoIP users. The application contains a list of the available codec types. The SDN controller requests information about flow statistics and port statistics from the OpenFlow switches and in case of traffic congestion, controller informs the SIP component and participating clients change audio processing codec. The controller requests statistics from the switches periodically in time spaces called polling period. Value given to polling period is crucial because in cases of small polling period, controller has an accurate recent view of network condition but requesting statistics continuously is a huge workload for the software switches resulting in bandwidth underutilization. On the other hand, in cases of huge polling period, OpenFlow switches are not congested but also the controller is not able to react instantly in cases of traffic congestion or link failure because an important amount of time will pass until it requests for next time statistics and discover the problem in the network. So, there is a trade-off in polling period value selection. Also, SIP functionality is used in [11] for video context. Statistics request method is met in [13], where the controller sends OpenFlow requests to the switches requesting port stats for selecting next path. In this work, a modified version of Djikstra algorithm for routing is applied. In more details, based on the flow and port stats controller receives, for every pair of switches i and j in the network evaluates the weight of the link between i and j as shown is Equation 3.1:

$$w(i,j) = \frac{C(i,j)}{C(i,j) - max(res(i,j), est(i,j))} \tag{3.1}$$

where C(i,j) denotes the capacity of the link between switches i and j, res(i,j) denotes the reserved bandwidth for the link and est(i,j) denotes the overall occupied bandwidth of the link. Quantity w(i,j) increases rapidly with utilization and in cases of heavy load traffic network performs better. In [84], SIP process is supported by a server who separates premium among common users to provide better Quality of Service and in [85] signal video adjustment is used. An interesting approach is shown in [86], where SIP functionality is used and all available paths are ranked except the optimal path. In case of rerouting decision, controller decides based on the path ranking for the next selected path.

There are some other remarkable approaches for QoS-aware networking. For instance, in [87], where multiple queues are used for packet forwarding, in order to achieve the desired bit-rate, shaping delay is applied to each one of the queues. In [16], a web-browser plugin called Yomo is used to identify flows that are used for TCP video transmission and it synchronizes them before receiving. [18] presents an alternative metric for evaluating QoE. The new metric is named as Structural Similarity (SSIM). If x is the initial video sequence VS_0 and y is the transmitted video VS_t their SSIM is denoted as SSIM(x,y). For QoE evaluation, in transmitter side $SSIM_0W$ is evaluated by comparing the original video VS_0 with a reference white video VS_W. Reference white-video is called a video sequence of white video frames of the same resolution and frame rate. Also, in receiver side $SSIM_{tW}$ is evaluated by comparing the received test video signal VS_t and the same reference white video VS_W. So, the satisfactory accuracy expressed by SSIM index be-

tween the original and test video sequence is denoted as $SSIM_{0t}$ and is given by Equation 3.2:

$$SSIM_{0t} = \frac{SSIM_{0W}}{SSIM_{tW}} \tag{3.2}$$

The whole process for SSIM evaluation is depicted in 3.4:

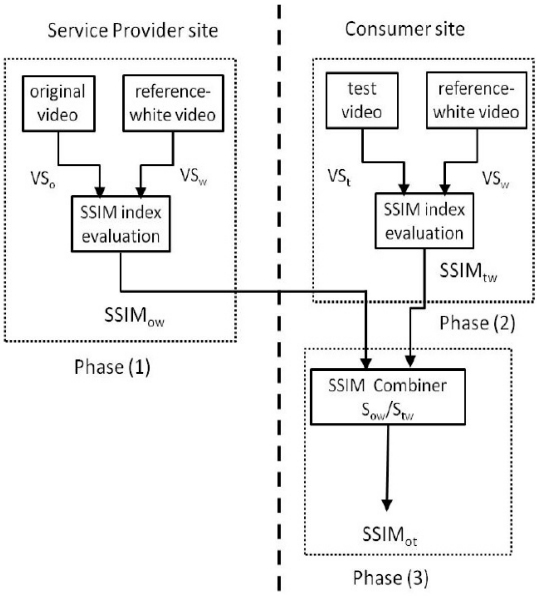


Figure 3.4: SSIM evaluation process [18]

Another interesting approach mentioned in [88] is that SDN controller does not request periodically stats from the network for deciding to reroute a specific flow. According to this method, the controller keeps a kind of history for each flow recording previous requests. So, it automatically assigns a path compatible with past QoS requests. In [24], a prototype QoS-aware Network Operating System is proposed (QNOX). The modular elements of this operating system are Service Element, Control Element, Management Element and Cognitive Knowledge Element. SDN controller duties are embedded into Control Element which monitors the network via Cognitive Knowledge Element and modify network behavior if it is required. Except from dynamic rerouting decisions, other approaches for guaranteed QoS propose use of virtual networks by service providers as the prototype described in [89]. Apart from SSIM metric, QoS is evaluated through Mean Opinion Score (MOS) [90], or in case of low QoE indicated by users, SDN controller decides to apply rerouting policies [91]. Instead of re-routing strategy, other studies focus on multipath routing for differentiated services where multiple paths between each pair of source and destination IPs [92]. Finally, a small amount of research is devoted in regarding Network As A Service approach (NAAS). In more details, in [93] via NaaS high level of abstraction is offered. In this approach, network calculus is used to evaluate service

curve and define general capacity profile. The capacity profile is evaluated as the convolution of the profiles of all links offered by individual network services for end-to-end transmission. Capacity profile is evaluated for path selection for each type of service in the network.

As a conclusion, Network Music Performance systems and QoS-aware applications supported with Software Defined Networking are two sides of the same coin. In related work, as analyzed in Chapter 3, each approach covers either the audio processing or networking perspective. Researches focus only on one side and accept the second as certain. For instance, they experiment on smart ways to reduce audio processing delay without considering that they can modify network performance. On the other hand, network researchers try to find optimal routing patterns for reducing network delay but they do not examine audio processing delay. In QoS-sensitive applications, which have excessive requirements and strict constraints, examining only one aspect given the other does not benefit application performance to a remarkable degree. Thus, experimenting on both sides and regarding one as a complement for the other, can lead to better performance as it will be shown in next chapter.

Chapter 4

Proposed architecture

In this chapter there is a detailed description of the proposed architecture. At first a general description of the problem and terminology takes place, then the used platforms/software is described and finally there is an analysis on the implemented schema.

Introduction As described in 1.1, Networked Music Performance systems belong to ultra-low delay sensitive applications. The maximum affordable end-to-end delay for NMP systems is about 25 ms. Equation 2.1 shows the mathematical expression of end-to-end delay evaluation in terms of audio processing and network delay. In more details, Equation 2.1 presents end-to-end delay as an aggregate of:

- Delay created by transmitter's sound-card during audio capturing
- Delay due to encoding process applied in transmitter side
- Network delay
- Delay due to decoding process applied in receiver side
- Delay created by receiver's sound-card during audio playing

In case of uncompressed data transmission, Equation 2.2 evaluates end-to-end delay as a function of network delay and sound-card delay. So, from the analysis that took place in 1.1, the major causes for end-to-end delay come from the modular components of the NMP system, which are the audio processing and network.

As shown in Chapter 3, there are two major types of approaches in NMP field. The first type is based on audio processing without considering network delay. This means that researchers focus on investigating new encoding methods, patterns for audio fragmenting, aiming to reduce delay increased due to audio processing. They regard network as a 'black-box' that has audio as input with certain delay and they regard it as constant overhead in their systems. Many of the approaches support Session Initiation Protocol, that enables signaling between nodes in order to negotiate for audio transmission parameters.

The second type of approaches examines the network aspect of an NMP without considering delay due to audio processing. This contains research for routing methods that eliminate network delay and delay due to packet switching but they do not examine the way that the audio signal is fragmented into data packets to transmit though the network. In Chapter 3, it is mentioned that although network delay is a key factor that affects end-to-end-delay, also audio delay can result in significant delay. For this reason, examining only one perspective of the problem cannot lead to powerful solutions.

In this master thesis, the proposed method combines both audio and network configuration for better application performance. This is feasible through close communication between application and network. Application side manipulates the audio process between transmitter and receiver. Network does not support conventional architecture but it allows Software Defined Networking. In 2.2, we describe all capabilities that SDN offers. In our approach, we exploit the ability of SDN controller that has a global view of network performance. This allows taking optimal routing decisions for audio packets. Also, controller can dynamically change network behavior in case of network congestion and link failure, which is very useful for delay-sensitive transmissions. The detailed description of the proposed functionality is analyzed in the next section.

4.1 Architecture

Classic SDN controller is assigned only with routing duties. It uses OpenFlow protocol to instruct switches and install flow rules in their forwarding tables. In our approach, controller can interact dynamically with the application side. This communication includes replying to application for path requests with certain values of delay and jitter. Controller also monitors all available paths in real-time to discover traffic congestion. In such a case, it dynamically decides to reroute audio flows to an alternative path which performs with less delay. Also, if there are no available paths that satisfy application's requirements, controller informs application to modify audio processing parameters to reduce audio processing delay. This can result in significant end-to-end delay decrease, which is crucial in ultra-low delay sensitive applications, like NMP systems. The target of this process is to overcome network delay increase by audio processing modification. The basic block diagram of the proposed approach is depicted in Figure 4.1 The major components of the proposed architecture are shown in 4.1. Each one of the entities in the list has a discrete role in the architecture.

- Transmitter
- Receiver
- Network
- SDN controller

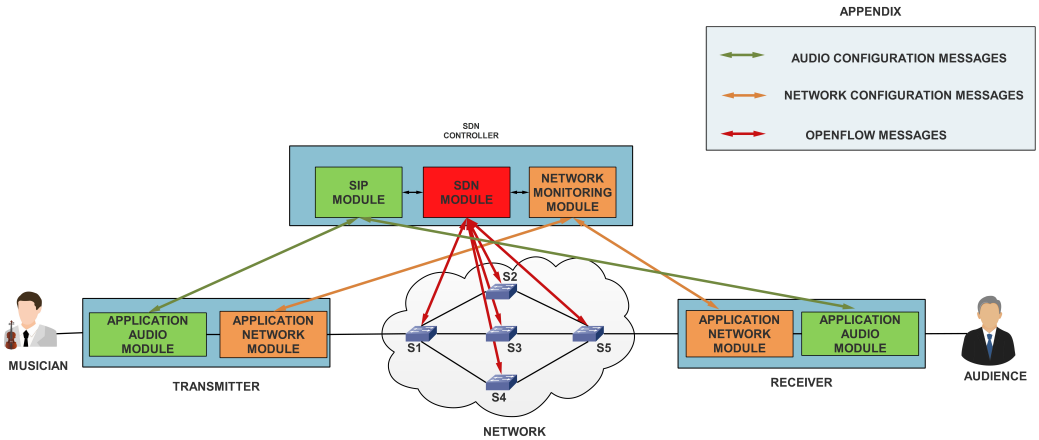


Figure 4.1: Proposed architecture

Transmitter is the entity that uses the application to transmit audio context through our network. For instance, transmitter can be a musician in a Teleorchestra application. Transmitter's duties are assigned in two modules: the first module is called Application Audio Module. Transmitter should interact with SDN controller dynamically during the live Networked Music Performance. Transmitter initially uses Application Audio Module when requesting a path from the controller with certain delay and jitter requirements. Transmitter's requirements are handled by SDN controller's SIP module which is assigned to find the path that satisfies transmitter's demands. Also, when audio processing modification is required, SDN controller instructs transmitter via Application Audio Module to change transmission pattern to overcome over-EPT delay cases. Application Audio Module is also assigned with acquiring sound data process from sound-card's input for audio transmission. Finally, Application Audio Module evaluates the delay that inserts transmitter's sound-card for capturing and processing, that will be presented in 4.2.

In addition to Application Audio Module, transmitter has another component called Application Network Module. The main task for this functionality is that it receives audio data acquired from transmitter's sound-card as input and transmits them through the network. For audio transmission, socket programming is applied. Due to the strict delay requirements of the application, User Datagram Protocol (UDP) [94] sockets are used. Apart, from audio transmission duties, Application Network Module is responsible for monitoring delay in real-time and jitter per path and informs the controller. The process of delay monitoring will be described analytically in section 4.3.

Transmitter's functionality is implemented in Mathworks Simulink platform which offer pre-compiled blocks that can be executed in real time. In more details, the used blocks are UDP Send and UDP Receive for User Datagram Protocol transmission through network and Matlab fcn which implements the logic of audio modification after controller's instruction. The script for transmitter implementation is in Figure 4.2.

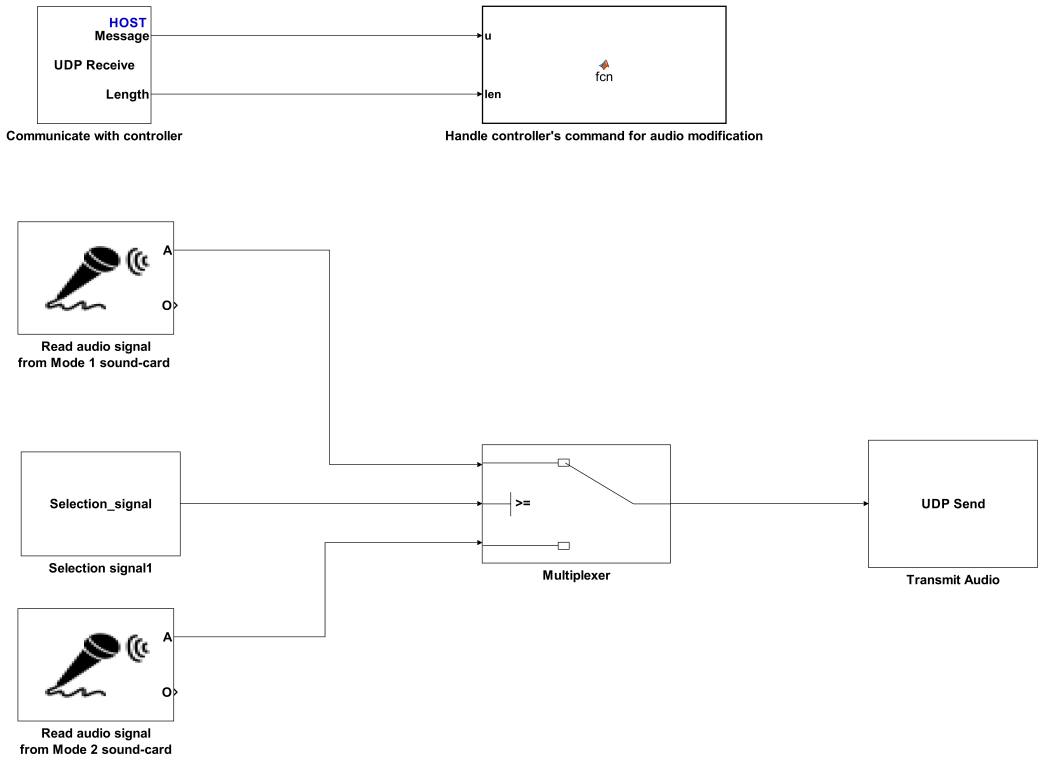


Figure 4.2: Transmitter Block Diagram

In Figure 4.2 we can see transmitter's modular components. At first, the block called "Communicate with Controller" enables receiving commands from the SDN controller of our application. It supports UDP protocol and is used for receiving instructions from controller for audio processing modification when it is required. In case that the transmitter receives such a command, it should inform also the receiver. This is implemented in block "Handle controller's command for audio modification". One of the roles of this block is that it opens a UDP socket towards the receiver to inform that audio settings should be reconfigured. Also, transmitter's script contains two blocks that are used for capturing audio data from sound-card. These blocks are "Read audio signal from Mode 1 sound-card" and "Read audio signal from Mode 2 sound-card". We used two sound-cards to implement audio reconfiguration because Simulink does not allow configuring parameters in blocks that have direct communication with hardware and drivers. Thus, we used two sound-cards for transmitting in different modes regarding sample rate and frame size values. Each time one sound-card only is active and this is defined by the variable "Selection signal" that is input in Multiplexer block which takes two signals as input and based on selection signal it allows only one signal to pass. Selection signal changes value when controller decides audio reconfiguration. The finally selected input audio signal is driven

to block called "Transmit Audio" that sends id through a UDP socket to the client virtual host into Mininet.

Receiver component represents the entity that receives the audio flow from the application. This role could be either another musician that participates in the application or either the audience that enjoys the live Networked Music Performance. Similar with transmitter, receiver contains Application Audio Module and Application Network Module. Application Audio Module is responsible for receiving audio flows from one or more transmitters and drive audio packets to the sound-card's speakers. Also, Application Audio Module receives from controller's SIP Module information for initial handshake between application and SDN controller and in case that audio processing modification is required. Finally, Application Audio Module evaluates the delay of receiver's sound-card and sends this measurement to the controller. Like transmitter's block diagram, the corresponding block diagram for Receiver role is depicted in Figures 4.3 and 4.4.

