Cross-layer Control for Adaptive Video Streaming over Wireless Access Networks

Abdallah S. Abdallah AbouSheaisha

Dissertation submitted to the faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Engineering

Allen B. MacKenzie, Chair

A. A. (Louis) Beex

Denis Gracanin

Luiz A. DaSilva

Mohamed S. AbouElnasr

Yaling Yang

February 19, 2016

Blacksburg, Virginia

Keywords: Wireless Networks, Cross-layer Optimization, HTTP Adaptive Video Streaming, Markov Decision Processes, Reinforcement Learning

Cross-layer Control for Adaptive Video Streaming over Wireless Access Networks

Abdallah S. Abdallah AbouSheaisha

(ABSTRACT)

Over the last decade, the wide deployment of wireless access technologies (e.g. WiFi, 3G, and LTE) and the remarkable growth in the volume of streaming video content have significantly altered the telecommunications field. These developments introduce new challenges to the research community including the need to develop new solutions (e.g. traffic models and transport protocols) to address changing traffic patterns and the characteristics of wireless links and the need for new evaluation methods that generate higher fidelity results under more realistic scenarios.

Unfortunately, for the last two decades, simulation studies have been the main tool for researchers in wireless networks. In spite of the advantages of simulation studies, overall they have had a negative influence on the credibility of published results. In partial response to this simulation crisis, the research community has adopted testing and evaluation using implementation-based experiments. Implementation-based experiments include field experiments, prototypes, emulations, and testbeds. An example of an implementation-based experiment is the MANIAC Challenge, a wireless networking competition that we designed and hosted, which included creation and operation of ad hoc networks using commodity hardware. However, the lack of software tools to facilitate these sorts of experiments has created new challenges. Currently, researchers must practice kernel programming in order to implement networking experiments, and there is an urgent need to lower the barriers of entry to wireless network experimentation.

With respect to the growth in video traffic over wireless networks, the main challenge is a mismatch between the design concepts of current internet protocols (e.g. the Transport Control Protocol (TCP)) and the reality of modern wireless networks and streaming video techniques. Internet protocols were designed to be deployed over wired networks and often perform poorly over wireless links; video encoding is highly loss tolerant and delay-constrained and yet, for reasons of expedience is carried using protocols that emphasize reliable delivery at the cost of potentially high delay.

This dissertation addresses the lack of software tools to support implementation-based networking experiments and the need to improve the performance of video streaming over wireless access networks. We propose a new software tool that allows researchers to implement experiments without a need to become kernel programmers. The new tool, called the Flexible Internetwork Stack (FINS) Framework, is available under an open source license. With our tool, researchers can implement new network layers, protocols, and algorithms, and redesign the interconnections between the protocols. It offers logging and monitoring capabilities as

well as dynamic reconfigurability of the modules' attributes and interconnections during runtime. We present details regarding the architecture, design, and implementation of the FINS Framework and provide an assessment of the framework including both qualitative and quantitative comparison with significant previous tools.

We also address the problem of HTTP-based adaptive video streaming (HAVS) over WiFi access networks. We focus on the negative influence of wireless last-hop connections on network utilization and the end-user quality of experience (QoE). We use a cross-layer approach to design three controllers. The first and second controllers adopt a heuristic cross-layer design, while the third controller formulates the HAVS problem as a Markov decision process (MDP). By solving the model using reinforcement learning, we achieved 20% performance improvement (after enough training) with respect to the performance of the best heuristic controller under unstable channel conditions. Our simulation results are backed by a system prototype using the FINS Framework.

Although it may seem predictable to achieve more gain in performance and in QoE by using cross-layer design, this dissertation not only presents a new technique that improves performance, but also suggests that it is time to move cross-layer and machine-learning-based approaches from the research field to actual deployment. It is time to move cognitive network techniques from the simulation environment to real world implementations.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This dissertation discusses two research problems and the relationship between them. The first problem is the lack of software tools to support implementation-based studies in networking research. The second problem is improving the performance of video quality controllers that run by the HTTP-based adaptive video streaming (HAVS) client applications over IEEE 802.11 access networks. In the following two sections, we briefly discuss our motivations and objectives in studying each of these problems and summarize our contributions.

1.1 Motivation and Objective

The last two decades witnessed the evolution of wireless networking in research, industry, and in daily use. However, many of the network and transport protocols in use were designed for wired networks and are poorly suited to the wireless environment. The research community has worked to address this issue through several approaches, including the applications of so called ”cross-layer solutions”. Unfortunately, cross-layer solutions have not moved beyond the research phase because of the challenges associated with evaluating them (e.g., the high cost of implementation either in simulation environment or testbeds due to following restricted stack models).

Many published proposals depend heavily on simulation, while few proposals are backed by implementation verified results. Significant discrepancies between simulation and experimental results are common due to the stochastic behavior of wireless channels. We have experienced this ourselves during the MANIAC Challenge [1]. Our empirical data showed that the wireless links were much more dynamic than predicted by widely used simulation models [2]. We believe combining simulation results with implementation-based results is crucial to provide better fidelity for wireless research.

Implementation-based experiments on wireless networks face logistical, evaluative, and implementational challenges. Logistical challenges include the need for significant numbers of people, equipment, and space. Evaluative challenges include the difficulty of repeatability and the lack of standard benchmarking scenarios.

Implementational challenges include the lack of experimental tools, the different ways the protocol stack is implemented across different platforms, and the high cost of implementation.

Motivated by the aforementioned observations from the MANIAC Challenge, we investigate the lack of experimental software tools as one of these challenges. We propose a new tool named the Flexible Internetworking Stack (FINS) Framework to address this problem. The tool seeks to address the requirements of the research community, while avoiding the disadvantages found in the previous tools, in order to lower the barrier for implementation-based wireless network experiments across the network stack. The tool achieves this by providing access to functions that are usually implemented within the operating system's kernel, in addition to a set of logging and reconfigurability features.

The last few years have seen incredible growth in the volume of streaming video content over the Internet through wireless technologies (e.g., WiFi, 3G, and LTE). The percentage of total Internet traffic which is video reached 51% in the year 2011 [3].

In the context of mobile devices and wireless access networks, video traffic volume exceeded 50% of the global mobile traffic in early 2012, before reaching 55% in 2014 [4]. Video traffic is expected to reach 72% of the global mobile data traffic by the year 2019. The percentage of the total mobile data traffic in 2014 which was actually offloaded through WiFi hotspots at public places, residential access points (APs), or cellular Femtocells is 46% [4].

HTTP-based adaptive video streaming (AVS) is the technology currently used to deliver video content on demand (VoD). It depends on storing multiple copies of each video clip, each copy representing a level of video quality with an associated minimum required data rate to be downloaded and played-back smoothly without discontinuity.

The files corresponding to different quality levels are segmented into equal number of segments of the same length with a granularity of a few seconds (e.g., 2, 4, 10). The video application installed on the client device is responsible for controlling which quality to stream and requesting the desired level when asking to download each segment. Each segment is requested by sending a new HTTP Get command over a new TCP connection or over an existing persistent TCP connection. Through the rest of this dissertation, we refer to the client thread that is responsible for choosing the quality level as the adaptive logic module (ALM) or the quality controller alternatively. It determines which quality to request based on the device capabilities (e.g., CPU, Graphics card, screen resolution) and the available connection bandwidth.

In general, current commercial video clients implement an ALM that estimates achieved E2E goodput periodically and requests the highest quality level whose minimum required data rate can be achieved, taking into account the amount of data already received and buffered to avoid an empty playback buffer (aka, buffer underrun). However, a fixed static scheme based on predetermined thresholds is usually used. Prediction or learning algorithms that would apply foresight are not currently used —present algorithms are myopic, considering only one step ahead.

Although various commercial solutions (e.g., Microsoft IIS Smooth Streaming [5], HTTP Live Streaming (HLS) IETF standard by Apple [6], the HTTP Dynamic Streaming (HDS) by Adobe [7], Dynamic Adaptive Streaming over HTTP (DASH)) exist, to the best of our knowledge none of the available products deploys an ALM that differentiates between wired and wireless connections. A differentiation is likely needed since end-to-end estimation is insufficient to adapt to changes caused by the dynamics of the wireless channel. Hence, there is a growing necessity to consider other components in designing the ALM to address the nature of wireless access links.

Although the research community has given more attention to the HTTP-based AVS (HAVS) in the wireless context over the last five years, as we discuss in the literature review in 2.2, the majority of proposals fall under three categories: proposals that adopt an obsolete technology, proposals that have been evaluated under unrealistic scenarios, and idealistic proposals that are too complex. For example, they may demand invasive modifications at both the client and server sides or introduce significant overhead, making their implementation costs too high for adoption.

The lack of an efficient and an easy to deploy quality controller for adaptive video streaming over wireless networks motivated us to tackle this problem. Our objective is to propose an efficient solution to enhance the utilization of the available bandwidth and to improve the end user Quality of Experience (QoE), while introducing limited modifications and adding minimal additional overhead. Hence, we present a cross-layer approach that uses reinforcement learning (RL) in the ALM to monitor the wireless link

and the TCP behavior then respond accordingly. RL enables the ALM to consider both the immediate and future achievable throughput. This problem also provides an example of how specific applications require cross-layer modifications to avoid degradation of performance under legacy TCP/IP protocols, which were not originally designed to function over wireless links. Hence, it is an interesting case study that combines simulation-based evaluation with implementation-based evaluation using our proposed tool, the FINS Framework.

1.2 Contributions

The contributions we made in this dissertation can be summarized as follows:

- We presented our design and implementation of the Flexible Internetwork Stack (FINS) Framework, a software tool to lower the barrier of implementation-based evaluation for the research in computer networking, especially wireless networks and mobile devices related scenarios. We made the tool available as open source through the GitHub repository.
- We designed and simulated a new heuristic cross-layer quality controller for HAVS clients running over IEEE 802.11 networks. the results showed that adapting video rate in the application layer based on specific measurements from the IEEE 802.11 MAC/PHY is a powerful approach. The proposed controller outperformed the

prior solution we used for comparison, especially under unstable channels and shorter initial buffering periods, which is a great advantage of our controller.

- We formulated the HAVS over IEEE 802.11 networks as a Markov decision process (MDP) and designed a new system model and reward function based on the insights we learned from the heuristic approach. Then, we presented an offline solution for the proposed system model, assuming that the behavior of the TCP congestion window dynamics and the bit error rate (BER) on the wireless link is known. The solution used a policy iteration algorithm to find the optimal quality controller.

- We presented an online solution for the proposed MDP system model using RL. The solution used the state-action-reward-state-action (SARSA) algorithm and successfully outperformed the prior solution we used for comparison. Our SARSA controller doubled the performance improvement achieved by the Q-learning controller. In addition to the simulation results, we implemented a prototype of the RL controllers and evaluated them under realistic experimental setup using the FINS Framework, the video player, and the DASH API approved by the DASH Industry Forum.

1.3 List of Publications

Here is a list of our publications on the FINS Framework:

- Abdallah A.S., MacKenzie A.B., DaSilva L.A., Thompson M.S.,"On Software Tools and Stack Architectures for Wireless Network Experiments", IEEE Wireless Communications and Networking Conference, WCNC 2011.

- Reed J., Abdallah A.S., MacKenzie A.B., DaSilva L.A., Thompson M. S., "The FINS Framework: Design and Implementation of the Flexible Internetwork Stack (FINS) Framework", IEEE Transaction on Mobile Computing (TMC), vol.15, no.2, pp.489-502, Feb. 1 2016

- Thompson M. S., Abdallah A.S., Reed J., MacKenzie A.B., DaSilva L.A., "The FINS Framework: An Open-Source Userspace Networking Subsystem for Linux", vol.28, no.5, pp.32-37, September-October 2014.

In addition to the paper published below, some of the results from Chapter 5 are currently prepared to be submitted to a conference paper and an extended version will be submitted to a journal paper.

- Abdallah A.S. and MacKenzie A.B., " New Cross-Layer Controller for Adaptive Video Streaming over IEEE 802.11 Networks", IEEE ICC 2015, London, UK.

Other publications by the author through the PhD program include:

- A. S. Abdallah, A. B. MacKenzie, V. Marojevic, R. B. Bacchus, A. Riaz, D. Roberson, J. Kalliovaara, J. Hallio, and R. Ekman, "Detecting the impact of human Mega-Events on spectrum usage", in 13th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, USA, Jan. 2016.

- Thompson Michael S., Hilal Amr E., Abdallah A.S., DaSilva L.A., MacKenzie A.B.,"The MANIAC Challenge: Exploring MANETs through Competition", in Proceedings of the 8th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), 2010, pp. 443452.
- Ghaboosi K., MacKenzie A.B., DaSilva L.A., Abdallah A.S., Latva-aho M., A Channel Selection Mechanism based on Incumbent Appearance Expectation for Cognitive Networks, IEEE Wireless Communications and Networking Conference, WCNC 2009.

1.4 Outline

This dissertation is divided into six chapters. Chapter 2 discusses the relevant background by reviewing the literature in the areas of implementation-based experimental networking tools and adaptive video streaming over wireless networks.

In Chapter 3, we present the Flexible Internetwork Stack (FINS) Framework, as our proposed solution to address the first research problem. The chapter discusses the FINS Framework architecture, design, and implementation in detail, and provides the the qualitative and quantitative performance assessments of the framework.

In chapter 4, we present a heuristic quality control approach for the HTTP-based AVS problem. The proposed approach exploits the fundamentals of wireless bandwidth

estimation found in the relevant literature and lessons that we learned from previous attempts on designing heuristic controllers.

In chapter 5, we present our system model that formulates the HAVS problem using the MDP framework, then we present he offline model-based solution and its online learning-based counterpart, each with its own performance evaluation results. Finally, we present and evaluate the prototype we have implemented using the FINS Framework.

We finally conclude with chapter 6, which provides an overview of our results and contributions, and highlights open questions and observations for future work.

Chapter 2

Literature Review [1]

This chapter introduces the fundamentals of the two research problems that we tackle in this dissertation. In this chapter, we present the definitions, concepts, and terminologies of each problem, then we review the most related work found in its literature.

The chapter is divided into two sections. In the first section, we present two classification taxonomies of previous attempts to create implementation-based experimental and prototyping tools, then we apply these taxonomies to review the previous work. We conclude the first section by discussing the features that we believe new tools are demanded to provide in order to fulfill the prospective users needs and the concerns to be considered when building a new tool to avoid previous mistakes.

In the second section, we cover the adaptive video streaming problem. The section

[1] The first section of this chapter is based on our publication [8]

covers the evolving of video streaming technologies to date. Then, we discuss the challenges specifically related to HTTP-based adaptive streaming over wireless access networks. The section concludes with a summary of related previous work with a focus on proposals that have used cross-layer designs and/or reinforcement learning and proposals that involve implementation-based evaluation or cover streaming over IEEE 802.11 wireless links.

2.1 Prototyping and Implementation-based Experimental Tools

In this dissertation, we use the terms *implementation-based tools*, *prototyping tools*,or *experimental tools* to refer to software packages, frameworks, architectural entities (e.g. stacks, connected graphs), and Application Programmable Interfaces (APIs) that can be used to prototype or fully implement new networking research proposals (e.g. a network layer, a routing protocol, or a congestion control algorithm) for testing and evaluation purposes.

The use of simulation environments (e.g. ns-2 [9], ns-3 [10], OPNET Modeler [11]) is still the most common testing and evaluation technique for wireless network algorithms and protocols despite growing questions about the fidelity of simulations [12]. This has motivated some researchers to try other techniques such as emulation

testbeds and implementation-based experimental tools. Emulation is a technique that combines simulation and implementation. In [13], an emulator is defined as a set of tools which allow the user to imitate a layer or more from the network stack in a simulation environment while running the remaining layers as real-world implemented systems. Emulation testbeds have many advantages but they inherit some of the disadvantages of simulation environments, in addition to the complexities of synchronizing the actual time clock with the simulation clock [14].

Implementation-based experiments are studies that run a full implementation of a network stack on top of a real-world platform such as a laptop or a handheld device. The MANIAC Challenge [1], and the study in [15] are examples of implementation-based experiments. Implementation-based experiments on wireless networks face a set of logistical, evaluative, and implementational challenges. Logistical challenges include the need for significant numbers of people, equipment, and space. Evaluative challenges include the difficulty of repeatability and the lack of standard benchmarking scenarios. Implementational challenges include the lack of experimental tools, the different ways the protocol stack is implemented across the different platforms, and the high cost of implementation.

Various software tools are needed to make implementation-based experiments a feasible approach for testing and evaluation of wireless network protocols. Such tools have been developed by individual researchers over the years, although their impact on the research community is still limited. A number of research studies have employed

implementation-based experiments, on mobile ad hoc networks (MANETs) as surveyed in [13]. However, the vast majority of MANET research still relies primarily on simulation.

This section presents an overview of the experimental tools that support implementation-based experiments across the network stack. We present two taxonomies for classifying these tools and then review significant previous work. Next, we discuss some issues related to the possible target platforms and hardware devices. The section concludes with a list of important features that are needed in future tools.

2.1.1 Classification Taxonomies

We begin by providing two taxonomies for implementation-based and prototyping tools. Our first classification scheme is based on architectural concept: *clean-slate tools* and *hybrid tools*.