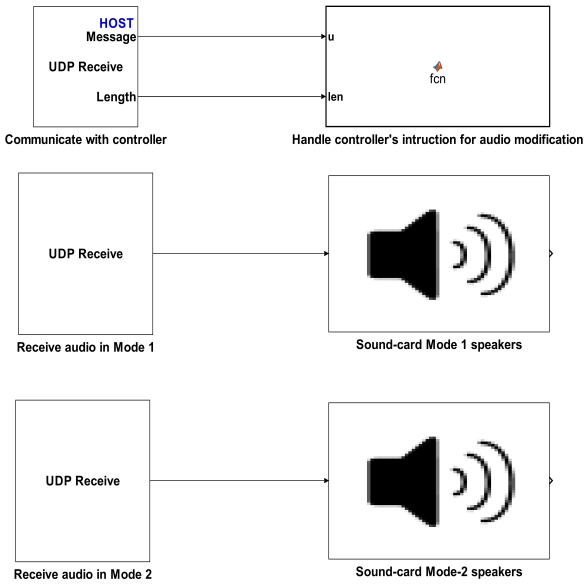


Figure 4.3: Receiving audio packets from network

Receiver initially receives audio packets from network. As we described in transmitter's case, packets are sent with different frame size values based on audio configuration settings that are used and whether audio processing modification is decided or not. So, similar with transmitter's role, receiver can receive using UDP sockets in two different modes due to the audio settings used. Reception through the network is implemented by "Read received audio signal from Mode 1 sound-card" and "Read received audio signal from Mode 2 sound-card" blocks in Figure 4.3. For supporting two different modes in audio settings, we used two pairs of sound-cards for playing received audio signal and also two sound-cards for acquiring initial that is sent through the network. Initial audio

signal is driven with cable from audio source to the receiver sound-cards. In our experimental setup (Figure 5.1), an MP3 device is used as audio source. The second set of sound-cards are named as "Read initial audio signal from Mode 1 sound-card" and "Read initial audio signal from Mode 2 sound-card" 4.4. Scripts depicted in Figures 4.3 and 4.4 are complementary scripts running parallel in receiver side. Initial signal is used to evaluate end-to-end delay for transmission between transmitter and receiver nodes. Received from network signal is played through sound-card's speakers. Speaker port in each sound-card is connected via an audio Jack cable to the microphone port. The signal is acquired via "Read received audio signal from Mode 1 sound-card" and "Read received audio signal from Mode 2 sound-card". According to the audio configuration mode that is used, we group each pair of sound-cards in transmitter and receiver in Mode 1 or in Mode 2. This separation is used to evaluate the delay between the initial and finally received signal. Delay evaluation is implemented through "Find Delay" clock that finds the delay in samples between two signals by evaluating the cross correlation between the two signals. The final value of delay is defined by a selection signal that controls a Multiplexer with the two values as input and according to selection signal's value one of the two is passed to output. Then, by dividing the delay value with current sample rate value we convert delay in seconds. In real world, this way of delay evaluation is impossible because we cannot have the initial signal to compare it with the received. Thus, delay evaluation is implemented using Equation 2.2 on SDN controller's side.

Network represents the network infrastructure that the application uses to transmit audio data from source (transmitter) to destination. Network consists of switches that support OpenFlow protocol. All switches are orchestrated by the SDN controller, that is implemented in SDN module. Network might have a well-known topology (single, tree, linear) or can be any possible custom topology. For network emulation purposes, Mininet described in 2.5 is used. The method in order to pass traffic through Mininet is analyzed in 4.12.

SDN controller is the main entity in the architecture schema. The functionality of the implemented SDN controller differs from the conventional duties of SDN controllers. Our proposed SDN controller contains three basic modules. The first module is SDN Module. It orchestrates network behavior my exchanging OpenFlow messages with switches, as described in section 2.3. In the implemented schema, POX controller described in 2.4 is used. The controller at first discovers the network topology. This is feasible by using the POX Discovery Module [95]. This module sends LLDP packets to all OpenFlow network switches and by examining their replies, it has a global view of the network. After discovering network topology, SDN Module examines the network topology and evaluates a number of link-disjoint paths between source and destination host. The controller receives as input the required number of disjoint paths. For instance, if the number of link-disjoint paths is equal to 3, SDN module looks for 3 link-disjoint paths between source and destination. This parameter is crucial because it increases path redundancy for connectivity between two hosts in case that a link fails during the process. This is a kind of multi-path

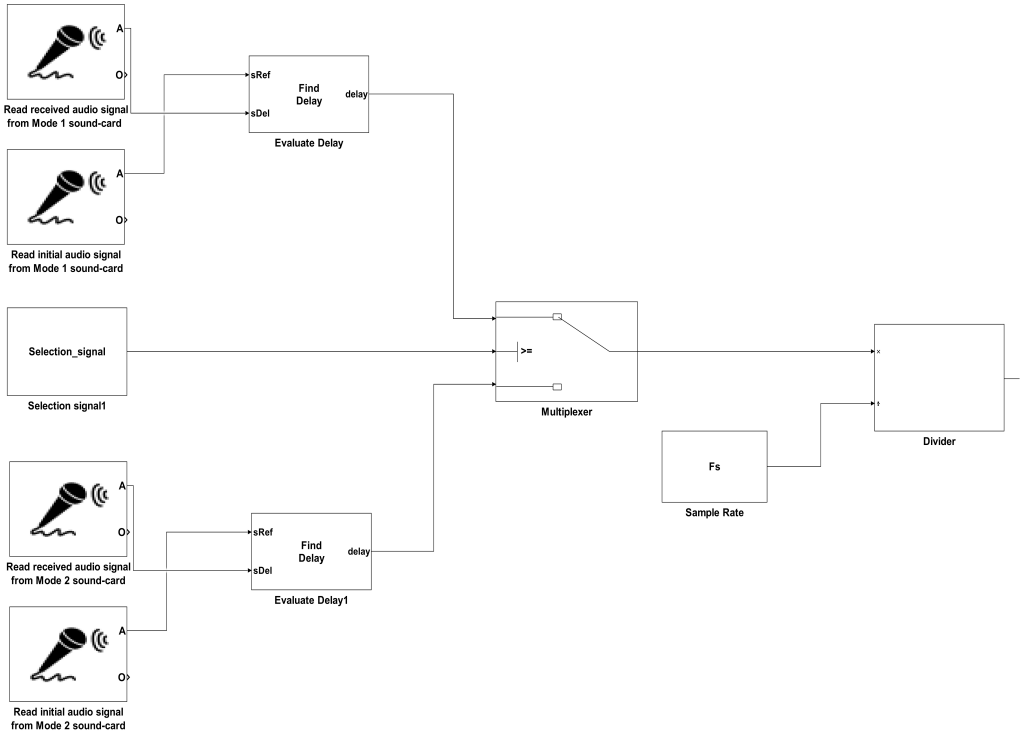


Figure 4.4: Evaluating delay in receiver side

routing that guarantees that the audio flows have nackup path and are not affected by link failure. Path evaluation is achieved by using python-package NetworkX [96]. This package represents the network topology as a graph and this allows path estimation between nodes in a simple way. After path evaluation, SDN Module installs flow rules for audio relaying between transmitter and receiver.

SDN controller is also equipped with a SIP Module, that allows interaction between controller and application. This module supports the new functionality that our implementation introduces which is the close interaction between application and network. SIP module initially is responsible for receiving and replying application requests for path assignment. In more details, SIP module receives the request and informs SDN module to evaluate the required number of link-disjoint paths and choose the path that satisfies the delay and jitter requirements. When the path is selected, SDN module informs SIP module and the reply is forwarded to the demanding application. As the streaming process continues, all available paths for transmission are monitored and in case of link failure or traffic congestion, audio flows are rerouted to better path. In case that all paths are congested and no one satisfies delay and jitter requirements, SIP Module informs the Application Audio Module of the transmitter to change audio processing pattern and also informs the corresponding Application Audio Module in receiver's side.

The last module that is contained into the proposed SDN controller is called Network Monitoring Module. This module is responsible for monitoring the delay and jitter for each path in the network. This information is crucial for the SDN controller because based on this, SDN assigns and changes paths. Network Monitoring Module receives measurements from transmitter's Application Network Module.

After the short introduction to the implemented architecture, in the next sections will be described some basic operations that the previously described modules are assigned such as sound-card delay evaluation, network delay monitoring, rerouting process etc. The full experimental setup that was used is shown if Figure 5.1.

4.2 Sound-card's blocking delay

Sound-card latency describes the delay of the signal from entering in sound-card's input till exiting sound-card's speakers port. This delay appears during capturing and receiving process. Sound-card's blocking delay is a function of two fundamental audio parameters: frame size and sample rate. Term frame size describes the number of samples that sound-card sends to output in every access. Sample rate describes the number of samples that are collected per second.

For sound-card's audio latency evaluation, Mathwork's Simulink blocks are used. In our experimental setup, an audio source (MP3) and 2 PCI sound-cards were used. Audio source's speaker port is connected via a 3.5 mm Jack cable with the microphone port of the first sound-card. For acquiring data from sound-card's microphone port, Simulink's Audio Device Reader component is used [97]. The functionality of the block for acquiring data from sound-card is shown in Figure 4.5. As a short description, sound passes through microphone to sound-card device. In the next step, audio signal is converted to digital by an Analog to Digital Converter (ADC) and is stored in a buffer. Then driver reads from this buffer and by using Audio Device Reader Simulink block data is formatted to preferred data type and is sent to the output of the block for further processing.

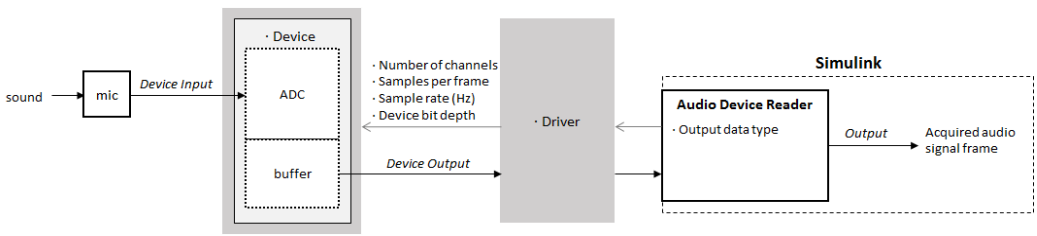


Figure 4.5: Audio Device Reader block diagram [97]

Audio signal is driven via Simulink software to speaker's output of the first sound-card using Audio Device Writer block [98]. Block's functionality is shown in Figure 4.6. This

block operates in the opposite way as Audio Device Reader block. It receives data as input into audio frames format, it sends audio data to audio driver and then to the sound-card's buffer. After this, Digital to Analog Converter converts data to analog and sends it to speakers.

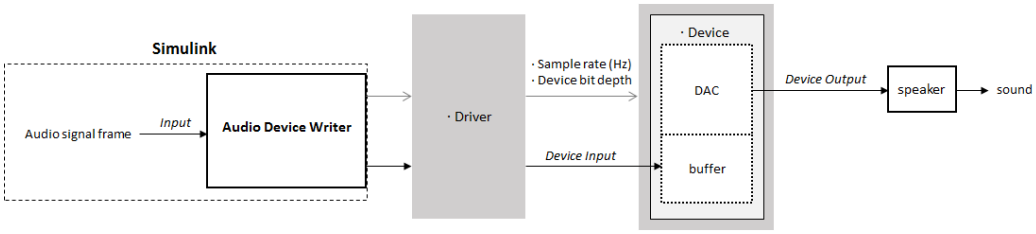


Figure 4.6: Audio Device Writer block diagram [98]

The speaker's port of the first sound-card is connected via a cable to the microphone port of the second sound-card. The goal of this setup is to evaluate the delay that the first sound-card inserts. The finally estimated delay for reading and recording audio in first sound-card is found by cross-correlation evaluation between the input signal of the first sound-card and the input signal of the second sound-card. Figure 4.7 describes the process for sound-card's blocking delay evaluation.

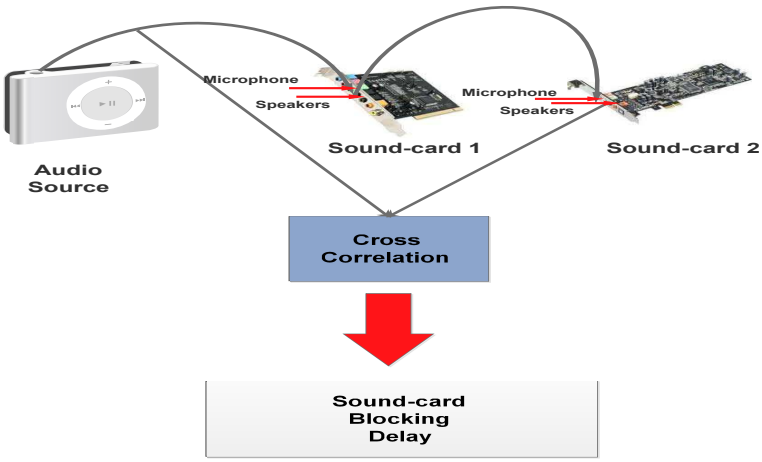


Figure 4.7: Blocking Delay Evaluation

Simulink script that implements sound-card's blocking delay evaluation is shown in Figure 4.8.

The described process is executed by Application Audio Module for each pair of transmitter-receiver that participate in audio transmission. After blocking delay evaluation has finished, the results are sent to the SDN controller's SIP module so given the network delay

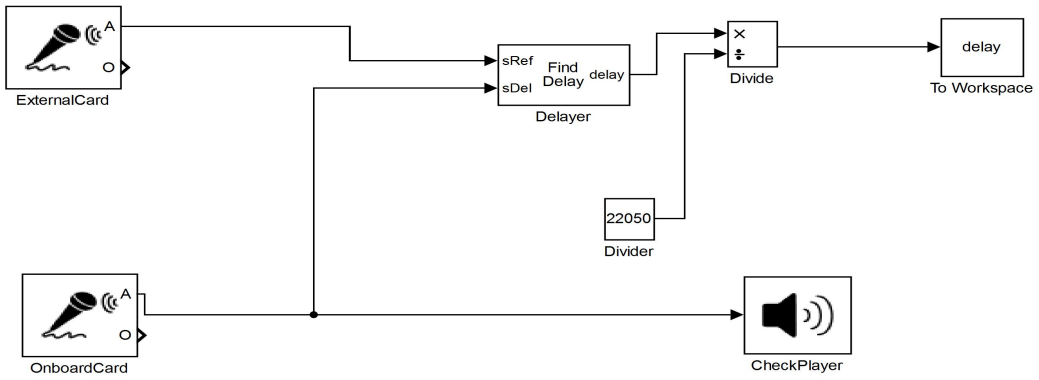


Figure 4.8: Delay evaluation

can estimate the mouth-to-ear delay from Equation 2.2. Application Audio Module in each node evaluates blocking delay for many combinations of sample rate and frame size values. Additionally, it executes this operation for a hundred times and evaluates the average blocking delay and via standard deviation the jitter of sound-card. Results from this analysis are kept in SIP module that creates a profile for the audio processing delay for each node. This information will be used in case that audio processing modification will be required to overcome network congestion problem by selecting sample rate-frame size values with lower blocking delay results. Figure 5.2 describes how frame size and sample rate affects sound-card's latency used in experimental setup.

4.3 Network delay

Equation 2.2 formulates the end-to-end delay evaluation method as a function of network delay and audio processing delay. In 4.2 we described the process that evaluates sound-card's blocking delay in each node for various combinations of frame size and sample rate values. This information is kept in SIP Module in SDN controller side and is used when a path request arrives or network delay is high and audio modification is required.

In 4.1 we mention that for audio transmission User Datagram Protocol (UDP) is used. UDP is quicker than TCP but it's more unreliable because no acknowledgment mechanism exists. In many approaches in described in Chapter 3, SDN controller use statistics about delay, jitter, flow stats, port stats for network monitoring. This information is acquired by sending requests to the OpenFlow switches about their stats. OpenFlow allows such process to estimate network performance. The frequency of these requests in related work is called polling period. As we described, small polling period values can offer real time network monitoring but also can cause traffic congestion to OpenFlow switches since they have to reply repeatedly to requests for stats to SDN controller. On the other hand, large values of polling period do not create computational overhead to the switches but this also does not offer accuracy in network monitoring. Generally, because

OpenFlow switches are software switches with limited computational resources, it is not recommended to avoid assigning them network monitoring duties.

In our approach, switches are not involved in network monitoring process. The complexity for network delay evaluation is assigned to the end hosts. In more details, if there are k-paths evaluated between two nodes and SDN controller wants to have real time view of the delay in these paths, one end host starts simultaneously sending UDP packets with specific format over each path to the destination node. Specific format means that each packet carries as data an id that characterizes the path that follows. Transmitter sends periodically one packet per second and waits for the reply. The time space between transmission timestamp and reception timestamp equals to the Round Trip Time (RTT). Transmitter evaluates running mean for RTT values among a number of recent measurements described by a window and jitter as the standard deviation of the values within this window. These statistics are sent to the SDN controller for real time network monitoring. Parallel transmission of these packets is feasible using multi-threaded programming.