In *clean slate tools*, researchers propose innovative architectures for the structures that connect the network components to each other within each node. Some proposals suggest structures that are not stacks. This category overlaps with projects that propose replacements for the TCP/IP stack. This area of research also serves Internet reinvention projects (e.g. the GENI project [16]). The clean-slate approach aims at avoiding the disadvantages of the layered architectures of the TCP/IP stack while retaining some degree of modularity. Projects such as the Autonomic Network Architecture (ANA) [17]

and the CellNet framework [18] are examples of clean-slate tools. For a detailed review of this area, the reader may review [19].

An example of a clean-slate architecture would be to have the different TCP/IP stack layers connected as a fully connected graph instead of their current layered connections and to allow all possible combinations of control or data flow. Another example would be to allow developers to implement all tasks of routing and medium access side by side with transport-layer tasks in a single module, breaking the concept of independence between different layers. These examples and similar radical solutions can be tested using a tool like ANA [17]. But they cannot be easily integrated with the current architecture of network nodes, local area networks, or the Internet. They also suffer from a lack of compatibility with current applications, hence special interfaces or APIs must be provided to enable such experimental work.

In contrast, *hybrid tools* modify or combine features of the layered stack with some clean slate features. Such modifications allow the implementation of cross-layer solutions and inter-communication between non-adjacent layers. An example of hybrid features is the ability to build new network-aware applications, which are end-user applications that take network status into account. This type of application is envisioned to run in cognitive network scenarios[20]. X-kernel [21], IRIS [22], X-layer [23], OpenOnload [24], and XL-interface [25] are examples of hybrid tools.

Our second taxonomy classifies the experimental tools into four classes with respect to their functional purposes. The first class contains the tools that aim to facilitate

developing and testing new protocols. For example, tools such as CLICK [26], ProtoLib [27], and Profab [28] provide researchers with a library of basic- and intermediate-level functions. These functions can be used to implement larger modules such as a protocol or even an entire layer. Some tools accelerate the development cycle by providing an API which is used to implement a virtual platform-independent layer. This virtual layer can be used to run experimental protocols on top of a real stack, sacrificing some performance for the sake of a unified implementation of the experimental protocols regardless of platform. Examples of platform-independent APIs include VIPE [29], and GEA [30].

The second class contains tools that provide generic interfaces to interact with the existing layers and protocols. These APIs provide a standard interface for the flow of data between layers, as well as reading or tuning parameters within a protocol or layer module during run time. They work as a kind of standard wrapper to access pre-implemented existing protocols or layers during an experiment. Examples of such tools include ULLA [31], DEC [32], Universal convergence layer (UCL) [33], and XIAN [34].

The third class contains tools that provide managerial features over a set of nodes. These tools give the user the ability to batch experimental scenarios, broadcast the scenarios among nodes, collect trace files, redirect the movement of nodes during the experiments, and reconfigure the network topologies. The goal of these tools is to give researchers the same flexibility and usability during implementation-based experiments

as provided by the network simulation environments. APE [35], and MTM [36] are examples of this class.

The last class covers the tools that provide monitoring and logging whether they adopt a centralized or distributed approach. Monitoring and logging tasks include trace collection, trace aggregation, topology visualization, and data filtering. Examples of this class include MMAN [37], APE View [35], and Promox [38].

2.1.2 Significant Examples of Experimental Tools

Here, we present a summary of the technical features of eight tools that are relevant to experimental wireless network research. Our criterion in selecting the tools in this set is their past or expected influence on the research community. Although some of the selected tools are not currently available due to licensing issues or because they are no longer supported, it is worth mentioning them as significant milestones on the way towards a widely deployed set of experimental wireless network tools.

- The CLICK Modular Router

  CLICK is a software tool for developing flexible, reconfigurable, reusable, and extensible routing modules. These routing modules are composed of basic packet processing modules called elements [26]. CLICK provides researchers with a rich library of basic elements. Examples include timers, counters, packet classifiers, schedulers, packet parsers, queue buffers, and interfaces to network hardware

devices. The elements are connected as a directed graph where the packets follow a preprogrammed path. CLICK is often referred to as a modular router because most of the architectural concepts of CLICK are inspired by the characteristics of routers. A configuration file is written using a declarative language. This file is responsible for creating the elements and setting up connections between them. The behavior of an element is described in a C++ class. A CLICK router module configuration can be built (realized) either as a Linux in-kernel driver or a user-space application. CLICK is widely deployed, as compared to the other tools surveyed. The main key to the success of CLICK in the research community is the continuous development of newer releases with better features and fewer bugs. Another important factor is the reusability of the generated elements.

- X-Kernel

  X-kernel [21] is an object-based framework for the development and implementation of network protocols as well as dynamic construction of the network stack. X-kernel has only two main classes of objects: protocol objects and session objects. Protocol objects implement network protocols such as TCP or IP, while session objects implement the behavior of the communication interface between any pair of protocols. Although the dependency relationships between protocols are statically defined at kernel configuration time, the sessions are opened and terminated dynamically during run time. Hence, the protocol objects are connected as a graph, which provides flexibility as compared to the

conventional layered stack. X-kernel also uses a set of libraries to implement the behavioral tasks of the protocol and session objects. These libraries include functions for manipulating messages, processing addresses, scheduling events, and binding objects. X-kernel can be run as either a user-space application on top of the network stack or as a network simulator using X-sim. Although X-kernel is object based, it is implemented in C. Unfortunately, the project is currently obsolete and the latest release does not support current operating systems. The most important concept in X-kernel is the ability to re-route the messages between different protocols dynamically during run time.

- OpenOnload

  OpenOnload [24] is a user-space version of the Linux kernel networking stack. The user-space re-implementation allows the Linux network stack to run mainly in user-space with minimal context switching and minimal code modifications. OpenOnload provides applications with backward compatibility, by supporting TCP/UDP/IP networking with the sockets API. The implementation incorporates several techniques such as generic sockets, memory mapping, and socket interception. The architecture adopted in OpenOnload is designed to address implementation challenges such as the embedded nature of sockets system calls inside the kernel, the need to maintain security guarantees, and the need to reduce overhead. OpenOnload is freely available for download, and is licensed under

GPLv2. It runs on Linux 2.6 kernels. It has been developed and released by Solarflare Communications, Inc. [39].

- Iris

  Iris is a flexible framework for building reconfigurable cognitive nodes on top of a software defined radio [40]. It adopts a generic component-based architecture where two component types are provided: radio components and layer components. Although the project has focused in the past on radio component issues such as supporting various SDR platforms, it provides a generic design to implement a whole cognitive network stack. Iris supports a core cognitive processing engine, which manages the flow of data among a set of functional blocks. These functional blocks are all implemented using the same generic component pattern.

  A main distinguishing feature of Iris is the ability to modify the stack structure during run time. Although this ability is implicitly provided in X-kernel, Iris explicitly allows the user to link and de-link components, pause and redirect both data and control flows between them, and tear down or add new components during runtime. Additionally, dynamic reading and modification of parameters within protocols are also supported during runtime. All implemented components currently available in the Iris library are related to the physical and medium access layers.

- The ANA Framework

  Autonomic Network Architecture (ANA) is a clean slate tool that aims to replace the conventional layered stack with self-learning, self-configurable, and self-forming networks [17]. The design is based on a core entity responsible for routing both data flows and control signals among the different functional blocks. A single node can have more than one core entity, which opens up the possibility of having multiple network stacks running on the same physical node or even on multiple physical adapters per node. An innovative addressing scheme is also proposed as part of the architecture to replace the IP addressing scheme. The main advantage of the central core routing entity in ANA is that dynamic modifications can be performed without pausing either the data flows or the control flows. However, this continuous routing of data flows creates a ceiling for the maximum achievable data throughput. ANA faces challenges inherited from the nature of clean slate solutions, such as the need to re-implement most of the existing protocols to be ANA compatible. The current version of ANA has few examples of modules and may require some effort to implement a TCP/IP application. However, the high reusability and flexibility of the ANA design may encourage researchers to adopt its approach for future experiments, especially with the growing interest in clean-slate-based projects (e.g. GENI , Future Internet Design (FIND), and Future Internet Research and Experimentation (FIRE)).

- UCL

Unified convergence layer (UCL) is a generic application programming interface (API) which provides the user with the ability to concurrently access multiple pairs of wireless physical and medium access control layers. UCL allows the network layer above all these pairs to implement cross-layer solutions by, for examples receiving link failure notifications or modifying the MAC layer retransmission behavior independently of the protocol implementations below. UCL also allows both data packets and control signals to flow seamlessly between the wireless interfaces and higher layers. This facilitates the implementation, testing, and evaluation of cross-layer solutions.

In order to provide these features while meeting performance requirements, UCL components had to be split between the user and kernel spaces. Unfortunately, UCL code has not been made publicly available for use, although the research results shown by the authors in [33] are promising.

- APE

  The Ad hoc Protocol Evaluation testbed is an experimental environment which provides a self-bootable and easily installable system for batching, broadcasting, configuring, and monitoring MANETs experiments [35]. The system has been specifically proposed to carry on an experimental comparison of routing protocols for MANETs. In addition to the managerial functions, it provides the user with the ability to collect traces to analyze connectivity, link changes, hop counts, and path optimality. APE also provides a GUI animation tool to visualize the network

topology while the experiment is running. Although the tool is available for download accompanied with helpful documentation, major updates are needed to support current versions of the Linux kernel and to support wireless adapters. APE and the aforementioned X-kernel are good examples of how the disruptive nature of research projects can lead to a short life time for such tools. The tools become obsolete quickly in spite of their promising results and expected reusability by the research community. This should encourage future proposers of new tools for experimental wireless network research to build an open source development community around their tools in order to guarantee their continuity.

- MMAN

  A Monitor for Mobile Ad hoc Network is a distributed monitoring tool for implementation-based experiments on MANETs [37]. MMAN provides features such as the ability to tolerate the loss of some management units, the ability to survive MANET partitioning, the employment of cooperative assessment between nodes to arrive at better evaluations, and visualization of the network topology during run time. MMAN introduces minimum interference to the MANET environment by using out-of-band communications between the monitoring nodes. However it requires passive distributed monitoring nodes in addition to the nodes in the main experiment. It also employs a separate central management node where all the traces are collected and analyzed concurrently. The cost of the extra nodes must be balanced against the ability to provide high fidelity evaluation

results, because this scenario guarantees that the monitoring or logging processes do not affect the network being monitored.

2.1.3 Platforms and Hardware Devices

The majority of the implementation-based experiments found in the literature use regular laptops that run a Linux operating system. With the dramatic growth of IP traffic initiated through handheld devices (e.g. smart phones, tablets, and PDAs), mobile devices should become an important target platform for experimental research. According to [3], 6% of consumer IP traffic in 2011 was generated by non-PC devices. This number is expected to be 20% by the year 2016. Tablet and smartphone traffic is expected to grow at rates of 116% and 119% respectively. If we add to this the expectation that 61% of the traffic will be generated through WiFi and cellular access networks, we can state that facilitating implementation-based wireless experiments on handheld devices is crucial for networking research om future years.

The following are some of the major reasons why handheld mobile devices are well-suited for networking research, especially when mobility is required:

1. Small size: mobile devices are much easier to transport and store than even the smallest laptops. Additionally, it is simple for a person to carry a device around with them.

2. Low power electronics: mobile devices are designed to have a long battery life; one that is measured in days not hours. This means that longer experiments are possible.

3. Simple interface: most mobile devices include touch screen interfaces which are easier to use while moving than laptop keyboards.

4. Low cost: most mobile devices cost $500 or less, meaning more devices can be purchased for the same amount of money.

5. Communication capabilities: most mobile devices have cellular connectivity, GPS receivers, at least IEEE 802.11b, and Bluetooth. This is more than you will find on most current laptops.

The major drawback of mobile devices is their limited computing resources. Mobile processors are less computationally capable than desktop and laptop processors. However this is acceptable since most networking experiments are not computationally demanding. Additionally, mobile devices have more limited storage space than larger devices. The current storage medium of choice is the microSD card, offering up to 16 and 32 GB of space; we expect this capacity will increase. Depending on the experiment, this may or may not be enough storage space. Since this storage is typically removable, it is possible to swap memory cards when needed. The power conserving abilities, specialized mobile-nature, and the low cost of mobile devices make them a perfect choice for experimental networking.

2.1.4 Features, Design Concepts, and Implementation Keys

Below, we summarize features that new tools should support to enable future wireless networking research. We also discuss the design concepts and implementation keys to consider when starting a new tool.

The features that we believe the new tool should make available are:

- Backward compatibility with legacy applications: A discouraging factor in previous proposals for replacing the layered network stack is the lack of support for legacy network applications. This leads to a high cost for adopting the new structure due to the need to re-implement the applications or at least the Socket API. As a result, these experimental tools lack the capability to use real-world applications; instead artificial traffic is generated. If a new tool enables the use of legacy applications by supporting integration with the BSD socket API, this disadvantage will be avoided.

- Running on handheld devices: It is challenging to carry out MANET experiments while carrying a laptop. Battery life and weight limit the experimental scenarios. The research community needs to enable implementation-based experiment on handheld devices such as Android platforms [41]. A new tool may support multiple-platforms if portability is taken into account as discussed later.

- Meters and knobs: Having a generic interface which allows the user to control the behavior of network protocols during run time, regardless of their internal implementation details, is the most important functional requirement for such

tools. This control may be applied manually from the command line, through a GUI, or even automatically through a cognitive engine module/process. This feature, which we refer to as "meters and knob, enables researchers to implement solutions that rely on event-driven notifications and dynamic responses. These are sufficient to realize many cross-layer solutions to improve the performance of the wireless network stack. Meters and knobs also provide an entry point to collect data traces for the logging process discussed below.

- Monitoring and logging: This requirement is related to the previous one. The main addition is that a process has to be responsible for collecting traces and metrics, recording these traces with time-stamps, and providing the ability to retrieve this information later. This process may be centralized or distributed among nodes. Out-of-band transfer of traces can be implemented to guarantee minimum influence on the performance of the experimental network. The ability to access time-stamped records during run time is another feature, which supports the implementation of cognitive protocols. Cognitive protocols may implement learning processes, which employ historical records to deduce statistical data. According to these statistics, changes in the network conditions are detected, and adaptation decisions are made.

- Reconfigurability of the network stack: Dynamic loading, linking, and de-linking of modules is particularly important to support experiments with dynamic (or cognitive) networks. The ability to add modules at run time and link or de-link previously loaded modules facilitates the implementation of various testing and

benchmarking scenarios. The clonable network stack project [42] is an example of support for reconfigurability features, where FreeBSD's kernel is modified to enable running multiple independent instances of the network stack concurrently. A whole protocol stack can be loaded then torn down, and other features are provided with minimum effect on network efficiency. In this dissertation, we are interested in smaller granularity, where a layer, protocol or an algorithm within a protocol can be replaced during run-time. For example, the user may want to change the TCP flavor (e.g. Tahoe, Vegas, or Reno) during run-time without interrupting the established connection(s).

In addition to these features, we present the following concepts and implementation keys after analyzing the literature to understand why researchers have not adopted some previous tools:

- Cost of implementation: As mentioned earlier, clean-slate tools provide the highest flexibility possible, however, they face high implementation costs. This includes the cost of the tool's infrastructure libraries, APIs and interfaces, as well as the cost of deploying such tools in experiments. Due to the major differences in the structure of the conventional network stack compared to the clean slate architectures, major modifications to (or a completely new implementation of) legacy protocols and applications might be required. This is a key concept to consider before deciding which architectural class to follow in creating a new tool.

- Continuity of support: The lack of continuity in research projects is why many

tools become obsolete. The absence of a development and support team to update the tool renders it obsolete. As long as the tool has not become a commercial product, the only way to maintain continuity is through a strong open source development community of researchers, network developers, and other interested members. Otherwise, the tool deployment rate will fade quickly like a lot of previous work (e.g., X-Kernel, and APE).

- Extendability: We define extendibility as the ability to add new features to the tool serving new objectives. One of the reasons previous tools have not been deployed more widely is that they are tightly customized to serve specific experiments. This reduces the ability of interested researchers to extend the tool's functionality in order to use it in other experiments. For example, modifying the XIAN tool [34] to pass measurements from the transport layer to the application layer, in addition to its purpose of passing measurements only from link layer to higher layers.

- Reusability: We define reusability as the ability to reuse components of the tool's implementation code into creating newer components. For example, reusing the same design and implementation of the timing block among several protocols that need timers. We have mentioned earlier how the high reusability of the CLICK modules contributes to its success. To achieve similar high reusability, future tools should adopt a modular architecture. Increasing the granularity within each module increases the reusability of these modules. An example of high modularity and granularity in an architecture is the way the basic elements in CLICK have

been used to implement middle level modules, which in turn integrated to implement high-level modules such as IPv4 and IPv6. Dependency also affects reusability as explained above.

- Portability: The ability to run the same tool on different platform makes a tool superior to one that runs on single platform. As mentioned earlier in Section 2.1.3, mobile devices are considered best fit platforms to carry on wireless mobile network experiments. A tool that runs on multiple platforms including handheld devices, will help researchers to tackle experiments that are currently challenging with the limited mobility and power constraints of conventional laptops. This is one of the main keys we have considered in our proposed framework. Dependency directly affects portability since binding the tool's implementation to a platform-dependent library or API decreases the chance of smooth porting.