Because UDP does not provide acknowledgment mechanisms, in the implemented approach, if a packet is not replied within a particular time space defined by a maximum threshold, we regard it as lost and we regard as delay a maximum value. This method is similar with traditional ping process, but it is threaded, fully configurable and also follows the same transmission mechanism with audio transmission. The proposed method for network delay evaluation is shown in Figure 4.9.

This threaded process is executed for each path. Transmitter keeps all measurements and evaluates a running mean value for having an accurate real time view of network's performance.

4.4 Application-Network communication

As described in 4.1, in this implemented architecture, application and network side cooperate during Networked Music Performance event. Initially, application via transmitter's Application Audio Module requests a path from SDN controller with certain delay and jitter requirements for transmission process. The SDN controller receives this application request and checks via Network Monitoring Module for the path that best matches to these requirements. In case that many paths satisfy the requirements, Network Monitoring Module evaluates the Euclidean distance between the pair of required delay and jitter values and every available path. The finally selected path is the one with the minimum distance from the required values. If no path satisfies the requirements, the Network Monitoring Module finds the best effort path. In any case, SDN controller informs via SIP Module the transmitter for the selected path to start audio streaming process.

As the streaming process goes on, Network Monitoring Module monitors all available

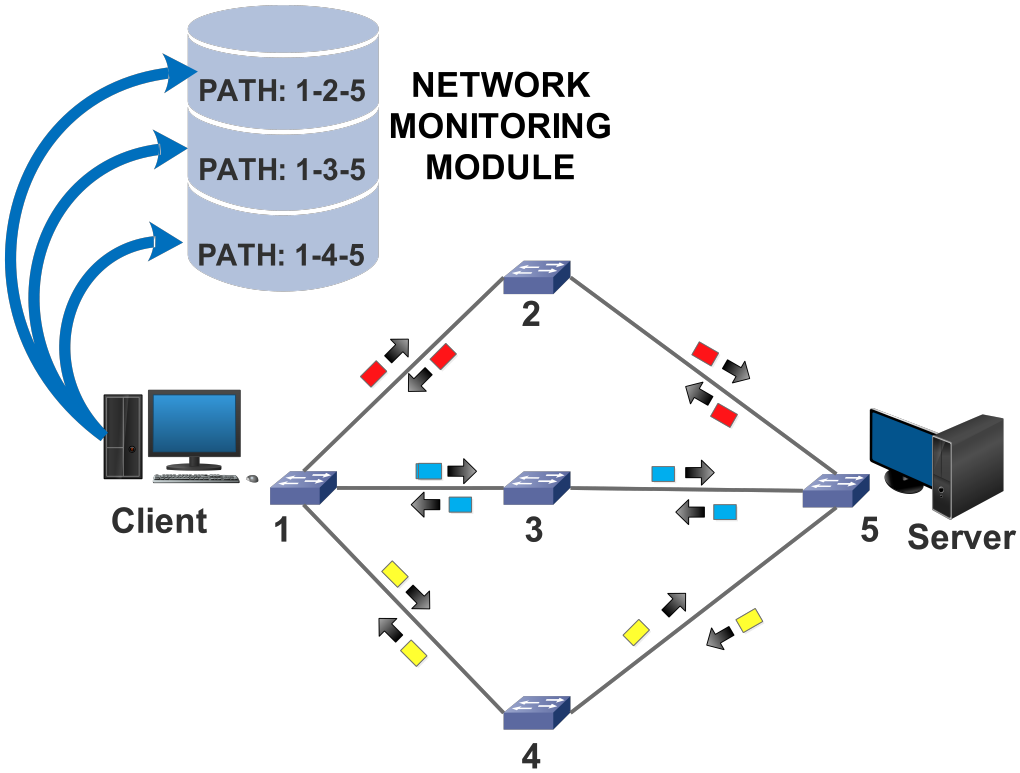


Figure 4.9: Proposed network delay evaluation

paths following the process described in 4.3. In case that a better path is found, SDN Module reroutes the audio flow to the path with less delay. In our implementation, a threshold delay value is defined to consider a path as better. If the difference between the currently used path and another path is greater than the threshold value, we consider it as better. So, in mathematical form, if A=$\{P_1, P_2, \ldots, P_n\}$ the set of n available paths with delay values D=$\{d_1, d_2, \ldots, d_n\}$ and P_u the path that is currently used with delay equal to d_u , the criterion for rerouting decision is formulated:

$$\forall d_k \in \ D \ evaluate \ diff(k) = d_u - d_k. \tag{4.1}$$

If any of diff(k) is greater than threshold value, rerouting is decided. If there are many paths whose delay difference from current path delay is greater than the threshold value, new path is the one which minimizes Euclidean distance. The above process is repeated until there is at least one path satisfying delay and jitter requirements. If there is no such a path, SDN controller informs the Application Audio Module of the transmitter and receiver to modify their audio processing pattern. This means to use a sample rate and frame combination with less audio processing delay. This will be decided based on the measurements that described in 4.2. SDN controller has a full view of each participant's sound-card performance through audio profiling for each participating node. The interac-

tion scenario between application and controller is shown in Figure 4.10. The flowchart for network delay monitoring, rerouting and audio processing modification decision is depicted in Figure 4.11.

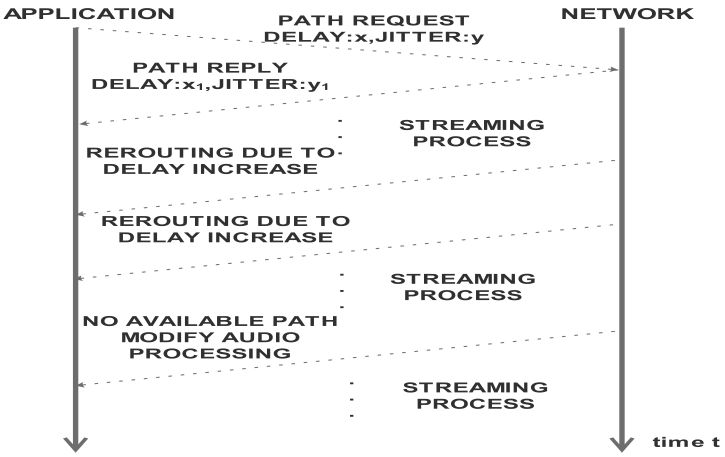


Figure 4.10: Application-Network communication

4.5 From Real World To Mininet

For a Networked Music performance, apart from the two nodes that one transmits audio to the other, also network connectivity is required. In real world, Internet plays this role. Millions of users per second send or receive data through the Internet. For the proposed architecture, the required network connectivity should support Software Defined Networking technology. In section 2.5, Mininet network emulator is introduced. Mininet allows emulating custom network topology that consists of hosts, switches and routers. It allows data transmission between hosts like being connected in real Internet and is pre-installed in a virtual machine and runs in a real computer. Mininet also is SDN-supported so emulated network topology can be manipulated by a SDN controller running in the same or in a remote computer with Mininet virtual machine. On the other hand, in a Networked Music Performance event, audio data captured from sound-card has to be transmitted through the network, received and played in receiver's side. Mininet allows network emulation but it does not fore audio hardware emulation. So, for the experimental setup, audio data captured from sound-card in one pc should travel through a topology emulated in Mininet and finally received and played by another real pc.

Passing traffic from a real pc into Mininet topology and receiving it to another pc requires communication between the real computers and the virtual hosts. Suppose two real computers that participate in a NMP event. The emulated topology in Mininet consists of five switches named as S_1, S_2, S_3, S_4, S_5. Real computer cannot pass traffic to the

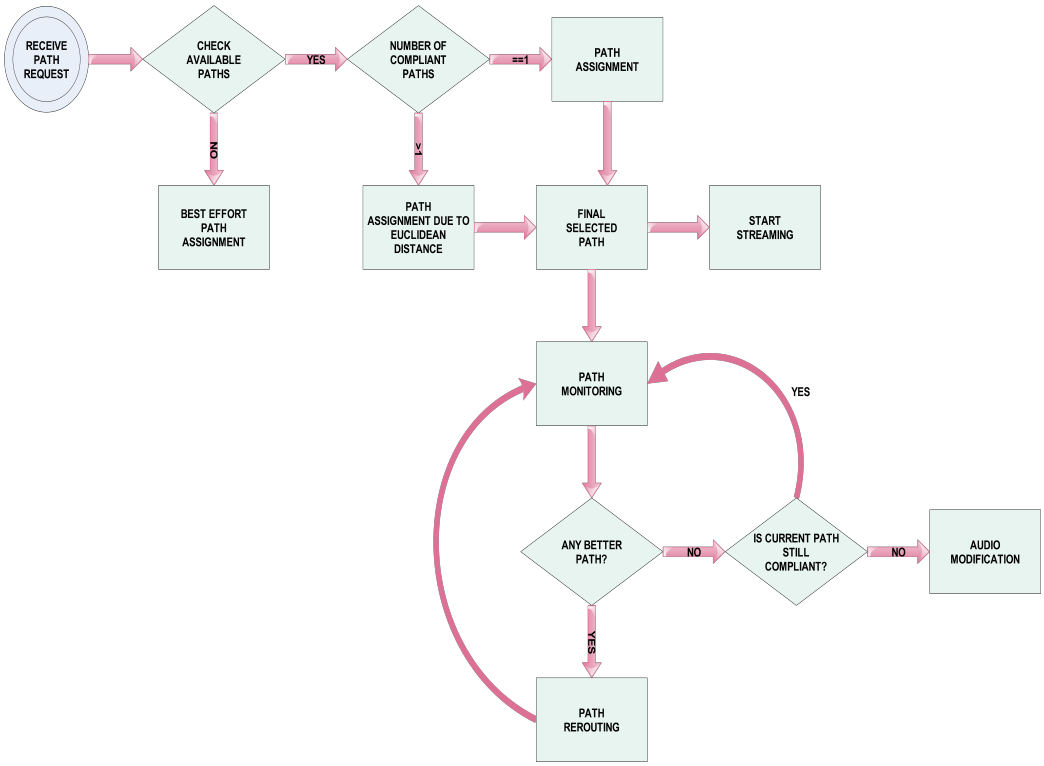


Figure 4.11: Network delay monitoring, rerouting and audio modification flowchart

emulated topology in Mininet because there is not a physical link between them. For this reason, two virtual hosts are added in Mininet topology. These virtual hosts act as relay servers. They are named as client and server. Client host should receive audio data from real transmitter computer, send it through Mininet topology and virtual host server receives audio data sent from client and forwards them to receiver computer. So, transmitter should be able to communicate with client and also server with the receiver's computer.

As described in [99], one solution to pass real traffic to Mininet emulated topology is to attach the IP address of the virtual machine that Mininet is running on to a switch participating in the topology. In more details, if eth0 is the network interface of Mininet virtual machine, the first step is to remove ip configuration from interface eth0. Then, IP address that eth0 previously had should me attached to a switch in the topology. This way transforms a switch to a gateway enabling communication between real and emulated world. Instead of using a topology switch as a gateway switch, we used an extra switch called gateway switch that only allows communication between transmitter-client and server-receiver. This switch is not evolved in path evaluation by controller's SDN Module. This complementary switch is directly connected with client and server and due to this, each of them has two network interfaces: one for communication with real world and one for communication through Mininet topology. Gateway switch is also an Open-

Flow switch but SDN Module does not install flow rules for this switch since rules for enabling communication are installed using ovs-ofctl tool [100]. The described topology is shown in Figure 4.12.

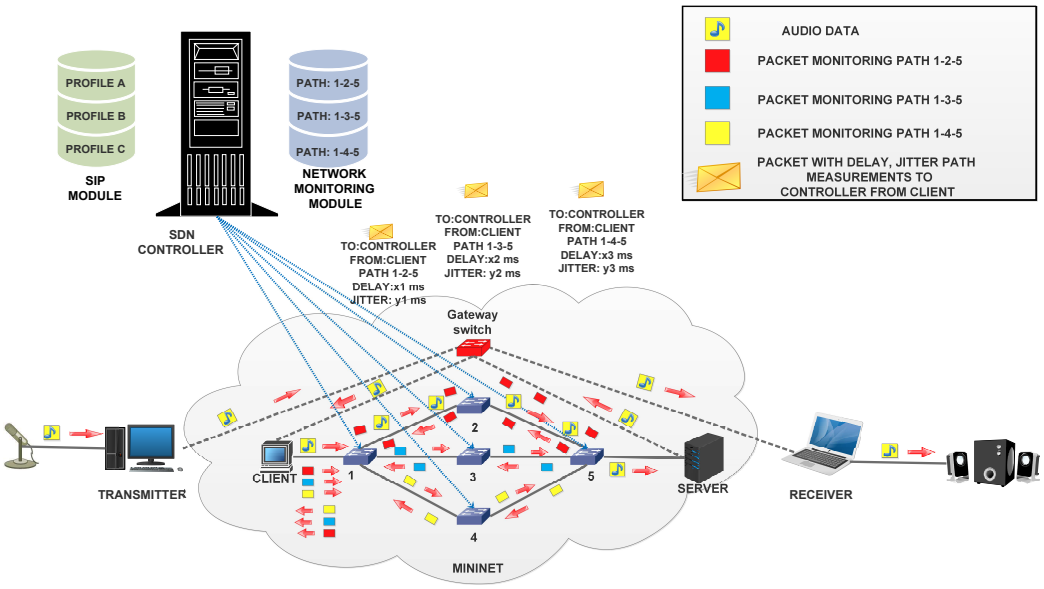


Figure 4.12: Communication between real and virtual hosts

In addition to audio transmission through Mininet network, SDN controller should have a view about delay and jitter path in each path of the topology. In 4.3 we described the mechanism for delay and jitter evaluation over a path. The difference between the transmission and reception timestamps equals to the current delay value of the path. SDN controller should receive this information and decide the path audio flows should follow to minimize network delay. So, client virtual host should send such delay-measuring packets over each path every second and inform the controller for the evaluated delay per path and simultaneously forward audio traffic sent from transmitter into Mininet. Also, server virtual host should act, apart from relay host that forwards audio to transmitter, also as echo server that replies to delay measuring packets client sends to it. Communication between client and SDN controller is feasible through gateway switch. So, messages with delay and jitter information shown in Figure 4.12 are sent from client through gateway switch to the SDN controller. When it receives these messages, Network Monitoring module stores this information and based on this information follows the process described in 4.11 decides whether rerouting event should be instructed or not.

4.6 A simple use case

In this section, a detailed description of a use case that our system supports will be analyzed. The example describes the case that two musicians join in our application and want

to perform a network music event. Each user uses audio hardware in order to process audio signal. Also, each user has SDN-supported network connectivity that consists of OVS switches that are orchestrated by the SDN controller.

As described in 4, the application assigns to each transmitter-receiver several-link disjoint paths. In this case, assume that the application has assigned three paths between transmitter and receiver. Suppose that set P is the set of assigned paths to this transmitter-receiver pair with P=$\{P_1, P_2, P_3\}$, with $P_1 = 1-2-5$, $P_2 = 1-3-5$ and $P_3 = 1-4-5$ denoting the paths of switches showed in Figure 4.13.

4.6.1 Audio profiling

The first step after joining in the application, is sound-card's blocking delay evaluation. SDN controller's SIP Module checks each musician's sound-card performance following the process described in 4.2. It tries different pairs of sample rate and frame size values and creates a kind of user profile containing this audio-based information. All audio profiles are stored in a data-base in controller's SIP Module. This profiling process is applied in both transmitter and receiver side. Profiling process is shown in Figure 4.13.

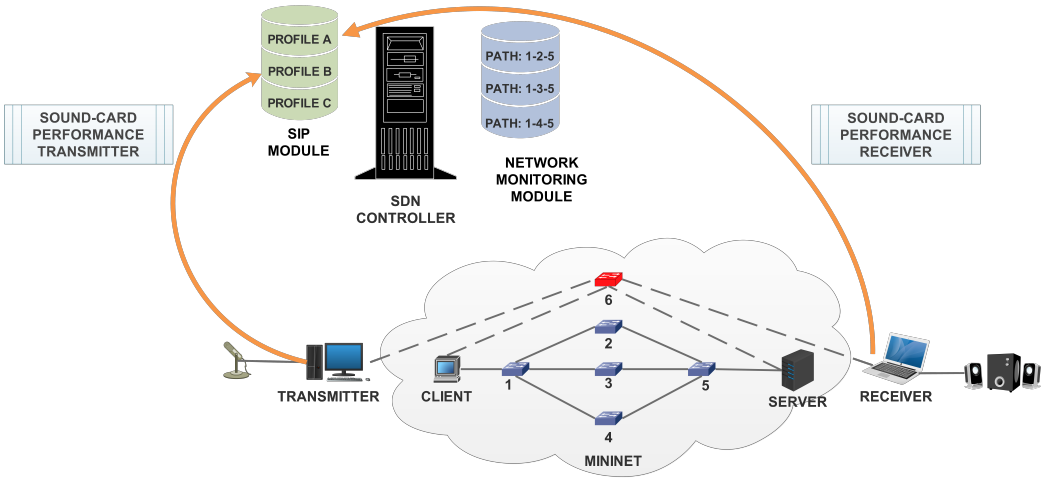


Figure 4.13: Sound-card audio profiling by SIP Module

4.6.2 Audio transmission

After Networked Music Performance application has created a profile for each participant, audio transmission is ready to start. Transmitter sends a path request to the controller requesting a path with certain delay and jitter requirements. SDN controller receives this request (Figure 4.14) and Network Monitoring Module selects the path that will be used for audio transmission.