- Dependency: A basic factor that affects extendibility, reusability, and portability is the list of dependencies. In general, the longer the list of dependencies, the more difficult it becomes to extend, reuse, or port the tool. Making major decisions based on features available only within a certain kernel version, a specific API, or a device driver eventually leads the tool becoming obsolete. Unfortunately, a distinguished tool like X-Kernel has fallen into this mistake.

The lists above are not inclusive but they cover the most important features and concepts. In table 2.1 we classify the set of tools discussed earlier with respect to the

classification taxonomies presented in section 2.1.1. In addition, we compare these tools according to the features and design concerns illustrated above.

Some Hybrid tools that belong to the development tools category, such as CLICK and X-kernel, have either no or limited support for the meters and knobs feature or the monitoring and logging feature. This is in contrast to hybrid development tools, which are equipped with cross-layer interface capabilities, such as Iris. The reason is that the cross-layer interfaces make it possible to implement its own meters and knobs code. Also, hybrid tools provide the user with a higher level of flexibility and accessibility across the network stack. This combination makes such tools more powerful for prototyping and testing cross-layer solutions and good candidates to implement experiments related to cognitive or reconfigurable nodes.

Clean slate architectures (e.g., ANA) in table 2.1 excel in terms of reconfigurability, extendability, and reusability, although they have the highest implementation cost due to the need to re-implement the whole network stack. They are more difficult to use because protocol developers and researchers need time to adapt to coding within the new non-layered architecture. Due to the burden of adapting to clean slate architectures as well as their high cost, future implementation-based experimental tools should adopt hybrid architectures to balance the trade-off between cost and usability. This allows researchers to focus on adding new features to implementation environments rather than reinventing the wheel. This is the architecture we adopt for the Flexible Internetwork Stack (FINS) Framework.

Table 2.1: Comparison of various experimental tools with respect to desirable features and design concepts.

Criterion	CLICK	ANA	X-kernel	IRIS	UCL	APE	MMAN
Architectural classification	Hybrid	Clean-slate	Hybrid	Hybrid	Hybrid	Hybrid	Hybrid
Functional classification	Development	Development	Development	Development + Interfacing	Interfacing	Managerial	Monitoring
Meters and knobs	Low	High	Low	Medium	Not Available	Not Available	Not Available
Backward compatibility with legacy applications	Low	High	Low	Medium	Not Available	Not Available	Not Available
Running on handheld devices	Low	High	Low	Medium	Not Available	Not Available	Not Available
Monitoring and logging	Low	High	Not Available	Medium	Not Available	High	High
Reconfigurability	Not Available	High	Low	Medium	Not Available	Not Available	Not Available
Cost of implementation	Medium	High	Medium	Medium	Low	Low	Low
Continuity of support	High	Medium	Not Available	Not Available	Not Available	Not Available	Not Available
Extendability	Medium	High	Medium	High	Not Available	Low	Not Available
Reusability	High	High	Medium	High	Low	Low	Low
Portability	Limited	Not Available	Not Available	High	Not Available	Not Available	Not Available

2.2 Adaptive Video Streaming

In this section, we review the evolution of video streaming over the last two decades. Then we look at the adaptive video streaming problem from the viewpoint of wireless last-hop access networks, and conclude with a discussion of the most relevant previous work.

2.2.1 The Evolution of Video Streaming

Video streaming over the Internet has gone through three main phases. In the first phase, the Real Time Streaming Protocol (RTSP) was widely used [43]. An RTSP client connects to an RTSP server to establish a connection before downloading the desired video file. If the video is available in multiple representations (resolution, streaming container file format), then switching between different representations requires reestablishing the connection and starting over to download the desired quality from the beginning. This is because each representation (e.g., resolution, streaming format) of the desired clip is basically stored on the server as a separate, single file. Different flavors were developed by running RTSP over UDP, over TCP, or even embedded in HTTP requests. The RTSP video session was controlled using the Real-Time Control Protocol (RTCP) [44]. Quality changes were only done manually based on user request. Common streaming file formats of this era are Microsoft ActiveMovie and Apple Quicktime. The most common video player application of this type was the RealPlayer.

In the second phase, the video streaming began a new era that was called HTTP progressive download streaming. This approach was still in use until recently on well known websites such as YouTube [45], which did not switch to a newer technology until the end of 2013. In progressive downloading, each video clip has a set of multiple quality levels (e.g., 240p, 320p, 480p, 720p, 1080p). For video-on-demand (VoD) services, the clip is encoded at these levels and a single file is generated for each level with the entire length of the clip regardless of how long it is, whether several seconds or few hours. The files are saved on the server (often distributed through content delivery networks (CDNs)). The end user uses a video application (e.g., a standalone application, or a browser plugin) to download the file over HTTP. A single HTTP Get request is sent per file over a separate TCP connection for the video content (for non-persistent connections, HTTP v1.0) or the request is pipelined over a persistent connection, such as the one originally used to download the web page (HTTP v1.1).

In the second phase, switching between different quality levels was done manually by the user—automatic switching was not provided yet. For a file about 90MB in size, representing a video clip a few minutes in length at average quality (720p), the complete file can be downloaded in less than a minute using a decent internet connection. If the user quits watching after the file is downloaded, then much of the work of the server and network has gone to waste. Dividing a clip into a larger number of files with smaller sizes could preserve the resources of content delivery networks (CDNs), Internet service providers (ISPs), and cellular service providers.

The third phase, which is the current state of the art is HTTP adaptive streaming (HAS) (also known as, HTTP-based adaptive video streaming (HAVS)). This technique has multiple current commercial examples with different implementation details. The bare bones version requires every clip to have multiple quality levels, where each level is made available on the server as a set of addressable segments (aka, chunks). Each chunk is an independently downloadable, decodable, and playable unit of media that is generated so that information about the past or next fragment is not needed. Chunks of same or different quality represent equal lengths of the video content (e.g., 2, 4, 6, 8, or 10 seconds). The optimal size of the fragments and whether it should be fixed or variable is a topic of research itself [46]. The servers are regular HTTP servers (e.g. Apache) since server-client interactions are all carried over HTTP.

The quality levels can be changed manually by the user through a user interface or automatically by the application. The ability to automatically switch quality levels adaptively based on network conditions is what gives the technology its name: *adaptive video streaming*. This adaptation feature can be either active or inactive by default according to preferences of the server, the client applications, or both.

Figure 2.1 illustrates the most common layout of HTTP adaptive video streaming. The server sends each client video fragments corresponding to a selected quality level. Current solutions use one of two feedback schemes. In the first scheme, the client application takes measurements, makes a decision regarding quality level, and sends the

server a request to raise or lower the video quality. In the second scheme, the client forwards the measurements to the server where the quality level decision is made.

The measurements are usually based on estimating the end-to-end (E2E) throughput or delay and on the device capabilities (e.g., decoding capability and screen resolution). The module that is responsible for making the quality request decision is called the quality controller, the rate adaptation algorithm, or the adaptation logic module (ALM)). We will use these terms interchangeably through the rest of this dissertation. The ALM is implemented in the video client application that supports HAS whether it is a standalone player or a video plugin for a web browser. Current commercial rate adaptation solutions are all heuristic. Feedback is sent periodically, with a period known as the adaptation window. The adaptation window is selected to be a multiple of the size of video chunks. For example, if chunks are four seconds long, then the adaptation window length will be 4, 8, 12, or 16 seconds. The server may respond to this adaptation feedback or ignore it according to the features supported for the playing clip. The server-supported features for each video clip are provided as meta-data before downloading begins. Different commercial solutions have different standards to manage exchanging and parsing meta-data of the video clips before starting to download chunks.

Since each chunk is saved as a separate entity, a meta-data scheme is necessarily. The scheme is used to assemble and represent the required information about the available quality levels and playlists into a single downloadable file. This file is initially

downloaded by the client application so that the application learns the available features supported by the server side. The meta-data file contents include the size of chunks, available video quality levels, and required decoding information. Several standards have been proposed to tackle the challenges related to these representation and storage problems (e.g. the optimum size of the chunks and the best encoding scheme). Examples of current solutions include Microsoft IIS Smooth Streaming [5], HTTP Live Streaming (HLS) by Apple (an IETF standard) [6], HTTP Dynamic Streaming (HDS) by Adobe [7], and Dynamic Adaptive Streaming over HTTP (DASH) (the ISO standard) [47, 48]. DASH focuses on presenting a unified standard packaging scheme that can work for all commercial solutions if adopted. Under DASH, server and client applications developers would not need to customize their solutions to a specific HAS solution. However, none of these solutions or standards proposes an algorithm for the adaptation logic module (ALM).

According to the experimental investigation by [45], the rate adaptation algorithm executed by the client side depends mainly on a bandwidth estimation process. The failure of the bandwidth estimation technique to detect the current end-to-end (E2E) conditions of the network can cause delayed feedback to the server. In this case, the adaptation decision takes longer to be determined and executed. The term *available bandwidth* is used to refer to the actual achievable E2E throughput. This differs from the definition of bandwidth typically used for wireless links. For the sake of clarity for the rest of this dissertation, we will use the term *achievable throughput* when we discuss the

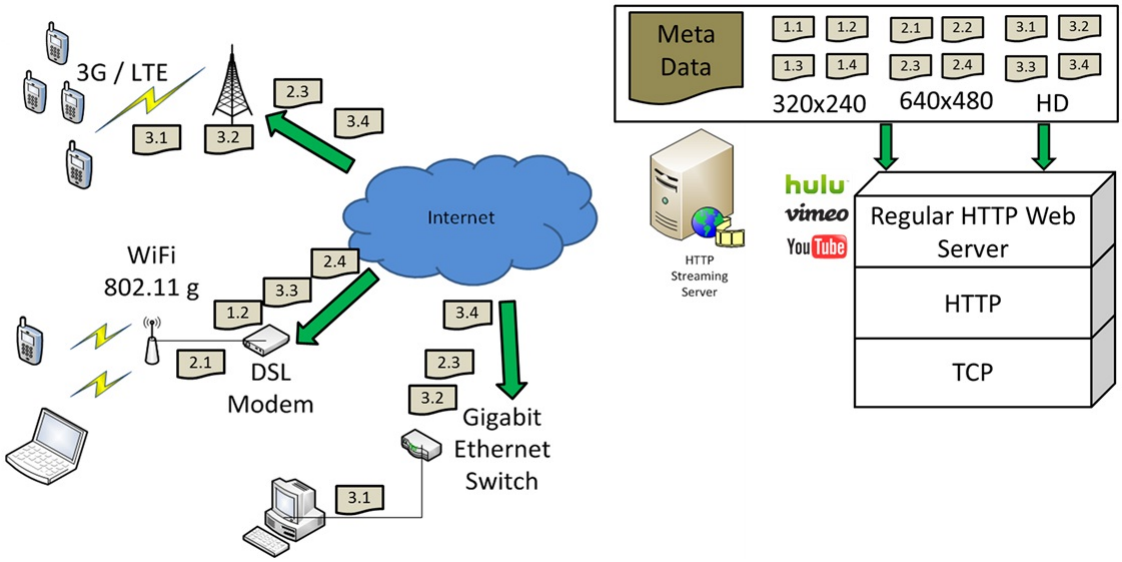


Figure 2.1: Adaptive HTTP video streaming state of the art.

E2E throughput and the term *available bandwidth* only when discussing the actual capacity available on wireless links.

The authors of [45] found that overestimating the achievable throughput leads to additional congestion at routers, a higher percentage of dropped packets, longer rebuffering delays, and more frequent re-buffering. Overestimating the achievable throughput is difficult to fix due to long adaptation process delays. On the other hand, being extra conservative in estimation leads to under-utilization of connection resources. According to [45], the Akamai scheme was running, on average, two quality levels below the attainable level under the given conditions.

According to the consumers survey report [49], the major elements of user satisfactions are initial delay, buffering (period and frequency), and video quality. The survey results report that after more than ten seconds of initial delay, 50% of consumers

quit watching the video. This indicates that using an initial buffering mechanism to mitigate the effects of E2E jitter may lead to significant loss of viewers. Therefore, determining an appropriate initial buffering period is important.

Viewers' perception of quality and patience also changes over time. In 2011, survey results suggested that rebuffering time comprising 1% of the total viewing time might reduce the viewing period by three minutes, on average. In 2014, viewers who received the same 1% rebuffering time reduced their watching time by an average of 14 minutes. These results are confirmed by a large study in [50], where researchers from the University of Massachussetts, Amherst and Akamai (a CDN provider) have studied the influence of buffering on viewers. On average, a consumer who receives 1% of buffering delay tends to watch 5% less of the desired video, and the viewing period continues to decrease as the buffering increases. Also, it has been found that viewers tolerate a longer initial buffering for long videos (e.g. a TV episode or a movie) than for short clips (e.g. a short clip on YouTube).

2.2.2 HTTP Adaptive Streaming (HAS) Over Wireless Access Networks

Since the YouTube service [51] began in 1995, the volume of IP traffic devoted to video streaming has increased dramatically. Over the last ten years, wireless access networks have become much more widespread. Despite this large scale use of wireless access

networks, the rich literature about TCP misbehaviors over wireless links, and the significant volume of video traffic with respect to the total mobile traffic as highlighted earlier, HAVS over wireless access networks has not received adequate attention. The problem has a significant influence on the overall performance of the Internet, consumer satisfaction, and changes to legacy economic models in media production and delivery services. The market for legacy TV services, media broadcast, and video content delivery services has been restructured over the last ten years. Certain services providers have vanished and new business models have emerged due to the widespread deployment of wireless access networks and the evolution of video streaming services.

This motivates us to focus on HTTP adaptive video streaming in wireless last-hop access networks. We are specifically interested in the IEEE 802.11 infrastructure standard mode (a.k.a. WiFi). WiFi not only dominates access to the Internet at residential locations, hotspots are commonly found in commercial venues (e.g., restaurants, hotels, malls), university campuses, government buildings, and city centers.

There are two major problems in using HAS over WiFi. The first issue is that HAVS runs over the Transport Control Protocol (TCP), which is not designed for this type of data. Instead, TCP was designed to provide reliable transfer of data streams that are not tolerant of packet loss. Video data is by nature tolerant to packet loss. and the decoding process includes techniques to mitigate different kinds of loss. Hence, the reliability provided by TCP is not desired as it imposes additional delay. Although same issue still

exists with TCP over wired links, TCP misbehavior over wireless links exacerbates the problem.

Unfortunately, TCP was designed with the assumption that the only source of packet loss is congestion at the routers. In wireless networks, TCP suffers from non-congestive losses such random loss, burst loss, and packet re-ordering. Several proposals exist to address these pitfalls [52–54]. TCP behavior affects video streaming in several ways. For instance, TCP over wireless may suffer from delays more frequently because non-congestive losses may trigger congestion avoidance. This contradicts the low latency required by video streaming in order to meet the playback deadlines of video frames. Preventing inappropriate triggering of the TCP congestion avoidance mechanism may result in a better HTTP video streaming experience.

The second problem is that the random nature of wireless channels and links may degrade the performance of adaptive video streaming applications in wireless last-hop networks (e.g. WiFi). The achievable end-to-end throughput in these networks is less stable and less predictable than in wired networks due to fluctuations in the wireless channel and link. This is strongly connected to the first issue since random or burst packet losses will deceive the TCP congestion avoidance algorithm.

A chain of incorrect decisions can propagate easily through the TCP/IP stack. For example: a frame transmission fails due to a sudden jump in bit error rate (BER) and retransmissions fail at the MAC layer, such that one or more packet(s) are dropped. When TCP acknowledgements are not received, the sender will retransmit the packet(s)

and shrink the sending window. Although the wireless link may have recovered immediately after the burst loss, the TCP window will take much longer to recover. This slow TCP window recovery wastes link opportunities and causes the ALM module to underestimate the the achievable throughput.

The main requirements for video streaming are high data rate and low latency. When a wireless link exists on an end-to-end path, it is expected that the bandwidth estimation scheme will perform more poorly as explained above. The random nature of wireless links increases the problem complexity due to noise, mobility, and MAC layer contention. Even prior to HAS, attention was given to the challenges associated with streaming video over wireless links. For example, the authors in [55, 56] have tried to address the challenges of video streaming over wireless ad hoc links. They suggest that for optimum E2E performance, sub-problems including video coding, reliable transport, and wireless resource allocation should be optimized jointly. In other words, a cross-layer solution is preferred. Several cross-layer solutions for video streaming over wireless have been presented including [57, 58].

2.2.3 Previous Work and Classification

Below is a summary of the most significant previous publications on video streaming. Although our focus is primarily on HTTP adaptive video streaming (HAVS) over WiFi links, we include some work that does not directly address HAVS or wireless because

we believe that the proposed solutions are portable. In addition to classifying previous proposals based on whether they have addressed video streaming in HAVS context or not, we also distinguish proposals that use cross-layer design from legacy, layered approaches. Furthermore, we investigate previous attempts that have used Markov decision processes (MDPs) and reinforcement learning, whether they present an offline model-based solution or an online model-free solution. Finally, we also classify the solutions according to the deployment location(s) (e.g., server side, client side, or intermediate nodes). Table 2.2 summarizes our findings.