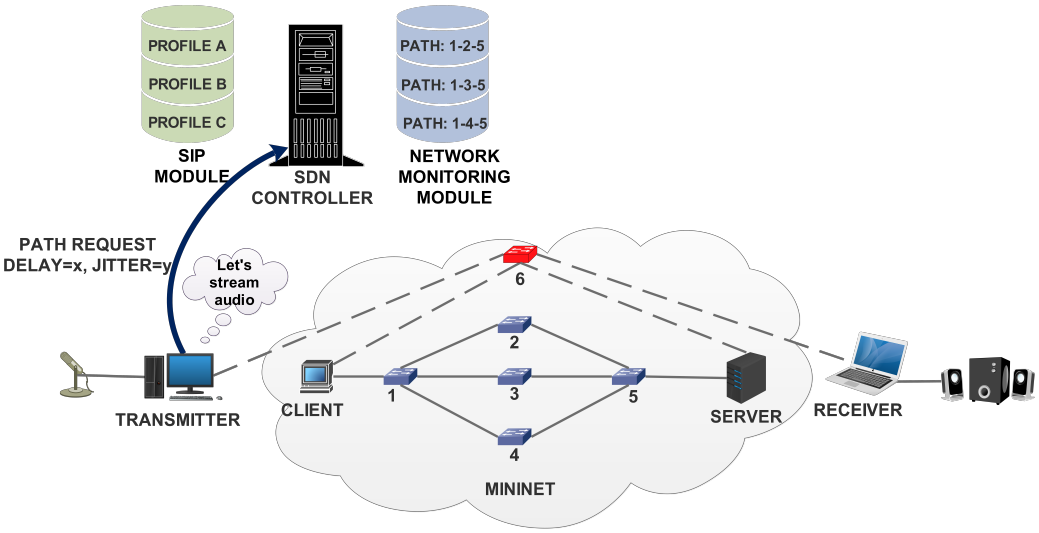


Figure 4.14: Path request by transmitter

SDN controller will use Network Monitoring Module to decide the path that will be used. The strategy for taking this decision is described in Section 4.3. Network Monitoring Module uses the process shown in Figure 4.9. Virtual client and server are used as hosts in Mininet topology to exchange packets between them for network delay evaluation in each path. These measurements are kept in a data-base in Network Monitoring Module in controller side. Network Monitoring Module checks the path requirements that transmitter sent and checks if available paths exist. If there are many available paths, it evaluates the Euclidean distance between delay and jitter values for each path and the required delay and jitter values. The path that has minimizes distance is finally selected for transmission. Assume, in our case that path $P_1 = 1 - 2 - 5$ is selected by Network Monitoring Module. SDN controller will inform the transmitter by sending a reply message (Figure 4.15. It also sends OpenFlow messages to switches that are involved into the selected path (switch 1, switch 2, switch 5) to install the corresponding flow rules in their forwarding tables for the selected path. Now, transmission is ready to start.

Transmitter receives controller's reply and starts audio transmission. It uses microphone to capture audio signal, and then audio signal is sent through the network using User Datagram Protocol. Audio packets should pass through Mininet topology according to the process described in section4.12. Client uses path $P_1 = 1 - 2 - 5$ for transmission and server receives these packets(Figure 4.16). Server forwards them to the receiver, who can listen live the audio signal sent. Network Monitoring Module, apart from replying to transmitter's path requests for path assignment is responsible for real-time monitoring all paths that are available for network connectivity between each pair of transmitter and receiver. So, in this example, Network Monitoring Module checks also the delay and jitter values of paths $P_2 = 1 - 3 - 5$ and $P_3 = 1 - 4 - 5$. As described in Figure 4.11, SDN

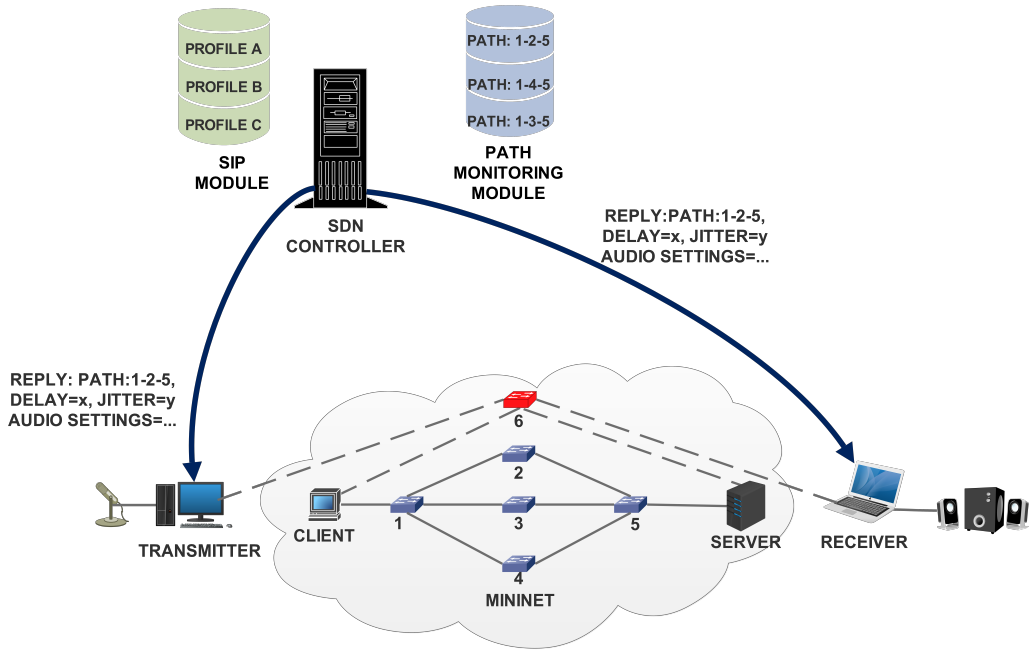


Figure 4.15: SDN controller's reply to path request

controller has defined a threshold value for deciding if audio flows should be rerouted to another path with less delay or not. It evaluates the differences in delay and jitter between the currently used bath and the other available paths. If there is at least one path whose delay is than the difference between currently used path and the threshold value, rerouting decision to this path is taken.

In real world, delay in links appears in cases of traffic congestion. When many users send data over the network simultaneously, in case data packets use the same paths, there is a great possibility for links in the path to be congested. Traffic congestion is a major problem for computer networks and if appears, Quality of Service is reduced due to network delay increase. So, traffic congestion cases should be avoided in delay sensitive services, especially in a NMP event where delay is an indicator for good performance. In Mininet, traffic congestion and delay increase cases can be emulated by using netem functionality [101]. Netem allows creating real world circumstances referring to delay, jitter and packet loss metrics. By using netem, user can add delay in an interface. This emulator is widely used in research community for testing protocols of wide area networks. So in our experiments, netem is used for creating or modifying delay over a link and check network's performance.

As audio transmission continues, Network Monitoring Module checks all available paths. We add delay using netem tool in the path $P_1 = 1 - 2 - 5$. While the difference between the delay of currently used path and the other available paths delay values remains below

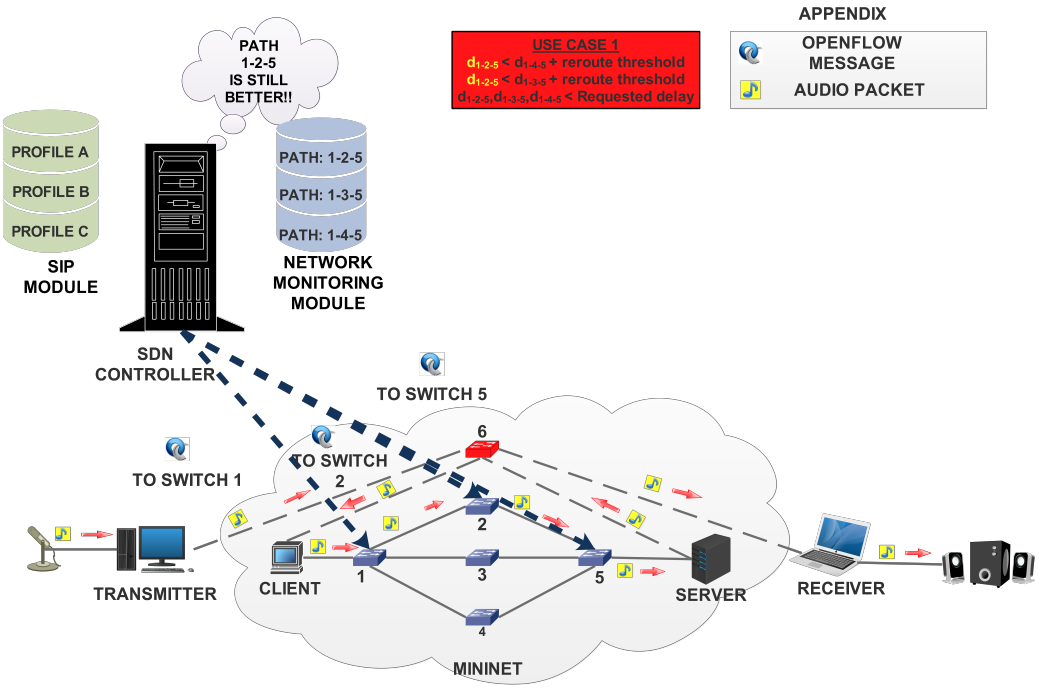


Figure 4.16: Audio transmission through network

the threshold value, no rerouting decision is taken. If difference in one path is greater than the threshold value, SDN instructs the audio flows to follow this path. This happens by sending the OpenFlow messages to the appropriate switches that are involved in the new selected path. Assume that if delay of path P_2 is denoted as d_2 and delay of path P_3 is denoted as d_3. Also, path P_1 performs with delay equal to d_1. Required delay is denoted as d_d.

- If $d_d > d_1 > d_2 > d_3$
- If d_1-d_2 > threshold value
- If d_1-d_3 > threshold value

then SDN controller will decide that rerouting should be applied and path $P_2 = 1 - 3 - 5$ will be selected as new path for audio flows. Thus, SDN controller sends OpenFlow messages to switches 1, 3 and 5 for installing new flow rules. After flow rule installation, audio packets will travel over path $P_2 = 1 - 3 - 5$ (Figure 4.17). Audio data use path P_2 to reach transmitter. Applying again delay emulation with netem tool to P_2, like previous case, will redirect audio flows to another path. In more details

- If $d_d > d_1 > d_2 > d_3$
- If d_2-d_3 > threshold value

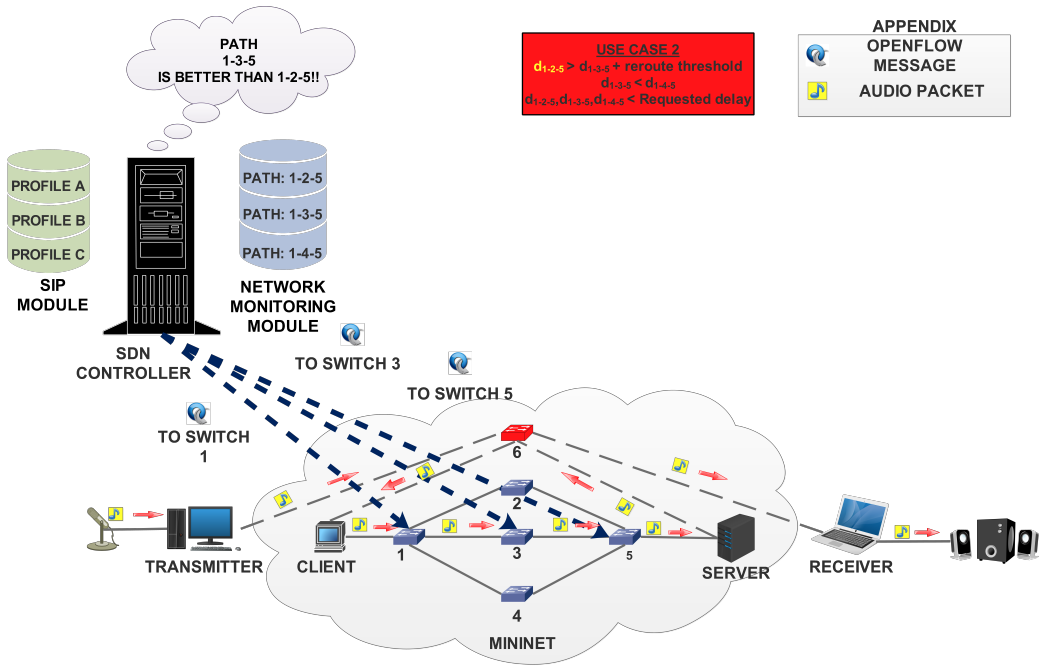


Figure 4.17: First rerouting decision to due to delay increase

- If d_2-d_1 > threshold value

then SDN controller will select path P_3 as next selected path. For this reason, SDN controller installs the corresponding flow rules to switches 1, 4, 5 and now path $P_3 = 1 - 4 - 5$ is used for audio transmission (Figure 4.18).

4.6.3 Audio processing modification

We can add add or modify delay value in every path. Suppose that we add in all available paths delay greater than the requested delay. This means that

- If d_d < d_1 , d_2 , d_3
- If d_3 < d_2
- If d_3 < d_2

If this happens and no path satisfying the path requirements exists, SDN controller informs the application to change audio processing profile in order to overcome network delay increase. This means that application should use an audio profile that performs less delay and jitter values and results in end-to-end delay below the Ensemble Performance Threshold. So SDN controller's SIP module looks for an audio profile that gives end-to-end delay below the delay constraint. When new profile is selected, SIP Module informs both transmitter and receiver side to change their audio processing parameters related to

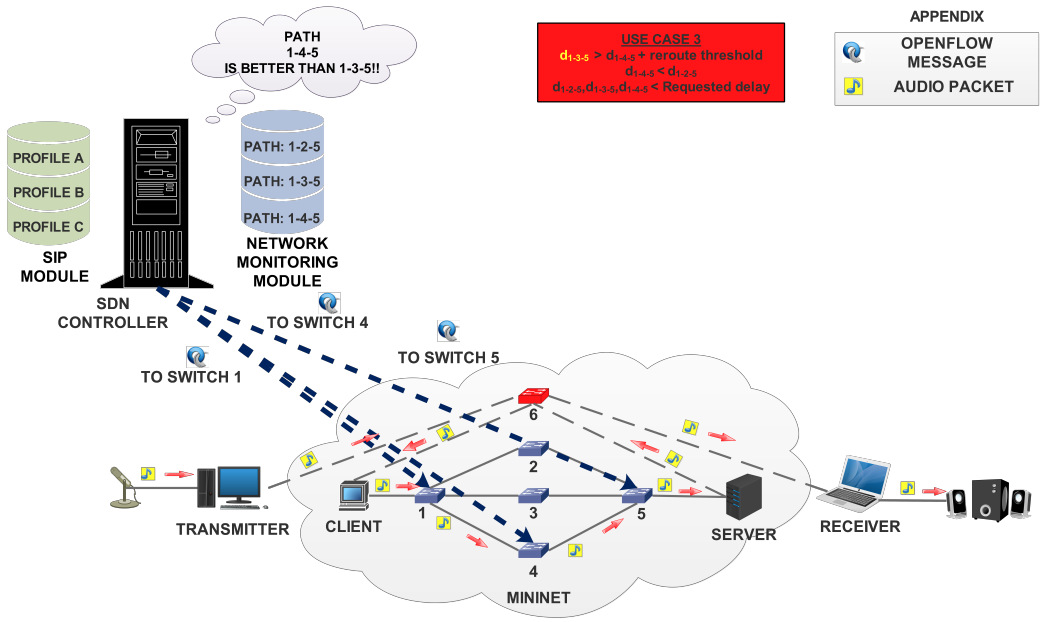


Figure 4.18: Second rerouting decision to due to delay increase

sample rate and frame size (Figure 4.19). This modification in audio configuration parameters may lead to bit-rate decrease and in worse quality. If network delay falls to affordable levels, our application returns to initial mode of audio configuration parameters.