2.2.3.1 Non-HTTP Adaptive Video Streaming over Wireless

In [59], the authors present a module called the Cross Layer Module (CLM) which consists of four components. The first component is the channel estimator, which estimates the quality of the IEEE 802.11 Medium Access Control (MAC) layer by monitoring the re-transmission counter and then forwarding the estimated status to the buffer manager. The second component is the buffer manager, which manages the transit buffer to make sure that it does not overflow or underflow. The third component is the transcoding controller, which adjusts the transcoding rate to switch to the rate best suited to the the channel conditions as estimated by the channel estimator.Adaptation is achieved by choosing between levels based on empirically estimated thresholds. The last component is the error-handling controller, which adjusts the parameters of the Forward Error Correction (FEC) and Automatic Repeat-Request (ARQ). The controller adjusts the

parameters to match the video frames time out (the threshold after which the frame is useless upon reaching the destination) and and the expected round trip time of the MAC layer frames, which carry the payload data. The proposal addresses video streaming over the RTP protocol, however some ideas are portable to HTTP video streaming. The performance of CLM has been evaluated using the ns-2 simulator [9] and the Evalvid video evaluation tool [60]. Also, partial implementation of the module using an IEEE 802.11 based testbed is provided in [61]. The testing setup is a server-client video streaming session that is implemented on two laptops in ad hoc mode.

Another attempt to provide a cross-layer framework is in [57], which offers a cross-layer proposal for video streaming in wireless ad hoc networks. The proposal aims to provide a best-effort quality of service (QoS) to stream videos in an ad hoc wireless environment by combining video rate adaptation and video-aware ad hoc routing at the network layer. The routing protocol used is called video DSA (VDSA). Basically, intermediate nodes that are willing to buffer video frames announce themselves during the route discovery phase so that video friendly routes are marked. In case of ARQ requests received from the destination, the volunteer nodes handle retransmitting the desired frames on behalf of the original source. A copy of the ARQ request is still forwarded to the original source. Evaluation has been done using the ns-2 simulator, and simulations show that the overall received video quality has been improved.

In [58], the authors present an end-to-end cross-layer solution that modifies both the application layer and the link layer. It depends on using priority-based ARQ and an

unequal error protection scheme. For unequal error protection, the FEC is applied on the level of packets instead of bits, after classifying every packet into one of four different classes according to its importance with respect to the video frame's content. This is customized based on the encoding scheme in use. The FEC parameters are assigned differently according the packet's class. The paper's authors use the RTP protocol running on top of the UDP protocol. The performance is evaluated using a customized socket-level simulation. The provided details are insufficient to replicate the results. Although the proposal addresses video streaming over the RTP protocol, some ideas are applicable to HTTP streaming over the TCP protocol.

The authors in [62] address receiver buffer size requirements for video streaming over TCP using memory-constrained devices. The authors propose an analysis to determine the right buffer size at the receiver and verify their model with the ns-2 simulator [9]. They claim that the use of the right buffer size can mitigate drawbacks of using TCP to run video streaming applications. This is because the rebuffering delay requirement for a long video stream is determined by the congestion control algorithm of TCP. The analysis shows that the receiver buffer requirement can be determined by the network characteristics and the desired buffer underrun probability (or re-buffering frequency). Simulation results validate the analytical model.

2.2.3.2 HTTP Adaptive Video Streaming (HAVS) over Wireless

The authors in [63] try to understand the solution that Akamai implements for video services in their CDN through reverse engineering. They choose the Akamai scheme as a representative due to similarity between it and other technologies. The Akamai end-to-end solution itself is already implemented on top of several technologies including IIS Streaming, Adobe HDS, and Apple HLS. An open source testbed including a Linux-based client, a network emulator package, and a kernel module to shape the download link bandwidth are used in the reverse engineering process. The results show that the adaption process suffers a long latency. In some cases, a 50 second transient period was required in order to match the actual available bandwidth. This is a significant period of time with respect to the full length of many video clips. Moreover, the emulator may actually be overestimating the stability of wireless links, meaning that performance in a realistic scenario could be even worse.

In [64], the authors present an optimization algorithm called Intelligent Switching-based Adaptive Video Streaming (ISAVS). It addresses flaws in IIS Smooth Streaming [5] with respect to wireless access networks. Current smooth streaming uses a heuristic method, where the client examines the real-time network conditions, CPU capability, and screen resolution after receiving a fragment, and then decides to request higher or lower bitrates from the server for the next fragment. The authors claim that evaluating the rate to be chosen must be a function of both the number of fragments/frames available

in the receiver buffer and the network conditions. Thus the proposed algorithm takes into account the buffer status at the receiver. The algorithm has been evaluated using the QualNet 4.5 simulator [65] and the EvalVid tool [60]. An IEEE 802.11 simulation is used to simulate the video stream with constant bit rate (CBR) traffic in the background. The proposed algorithm clearly reduces the occurrence of playback interruptions, which greatly degrade the Quality of Experience (QoE).

From the proposals discussed above and others, we classify cross-layer solutions that make adaptation decisions at the server into three main categories:

- Proposals that use feedback from the client on the E2E achievable throughput in order to adaptively control the sending rate on the server side. The source sending rate is adapted by switching to a different pre-encoded quality level or by tweaking the transcoding parameters on-the-fly. (On-the-fly transcoding is rarely used for video on demand.)

- Proposals that use feedback from the client to modify parameters at the intermediate nodes (e.g., routers). For example, modifying the error control mechanisms (e.g. Forward Error Correction (FEC) or Automatic Repeat Request (ARQ)) at the intermediate routers or server.

- Proposals that use feedback from the client in order to modify the wireless parameters (e.g. physical channel rate, sending power, or modulation scheme) of the wireless last-hop with or without coordination with the streaming server.

On the other hand, cross-layer solutions that make adaptation decisions on the client-side can be divided into:

- Proposals that depend on reading meters from network stack layers below the application layer and making a decision at the application layer (e.g., request lower resolution). These are known as context-aware or network-aware applications.
- Proposals that modify network protocols (within the client's TCP/IP stack) to become aware of video traffic. For example, redesigning the Automatic Repeat Request (ARQ) algorithm on the client-side with respect to video streaming. Video contents are by nature tolerant of slight corruption so sending an ARQ request may be less useful than passing the corrupted packets to the application layer within the playback time constraints.

Generally, other cross-layer paradigms described in [66] can be used to improve the performance of video traffic over wireless links, but some are easy to implement while others have high implementation costs.

2.2.3.3 Learning-based Adaptive Video Streaming

Model-based Network Conditions

In several publications [67–75],Van der Schaar et al. presented cross-layer optimization frameworks that rely on model-based Markov decision processes (MDPs). For example, [67, 68, 70] present a theoretical framework for cross-layer joint optimization using

Markov decision processes (MDPs). The main advantage of this approach over the well-known Network Utility Maximization (NUM) [76] approach is that MDP-based approaches take into account the future consequences of actions taken at the current time, while the NUM approaches provide myopic solutions that are shortsighted. The framework aims to optimize traffic over a single-hop wireless link but is portable to other scenarios within limits.

The framework models each layer in the TCP/IP stack as a sub-model having its own state space, action space, and reward function. The set of actions that affects each layer's model state is composed of local (layer-specific) actions and of external actions (committed by other layers). The proposed approach assumes that all layers are synchronized, hence the entire system can be described with multidimensional, discretized state space. Then, the layers are solved jointly using layered value-iteration. The algorithm solves each layer's value function iteratively.

The authors have proposed three scenarios under their framework. The first scenario assumes each layers sub-model uses only its own parameters without access to other layers. The second scenario assumes direct access to local variables between layers. The first and second scenarios do not follow the constraints of the layered architecture. To provide an alternative that does not break the layered design of the TCP/IP stack, they have presented a third scenario, where the local parameters are only accessible through a message passing interface.

In [77], the authors use the aforementioned framework in alive video transmission

application over a slow-varying flat-fading wireless channel. The goal is to adapt the sending rate of the transmitter side assuming that a dynamic video flow arrives at the sender's post-encoding buffer, before the data is divided into packets, and scheduled to be sent over the wireless link.

The objective is to optimize the perceived quality while minimizing the total transmission energy. Parametric models for the PHY, MAC, and APP layers are proposed but the transport and network layers are neglected. This model-based scenario assumes that each layers state transition matrix is known a priori. To solve the proposed joint model, [77] presents a realtime version of the traditional offline dynamic programming value-iteration algorithm. The realtime algorithm, known as realtime dynamic programming (RTDP), makes realtime decisions to execute and update the policy simultaneously. In addition, a modified version of the RTDP algorithm, known as Adaptive RTDP (ARTDP), is presented to gather measurements (e.g., channel conditions) and update the system model parameters dynamically during runtime.This version addresses scenarios in which the system dynamics are unknown, partially unknown, or non-stationary. In this work, the authors assume that a sub-module in the application layer handles the optimization centrally after being provided with full access to all layers' parameters. In [69], a modified RTDP version called layered RTDP (LRTDP) is presented which solves each sub-model locally at each layer, while gaining necessary external information through message exchange.

In [73], the same team investigates the use of a Multiuser MDP (MU-MDP)

policy-making approach to deal with multimedia transmission in cooperative multi-hop wireless network. The global utility function in this case optimizes both the perceived video quality at each node and the overall usage of energy. In such cooperative networks, intermediate nodes offer forwarding services according to a reward model that governs the tendency to forward the received packet or to save energy, assuming that if everyone drops packets instead of forwarding then only nodes, which have direct links to teach other, can communicate. Therefore, the algorithm to solve the MU-MDP is basically a pricing-based resource allocation algorithm that governs how much energy each node is willing to spend on forwarding multimedia data to its receivers. The forwarding cost is represented in terms of the consumed energy and the future expected congestion due to forwarded data.

The authors assume a single sink node that is the final destination for all the traffic generated from various sources within the network. Links are accessed in a TDMA manner, while control information (e.g., information required by the MU-MDP algorithm) is exchanged between the sink node and other nodes via a random access algorithm, that is a modified version of the RTS/CTS algorithm used on CSMA/CA channels.

In [75], Van der Schaar et al. go back to the problem defined and solved in [71] to present a multiuser version of the same problem, where users share network resources and optimize the scheduling and sending of video packets by implementing the aforementioned cross-layer framework at each node. However, a global optimal solution

is desired across all nodes, not merely an optimal solution for each node individually. A crucial challenge in solving the problem in distributed fashion is that users' actions are coupled (i.e., each action made by a user alters the network environment). The major contribution is a new approach to decouple the users actions in this MU-MDP scenario, so that each node can locally find an optimal foresighted cross-layer solution (e.g., an optimal packet schedule given network conditions). Both an offline and an online learning algorithm are presented to solve the system.

In contrast to all their aforementioned efforts, in [72, 74] Van der Schaar et al. take into account the transport layer dynamics. Specifically, they propose a learning-based TCP congestion window controller. In [72], the authors modified the online model-free learning algorithm to find the optimal sending rate in [71]. As a result of this modification, the algorithm into account the delay components with respect to the multimedia data in selecting the congestion window, resulting in a media-friendly TCP protocol. The major modification is altering the approach to consider finite horizon version of the MDP problem, hence the algorithm considers future gains and losses with a limited foresight of K steps. Simulation results showed noticeable improvement in perceived video quality in real-time streaming scenarios, where the playback delays are highly restricted. The authors assumed fixed-size RTP protocol packets fed by the APP layer into the TCP send buffer. The solution not only improved the received video quality, but also maintained fairness between the intended video flow and other TCP streams simulated in the background.

Since providing the MDP-based TCP congestion controller with sufficient statistical knowledge about the multimedia data and the network environment is challenging, Van der Schaar et al. in [74] modeled the problem as an incomplete observation MDP problem (a.k.a. partially observable MDP (POMDP)). After formulating the problem as a POMDP, they used an online reinforcement learning algorithm to solve the problem. The online algorithm improves the received quality of simulated real-time multimedia traffic, while maintaining fairness with other background TCP streams.

Online Learning of Network Conditions

The most recent significant work on HAVS using reinforcement learning are the publications by De Truck et al. found in [78–81]. Also, in [82], they propose heuristic adaptive logic on the client side combined with heuristic, video-aware priority queues at intermediate proxy nodes. In [79], they presented an adaptive video streaming client using reinforcement learning. They used a state function of two dimensions representing the current buffer level and the current estimated bandwidth and a reward function consisting of three components: video quality, frequency of quality changes, and buffer underrun events. A classic Q-learning algorithm [83] was used, and the system was simulated with the ns-3 network simulator using 3G bandwidth traces and video traces from the DASH test vectors dataset [84]. The proposed solution outperformed the Microsoft Smooth Streaming (MSS) commercial solution under the simulation settings.

MSS relies on a heuristic bandwidth estimation algorithm that also monitors the buffer level.

The same authors presented an enhanced algorithm known as the FA-Q-learning adaptive video client in [81] that improves the exploration ability of the traditional Q-learning algorithm by combining a value-difference based exploration algorithm with the Softmax action selection algorithm [85]. The proposed client controller has been tested using ns-3 with background traffic and preset bandwidth configurations. Authors reported that the FA-Q solutions outperformed the traditional Q-learning only client.

In [80], the same research team improved their FA-Q-learning adaptive client by adding a heuristic layer to the reinforcement learning to exploit some domain knowledge. In particular, buffer-size panic thresholds (an idea already used in the MSS solution) were added to override decisions when the buffer level drops or grows beyond these values. In general, the addition of heuristic tunning to the reinforcement learning approach reduced the buffer over-filling problem.

Furthermore, same group of researchers has extended their single-client RL-based solution to cover multiple client scenarios in which each video client runs an RL agent-based quality controller. The multi-agent approach described in [78], uses the FA-Q-learning algorithm equipped with the enhanced exploration feature to optimize the global reward function instead of the usual local function. Since optimizing for global fairness requires exchanging local information between the agents (e.g., locally estimated available bandwidth and local quality decisions), the authors proposed

adding a proxy server between the server and client. The proxy is responsible for communicating information and coordinating the exchange of reward estimates. The solution was tested by ns-3 simulation, and results show that it is able to increase fairness remarkably compared to a scenario in which each client runs the FAQ-learning agent optimizing only the local utility function or in which users run the baseline MSS solution.

It is important to note that all aforementioned learning-based work by De Truck et al. did not address last-hop wireless link scenarios directly, except for [79], in which available bandwidth was simulated using a 3G network trace dataset. In the rest of work, the variable bandwidth pattern was generated synthetically and was not as challenging as could be presented by WiFi or 3G links in real world scenarios.

Table 2.2: Classification of previous work on adaptive video streaming.

Criterion	Server-based or Router-based	Client-based	Server/Routers + client	Cross-layer	Offline Model-based (MDP or POMDP)	Online learning (RL)
Non-HTTP streaming						
[61], [59]	✓			✓		
[57]	✓			✓		
[58]	✓			✓		
[62]		✓				
[67, 68, 70]	✓			✓	✓	
[77]	✓			✓	✓	✓
[71]	✓			✓	✓	✓
[73]			✓	✓	✓(MU-MDP)	
[75]			✓	✓	✓(MU-MDP)	✓
[72]	✓			✓	✓(Finite-Horizon)	✓
[74]	✓			✓	✓(POMDP)	✓
HTTP streaming						
[64]		✓				
[63]		✓				
[78–81]		✓				✓
[82]	✓			✓		

Finally, we mention some beneficial reviews and surveys. The study in [86] provides a rich survey on video streaming over IEEE 802.11, not focusing necessarily on HTTP-based proposals. Also, the study in [87] presents insights regarding the different metrics (e.g., objective, subjective) that can be used to judge the quality of experience (QoE) perceived by viewers in the HAVS context. Similarly, the study in [88] categorizes various subjective and objective metrics with a focus on metrics for video on demand (VoD) services. Given the importance of YouTube on the Internet, researchers have given it a lot of attention, studying relevant streaming solutions, QoE metrics, and performance over various wireless technologies (e.g., WiFi, cellular), e.g. [89–93].

2.3 Conclusions

In this chapter, we reviewed the literature related to research presented in this dissertation. In the first section, we reviewed and classified previous work on experimental tools that enable implementation-based networking experiments, and we discussed the lessons learned from existing tools including desired features for new tools as well as design and implementation concerns that should be considered when proposing a new tool. The second section of the chapter presented the basic terminology and background related to adaptive video streaming. We reviewed the literature on video streaming focusing on the previous work that is most relevant to HTTP adaptive

video streaming over wireless links, especially IEEE 802.11 networks. Also, we classified prior work used similar approaches to those that we adopt in chapters 4 and 5.

Chapter 3

The Flexible Internetwork Stack (FINS) Framework: Current Research Status[1]

This chapter describes a new prototyping and implementation-based experimental tool, called the Flexible Internetwork Stack (FINS) Framework. The FINS Framework is a software framework which aims to lower the barrier for implementation-based wireless network experiments across the network stack by providing access to functions that are usually implemented within the operating system's kernel. The FINS Framework is inspired by the MANIAC Challenge competition [**?**], which provided insight into the real performance of the TCP/IP stack in wireless ad hoc networks. The technical challenges faced to implement the MANIAC Challenge API and the logistical challenges

[1]The work described in this chapter was previously published in [8, 94–96]. This material is based upon work supported by the National Science Foundation under Grant Nos. 0916300 (Virginia Tech) and 0916283 (Bucknell University).

of conducting the MANIAC experiments motivated us to develop the FINS Framework for use by the network research community.

The chapter is partitioned into four sections. First, we discuss the architecture we decide to adopt for the FINS Framework. Then, we present our implementation schemes from both networking and software development points of view. Next, we provide a thorough evaluation for the framework system performance in terms of achievable sending and receiving rates as well as loss ratio due to any processing delay. In addition, we provide qualitative experimental scenarios, where the framework has been deployed in experimental or real networking scenarios to prove its compatibility or to verify specific features such as supporting legacy applications. Finally, we conclude with a summary of the contributions made till the end of the project and suggestions for future directions.

3.1 System Architecture and Design

In this section, we discuss the architecture we chose for our new tool, guided by the architectural classification we provided in Section 2.1.1. We not only followed these guidelines, but also fulfilled most of the requirements and specifications discussed in Section 2.1.4.

3.1.1 Design Goals

We start this section by highlighting the system design goals (aka, high level system features) that we have extracted from reviewing the literature and studying the possible users requirements. The FINS Framework was created with a specific set of design goals in mind. The focus of these was to ensure that the result would be useful to numerous research groups, require minimal work to get started, and solve the problems noted earlier. A list of the specific goals of the FINS Framework follows.

- Works with existing hardware: A primary focus of the FINS Framework is to support commodity and existing equipment that researchers already have and use or can obtain easily. Ideally, the tool should be portable with minimal effort to new hardware as it becomes available.

- Userspace development: As previously discussed, creating userspace applications is easier than developing in kernel space. In addition to requiring less development time, userspace applications can leverage existing high-level libraries and developer knowledge. Userspace programs are less likely to cause a system crash, which saves time and frustration. Code may be portable from network simulations (e.g., ns-2 and OPNET). All of these advantages lead to faster develop- ment with less initial overhead.

- Ready to run release: Another major requirement is that the tool be ready to run

upon download. We define this as includ- ing implementations of major network protocols in a state that allows the use of the tool without modification.

- Supports existing applications and protocols: The FINS Framework must support existing applications without recom- pilation or other changes. Limiting the number of applications that are supported, requiring users to rebuild every networking application, or, in the worst case, requiring new applications to be written inhibits adoption. Additionally, implementations of common network protocols should be provided for direct use or modification.

- Handheld computing device support: Unlike laptops, smart- phones and tablets are better suited for truly mobile network- ing experiments, and are more desirable platforms than laptops for certain types of experiments. Handheld devices are often less expensive, more portable, and have a longer battery life than laptops.

- Flexible architecture: The ongoing interest in cross-layer and cognitive networking research requires data and control exchange to go beyond what is easily accomplished in the tra- ditional stack. Protocols may need to communicate in order to optimize system functionality to meet specific goals. The result of these goals is an approach based on a modu- lar networking protocol subsystem that runs in userspace on desktop Linux and Google Android. Everything is written in C for speed and code reuse. To allow for quick adoption, FINS versions of major protocols are provided as well. Finally, exist- ing network applications are supported.

3.1.2 The Framework Structure and Modules

The FINS Framework is an *experimental tool* that belongs to the hybrid architecture category. The major components of the framework are a central module named the *switch* and the *modules* (around the switch) forming a star structure as shown in Figure 3.1-left. The switch transfers network traffic, and framework's intra-communication control messages between modules based on linking information. The linking information is kept in a dynamically updatable table that defines where traffic units go when they leave a module. The framework is designed to be fully-connected where any module can communicate with any other module simply by placing a linking record in the linking table. This approach differs from the traditional stack approach where protocols can only communicate with the protocols that are "above" or "below" them as shown in Figure 3.1-right. Additionally, this approach adds capabilities to easily implement cross-layer protocol design and network data flow manipulation.

The hybrid structure of the FINS Framework builds off of the traditional data-link layer and introduces a flat approach for the network to application layers, while maintaining seamless backward compatibility with legacy network applications. Users of the FINS Framework thus avoid the high cost of re-implementing hardware-specific lower layers and FINS enables experiments that incorporate realistic traffic scenarios, namely real-world applications instead of custom tailored traffic generators. The non-layered nature of the architecture means any module can communicate with any

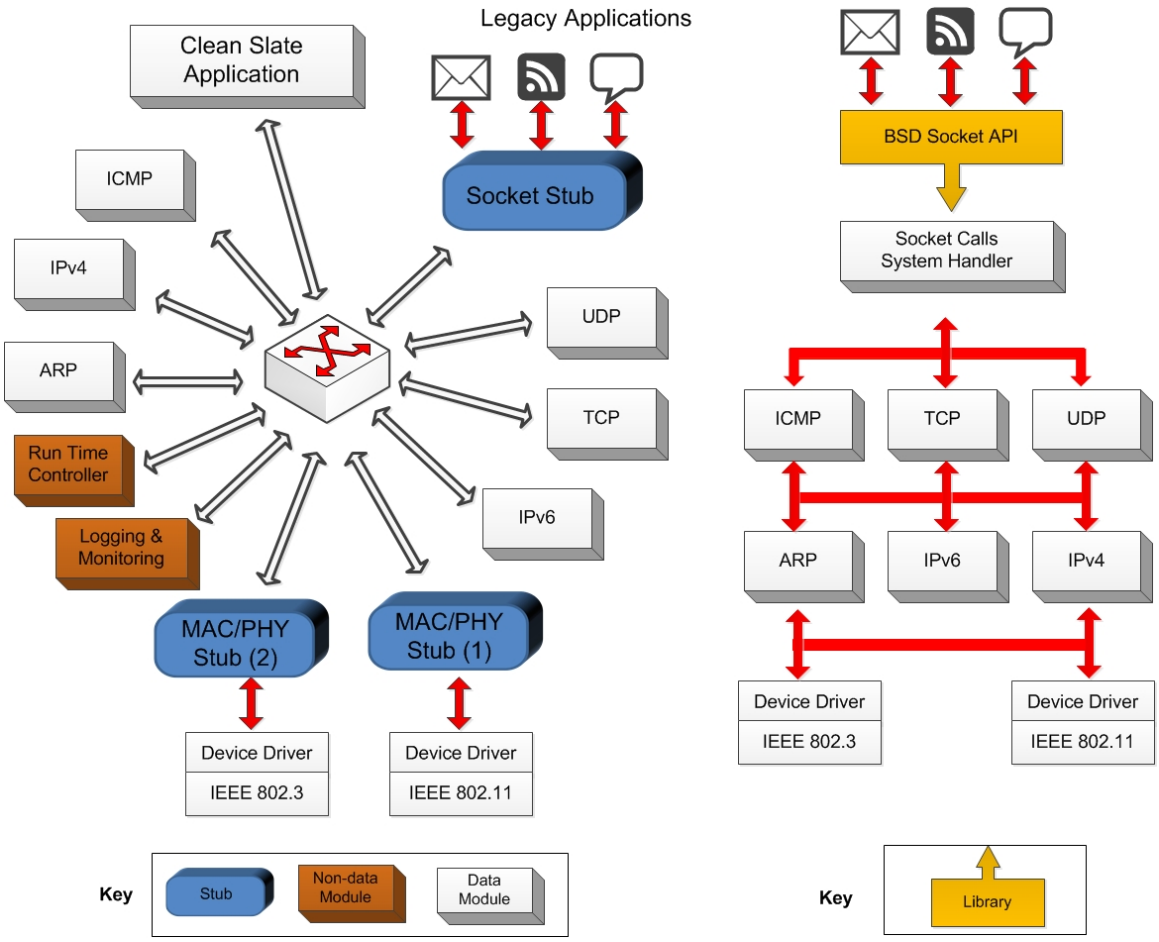


Figure 3.1: The FINS Framework architecture (left) vs the conventional TCP/IP stack (right)

other module, allowing for greater access to *meters* and *knobs* across layers as well as support for cross-layer protocol design.

Our architecture also allows experimenters to implement applications which are connected directly to the FINS Framework without using the BSD or glibc socket API in a clean-slate fashion (right side of Figure 3.1). The ability to implement clean-slate applications allows researchers to explore new experimental scenarios, such as implementing and testing a context-aware application that varies its traffic pattern based on the surrounding network conditions. The ability to connect directly to the framework not only allows context-aware applications through access to the internal conditions of protocols, but also cognitive applications that can control some protocol parameters or even direct management features within the FINS Framework. This is helpful for researchers working on cognitive nodes and networks [97].

Each end-user network node (e.g. laptop, tablet, smartphone) within an experiment is assumed to run a single instance of the FINS Framework. The FINS Framework follows a modular architecture where its core is organized into several modules. There are three categories of modules in the framework: data modules, non-data modules, and stub modules.

- Data modules act on ingress and egress network traffic as it is processed in the node (e.g. ARP, TCP, UDP).

- Non-data modules observe network traffic and interact with other modules; these modules may not act on or modify network data (e.g. logging & monitoring).
- Stub modules are modules that enable integrating the FINS Framework with existing mechanisms outside of the framework (e.g. socket stub, MAC/PHY stub).

The data and the control traces logged by the logging module can be used in different ways to support repeatability and controllability of experiments. Firstly, in order to fix one or multiple parameters of the running modules while studying others, the researchers can use logged traces to regenerate the previously logged behavior of a specific entity in an experiment. Furthermore, for every experimental setup where the FINS Framework is deployed, the egress traffic of one node is the ingress traffic of others. Therefore, the traces collected by the various logging modules can be utilized to fix and repeat the external conditions up to some extent. In implementation-based experiments, there is always a tradeoff between the repeatability and practicality due to the random nature of external conditions (e.g., background noise, interference). Secondly, logged traces can be reused by network simulators and emulators to generate realistic conditions for an experiment.

Modules may vary greatly, but must all follow common guidelines and interface with the switch in the same way. They are meant to interact with other modules through loosely coupled generic interfaces and to be implemented at differing levels of granularity, such as at the protocol, algorithm, or library level. This aligns with the concepts of generic interfaces and code reusability mentioned previously.

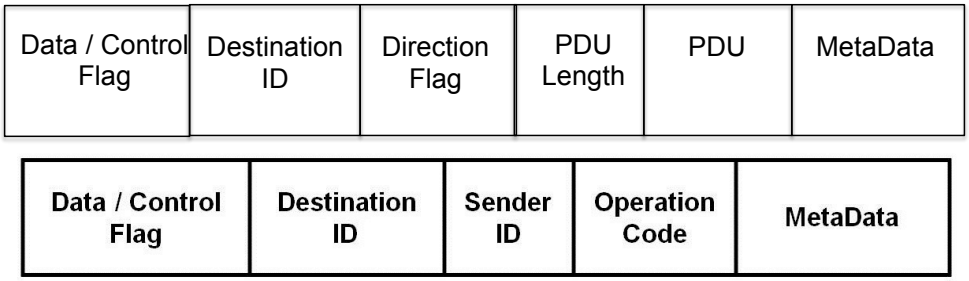


Figure 3.2: Structure of FINS data frames (top) and FINS control frames (bottom). Data frames carry network data in the PDU field as traffic passes through the framework, while control frames are used to communicate between modules through the operation code and metadata fields.

3.1.3 FINS Frames

Communication between modules is accomplished through two types of frames based on the traffic type: *data frames* and *control frames*.

- Data frames encapsulate ingress or egress network traffic that flows through the framework (top of Figure 3.2).

- Control frames encapsulate messages exchanged between the modules in the framework and do not carry network traffic; they are typically used for management and supervisory functions (bottom of Figure 3.2). The control messages scheme is the key to realize distinguishing features of the FINS Framework such as the support of meters and knobs, the monitoring and logging ability, and the reconfigurability of the networking stack. The internal structure, subtypes, and the role of control frames to support the framework features are explained thoroughly below.

Data and stub modules are able to send both data and control frames; however, since non-data modules may not act on or modify network data they can only send control frames. Both data and control frames share a first field used to differentiate frame type, followed by a second field holding the value of the destination ID. Subsequent fields vary depending on frame type and other parameters.

The FINS data frame consists of six fields. The third field is called the direction flag. It is presented to allow backward compatibility with the legacy layered architecture, where a direction flag processed internally by the destined protocol module to determine whether the frame is on the outgoing or incoming direction with respect to the conventional layers. This is helpful for reusing old protocol implementations with minimum modifications. In contrast, newly implemented protocol modules depend on link ID and modules IDs configured by the user in the framework configuration file. The next field is the length of the PDU in bytes, which is followed by the PDU data field, where the encapsulated data is stored. The FINS data frames encapsulate conventional TCP/IP stack' data units such as UDP datagrams, TCP segments, and IPv4 packets. The last field is called the Meta-data, which is used to exchange special key-value pairs between the modules. This is used to carry additional management and control information between layers associated with the encapsulated data.

The control frames have a more flexible format, while sharing the first and second fields with the data FINS frames. The third field is the sender ID of the frame, which is necessary in case of control frames, because processing of the frame depends in part on

Table 3.1: A list of possible Operation and their use.

Operation	Use
Parameter Read Request	Request the value of a module's parameter
Parameter Read Reply	Respond with the value of the module's parameter
Event Listen	Register to be notified when an event occurs in a module
Event Alert	Notify a listening module the event occurred
Parameter Write Request	Request to change the value of a module's parameter
Parameter Write Confirmation	Confirm whether the parameter's value was changed
Procedure Execute Request	Request the execution of a procedure
Procedure Execute Reply	Respond with the result of the procedure
Error Log	Report that a non-termination error has occurred

which module it comes from. The next field is the operation code which determines what operation the receiver module should execute. The last field contains metadata, including special key-value pairs that are associated with the sender, operation code, etc. Control frames expose *meters* for other modules to measure through two modes: *polling mode* and *event-driven mode*. Polling mode is performed using the Parameter Read Request and Parameter Read Reply operations, while the event-driven mode is accomplished using Event Listen and Event Alert. The other half of the concept, *knobs* through which modules can be affected, is realized through Parameter Write Request and Parameter Write Confirmation. Nine operations are currently defined and supported in the available releases of the FINS Framework. These operations and a brief summary of their uses are listed in Table 3.1.

3.1.4 Switch and Linking Table

All modules within the framework communicate using frames that pass through the switch, with the path of frames controlled by the linking table. This table, which is analogous to a routing table, specifies the receiver module(s) of a frame based on the sending module and virtual link it is sent over. This allows both the user and module designer to set and change how data flows through the system, as both can reconfigure the stack through the linking table. Virtual links are typically differentiated by the type of communication, traffic behavior, and/or direction through the protocol suite (egress or ingress). For example a module might have separate links for sending control frames and data frames to the same module or possibly separate links to send UDP and TCP traffic to different modules. Finally, the use of a central switch adds the ability to perform runtime reconfiguration through changing the linking table and the effect of "pausing" the stack by halting the routing of frames.

The design of the switch and the linking table are closely coupled to the objectives of the FINS Framework and embody some of the tradeoffs that were made to support these goals. We decided on an architecture with a central switch to provide maximum flexibility, at a small cost of additional rebuffering. Flexibility was favored again by dynamically allocating module IDs at configuration and load time instead of relying on static allocations. FINS also makes the main thread responsible for supervising and

manipulate the linking table based on user inputs, in order to support runtime reconfigurability.

3.2 Development and Implementation

In this section, we discuss the implementation details of the FINS Framework different versions (e.g., v0.8, v0.9, v1.0) through the latest unofficial release, v1.1, which currently runs on Linux and Android platforms. All versions are available for downloading through `http://finsframework.org/`. The discussion focuses on the three major parts that implement the framework: the *FINS Framework core process*, the *FINS Framework socket stub module*, and the *FINS Framework MAC/PHY stub module*. In version we strove to reproduce the existing capabilities offered by the traditional stack with the goal that subsequent versions would exhibit improved support for configurability at the MAC/PHY layer and more interactivity through additional modules. As such, the switch and modules for the socket stub, Real-Time Manager (RTM), ICMP, TCP, UDP, IPv4, ARP, and the MAC/PHY stub have been implemented and connected as shown in Figure 3.3.