This decision, apart from end-to-end delay decrease, may cause problem in sound's quality. This depends on the hardware and quality that each sound-card supports. In any case, if network delay increases more, and using the selected audio profile does not offer below-EPT end-to-end delay values, SDN controller can select a new profile and instructs transmitter and receiver to adopt it. In worst case, if network delay is very high, SDN controller will use all available profiles and will give the minimum possible end-to-end delay to application users.

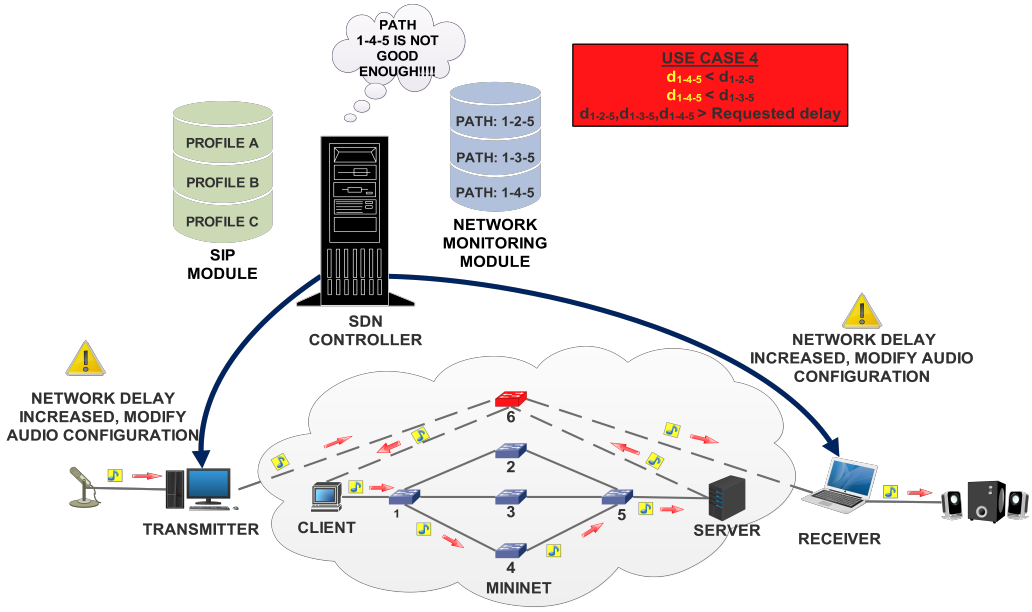


Figure 4.19: Audio processing modification decision

Chapter 5

Evaluation

In this chapter, we will introduce and analyze the results that came from our proposed architecture. Based on these results we will analyze the performance of the implemented architecture and factors that affect it's performance.

5.1 Experimental setup

For our experiments, 2 real computers were used. In each computer also a virtual machine was running in order to represent the four modular components of the architecture. As we described in section 4.1, these components are:

- Transmitter
- Receiver
- SDN-supported network
- SDN controller

We also used four sound-cards to be able to switch between two audio configuration sets in case that audio modification is required by the system. Each pair of the four sound-cards supports different audio configuration mode. The full view of experimental setup is shown in Figure 5.1.

Mininet runs in a virtual machine in transmitter computer and emulates network topology. SDN controller is running on receiver computer in a virtual machine. For network connectivity between the two real computers a Linksys router is used.

During experimental process, audio source is playing music. Audio play is driven with audio Jack cable to transmitter computer where it is processed in two audio configuration modes. This audio signal is also driven to receiver computer where it is also processed in

Figure 5.1: Experimental setup

two different audio configuration modes for delay evaluation via cross correlation. When Mininet topology is created in transmitter's side, SDN controller which runs on receiver's computer connects and discovers the full network topology. It is important that the application is tested in many different network topology types without any problem. In Mininet topology, client and server virtual hosts start exchanging periodically packets and evaluate network delay in each path, as described in Figure 4.9.

Suppose that the two real computers, transmitter and receiver have recently join the SDN-supported Networked Music Performance application and want to transmit audio context using the application. SIP module in SDN controller checks each sound-card's performance in transmitter and receiver in various sample rate and frame size combinations and creates an audio profile for each user, kept in a database as described in Figure 4.7. These profiles will be used when the transmitter sends a path request to SDN controller for audio transmission.

Transmitter and receiver are regular users so for audio transmission three link-disjoint paths are assigned for connectivity between them. In case of premium users, the number of paths between transmitter and receiver is larger to increase connection redundancy. Transmitter sends a path request to the SDN controller for transmission to receiver in cer-

tain delay and jitter requirements (Figure 4.14). SDN controller checks the available paths for these users and based on the measurements that it has for network delay assigns the path that matches best to their demands (Figure 4.15) and informs transmitter and receiver for the path and also the audio settings that they can use for their transmission. This initial handshake between transmitter and SDN controller is implemented in MATLAB and Simulink scripts.

When transmitter reveives SDN controller's reply, it starts audio transmission through Mininet. Simultaneously, receiver can evaluate in real-time the delay by the cross-correlation evaluation between initial and received signal (Figure 4.4) and the SDN controller also by using the information about audio profiles for each user and network measurements that it has in Equation 2.3. During experiment, when a better path is found, SDN controller installs the corresponding flow rules to OpenFlow switches and it reroutes audio flows. When no path satisfies delay and jitter requirements, is informs transmitter and receiver to modify their audio modes. This is the summary of our application's functionality and in the next sections we will discuss the results of the experimental process.

5.2 Sound-card blocking delay

Following the method described in Figure 4.7, SIP Module of the SDN controller forms a table similar with 5.1 about sound-card's delay evaluation for each user of the application. It describes a part of user's audio profile in our application.

Frame Size (samples)	**Sample Rate (Hz)**	**Blocking Delay (ms)**
64	22050	2.290709520523000
128	22050	5.40756032663547
256	22050	11.8374644877110
512	22050	23.9535662749948
64	44100	1.05316310098900
128	44100	2.51817673929500
256	44100	5.07265410981300
512	44100	11.5306774050616

Table 5.1: Sample of user audio profile

We tried 22050, 44100, 48000, 192000 Hz for sampling rate values and 64, 128, 256, 512, 1024, 2048, 4096 samples as frame size values. Figure 5.2 depicts sound-card's performance in various sample rate and frame size combinations.

As we can see, blocking delay decreases as we decrease frame size and increase sample

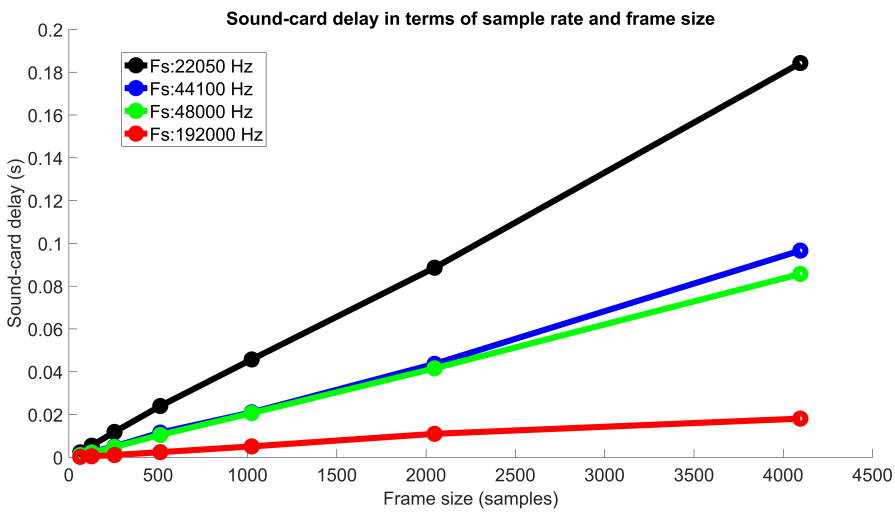


Figure 5.2: Measured sound-card's blocking delay in terms of frame size and sample rate

rate. A metric for empirical blocking delay evaluation is

$$blocking\ delay = \frac{frame\ size}{sample\ rate} \tag{5.1}$$

For instance, a combination of frame size equal to 64 samples and sample rate 22050 Hz is almost equal in delay terms to the combination for frame size equal to 128 samples and sample rate 44100 Hz. In practice, blocking delay equals

$$blocking\ delay = \frac{frame\ size}{sample\ rate} + hardware\ overhead \tag{5.2}$$

Hardware overhead depends on the quality of the sound-card and on settings that are recommended of each sound-card use. Expensive professional sound-cards perform the same in all sample rate and frame size combinations but conventional sound-cards depend especially on sample rate value that are used.

5.3 Path selection and rerouting decision

The main purpose of our implementation is to provide a mechanism that given audio settings of each user and network to provide communication between transmitter and receiver below the Ensemble Performance Threshold value. Thus, SDN controller monitors in real time network delay in each path and assigns the path with minimum delay to satisfy EPT constraint. Also, if during transmission better path is found, it dynamically reroutes audio flows to this path. In case that all paths are congested, it interacts with application side to modify audio configuration and absorb network delay increase. If this does not

result in below-EPT end-to-end-delay because network delay has is totally congested, our application provides the best-effort path for minimizing end-to-end delay.

Since SDN controller has information about audio profiles for each user, by using Equation 2.3, it can evaluate the maximum affordable network delay for audio transmission before interacting with application for audio settings reconfiguration. Figure 5.3 shows the analysis for the relation between blocking delay and network delay for each pair of frame size and sample rate values in order to have up-to EPT end-to-end delay.

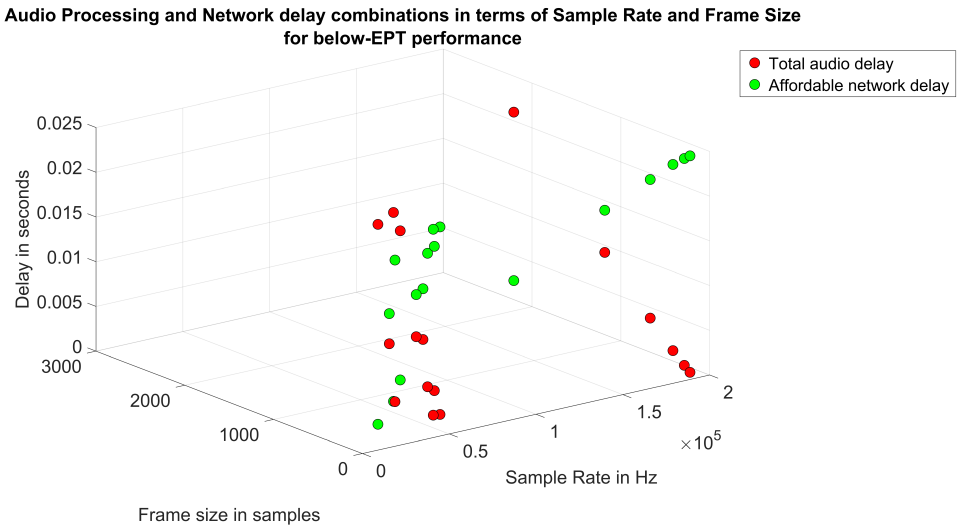


Figure 5.3: Maximum affordable network delay for each audio profile

We will perform the results of our implemented architecture through an example that we tried as use case for our system. In this example, we will present also some parts of the user interface from the system and also evaluate it's performance. Suppose that we have the topology shown in Figure 5.4.

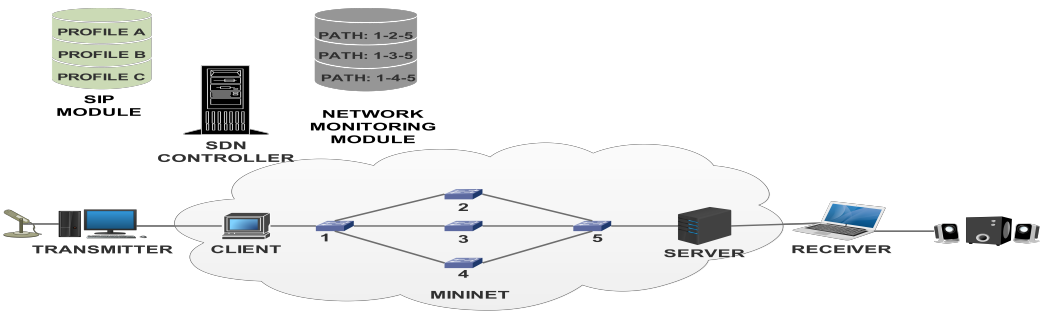


Figure 5.4: Example topology

SIP module has collected audio information from both transmitter and receiver, available

paths are monitored in real time and this information is stored in Network Monitoring Module. For network monitoring we used a window for 40 measurements for evaluating moving average delay and jitter.Suppose transmitter sends and application request to the SDN controller, as shown in Figure 5.5.

```
Please enter the requested delay requirements of the application in msec: 14.1849
Please enter the requested jitter requirements of the application in msec: 4
REQ_PATH>FROM:192.168.1.118,TO:192.168.1.119,WITH:DEL=14.1849,JIT=4
Initial handshake message from controller just received

offered_network_delay =

    0.3302


offered_network_jitter =

    1.3841

No audio modification is required
Default Audio configuration finished.......
Default configuration for transmission decided with sample rate equal to 22050 samples/s and frame size equal to 128 samples...
Also receiver side will be informed.
Now you can start streaming!!!!!!!
Your current path has 0.33018 msec delay and 1.3841 msec jitter
```

Figure 5.5: Application request screenshot from transmitter

SIP module in SDN controller receives application's request 5.6 and checks the available paths via Network Monitoring Module in order to select a path.

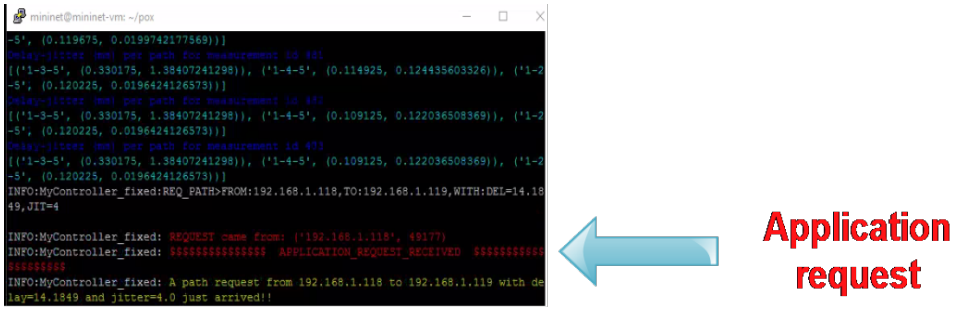


Figure 5.6: Application request screenshot from controller

SDN controller uses Euclidean distance and finally selects path 1-3-5 for audio transmission (Figure 5.7). It also informs transmitter and receiver (Figure 5.8) for selected path delay and jitter and also for the audio configuration that is recommended. The initial audio configuration set refers to sample rate equal to 22050 Hz and frame size equal to 128 samples (Mode 1). The alternative audio mode (Mode 2) that will be used refers to sample rate equal to 44100 Hz and frame size 128 samples. The aggregate (for both sound-cards in transmitter receiver) blocking delay is evaluated equal to 10.8 ms for Mode 1 and 2.1 ms in Mode 2. So the gain between the two modes is equal to 8.7 ms. This means that our system can absorb up to 8.7 ms increase in the network by re-configuring audio parameters.

Transmitter receives the SDN controller's reply and starts transmitting audio to receiver node. For testing the performance of our system, we used netem tool to add delay in each

Figure 5.7: Application reply screenshot from controller

```
For transmission start, frame size assigned to 128 samples and sample rate assigned to 22050

 samples/s
```

Figure 5.8: Application reply received in receiver side

interface and check system's behavior. Figure 5.9 shows the evolution of path selection mechanism. Threshold value for deciding rerouting to a new path is 2 ms.

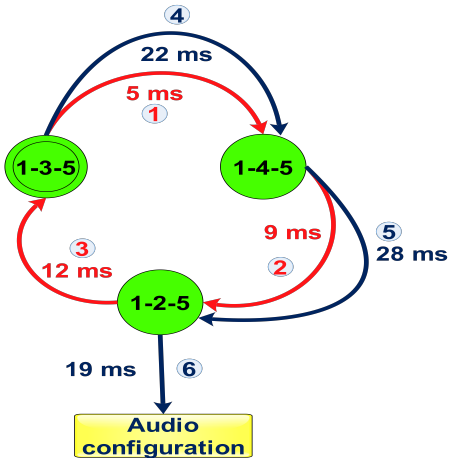


Figure 5.9: Selected path into Finite State Machine form

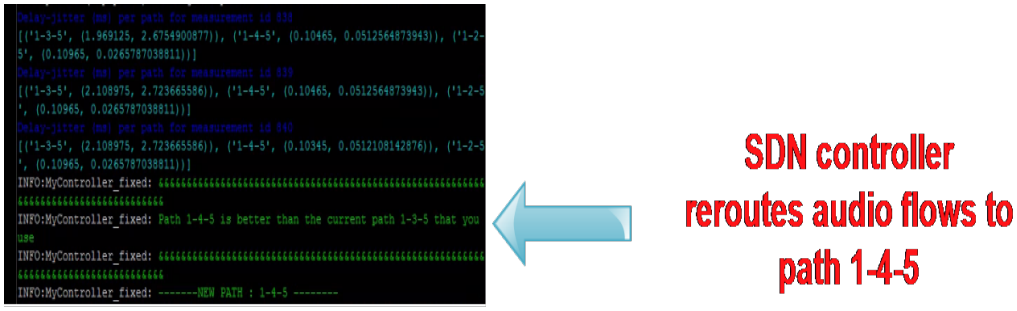
Above each arrow, the added delay value is shown. Numbers in the FSM show the sequence of the events. Table 5.2 shows the time in second each transmission from current to next path takes place.

In case for rerouting, SDN controller informs the user as shown in Figure 5.10.

When no available path satisfies EPT constraint, SDN controller informs both transmitter and receiver to modify their audio settings. Messages for audio configuration are shown in Figure 5.11 (SDN controller), in Figure 5.12 (transmitter) and in Figure 5.13 (receiver).

Time (sec)	Current state	Next state
161	-	1-3-5
280	1-3-5	1-4-5
319	1-4-5	1-2-5
377	1-2-5	1-3-5
446	1-3-5	1-4-5
493	1-4-5	1-2-5
564	1-2-5	Audio configuration

Table 5.2: Transmission schedule



```
Delay-jitter (ms) per path for measurement id 838
[('1-3-5', (1.969125, 2.6754900877)), ('1-4-5', (0.10465, 0.0512564873943)), ('1-2-
5', (0.10965, 0.0265787038811))]
Delay-jitter (ms) per path for measurement id 839
[('1-3-5', (2.108975, 2.723665586)), ('1-4-5', (0.10465, 0.0512564873943)), ('1-2-5
', (0.10965, 0.0265787038811))]
Delay-jitter (ms) per path for measurement id 840
[('1-3-5', (2.108975, 2.723665586)), ('1-4-5', (0.10345, 0.0512108142876)), ('1-2-5
', (0.10965, 0.0265787038811))]
INFO:MyController_fixed: &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&
INFO:MyController_fixed: Path 1-4-5 is better than the current path 1-3-5 that you
use
INFO:MyController_fixed: &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&
INFO:MyController_fixed: -------NEW PATH : 1-4-5 --------
```

Figure 5.10: Rerouting example

Figure 5.14 shows the delay in each of the three link disjoint paths in out topology. Also, all rerouting events and audio configuration decision are depicted in the same figure. We can see that if transmitter uses Mode 1 for transmission which gives 10.8 ms aggregate blocking delay for both sound-cards, the maximum affordable network delay that allows end-to-end delay up to EPT value is 25-10.8151=14.1849 ms. When all paths are above this value, SDN controller instructs transmitter and receiver to switch to audio Mode 2. In Figure 5.14 we can see that after 780 seconds the best available path has delay equal to 19-20 ms which is greater than the maximum affordable network delay.

5.4 Results

Figure 5.16 shows the fundamental metric that defines the performance of our system, which is end-to-end delay. We can see in details the exact value of network delay per second and end-to-end delay. It's worthwhile to mention that the novel element of this thesis, which is close communication between network and application for below EPT end-to-end delay, is also confirmed, since audio configuration decreases end-to-end delay. Also, by comparing (Figure 5.15) delay estimated by SDN controller via Equation 2.3 and end-to-end delay evaluated in receiver, we can see that they are almost the same. This is critical because in real world it is impossible to have a copy from initial signal for delay evaluation via cross correlation method. So, using Equation 2.3 allows estimating

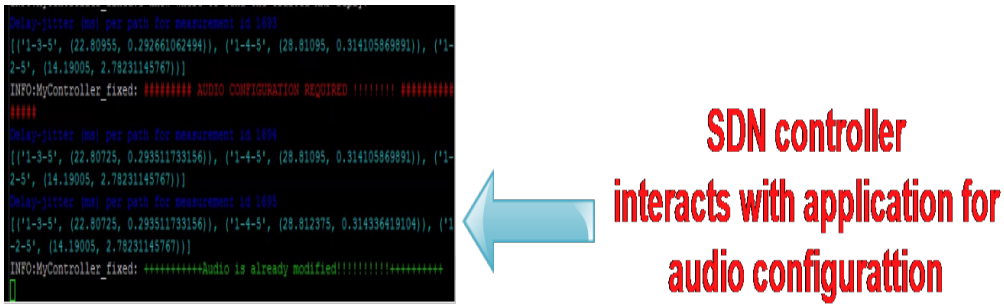


Figure 5.11: Network-application communication example

```
MOD AUDIO COMMAND RECEIVED FROM CONTROLLER!!!!!!
Sample rate changed to 44100 samples/s from 22050 after controller MOD_AUDIO command
Frame size changed to 64 samples from 128 after controller MOD AUDIO command
```

Figure 5.12: SDN controller informs transmitter for audio configuration

end-to-end delay in NMP systems given blocking delay and network delay.

Purple line shows the end-to-end delay without communication between network and application and red line shows the end-to-end delay after our method is applied. We can see that in first case the end-to-end delay is about 30.6008 ms and due to communication between network and application this is reduced to 21.8920 ms, which is accepted due to EPT constraint and offers remarkable decrease by 8.71 ms. Network delay reaches to 19.7857 ms which is 19.7857-14.1849= 5.6008 ms over the maximum affordable network delay that allowed audio Mode 1. Our application can afford still 25-21.8920=3.1080 ms without violating EPT constraint. The gain in delay terms that interaction between transmitter and receiver offered in percentage scale is evaluated as:

$$gain = \frac{end - to - end\ delay\ mode1 - end - to - end\ delay\ mode2}{end - to - end\ delay\ mode1} * 100\% \quad (5.3)$$

Also we can define gain metric referring to blocking delay modification and is evaluated as

$$gain_{audio} = \frac{total\ blocking\ delay\ mode1 - total\ blocking\ delay\ mode2}{total\ blocking\ delay\ mode1} * 100\% \quad (5.4)$$

Gain value in Equation 5.3 shows the degree that end-to-end delay is reduced compared to the case that no audio modification was decided. On the other hand, gain value from Equation 5.4 shows the degree that only blocking delay is reduced due to audio modification in a mode with reduced audio latency. In simulation experiments, we tried various combinations of audio configuration parameters. For instance, in case that audio mode 1

```
MOD_AUDIO:
Sample rate  changed to 44100 samples/s from 22050 after controller MOD_AUDIO command
Frame size changed to 64 samples from 128 after controller MOD_AUDIO command
```

Figure 5.13: SDN controller informs receiver for audio configuration

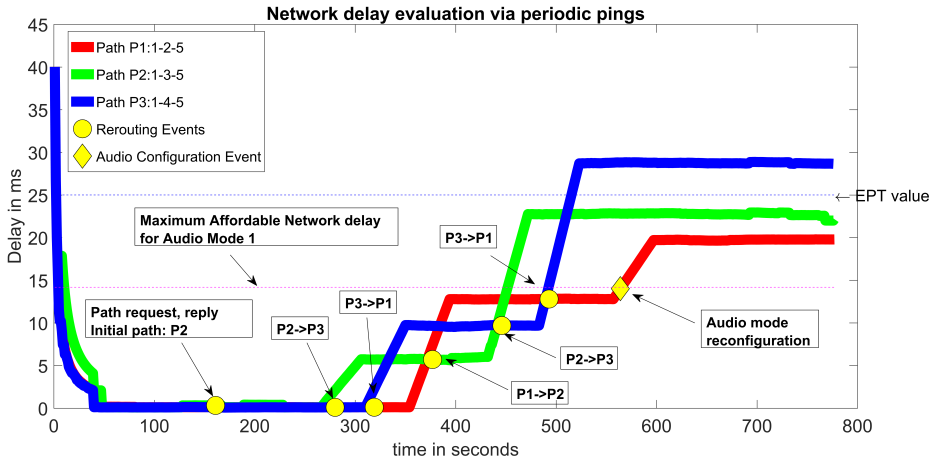


Figure 5.14: Network delay in each path of the topology

refers to sample rate equal to 44100 Hz and frame size equal to 512 samples and mode 2 can be either with sample rate equal to 44100 Hz and frame size equal to 64 samples (case 1), 128 samples (case 2) or 256 samples (case 3) the gain referring to end-to-end delay and blocking delay in each case is shown in Table 5.3. For this simulations, we increased network delay 0.08 ms per second and we consider that network delay is equal to 1 second for time space until t=10 seconds . Figure 5.17 shows delay evolution for each audio configuration mode. Audio configuration is decided at t=21 seconds. Values for blocking delay in each audio mode are described in Table 5.1.

Sample rate (Hz)	**Frame size (samples)**	**Gain percentage (%)**	**Gain audio percentage(%)**
44100	64	59.29	90.8
44100	128	51	78.16
44100	256	36.546	56

Table 5.3: Gain percentage simulation results

In practice, gain percentage is case of switching between two audio modes is given by the equation 5.3 but end-to-end delay is equal to:

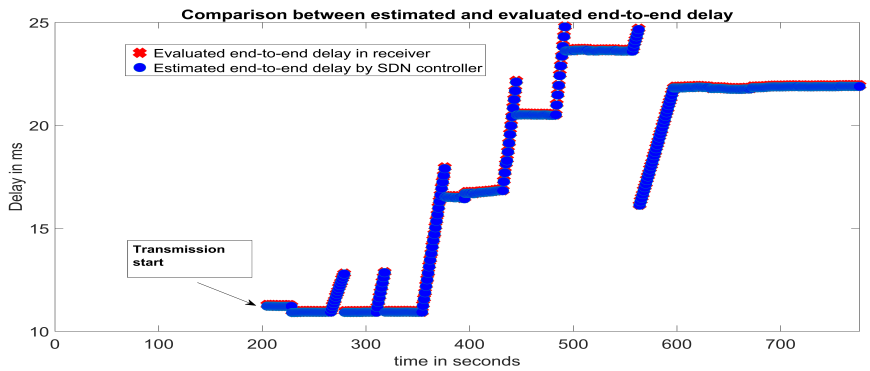


Figure 5.15: Comparison between evaluated and estimated end-to-end delay

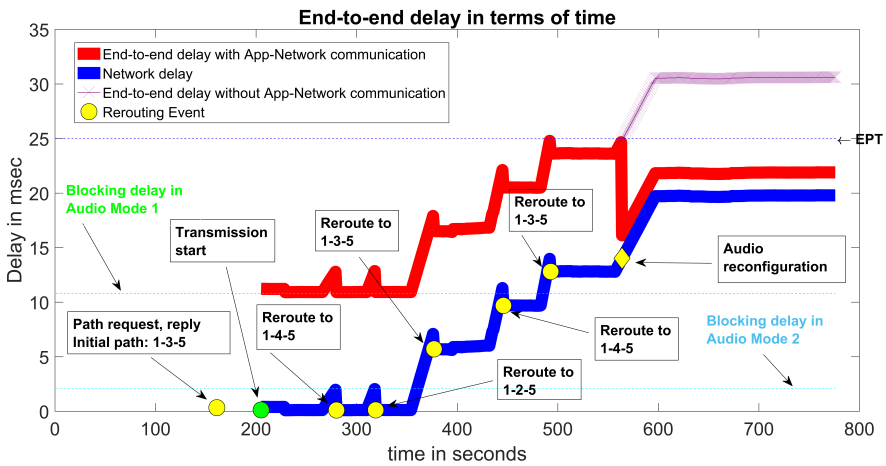


Figure 5.16: End-to-end delay evaluation

$$end - to - end\ delay = network\ delay + 2 * sound - card\ delay \tag{5.5}$$

Combining Equations 5.5 and 5.3 we evaluate gain as (network delay is common for two audio modes):

$$gain = \frac{blocking\ delay\ mode1 - blocking\ delay\ mode2}{network\ delay + blocking\ delay\ mode1} * 100\% \tag{5.6}$$

So, by 5.6 we evaluate gain percentage for switching to another audio mode as a function of network delay and difference between the blocking delay values of the two audio modes.

From table 5.3 it is shown that for given sample rate, as frame size increases end-to-end delay also increases. This happens because frame size describes the number of samples

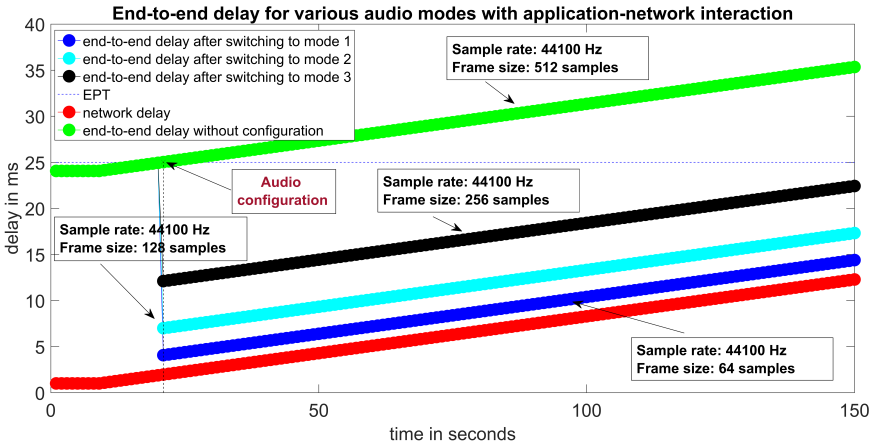


Figure 5.17: Simulation results for end-to-end delay in various audio modes

that sound-card gives to output per access. For larger frame size cases sound-card should wait more to produce the required number of samples in the output. So, for minimizing blocking delay and generally end-to-end delay frame size should be kept in low levels.

Chapter 6

Discussion

In Chapter 3 we described the related work about Networked Music Performance systems and generally SDN-based QoS aware systems. The main problem in NMP approaches is that researchers due to their interest area examine only one side of the problem and they don't take into account the other side. Audio researchers focus on ultra-low delay encoders and try to minimize end-to-end delay by reducing audio processing delay. On the other hand, network researchers examine NMP via network perspective as an example of QoS-aware system and they incorporate Software Defined Networking in their architecture to be flexible in dynamic path selection for minimizing network delay without examining audio processing aspect. The novel feature of our approach is that it does not examine only one side of the problem but it tries to minimize both network and audio processing delay and finally end-to-end delay. This is possible via close communication between application and network. Initially application requests a path with certain delay and jitter requirements and network assigns this path. Additionally, as transmission process continues in case that this path is congested, SDN controller dynamically reroutes audio flows to alternative paths without causing problem to users experience. In case that network cannot absorb increased delay, it informs application to modify audio processing mode and reduce as possible end-to-end delay.

Apart from the main difference mentioned above compared with previous similar approaches, there are also differences in the strategy each approach uses. For instance, we described that a large number of SDN-based QoS-aware application prioritize the flows in queues and based on Type of Service or User privileges (separation between normal and premium users) they provide paths to users. In our approach, each transmitter-receiver pair is assigned a group of available paths and SDN controller selects audio transmission path among them without any type of classification. Moreover, our approach can support multicast transmission for increasing redundancy by assigning multiple paths for each audio flow but this is an extra feature. We also use Session Initiation Protocol for signaling purpose between transmitter and receiver but also in this handshake process participates SDN controller which initially assigns network paths and also instructs audio modification if it can't reduce end-to-end delay via rerouting method.

Another major difference between our architecture and other QoS-aware projects that use live path monitoring is the way that we acquire information about network conditions. In many approaches that we describe in 3, SDN controller asks periodically OpenFlow switches for statistics about flows and ports. The time between two consecutive requests is called polling period. In case that small polling period is used, switches should cope with an important overhead due to these requests. However, because switches have limited resources, we try to avoid assigning them monitoring duties. They are just used as forwarding devices. We created an alternative path monitoring mechanism with transmission and reception packets of specific format and evaluating path delay as time difference between transmission and reception for each path. Transmission of these packets takes place every second between virtual hosts in Mininet. This method exploits the resources that end hosts have which is preferred than requesting repeatedly the switches. Also, in many QoS-aware applications rerouting to a new path is enabled when users report the problem via Mean Opinion Score (MOS) or QoE. Our approach enables rerouting mechanism automatically when a path with the difference of current path delay and any other available path is greater than a threshold value and does not depend on users report or is not affected by users opinion.

As a conclusion, the architecture we proposed is an intersection of the major types of approaches about Networked Music Performance systems, audio processing-based and network based. This combination extends the flexibility of the system because it can deal with congestion cases and if this is not adequate, audio modification can offer below-EPT transmission or at least offer minimized end-to-end delay, which in cases of ultra-low delay sensitive applications is remarkable.

Chapter 7

Conclusions and Future Work

Conclusions

In this master thesis we introduced and demonstrated a method for close collaboration between application and network for system performance improvement. This model can be applied in all QoS-aware applications that should provide services satisfying certain requirements related with end to end delay, jitter, packet loss and error rate. We selected Networked Music Performance (NMP) systems to apply this concept, because this type of application belongs to ultra-low delay sensitive applications that have strict delay constraints at the level of milliseconds.