3.2.1 Major Processes and Modules: Version 1.0 and Beyond

The FINS Framework is written in C and implements most of the platform in user-space on top of the Linux kernel. The following two platforms served as target platforms:

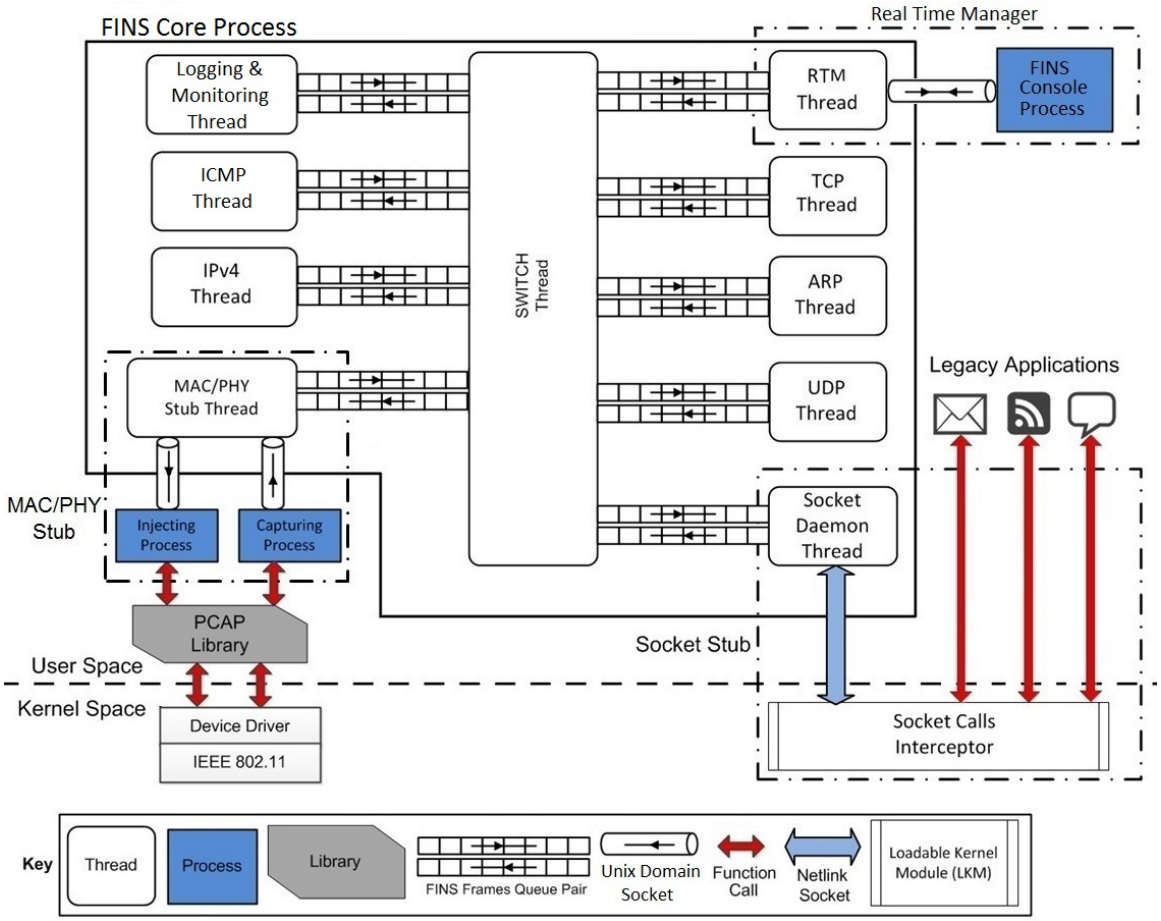


Figure 3.3: FINS Framework implementation and flow of data.

- Laptop computers running Ubuntu 12.04 and Linux kernel 3.2 with an IEEE 802.11 wireless interface,
- and Nexus 7 tablets running Android 4.2.2 and Linux kernel 3.1.

Implementing most of the FINS Framework in user-space means users avoid kernel-space programming, are able to reuse existing libraries and/or FINS Framework code, and recover more quickly from mistakes, collectively speeding development. Finally, C is portable to Android handheld devices, as the Android OS is built on top of a version of the Linux kernel and has built-in support for running native applications.

3.2.1.1 FINS Framework Core Process

A majority of the components in the architecture are implemented as part of the core process, which is a single, multi-threaded process in user-space. The switch is implemented as a single thread and each module is implemented using at least one thread. The modules and switch interact through pairs of input (switch-to-module) and output (module-to-switch) queues, with a pair associated to each module. A module is expected to receive FINS frames through its input queue, process them, and send any response or internally generated frames through its output queue, thus providing a unified interface between the switch and each module.

The switch is expected to read frames from the module output queues using a round robin mechanism, determine the receiver module(s) for each frame based on the

configuration stored in its linking table, and push the frame onto the corresponding module's input queue. The linking table should be configured with all ingress and egress data traffic paths as well as control traffic paths between modules. Table 3.2 shows an example linking table configured to rebuild the traditional stack using the modules presented in Figure 3.3.

The linking table determines a frame's receiver(s) using its sender and destination ID. For a simple example, consider the case when the MAC/PHY stub thread intends to send an Ethernet frame, but lacks the corresponding MAC address Figure 3.4:

C1. The MAC/PHY stub thread recognizes it needs a MAC address and creates a Read Parameter Request control frame with the appropriate metadata to communicate with the ARP module. The MAC/PHY stub module inserts the frame into its output queue with the destination ID set to its link ID associated with ARP traffic (02).

C2. The switch thread reads the frame from the MAC/PHY stub output queue and uses the sender and destination ID (02) to search the linking table for the receiver module IDs (7 and 8), since there are multiple destinations the frame is copied to and the switch pushes one into each input queue of the receiver modules (ARP and Logging & Monitoring).

C3. The ARP thread reads the frame from its input queue and searches the locally cached MAC addresses using the metadata provided in the frame. If no up-to-date MAC address is found, it sends out ARP requests using data frames and resolves

Table 3.2: An example of configuration records from the FINS Framework linking table

Sender Module	Sender Module ID	Destination Link	Receiver Module	Receiver Module ID
Socket Stub	1	01	ICMP	2
Socket Stub	1	02	TCP	3
Socket Stub	1	03	UDP	4
ICMP	2	01	Socket Stub	1
ICMP	2	02	TCP	3
ICMP	2	03	UDP	4
ICMP	2	04	IPv4	5
TCP	3	01	Socket Stub	1
TCP	3	02	ICMP	2
TCP	3	03	IPv4	5
UDP	4	01	Socket Stub	1
UDP	4	02	ICMP	2
UDP	4	03	IPv4	5
IPv4	5	01	ICMP	2
IPv4	5	02	TCP	3
IPv4	5	03	UDP	4
IPv4	5	04	MAC/PHY stub	6
MAC/PHY stub	6	01	IPv4	5
MAC/PHY stub	6	02	ARP	7
MAC/PHY stub	6	02	Logging & Monitoring	8
ARP	7	01	MAC/PHY stub	6
ARP	7	01	Logging & Monitoring	8

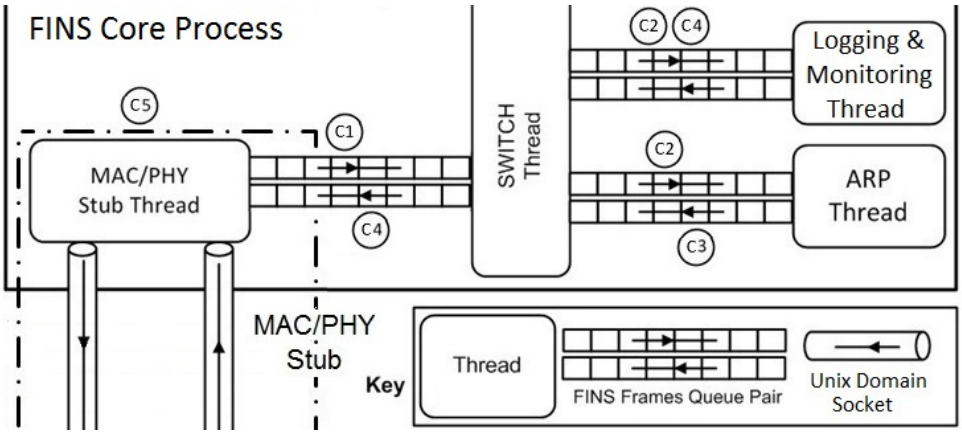


Figure 3.4: Steps of example interaction between the MAC/PHY stub and ARP modules.

the address. After resolving the address, the ARP thread changes the control frame's operation code to Read Parameter Reply, adds the appropriate information to the metadata, and inserts the frame into its output queue with the destination ID set to its link ID associated with MAC/PHY stub traffic (01).

C4. The switch thread reads the frame from the ARP output queue, uses the sender and destination ID (01) to search the linking table for the receiver module IDs (6 and 8), since there are multiple destinations the frame is copied to and the switch pushes one into each input queue of the receiver modules (MAC/PHY stub and Logging & Monitoring).

C5. The MAC/PHY stub thread reads the frame from its input queue, retrieves the necessary information from the metadata, and completes the Ethernet frame.

This example illustrates the use of control frames to communicate between modules and the interaction between the linking table, switch, and modules. The frames sent to

the Logging & Monitoring module were not discussed, as the module simply logs their contents; however, they showcase how transparent logging is possible through simple manipulation of the linking table.

In versions 0.8 and 0.9, the linking table was implemented in a centralized approach, where the switch module had a single copy of the entire table. We have made an improvement in Version 1.0 so the linking table has been optimized and implemented in a distributed manner, with each module receiving a subset of the linking table, performing the search for module ID(s) when sending, and copying frames when multiple receivers are found. Thus the destination ID field in frames stores the module ID of the receiver and the switch simply forwards frames to their destination. A subsequent optimization is that replies to control frames (e.g. Parameter Read Reply, Parameter Write Confirmation, Procedure Execute Reply) can be sent to the module ID contained in the sender ID field without a lookup in the local linking table. Later step-by-step examples will describe traffic flow using the distributed implementation of the linking table.

3.2.1.2 The Socket Stub Module

Supporting backwards compatibility of unmodified legacy applications is an important factor in implementing the socket stub module. Many of the reviewed tools performed tradeoffs between efficiency, ease of development, and flexibility by implementing their mechanism in either kernel-space or user-space. The socket stub module (Figure 3.3,

bottom right) combines the benefits of both by being implemented as two separate components: the *socket calls interceptor* in kernel-space and the *socket daemon thread* in user-space.

The socket calls interceptor is meant to be a lightweight component that intercepts pertinent BSD socket system calls and relays the necessary information to and from the socket daemon thread in the core process. To do this the socket calls interceptor is implemented as an LKM that unregisters the traditional internet socket family (AF_INET) and replaces it with our own family. This utilizes the behavior of the kernel to direct only appropriate socket calls to the socket calls interceptor and avoids re-implementing functionality provided by the kernel, such as handling cloned sockets or interactions with file descriptors. The socket calls interceptor communicates with the socket daemon thread over a Netlink connection (Figure 3.3, blue arrow) in client-server fashion, whereby the inherently parallel system calls are serialized and passed through the outgoing Netlink connection in order of arrival.

The socket daemon thread within the core process is attached to the other end of the Netlink connection and handles tasks that were originally implemented by the socket system call handlers and the network subsystem inside the kernel, such as maintaining the status and structures of opened sockets. An example walk-through of the steps in intercepting a socket call is depicted in Figure 3.5, as follows:

1. In user-space, an application calls socket() to create a socket for ICMP, TCP or UDP transfer.

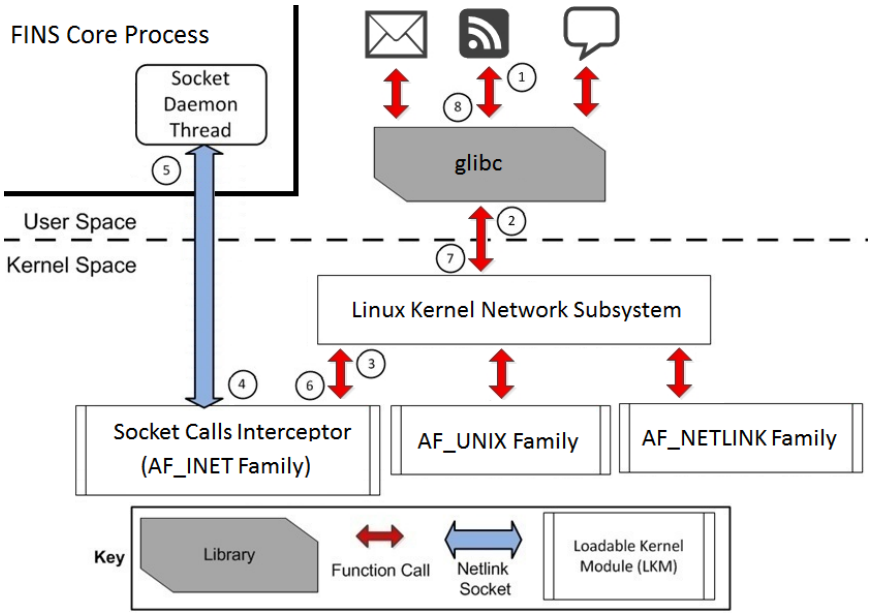


Figure 3.5: Implementation and data flow for the FINS Framework socket stub module.

2. Glibc converts the socket() call into a system socket() call that goes to the network subsystem in the Linux kernel.

3. In kernel-space, the network subsystem demuliplexes each call to its respective family and for the AF_INET family the call is directed to the socket calls interceptor. Through this, other types of socket communication are left untouched and are directed to their traditional handlers, e.g. Netlink, Unix domain sockets, etc.

4. The socket calls interceptor catches the call and creates the minimum necessary kernel socket objects, setting the socket operations (bind(), connect(), etc) to corresponding functions in the socket calls interceptor. This enables the kernel to track the socket and directs future system calls to the appropriate socket calls interceptor function (bind() $\rightarrow$ FINS_interceptor_bind()). The socket calls

interceptor serializes the call, forwards it to the socket daemon thread in the core process through the Netlink connection, and waits for a response.

5. In user-space, the socket daemon thread processes the socket() request, creates a new socket record in the FINS Framework socket database, and constructs any socket-related objects. The result of processing the call (for socket() a success/failure status) is serialized and transmitted back to the socket calls interceptor through the Netlink connection.
6. In kernel-space, the socket calls interceptor receives the result, deserializes and handles it accordingly, and then returns an associated value to the network subsystem.
7. Depending on the return value, the kernel returns either a socket descriptor or an error to glibc.
8. In user-space, glibc returns the result to the application.

Once a FINS Framework socket is created, future system calls to the socket pass through the network subsystem to a corresponding function in the socket calls interceptor. A similar process is conducted to serialize and shuttle the call to the socket daemon thread with an accompanying procedure to deserialize and handle the socket daemon thread's return.

Of the many possible mechanisms for intercepting system calls, we found the two-component LKM solution to be the least invasive and most advantageous. By

replacing AF_INET using a lightweight LKM, interception is seamless without modifying any kernel code, meaning that there is no need to recompile the Linux kernel. This avoids many of the issues related to kernel-space programming and allows for the same socket calls interceptor code to be used for kernel versions 2.6 to 3.8 on both Ubuntu and Android with only negligible changes. In addition, this intercepts all calls to internet sockets whether from dynamically or statically linked applications. Unfortunately, some minor overhead is introduced for each call as all necessary information must be shuttled from kernel-space to user-space and vice versa.

3.2.1.3 The MAC/PHY Stub Module

The MAC/PHY stub module (Figure fig4, middle left) connects to the network and provides portability across platforms. Implementation of the MAC/PHY layers differs significantly among platforms due to interactions among the kernel, the device driver, and the network adapter's firmware. The current MAC/PHY stub module is implemented using the Pcap library [98], a portable C/C++ library for network traffic capture and injection. Using the Pcap library allows the user to capture and inject Ethernet frames, and get/set some basic network adapter parameters.

The MAC/PHY stub module is implemented as three components: the *MAC/PHY stub thread*, the *capturing process*, and the *injecting process*. The MAC/PHY stub thread is a thread within the core process that works as a multiplexer/demultiplexer of the traffic between the switch thread and a networking interface. It is connected through a Unix

domain socket to the capturing process and through a second Unix domain socket to the injecting process. In order to accelerate the processing of frames and enhance the overall performance of the FINS Framework, the generation and serialization of Ethernet frames is implemented in the MAC/PHY stub thread, while their injection and capture are isolated into the two processes outside the core process (Figure 3.3, lower left). In effect, to send an IP packet the MAC/PHY stub thread generates an appropriate MAC frame header, encapsulates the packet, serializes the frame, and sends it over the domain socket to the injecting process. In turn, the injecting process forwards the stream to the network adapter's buffer by calling the Pcap library injection function. The operation is reversed when capturing traffic from a network adapter.

The default MAC/PHY stub module in Version 1.0 is limited because it does not provide control over the IEEE 802.11 medium access and physical layers' details such as retrieving the IEEE 802.11 specific headers, reading the IEEE 802.11 channels parameters, or controlling interface parameters such as power levels, and switching to monitoring mode. The current implementation of the MAC/PHY stub allows only the following features: disabling/enabling the network adapters, switching between the basic WiFi modes (infrastructure, and adhoc), and using the loopback interface. However, a modified module that is available in the development tree provides access to read statistics from the PHY layer's rate adaptation algorithm (e.g., minsrel). The module polls data from the generic *mac80211* loadable module, found in the Linux

implementation of the IEEE 802.11 protocol. We will refer to this modified MAC/PHY stub in Section 3.3.2.

3.2.1.4 The RTM and Console

Two minor but essential parts of the FINS Framework are the *Real-Time Manager (RTM) module* and the *FINS Framework console*. The RTM module is a non-data module that receives input from outside of the framework and allows runtime management and supervision of the framework. The FINS Framework console is a command line application that connects to the RTM through a Unix domain socket and can either act as an interactive prompt or simply receive status updates from the framework. Through the RTM and console, advanced monitoring and logging is possible as well as runtime reconfiguration of the linking table and modules.