Our method manages to absorb an important amount of network delay increase by modifying audio configuration parameter set as we described in Chapter 5 and generally benefits system's performance by providing better QoS service to users. Gain degree depends on the network condition and the available modes that are offered by application for switching in case of excessive network delay increase. The whole architecture is based on the recent trend in computer networks called Software Defined Networking. Apart from NMP systems, our approach can be extended to various applications used in daily life as we describe in this chapter.

Through interaction between application and network we offer better network utilization as optimal path routes are used for data delivery, avoiding traffic congestion and link failure problems. Also, application can use network information to inform users about bad network connectivity in case that it can't cope with the problem. Generally, because for most of recent applications Internet connectivity is indispensable for data exchange, collaboration concept can benefit both sides for improvement. This means that applications would be more flexible in sudden network changes without interfere users and computer networks will be used for satisfying certain application requirements described in Service Level Agreements. Also, Internet Service Providers can adjust their network infrastructure easily to new demands due to technological breakthroughs in coming years. In any case, concept introduced in this master thesis is a prototype that can be applied instantly

in real world conditions.

Technical limitations

In our approach, due to the software used, there were some constraints during experimental process. First, network topology is emulated via Mininet running on a virtual machine. This machine has limited computational resources and is not proposed for large-scale network emulations because it shares the available resources related to CPU and memory among hosts and switches. So, in case of large-scale networks a convenient computer is not recommended. Also, Mininet by default is designed to be isolated from Internet and in case that real traffic should pass through Mininet, user should configure it by attaching the corresponding interfaces to real hardware interfaces, as described in 4.5. This might create some overhead in terms of delay because of Mininet limited resources. Also, users should not forget that Mininet emulated hosts share the host file system and PID space and this requires attention when running scripts that affect file system. Finally, as all emulators, when using Mininet in real-time applications users should be careful when measuring in real-time outside Mininet and in virtual time inside Mininet. Generally, Mininet is recommended for emulating network topologies for experiment purposes but it requires attention when cooperating with other applications running on host machine.

Furthermore, in our architecture we used POX as an SDN controller. As we described in 2.4, POX is Python based controller used by research community. It supports single-threaded architecture so parallel operations are allowed but it executes them as being sequential. In our case, for instance, when client virtual host sends simultaneously measurements for delay and jitter to SDN controller, it receives them one by one. So, if we have many paths to monitor and receive measurements, controller will not react as instantly as required in case of network changes. So, POX can be used in small scale topologies when parallel path monitoring for taking optimal routing decisions is required. Furthermore, POX does not provide support for IP fragmented packets. In real network, the maximum available packet size that can be transmitted without fragmentation is defined by Maximum Transmission Unit (MTU). On Ethernet, the maximum amount of data in a frame is ordinarily 1500 bytes, which leaves at most 1472 bytes for application data to avoid fragmentation, assuming 20 bytes for the IPv4 header and 8 bytes for the UDP header. POX cannot handle fragmented packets so this limited the range of possible frame size values that we could use. Packets that we transmit through network consist of double precision numbers which have size 8 bytes. As a consequence, the maximum available frame size that avoids IP fragmentation is 128 samples ($128 * 8 = 1024$ bytes).

Finally, as we described in 4.1, we used four sound-cards for performing audio mode switching due to controller's instruction. This happened because the software tool we used for capturing and transmitting audio from sound-card to network, Simulink, does not allow modifying parameters in blocks that require access to real hardware on runtime. So, each audio mode is supported by a pair of transmitter-receiver sound-card operating in different sample rate and frame size conditions. In real world, a software that allows

this modification would be useful to change dynamically to new sample rate and frame size values.

Future work

Close collaboration between application and network is effective for ultra-low delay sensitive cases such as NMP systems. Apart from this type of service, there are various daily life applications that can support this concept. In this section, we describe various.

A modified version of our system could incorporate ultra-low delay encoding/decoding methods (e.g. Opus [8]) for modifying bit-rate based on network condition. Bit-rate describes the number of bits required for signal representation per time unit. Bit-rate is evaluated from Equation 7.1

$$bit\ rate = \frac{number\ of\ bits\ per\ sample}{sample\ rate} \tag{7.1}$$

In this case, encoder and decoder would also interact with network and modify encoding/decoding method by changing the number of bits used for audio representation in case of traffic congestion, offering bit-rate adoption to network condition. A proposed architecture is shown in Figure 7.1.

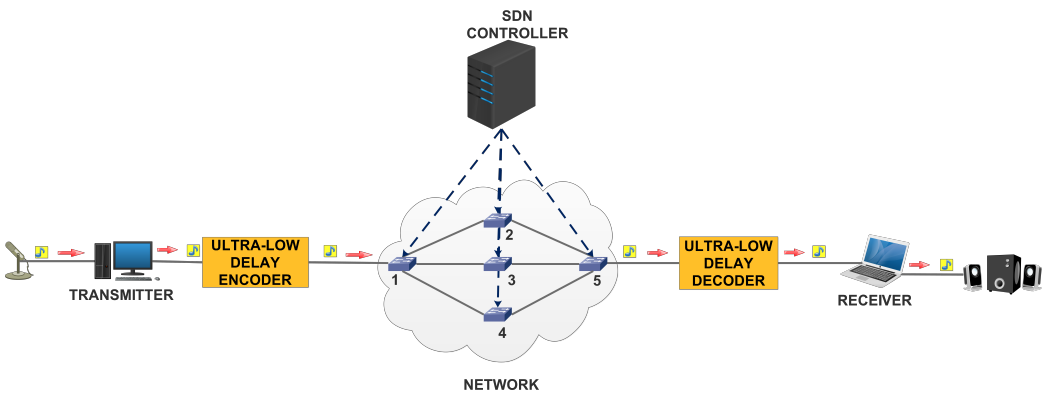


Figure 7.1: Future work with encoding in NMP

Additionally, apart from audio services, we can support also video teleconferences that require instant interaction between participants located in different places. In this case, SDN controller should assign also paths for video packets that should have same delay as audio packets and in case of traffic congestion, it reroutes them to new paths or modify multimedia processing pattern. Also, a similar application type that can adopt our concept is online gaming, that requires ultra-delay interaction between players. An example of this prototype is shown in Figure 7.2.

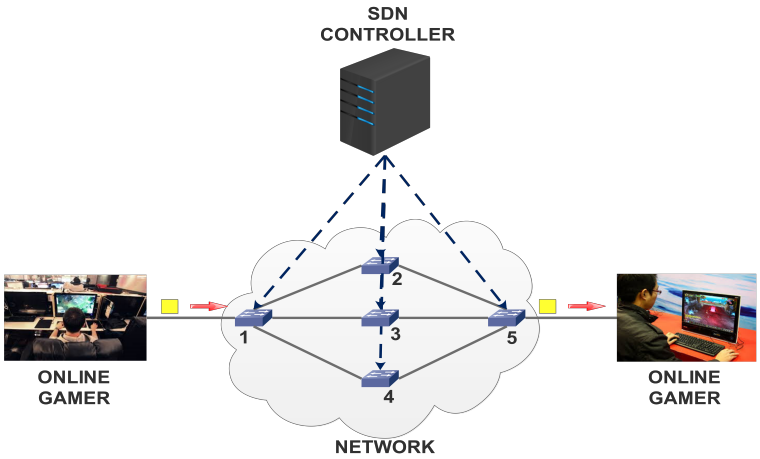


Figure 7.2: Future work in online games

Multimedia domain is not the only area that our approach can benefit. Also, in wireless sensor networks, SDN controller can monitor wireless network and select the path with minimum delay between the sensor and the measurement collector. This can be useful in cases of wireless sensors monitoring gas leakage for instance and instant reaction is required to deal with the problem. There are various researches incorporating SDN in wireless sensor networks studies ([102], [103], [103], [104]) that introduce SDN routing policies applied in wireless sensor networks.

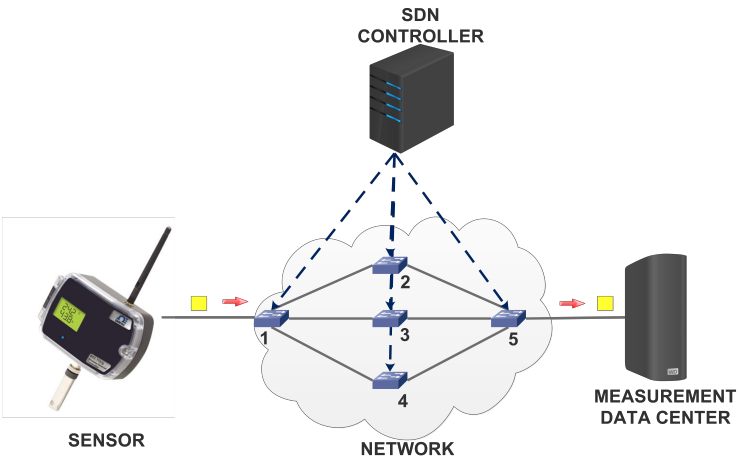


Figure 7.3: Future work in wireless sensor networks

Also, the proposed methodology can be applied to health care applications. Patients use wearable devices that transmit periodically measurements related to health indicators such as blood pressure, vision quality, and body temperature. In case that a patient needs instant treatment, device transmits an emergency message and our system can provide paths

with minimum delay for message or measurements transmission. Related works about SDN-based healthcare applications are [105] and [106]. Also, telesurgery can be benefited by our approach that requires also minimized delay across paths. SDN can be used in telesurgery applications for path establishment between doctor and patient that has certain delay value and guaranteed Quality of Servive ([107], [108]). Moreover, communication between application and network can benefit online financial transactions that require secure and instant reaction, like stock market tradings. In this case, users require real time monitoring for current stock market prices and also if they want to execute an online transaction, they want to accomplish the transaction instantly. Our proposed method can help towards this direction and generally online transactions can be benefited (Figure 7.4). Finally, another type of service that requires delay guaranteed communication is online bet services. Many users around the world use applications in mobile devices or computers and while a sport event takes place, they use Internet for online betting purpose. In this case, the network should react instantly to their bet operations so delay should be eliminated. Our method could contribute in this problem, providing the best possible path that satisfies the delay requirements or provide multiple paths for secure transactions between users.

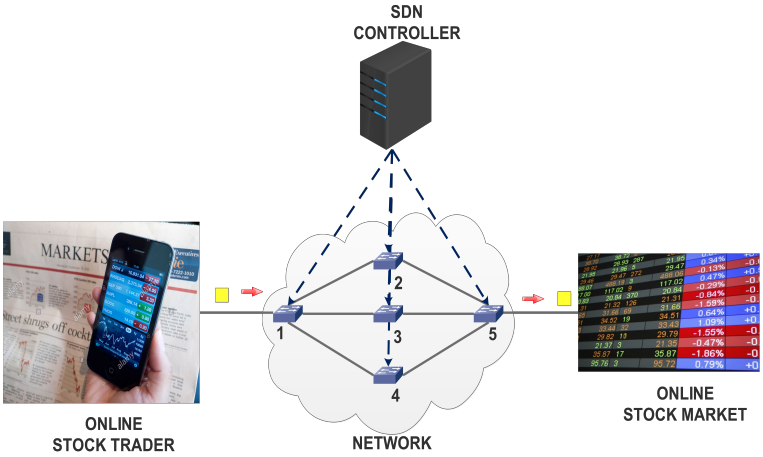


Figure 7.4: Future work in online transactions

As a conclusion, close collaboration between application and network is very useful concept for all applications that require minimized end-to-end delay and instant interaction between users connected to Internet. All the above proposed solutions to embed this concept to real systems aims to enhance applications' performance and generally quality of service and user experience.

Bibliography

[1] A. Carôt and C. Werner, "Network music performance-problems, approaches and perspectives," in *Proceedings of the "Music in the Global Village"-Conference, Budapest, Hungary*, vol. 162, 2007, pp. 23–10.

[2] G. Xylomenos, C. Tsilopoulos, Y. Thomas, and G. C. Polyzos, "Reduced switching delay for networked music performance," in *Packet Video Workshop (Poster Session)*, 2013.

[3] C. Chafe, M. Gurevich, G. Leslie, and S. Tyan, "Effect of time delay on ensemble accuracy," in *Proceedings of the International Symposium on Musical Acoustics*, vol. 31, 2004, p. 46.

[4] A. Carôt, U. Krämer, and G. Schuller, "Network music performance (nmp) in narrow band networks," in *Audio Engineering Society Convention 120*, May 2006. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=13528

[5] M. Goto, I. Hidaka, H. Matsumoto, Y. Kuroda, and Y. Muraoka, "A virtual jazz session system: Virja session," *Transactions of Information Processing Society of Japan*, vol. 21, 1910.

[6] M. Lutzky, M. Schnell, M. Schmidt, and R. Geiger, "Structural analysis of low latency audio coding schemes," in *Proc. 119th AES convention*, 2005.

[7] S. Wabnik, G. Schuller, and F. Kraemer, "An error robust ultra low delay audio coder using an ma prediction model," in *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*. IEEE, 2009, pp. 5–8.

[8] J.-M. Valin, G. Maxwell, T. B. Terriberry, and K. Vos, "High-quality, low-delay music coding in the opus codec," *arXiv preprint arXiv:1602.04845*, 2016.

[9] A. Carôt, P. Rebelo, and A. Renaud, "Networked music performance: State of the art," in *Audio engineering society conference: 30th international conference: intelligent audio environments*. Audio Engineering Society, 2007.

[10] E. Chemeritskiy and R. Smelansky, "On QoS management in SDN by multipath routing." IEEE, Oct. 2014, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/6995581/

[11] H. Owens II and A. Durresi, "Video over Software-Defined Networking (VSDN)," *Computer Networks*, vol. 92, pp. 341–356, Dec. 2015. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1389128615003199

[12] J. Yan, H. Zhang, Q. Shuai, B. Liu, and X. Guo, "HiQoS: An SDN-based multipath QoS solution," *China Communications*, vol. 12, no. 5, pp. 123–133, May 2015. [Online]. Available: http://ieeexplore.ieee.org/document/7112035/

[13] S. Tomovic, N. Prasad, and I. Radusinovic, "SDN control framework for QoS provisioning." IEEE, Nov. 2014, pp. 111–114. [Online]. Available: http://ieeexplore.ieee.org/document/7034369/

[14] H. E. Egilmez, S. Civanlar, and A. M. Tekalp, "An Optimization Framework for QoS-Enabled Adaptive Video Streaming Over OpenFlow Networks," *IEEE Transactions on Multimedia*, vol. 15, no. 3, pp. 710–715, Apr. 2013. [Online]. Available: http://ieeexplore.ieee.org/document/6376227/

[15] B. Bhattacharya and D. Das, "SDN based architecture for QoS enabled services across networks with dynamic service level agreement." IEEE, Dec. 2013, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/6802868/

[16] M. Jarschel, F. Wamser, T. Hohn, T. Zinner, and P. Tran-Gia, "SDN-Based Application-Aware Networking on the Example of YouTube Video Streaming." IEEE, Oct. 2013, pp. 87–92. [Online]. Available: http://ieeexplore.ieee.org/document/6680564/

[17] n. L. Valdivieso Caraguay, A. Benito Peral, L. I. Barona López, and L. J. García Villalba, "SDN: Evolution and Opportunities in the Development IoT Applications," *International Journal of Distributed Sensor Networks*, vol. 10, no. 5, p. 735142, May 2014. [Online]. Available: http://journals.sagepub.com/doi/10.1155/2014/735142

[18] H. Koumaras, M.-A. Kourtis, C. Sakkas, G. Xilouris, and S. Kolometsos, "In-service Video Quality assessment based on SDN/NFV techniques." IEEE, May 2016, pp. 1–5. [Online]. Available: http://ieeexplore.ieee.org/document/7500385/

[19] I. Bueno, J. I. Aznar, E. Escalona, J. Ferrer, and J. A. Garcia-Espin, "An OpenNaaS Based SDN Framework for Dynamic QoS Control." IEEE, Nov. 2013, pp. 1–7. [Online]. Available: http://ieeexplore.ieee.org/document/6702533/

[20] R. Durner, A. Blenk, and W. Kellerer, "Performance study of dynamic qos management for openflow-enabled sdn switches," in *Quality of Service (IWQoS), 2015 IEEE 23rd International Symposium on*. IEEE, 2015, pp. 177–182.