3.2.2 Traffic Flow Walkthrough

Fig. 3.6 illustrates a typical traffic flow using an example stack that rebuilds the traditional TCP/IP stack. This is the same stack mentioned earlier whose configuration is given in Table 3.2. The steps of the outgoing traffic flow are indicated using circles labeled D1-D9 and will be discussed thoroughly, while only differences will be mentioned for the incoming traffic flow. For the outgoing traffic flow, consider the case when a running UDP application sends data:

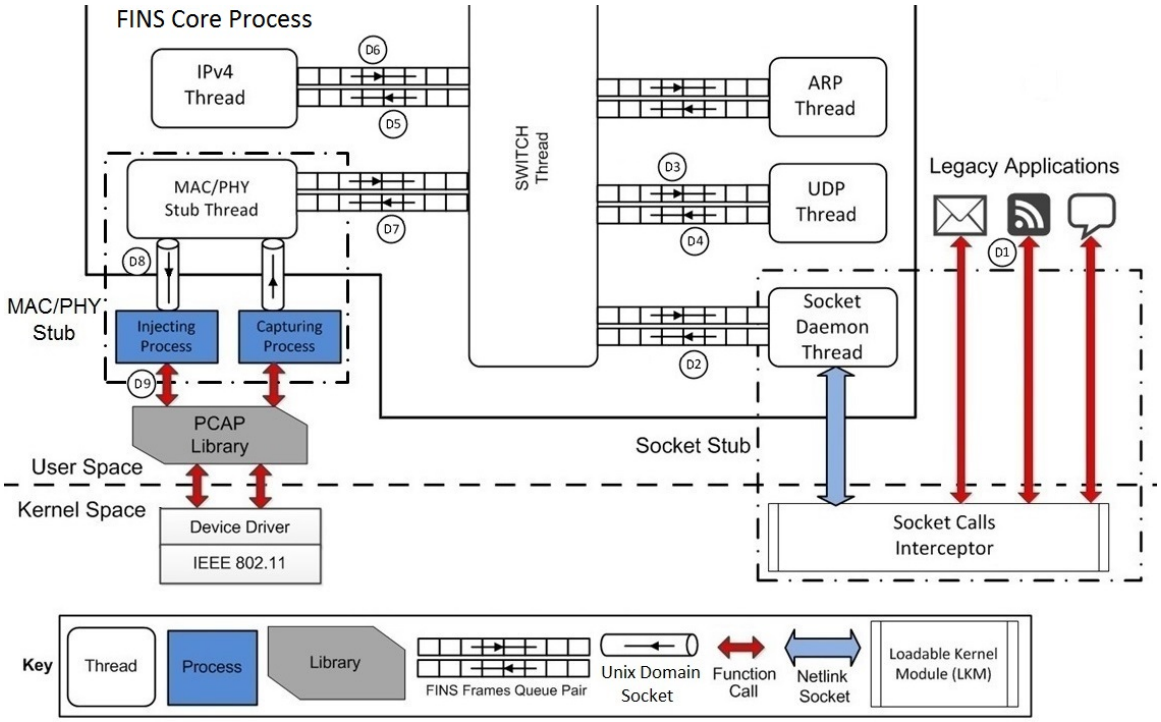


Figure 3.6: UDP/IP example stack using the FINS Framework.

D1. A UDP application sends a buffer of raw data through the glibc socket API.

D2. The socket daemon thread receives the data from the socket calls interceptor, encapsulates the data into a data frame, and uses its link ID associated with UDP traffic (03) to search its local subset of the linking table for a receiver module ID (4). The socket stub module inserts the frame into its output queue with the destination ID set to the UDP module (4).

D3. The switch thread reads the frame from the socket stub output queue and pushes it into the input queue of the UDP module as directed by the destination ID (4).

D4. The UDP thread reads the frame from its input queue, decapsulates the raw data, and extracts necessary information from the metadata, such as the sending and

receiving IP addresses/ports. The UDP thread performs its UDP related functions, creates a UDP datagram, and inserts it into the data frame. It then uses its link ID associated with traffic going down the stack (03) to search its local subset of the linking table and pushes the frame onto its output queue with the destination ID set to the IPv4 module (5).

D5. The switch thread reads the frame from the UDP output queue and pushes it into the input queue of the IPv4 module as directed by the destination ID (5).

D6. The IPv4 thread reads the frame from its input queue, extracts the encapsulated UDP datagram and any required metadata. Then, a new IPv4 packet gets built based on the metadata forwarded from the UDP module and is encapsulated into the data frame. After searching its linking table using its link ID associated with downward traffic (04), the IPV4 thread pushes the frame onto its output queue destined for the MAC/PHY module (6).

D7. The switch thread reads the frame from the IPv4 output queue and pushes it into the input queue of the MAC/PHY stub module as directed by the destination ID (6).

D8. The MAC/PHY stub thread reads the frame, extracts the IPv4 packet and the metadata. Then, it builds an Ethernet frame to be sent through the Pcap library. The MAC/PHY stub thread attempts to resolve the MAC address using its internal copy of previously found MAC addresses, contacting the ARP module if the

corresponding MAC address is not found (Fig. **??**). Once the MAC address has been acquired it is used to finish the Ethernet frame, which is then pushed into the injection Unix domain socket that carries data to the injecting process.

D9. The injection process reads the Ethernet frames buffered into its input Unix domain socket. Then, it sends each frame over a separate call to the Pcap library, which pushes the frames to the interface device driver.

Note that these steps may not occur directly after each other, as the modules may process other frames in between or conduct module-to-module communication to retrieve necessary information. This is referenced in Step D8, where conditions may necessitate a control flow to request the MAC address.

The incoming data flow is generally the reverse of the outgoing flow taking into account the following:

- The Pcap library captures the incoming Ethernet frames after the device driver replaces the original MAC and PHY header with a generic Ethernet header. The frames are pushed into a special Pcap buffer space which is read using a specific call back function.
- The capturing process serializes each captured frame and pushes it into the Unix domain socket connecting the capturing process to the core process.
- The steps then generally proceed in reverse order of the outgoing flow explained earlier.

- Eventually, the incoming data is pushed into the socket daemon thread's input queue. After reading the frame from the queue, the frame metadata is used to search the socket database for the target socket. If the destination socket is found, the frame is pushed into a separate receiving internal queue, which is one of the socket layer related objects created upon socket creation.

- Whenever the socket stub detects a socket receiving call from the application, the next data frame is read from the internal queue. The encapsulated raw data within the frame is serialized and sent over the Netlink connection to the socket calls interceptor. Then the socket calls interceptor returns the data to the application through the Linux kernel socket API.

It is clear from the architecture and implementation details discussed above that the FINS Framework strongly supports wireless experiments by enabling experiments on Android-based mobile devices and by facilitating the implementation of cross-layer solutions and context-aware applications. By supporting mobile devices whose battery life may extend to a couple of days, the FINS Framework allows researchers to consider experimental mobility scenarios that not otherwise have been possible to explore.

The study of cross-layer solutions and context-aware applications has been primarily pursued through simulations, due to implementation challenges using the legacy stack. The FINS Framework eases the implementation of cross-layer solutions for layer three and above. With the FINS Framework, even experiments that involve meters

and knobs below layer three are relatively simple to implement. We illustrate an example of such a cross-layer solution in 3.3.2.1.

3.2.3 Enabling FINS Framework over Android Platforms

One goal of the FINS Framework is the ability to run wireless experiments on handheld devices, hence we have planned to make the framework runnable on handheld devices with few or no modifications. The most promising handheld platforms for the framework are the Android-based platforms (e.g. smart-phones and tablets) since Android has a Linux-like kernel. Throughout the architecture, design and implementation we have considered the factors that may affect portability. We have made decisions to support portability like splitting the socket stub into two submodules, wrapping the MAC/PHY into a stub, and running the framework core in user-space, in addition to reducing the number of dependencies and carefully selecting libraries and APIs to be used. Hence, version 1.1 of the FINS Framework is planned support both the Linux and Android platforms.

This is a great advantage over previous tools including the well-known Click router [26]. It opens the door to carry on different sorts of wireless implementation-based experiments in nomadic and fully mobile scenarios, in WiFi ad hoc and infrastructure modes, and even in cellular access networks (e.g., 3G, 4G, LTE+). The lighter weight, longer battery life, more dynamic mobility, and more interactive user

interfaces of Android handheld devices, compared to traditional Linux laptops, present a rich environment for researchers.

We believe fragmenting the source code tree into two branches will discourage development efforts from the research community. This is why we have decided to maintain single tree of source code that is cross-compilable to run on Android. In order to achieve this, we have to satisfy implementation constraints forced by both Linux and Android. Based on our thorough investigation, there are three main challenges in order to run the same version of the FINS Framework on both Android and Linux devices. The first is related to the application layer, the second is related to operating system library dependencies, and the third is related to the the MAC/PHY subsystem in the Android kernel.

The first challenge is that applications that run on Android are not native applications. They are JAVA applications that run via a modified version of the Dalvic JAVA Virtual Machine (JVM). Hence we must enable the FINS Framework core process to run as native process or wrap it into a standard Android application. Yet it must be able to maintain full functioning privileges required such as the multi-threads core and the use of interprocess communications.

The second challenge is to make sure that all our library dependencies are fulfilled. In order to avoid dependency troubles , we have carefully chosen the libraries on which we rely. Our list include LibPcap library, LibConfig library, and POSIX pthreads. LibPcap is a system independent library that is compilable with Android. In fact, the Android

source code tree already includes Tcpdump which is a tool for network monitoring and data acquisition that depends on LibPcap. The second library is LibConfig which we use to implement the Meta-data field within the FINS frames. LibConfig has a fully reentrant parser which is an advantage since the framework core process is multi-threaded. Using this reentrant parser, different threads running simultaneously can parse the Meta-data embedded into the FINS frames. Although the library itself is not thread-safe, we fix this by using a common synchronization mechanism that is supported by both Linux and Android.

The only obstacle remains is that Android user-space does not use the GNU C implementation of the C standard library (glibc), however it uses an implementation called Bionic libc. Bionic is a modified variation of the BSD standard C library. Comparing to glibc, Bionic is faster and has smaller memory footprint which was a crucial advantages in the early days of Android platforms. However Bionic does not have a complete implementation of the POSIX pthreads. It only covers the basic needs of the Dalvik JVM threads. In order to survive these constraints, we implement the FINS Framework multi-threaded core process using only the limited features found in Bionic. The main obstacle is the lack of the function (*thread_cancel*). Although a thread can exit voluntarily, the parent thread cannot terminate the child thread. This ability to kill a child thread is needed to unload a module's thread whenever needed to reconfigure the stack. We handle this by having a status flag checked by the module itself. Whenever the flag is set, the module accomplishes the required memory garbage collection then exits.

The last challenge is the difference between the IEEE 802.11 stack implementation in the Linux kernel versus the Android kernel. Android WiFi device drivers still depend on the ancestor of the softMAC API called the Wireless Extension (WE). Most of the manufacturers even provide the drivers as binaries without sharing source code trees. This is sufficient if an experiment needs only to capture and inject traffic which is fulfilled by the generic MAC/PHY stub. But with this hard constraint, implementing the customized IEEE 802.11 stub is not possible in case of the Android platform's built-in WiFi devices. In order to enable experiments that requires the meters and knobs provided by the IEEE 802.11 stub, we suggest the use of an external USB wireless card. These have Linux open source drivers that are compatible with the WE API. Then we implement the stub by modifying the driver's lowest layer since the softMAC does not exist.

3.3 System Evaluation

This section is segmented into two parts. The first part covers four experiments. The first and second experiments quantify the performance of the main critical components within the FINS Framework (e.g., socket stub, MAC/PHY stub) in order to identify the possible system bottlenecks. We also compare the results attained from FINS to what is achieved using the Click Router in the same scenario. The Click Router has been chosen because it is one of the most widely deployed tools for experimental network research and its latest

official release still functions on current systems. We use the Click Router in userlevel mode. Experiments three and four test the performance of the FINS Framework when it implements a fully functioning networking stack that includes commonly used network protocols.

The second part of this section discusses more complex experiments that we have successfully implemented utilizing the FINS Framework, to illustrate the value of the FINS Framework as a research tool. This includes experiments involving routing, on-demand video streaming, instant messaging, and voice chat.

3.3.1 Performance Evaluation

While flexibility is a primary goal of the FINS Framework, a minimum level of performance is needed for it to be usable in a variety of scenarios. Our goal was to support data rates up to 54 Mbps, the maximum data rate of IEEE 802.11g. The equipment used in the following tests included:

- A Netgear N300 IEEE 802.11b/g/n wireless router access point.
- Two Dell Inspiron 1525s laptops with Intel Core 2 Duo processors running at 2 GHz, 3 GB of RAM, Broadcom IEEE 802.11b/g wireless interfaces, and Marvel fast Ethernet controllers. Both laptops were running Linux Ubuntu 10.04 with Linux kernel version 3.2.0.

- Two Google Nexus 7 tablets produced by ASUS. Both tablets were running Android 4.2.2 with Linux kernel version 3.1.10.

For each of the following experiments, the end-to-end throughput was measured at the application level using the widely used networking tool *iperf*.

3.3.1.1 Experiment 1

This experiment was intended to evaluate the socket stub module and determine the maximum sending throughput of application data through the module. This is important since socket calls must be shuttled to the core process and a bottleneck in this procedure would slow the execution of applications, potentially changing their behavior. To evaluate this, Experiment 1 utilized a single computer, not connected to the network, running a FINS Framework which consisted of only the switch, the socket stub module, and a custom logging module that reported throughput. The switch linking table was configured so that frames from the socket stub module traveled to the logging module, which simply measured the application throughput. On top of this stack ran an instance of the iperf application in client mode, which was set to generate a 60 second period of UDP traffic sent through the local loopback in 1.47 kB datagrams.

We have designed a similar experimental scenario to test the Click Router. The latter does not replace or modify the socket interface or the socket handling subsystem in order to capture the application traffic. Instead, the Click Router has a socket module which we

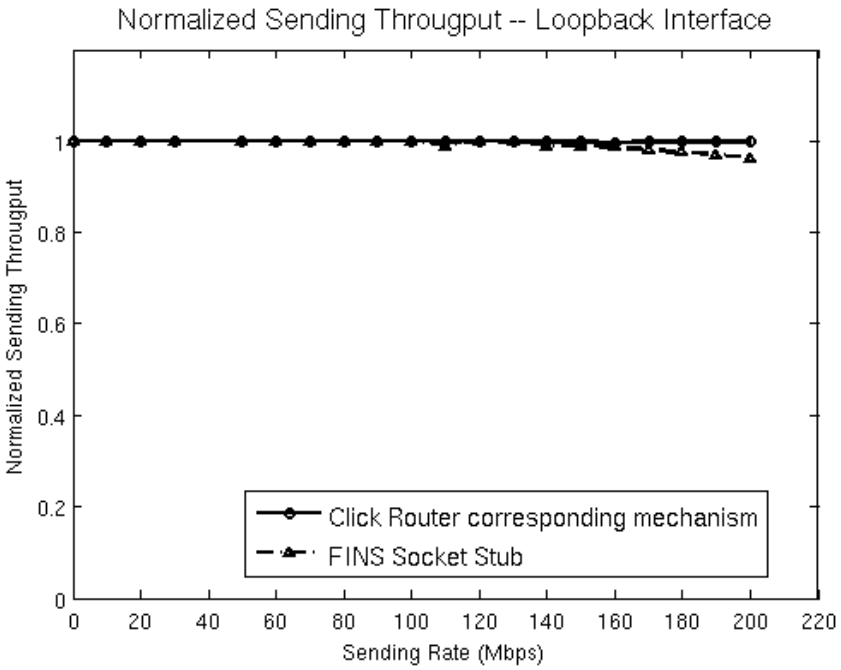


Figure 3.7: The observed throughput through the socket stub module normalized by the data rate attempted by iperf.

have used to implement a UDP server that listens to the loopback address, particularly UDP port number 5001, which is the default destination port of the regular iperf servers. Hence, any datagram generated by the UDP client gets redirected to the Click's socket module after the packet hits the network layer within the kernel.

The results of this experiment (Fig. 3.7) show that the FINS socket stub module is able to handle data rates up to 169 Mbps with a drop rate less than 0.01. This suggests that the socket stub module and the data transfer between the socket calls interceptor and the socket daemon thread is reliable and will not create a bottleneck for the FINS Framework in networking experiments. However, it is important to note that Netlink connections have been shown to operate at speeds an order of magnitude faster,

indicating that the limiting factor for the socket stub module is most likely the processing done in the socket daemon thread. Since the core process is implemented as a single process with many threads, the throughput of the socket stub module may decrease in practice due to scheduling constraints.

On the other hand, the Click Router achieved a normalized throughput close to the maximum sending rate we used (200Mbps). Basically, the setup of this experiment does not force the Click Router to cause any significant drop of packets since there is no significant processing delay (decapsulation or re-encapsulation).

3.3.1.2 Experiment 2

The goal of Experiment 2 was to evaluate the current implementation of the MAC/PHY stub module and observe the maximum receiving throughput that our packet capturing mechanism can support before dropping frames. This is pertinent because the Pcap library will overwrite previously captured frames in its buffer if the capturing process does not poll and process the frames fast enough. Pcap opens a raw socket with the device driver to sniff the packets before they reach the protocol's family's handler at the bottom of the kernel networking stack. The maximum buffer size of the opened socket is 65535 Bytes. The Click Router uses the Pcap library to capture the packets as well.

To evaluate this the traditional stack and the FINS Framework were run in tandem and iperf was used to send traffic through the traditional stack to the local loopback

address. When these packets were "received" by the node, the FINS Framework was able to intercept these packets through the Pcap library. As such, the setup for this experiment consisted of a single, non-networked computer that was running a FINS Framework which consisted of only the switch, the MAC/PHY stub module, and a custom logging module that reported throughput.