[21] T. Lin, J.-M. Kang, H. Bannazadeh, and A. Leon-Garcia, "Enabling SDN applications on Software-Defined Infrastructure." IEEE, May 2014, pp. 1–7. [Online]. Available: http://ieeexplore.ieee.org/document/6838226/

[22] S. Sharma, D. Staessens, D. Colle, D. Palma, J. Goncalves, M. Pickavet, L. Cordeiro, and P. Demeester, "Demonstrating resilient quality of service in Software Defined Networking." IEEE, Apr. 2014, pp. 133–134. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6849191

[23] M. P. Fernandez, "Comparing OpenFlow Controller Paradigms Scalability: Reactive and Proactive." IEEE, Mar. 2013, pp. 1009–1016. [Online]. Available: http://ieeexplore.ieee.org/document/6531863/

[24] Kwangtae Jeong, Jinwook Kim, and Young-Tak Kim, "QoS-aware Network Operating System for software defined networking with Generalized OpenFlows." IEEE, Apr. 2012, pp. 1167–1174. [Online]. Available: http://ieeexplore.ieee.org/document/6212044/

[25] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in SDN-OpenFlow networks," *Computer Networks*, vol. 71, pp. 1–30, Oct. 2014. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1389128614002254

[26] J. Batalle, J. Ferrer Riera, E. Escalona, and J. A. Garcia-Espin, "On the Implementation of NFV over an OpenFlow Infrastructure: Routing Function Virtualization." IEEE, Nov. 2013, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/6702546/

[27] A. Lara, A. Kolasani, and B. Ramamurthy, "Simplifying network management using Software Defined Networking and OpenFlow." IEEE, Dec. 2012, pp. 24–29. [Online]. Available: http://ieeexplore.ieee.org/document/6524222/

[28] F. Hu, Q. Hao, and K. Bao, "A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6819788

[29] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[30] "Beacon," https://openow.stanford.edu/display/Beacon.

[31] "Floodlight," http://Floodlight.openowhub.org/.

[32] "MUL," http://sourceforge.net/p/mul/wiki/Home/.

[33] Z. Cai, "Maestro: achieving scalability and coordination in centralizaed network control plane," Ph.D. dissertation, Rice University, 2012.

[34] "RYU," http://osrg.github.com/ryu/.

[35] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "Onos: Towards an open, distributed sdn os," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: ACM, 2014, pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/2620728.2620744

[36] A. Bagewadi and R. Babu, "Towards an ethernet learning switch and bandwidth optimization using pox controller," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 3, no. 7, pp. 7531–7535, 2014.

[37] R. Khondoker, A. Zaalouk, R. Marx, and K. Bayarou, "Feature-based comparison and selection of Software Defined Networking (SDN) controllers." IEEE, Jan. 2014, pp. 1–7. [Online]. Available: http://ieeexplore.ieee.org/document/6916572/

[38] Ligia Rodrigues Prete, A. A. Shinoda, C. M. Schweitzer, and R. L. S. de Oliveira, "Simulation in an SDN network scenario using the POX Controller." IEEE, Jun. 2014, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/6860403/

[39] S. Kaur, J. Singh, and N. S. Ghumman, "Network programmability using pox controller," in *ICCCS International Conference on Communication, Computing & Systems, IEEE*, no. s 134, 2014, p. 138.

[40] S. Sezer, S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, Jul. 2013. [Online]. Available: http://ieeexplore.ieee.org/document/6553676/

[41] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced study of SDN/OpenFlow controllers." ACM Press, 2013, pp. 1–6. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2556610.2556621

[42] "POX," https://github.com/noxrepo/pox.

[43] "POX," https://openflow.stanford.edu/display/ONL/POX+Wiki.

[44] F. Keti and S. Askar, "Emulation of Software Defined Networks Using Mininet in Different Simulation Environments." IEEE, Feb. 2015, pp. 205–210. [Online]. Available: http://ieeexplore.ieee.org/document/7311238/

[45] K. K. Sharma and M. Sood, "Mininet as a container based emulator for software defined networks," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 4, no. 12, 2014.

[46] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks." ACM Press, 2010, pp. 1–6. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1868447.1868466

[47] V. Antonenko and R. Smelyanskiy, "Global network modelling based on mininet approach." ACM Press, 2013, p. 145. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2491185.2491211

[48] "Traffic Control," https://wiki.linuxfoundation.org/networking/netem.

[49] G. Baltas and G. Xylomenos, "Ultra low delay switching for networked music performance." IEEE, Jul. 2014, pp. 70–74. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6878798

[50] A. Carôt, U. Krämer, and G. Schuller, "Network music performance (nmp) in narrow band networks," in *Audio Engineering Society Convention 120*. Audio Engineering Society, 2006.

[51] Xiaoyuan Gu, M. Dick, Z. Kurtisi, U. Noyer, and L. Wolf, "Network-centric music performance: practice and experiments," *IEEE Communications Magazine*, vol. 43, no. 6, pp. 86–93, Jun. 2005. [Online]. Available: http://ieeexplore.ieee.org/document/1452835/

[52] D. Akoumianakis, C. Alexandraki, V. Alexiou, C. Anagnostopoulou, A. Eleftheriadis, V. Lalioti, A. Mouchtaris, D. Pavlidi, G. C. Polyzos, P. Tsakalides, G. Xylomenos, and P. Zervas, "The MusiNet project: Towards unraveling the full potential of Networked Music Performance systems." IEEE, Jul. 2014, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/6878779/

[53] "Session initiation protocol, url = http://www.networkworld.com/article/2332980/lan-wan/lan-wan-what-is-sip.html,."

[54] H. Nam, K.-H. Kim, J. Y. Kim, and H. Schulzrinne, "Towards QoE-aware video streaming using SDN." IEEE, Dec. 2014, pp. 1317–1322. [Online]. Available: http://ieeexplore.ieee.org/document/7036990/

[55] T. Nurmela, "Session initiation protocol," in *Seminar on Transport of multimedia streams, University of Helsinki*. Citeseer, 2007.

[56] A. Ali, N. Ahmad, M. S. Akhtar, and A. Srivastava, "Session initiation protocol," *International Journal of Scientific and Engineering Research*, vol. 4, no. 1, pp. 1–6, 2013.

[57] H. Sinnreich and A. B. Johnston, *Internet communications using SIP: delivering VoIP and multimedia services with Session Initiation Protocol*, 2nd ed. Indianapolis, IN: Wiley Pub, 2006, oCLC: ocm65340953.

[58] G. Camarillo, R. Kantola, and H. Schulzrinne, "Evaluation of transport protocols for the session initiation protocol," *IEEE Network*, vol. 17, no. 5, pp. 40–46, Sep. 2003. [Online]. Available: http://ieeexplore.ieee.org/document/1233916/

[59] "ejamming, url = http://www.ejamming.com,."

[60] A. Barbosa, "Displaced soundscapes," *Computer-supported cooperative work for music applications*, 2006.

[61] "ninjam, url = http://www.ninjam.com,."

[62] G. Hajdu, "Quintet. net-a quintet on the internet." in *ICMC*, 2003.

[63] "Digital musician link, url = http://www.digitalmusician.net,."

[64] "Vsttunnel description, url = http://www.vstunnel.com,."

[65] J.-P. Cáceres and C. Chafe, "Jacktrip: Under the hood of an engine for network audio," *Journal of New Music Research*, vol. 39, no. 3, pp. 183–187, 2010.

[66] A. A. Sawchuk, E. Chew, R. Zimmermann, C. Papadopoulos, and C. Kyriakakis, "From remote media immersion to distributed immersive performance," in *Proceedings of the 2003 ACM SIGMM workshop on Experiential telepresence*. ACM, 2003, pp. 110–120.

[67] C. Alexandraki, P. Koutlemanis, P. Gasteratos, N. Valsamakis, D. Akoumianakis, G. Milolidakis, G. Vellis, and D. Kotsalis, "Towards the implementation of a generic platform for networked music performance: the diamouses approach." in *ICMC*, 2008, pp. 251–258.

[68] H. Kumar, H. H. Gharakheili, and V. Sivaraman, "User control of quality of experience in home networks using SDN." IEEE, Dec. 2013, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/6802847/

[69] S. Sharma, D. Staessens, D. Colle, D. Palma, J. Goncalves, R. Figueiredo, D. Morris, M. Pickavet, and P. Demeester, "Implementing Quality of Service for the Software Defined Networking Enabled Future Internet." IEEE, Sep. 2014, pp. 49–54. [Online]. Available: http://ieeexplore.ieee.org/document/6984051/

[70] R. Durner, A. Blenk, and W. Kellerer, "Performance study of dynamic QoS management for OpenFlow-enabled SDN switches." IEEE, Jun. 2015, pp. 177–182. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7404730

[71] C. Sieber, A. Blenk, D. Hock, M. Scheib, T. Hohn, S. Kohler, and W. Kellerer, "Network configuration with quality of service abstractions for SDN and legacy networks." IEEE, May 2015, pp. 1135–1136. [Online]. Available: http://ieeexplore.ieee.org/document/7140446/

[72] H. E. Egilmez and A. M. Tekalp, "Distributed QoS Architectures for Multimedia Streaming Over Software Defined Networks," *IEEE Transactions on Multimedia*, vol. 16, no. 6, pp. 1597–1609, Oct. 2014. [Online]. Available: http://ieeexplore.ieee.org/document/6819050/

[73] D. Adami, L. Donatini, S. Giordano, and M. Pagano, "A network control application enabling Software-Defined Quality of Service." IEEE, Jun. 2015, pp. 6074–6079. [Online]. Available: http://ieeexplore.ieee.org/document/7249290/

[74] R. F. Diorio and V. S. Timóteo, "A Platform for Multimedia Traffic Forwarding in Software Defined Networks." ACM Press, 2015, pp. 177–180. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2820426.2820464

[75] E. Hilmi E., D. S. Tahsin, B. K. Tolga, and T. A. Murat, "OpenQoS: An OpenFlow Controller Design for Multimedia Delivery with End-to-End Quality of Service over Software-Defined Networks."

[76] R. Alvizu and G. Maier, "Can open flow make transport networks smarter and dynamic? An overview on transport SDN." IEEE, Jun. 2014, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/6867771/

[77] E. Liotou, G. Tseliou, K. Samdanis, D. Tsolkas, F. Adelantado, and C. Verikoukis, "An SDN QoE-service for dynamically enhancing the performance of OTT applications." IEEE, May 2015, pp. 1–2. [Online]. Available: http://ieeexplore.ieee.org/document/7148106/

[78] L. Bertaux, A. Hakiri, S. Medjiah, P. Berthou, and S. Abdellatif, "A DDS/SDN Based Communication System for Efficient Support of Dynamic Distributed Real-Time Applications." IEEE, Oct. 2014, pp. 77–84. [Online]. Available: http://ieeexplore.ieee.org/document/6957179/

[79] S. Gorlatch, T. Humernbrum, and F. Glinka, "Improving QoS in real-time internet applications: from best-effort to Software-Defined Networks." IEEE, Feb. 2014, pp. 189–193. [Online]. Available: http://ieeexplore.ieee.org/document/6785329/

[80] J. W. Guck and W. Kellerer, "Achieving end-to-end real-time Quality of Service with Software Defined Networking." IEEE, Oct. 2014, pp. 70–76. [Online]. Available: http://ieeexplore.ieee.org/document/6968971/

[81] O. Dobrijevic, A. J. Kassler, L. Skorin-Kapov, and M. Matijasevic, "Q-POINT: QoE-Driven Path Optimization Model for Multimedia Services," in *Wired/Wireless Internet Communications*, A. Mellouk, S. Fowler, S. Hoceini, and B. Daachi, Eds. Cham: Springer International Publishing, 2014, vol. 8458, pp. 134–147. [Online]. Available: http://link.springer.com/10.1007/978-3-319-13174-0_11

[82] Q. Wang, K. Xu, R. Izard, B. Kribbs, J. Porter, K.-C. Wang, A. Prakash, and P. Ramanathan, "GENI Cinema: An SDN-Assisted Scalable Live Video Streaming Service." IEEE, Oct. 2014, pp. 529–532. [Online]. Available: http://ieeexplore.ieee.org/document/6980423/

[83] P. D. S. Maribondo and N. C. Fernandes, "Avoiding Voice Traffic Degradation in IP Enterprise Networks Using CAoS." ACM Press, 2016, pp. 34–36. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2940116.2940129

[84] M. A. Karaman, B. Gorkemli, S. Tatlicioglu, M. Komurcuoglu, and O. Karakaya, "Quality of service control and resource prioritization with Software Defined Networking." IEEE, Apr. 2015, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/7116149/

[85] J. J. Quinlan, A. H. Zahran, K. Ramakrishnan, and C. J. Sreenan, "Delivery of adaptive bit rate video: balancing fairness, efficiency and quality." IEEE, Apr. 2015, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/7114736/

[86] A. Kassler, L. Skorin-Kapov, O. Dobrijevic, M. Matijasevic, and P. Dely, "Towards qoe-driven multimedia service negotiation and path optimization with software defined networking," in *Software, Telecommunications and Computer Networks (SoftCOM), 2012 20th International Conference on*. IEEE, 2012, pp. 1–5.

[87] A. Ishimori, F. Farias, E. Cerqueira, and A. Abelem, "Control of Multiple Packet Schedulers for Improving QoS on OpenFlow/SDN Networking." IEEE, Oct. 2013, pp. 81–86. [Online]. Available: http://ieeexplore.ieee.org/document/6680563/

[88] F. Ongaro, "Enhancing quality of service in software-defined networks," Ph.D. dissertation, ALMA MATER STUDIORUM-UNIVERSITY OF BOLOGNA, 2014.

[89] M. Shimamura, H. Yamanaka, Y. Uratani, A. Nagata, S. Ishii, K. Iida, E. Kawai, and M. Tsuru, "Architecture for resource controllable NVE to meet service providers' dynamic QoS demands." IEEE, Oct. 2013, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/6724517/

[90] M. T. Diallo, F. Fieau, E. Abd-Elrahman, and H. Afifi, "Utility-based approach for video service delivery optimization," in *ICSNC 2014: International Conference on Systems and Network Communication*, 2014, pp. 5–10.

[91] W. Zhang, W. Guo, C. Li, and Y. Wen, "User-controlled QoE Adjustment based on Software Defined Networking." OSA, 2014, p. AF1G.3. [Online]. Available: https://www.osapublishing.org/abstract.cfm?uri=ACPC-2014-AF1G.3

[92] Y. Jinyao, Z. Hailong, S. Qianjun, L. Bo, and G. Xiao, "Hiqos: An sdn-based multipath qos solution," *China Communications*, vol. 12, no. 5, pp. 123–133, 2015.

[93] Qiang Duan, "Network-as-a-Service in Software-Defined Networks for end-to-end QoS provisioning." IEEE, May 2014, pp. 1–5. [Online]. Available: http://ieeexplore.ieee.org/document/6839919/

[94] "Rfc 768 - ietf, url = https://https://www.ietf.org/rfc/rfc768.txt,."

[95] "Pox discovery module, url = https://github.com/CPqD/RouteFlow/blob/master/pox/pox/openflow/discovery.py,."

[96] "Python networkx, url = https://networkx.github.io/,."

[97] "Audio device reader, url = https://www.mathworks.com/help/audio/ref/audiodevicereader.html,."

[98] "Audio device writer, url = https://www.mathworks.com/help/audio/ref/audiodevicewriter-class.html,."

[99] "Mininet hosts talking to internet, url = http://blog.mcchan.io/attach-mininet-to-the-internet,."

[100] "Open vswitch manual, url = http://openvswitch.org/support/dist-docs/ovs-ofctl.8.txt,."

[101] "NETEM," https://wiki.linuxfoundation.org/networking/netem.

[102] A. De Gante, M. Aslan, and A. Matrawy, "Smart wireless sensor network management based on software-defined networking," in *Communications (QBSC), 2014 27th Biennial Symposium on*. IEEE, 2014, pp. 71–75.

[103] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "Sdn-wise: Design, prototyping and experimentation of a stateful sdn solution for wireless sensor networks," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*. IEEE, 2015, pp. 513–521.

[104] T. Luo, H.-P. Tan, and T. Q. Quek, "Sensor openflow: Enabling software-defined wireless sensor networks," *IEEE Communications Letters*, vol. 16, no. 11, pp. 1896–1899, 2012.

[105] M. Z. Hasan, "Intelligent healthcare computing and networking," in *e-Health Networking, Applications and Services (Healthcom), 2012 IEEE 14th International Conference on*. IEEE, 2012, pp. 481–485.

[106] L. Hu, M. Qiu, J. Song, M. S. Hossain, and A. Ghoneim, "Software defined healthcare networks," *IEEE Wireless Communications*, vol. 22, no. 6, pp. 67–75, 2015.

[107] V. Kotronis, X. Dimitropoulos, R. Klöti, B. Ager, P. Georgopoulos, and S. Schmid, "Control exchange points: Providing qos-enabled end-to-end services via sdn-based inter-domain routing orchestration," *LINX*, vol. 2429, no. 1093, p. 2443, 2014.

[108] R. Rayman, S. Primak, R. Patel, M. Moallem, R. Morady, M. Tavakoli, V. Subotic, N. Galbraith, A. van Wynsberghe, and K. Croome, "Effects of latency on telesurgery: an experimental study," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2005, pp. 57–64.