The internal queues between the FINS modules (the MAC/PHY stub and switch) are capped at 100,000 frames but we never have observed such a backlog. Although this is the default cap, it is designed to address extreme cases of very small Ethernet frames (e.g. 64 bytes, with only 18 bytes of payload). In such a case, with a rate of 10 Mbps, the corresponding rate of packets is 69444 packets/second. If the target research experiment expects handling an extreme case beyond a 100,000 frame buffer limit, then the FINS adopters can easily adjust this parameter.

The switch linking table was configured so that frames from the MAC/PHY stub module traveled to the logging module, which measured the throughput of application data. Since this particular FINS Framework excluded the socket stub module, the traditional TCP/IP stack was still accessible and was used to run the iperf application. Once again, iperf ran in client mode and was set to generate 60 second trials of UDP traffic sent through the local loopback in 1.47 kB datagrams. The local loopback was used to avoid the increased probability of dropping frames due to a wireless channel or potential throughput limitations in using a LAN cable.

Fig. 3.8 shows that the MAC/PHY stub module maintained a packet loss rate of 0%

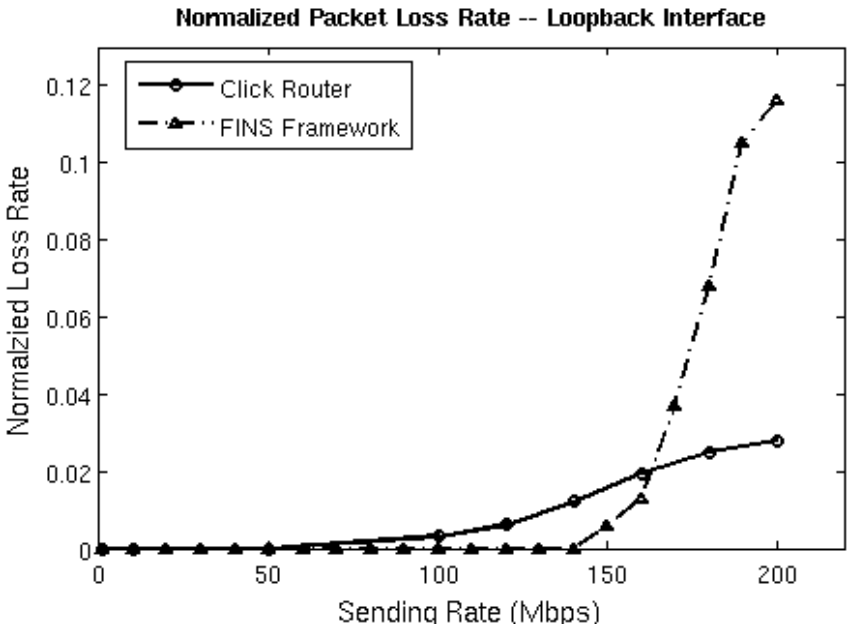


Figure 3.8: The packet loss rate through the MAC/PHY stub module at varying data rates.

at incoming data rates up to 140 Mbps. The packet loss rate appeared to increase rapidly at rates above 140 Mbps. Throughput measurements from the logging module showed that zero percent (0%) of the packet loss occurred internal to the FINS Framework, indicating that the packet drops shown in Fig. 3.8 occurred at the Pcap library level. Our analysis suggests that this is caused by the capturing process blocking on the Unix domain socket connecting it to the MAC/PHY stub thread, which effectively couples the processing delay of handling a frame in the MAC/PHY stub thread with how quickly the capturing process can retrieve frames from Pcap. As with the socket stub module, this processing delay may increase under high scheduling demand.

The Click Router started to suffer dropped frames at a lower rate, specifically above 60Mbps. However the drop rate increased smoothly through 200Mbps. The Click Router

evaluation by the authors in [99] mentioned achieving a forwarding rate of 333,000 64-byte packets per second using less capable hardware. We believe this previously reported rate was achieved by the kernel-level mode, not by the user-space mode.

The simplest implementation of this test using the Click Router includes three modules (FromDevice, Counter, and Discard). The Click Router runs as single process, hence it does not use any interprocess communication to connect the built modules. In case of the FINS Framework, there is an overhead of interprocess communication since the capturer and the FINS core are connected via UNIX domain socket. This increases the number of memory operations and consumes additional CPU slots, hence the system may get into a livelock. This explains how the normalized throughput decreases as the packets rate increases. In addition, at a point of time the iperf client itself will consume a significant portion of the available CPU slots while generating higher sending rate.

Generally, we believe that these experiments demonstrate that the FINS Framework achieves the desired level of performance, given the degrees of freedom and benefits provided to users by altering the kernel networking stack to run mostly in the userspace.

3.3.1.3 Experiment 3

Experiments 3 and 4 use two setups to contrast the maximum performance achievable by the FINS Framework (found using a wired setup) to the performance observed in a

wireless environment. The UDP and TCP results recorded in both experiments are shown in Tables 3.3 and 3.4, respectively. In addition, benchmark values for the traditional Linux stack were recorded for reference and are also shown. During Experiment 4, the results of the traditional stack were used to estimate the wireless channel, which is why the drop rate is not listed.

In the third experiment, we used iperf to measure the maximum end-to-end goodput achievable by the FINS Framework when sending or receiving data. To observe the sending rate we connected the two laptops using a 100 Mbps LAN cable (blue arrow in Figure 3.9). On one of the computers an iperf client was run on top of a FINS Framework, while the other computer ran an iperf server on top of the traditional Linux stack. The FINS Framework used for the experiment was the rebuilt stack depicted in Fig. 3.6 and whose linking table was shown in Table 3.2. Finally, all processes that were not required by the operating system were killed on both computers for the duration of this experiment. For the receiving rate the same setup was used with the traffic stream reversed – the iperf server was run on the FINS Framework and the iperf client application on the traditional stack.

In Experiment 3, it was found that the FINS Framework is able to send a maximum of 87.1 Mbps through UDP and 83.6 Mbps using TCP. These values are close to the observed throughputs for the traditional stack (95.4 Mbps and 94.4 Mbps respectively) when limited by a 100 Mbps LAN cable. Receiver results where more nuanced, with the FINS Framework currently able to receive at a rate equal to that of the traditional stack

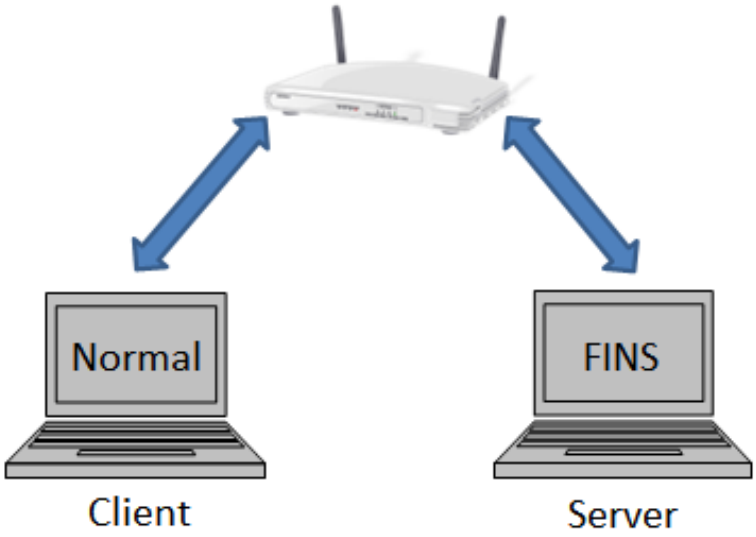


Figure 3.9: Experimental setup for Experiment 3.

for UDP (95.4 Mbps) and a significant fraction for TCP (48.1 Mbps). The major performance difference between UDP and TCP receive results is likely due to our current implementation of the TCP module and not the stack as a whole. In any case, the results demonstrate the ability to send and receive at speeds that would be enough to support the theoretical data rates of IEEE 802.11a/b/g and may be enough for the rates practically achieved by IEEE 802.11n technologies.

3.3.1.4 Experiment 4

This experiment was designed to observe the maximum end-to-end goodput achievable by the FINS Framework in a wireless environment. As such, the previously described Android tablets were set to work in Wi-Fi infrastructure mode and comprised a basic service set (BSS) in conjunction with the access point. Similar to Experiment 3, when

Figure 3.10: Experimental setup for Experiment 4.

sending one of the computers ran an iperf client on top of a FINS Framework, while the other computer had an iperf server on top of the traditional Linux stack. The traffic stream was reversed when collecting receiving rates. Experiment four setup is shown in Figure 3.10

Using the traditional stack, an estimate of the maximum goodput possible between end nodes in this wireless setup was found to be 15.7 Mbps for UDP and 13.0 Mbps for TCP. Within these constraints, the FINS Framework achieved a maximum sending rate of 15.7 Mbps for UDP and 13.0 Mbps for TCP through the iperf server application. For receiving, the framework performed equally well (15.6 Mbps and 13.0 Mbps) with the observed packet loss ($< 0.4\%$) occurring at the link level. Throughout this entire experiment the FINS Framework maintained a zero percent (0%) dropping rate within the stack, with the low maximum goodput estimates provided by the traditional stack caused by the noisy wireless environment. Analysis of the observed traffic and the good wireless performance for the FINS Framework relative to the traditional stack suggest that while our TCP module may have flaws receiving at higher rates it responds robustly to moderate segment loss.

The evaluation results from Experiments 1-4 suggest that version 1.0 of the FINS Framework is currently able to support implementation-based Wi-Fi networking experiments for IEEE 802.11a/b/g standards on Ubuntu and Android.

Table 3.3: UDP Results for Experiment 3 (laptops, LAN) and Experiment 4 (tablets, Wi-Fi).

		FINS Framework			Traditional Stack		
Network	Action	Goodput (Mbps)	Jitter (ms)	Drop (%)	Goodput (Mbps)	Jitter (ms)	Drop (%)
Exp. 3	Sending	87.1	0.242	0.0615	95.4	0.222	0.00610
	Receiving	95.4	0.242	0.135	95.4	0.222	0.00610
Exp. 4	Sending	15.7	1.40	0.0575	15.7	1.36	-
	Receiving	15.6	1.74	0.397	15.7	1.36	-

Table 3.4: TCP Results for Experiment 3 (laptops, LAN) and Experiment 4 (tablets, Wi-Fi).

		FINS	Trad Stack
Net	Act	Goodput (Mbps)	Goodput (Mbps)
Exp. 3	Send	83.6	94.4
	Recv	48.1	94.4
Exp. 4	Send	13.0	13.0
	Recv	13.0	13.0

3.3.2 Evaluation Through Experimental Scenarios

Extensive testing with many networking applications was conducted in the lab to ensure the transparent support of tools commonly used in research. Each of the applications were successfully used unmodified and without recompilation; some examples include: *ifconfig*, *ping*, *iperf*, *dig*, *traceroute*, *Telnet*, *Wireshark*, *Firefox*, and *Skype*. Various experimental scenarios have been successfully implemented and tested using the FINS Framework in qualitative trials that occurred in the lab. We discuss below a set of the applications-based experiments that we ran in our lab and collectively in a workshop attended by external experimenters.

The experiments can be divided into two categories, the first category uses synthetically generated traffic over a controlled wireless environment, while the second category depends on ordinary Internet usage scenarios (e.g., file downloading, video-on-demand (VoD) streaming, real-time multimedia).

3.3.2.1 Synthetic Traffic Experiments

The first experiment focuses on packet forwarding. As shown in Fig.3.11, the experiment utilizes several laptops as stationary ad hoc nodes, each is equipped with an IEEE802.11g NIC. The nodes form a ring topology before a specific node (e.g., node 2) generates traffic destined to a node outside the set of its first hop neighbors. Each node forwards packets on a packet-by-packet basis that depends on investigating the

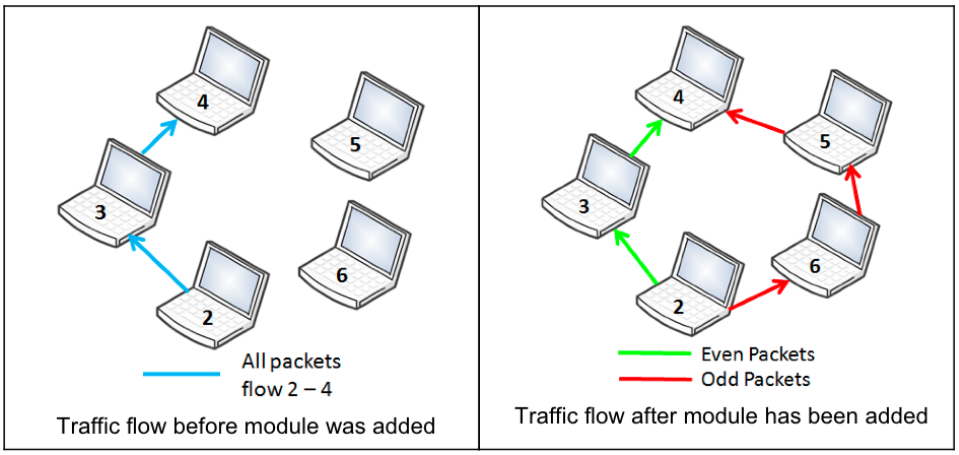


Figure 3.11: Forwarding scheme before and after adding the new forwarding module.

application payload (i.e., a serial number field). We use the FINS Framework to add a new thread, independent from the IPv4 thread, to implement the payload check and forwarding rules. In addition, the experiment implements a logging module (i.e., RTM thread) connected to our implementation of the FINS console process to log and display the ongoing experiment events.

The second experiment implements a context-aware cross-layer solution to adapt the data generation rate of a variable bit rate (VBR) application (e.g., media streaming) based on the available wireless bandwidth estimated at the MAC level. The experiment includes 8 pairs of laptops equipped with IEEE802.11g NICs. Each pair consists of a sending node and a receiving node. Nodes contend with each other to send VBR data through the same IEEE802.11n access point. The data generation application collects logs from the minstrel module through a modified version from the FINS Framework MAC/PHY stub module. We developed the latter so that it communicates with our modified version from the Broadcom Wi-Fi device driver over a generic netlink socket,

which allows bidirectional communication between the kernel and userspace. No recompilation of the Linux kernel is required since the modified device driver is a loadable kernel module (LKM).

The traffic generator (sending side) receives the exponential weighted moving average (EWMA) statistics from the minstrel rate adaptation algorithm [100], which is run by the mac80211 kernel module on top of the device driver. This experiment is strongly related to our ongoing research on adaptive video streaming. Similar to the previous example, development costs related to the FINS Framework were low, and modifying the driver made up the bulk of the work. Through the FINS Framework, researchers have access to cross-layer information with more detail and granularity than possible through the traditional stack, an important factor to accelerate research in many areas such as cognitive networks [97].

Other experiments focused on the ability to monitor and control network behavior at runtime, potentially for classroom educational uses. For instance, one scenario involved controlling TCP behavior and variables (e.g. Fast retransmit, GBN, timeouts, etc.) while observing the impact on performance in real-time. Another experiment showcased Android and the ease of use when performing mobile experiments.

During the summer of 2013, our group hosted an all-day workshop that focused on experimental wireless networking research and introduced the FINS Framework through a series of hands-on exercises. During the all-day workshop, the attendees had the chance to practice by themselves the steps to run each of these synthetic traffic experiments.

3.3.2.2 Internet-based Experiments

Previous tools (e.g., Click Router, ANA) suffer from the lack of or high cost of integration with legacy applications or heterogeneous networks. The next set of experiments illustrates the backward compatibility of the FINS Framework. During this set of experiments, the FINS Framework node mainly connects to the Internet using 100Mbps Ethernet connection via Virginia Tech campus network. However, numerous runs have been also repeated using off-campus Internet connections at coffee shops or residential buildings.

Three experiments were chosen for this evaluation: file downloading, video streaming, and delay-sensitive media communication. The first test was downloading a file using both HTTP and FTP. Using the Firefox browser, we downloaded a roughly 700-Mbyte ISO file from the Ubuntu website. Measurements have verified that download speed has been only limited by the end to end connection.

The second experiment successfully runs multiple on-demand high definition (HD) YouTube videos over the Internet using the *Firefox* browser. The video sessions are adaptive HTTP video streaming sessions that run over TCP. This state of the art video technology adapts according to the throughput and delay conditions [63]. The video sessions have included video clips with length varies between 3-minutes to 45-minute-long and with quality that varies between 144p to 1080p (HD). It has been verified that the FINS Framework components did not introduce any significant

abnormality to the video application. The experiment was repeated using a wireless connection to examine different throughput and delay conditions. Results did not differ with respect to how the FINS Framework performed. The only changes observed were due to the wireless bandwidth.

The final experiment aims at testing how much the processing delay introduced by the FINS Framework (if any) may affect the performance of a real-time delay sensitive Internet application, which has stringent timing requirements. Hence, we decided to setup up a series of Skype calls over the Internet between two nodes, separated by nine hops over the Internet. The first node runs the Skype [101] application on the FINS Framework, while the second runs Skype over traditional stack. Skype communication can be carried over TCP or UDP. When using UDP, the call was successful, with jitter varying from 35 to 400 ms and packet loss varying from 2 to 15 percent. Despite high jitter variation and some packet loss, the call quality and usability were reasonable. At this time, the FINS Framework implementation of TCP cannot meet the stringent timing requirements of a Skype voice call. This is an area of investigation and future work. Additionally, video calls were partially successful. Currently, we are revising our TCP implementation to meet more restricted time requirements, although it works well for streaming video-on-demand (VoD) contents.

The results from these experiments show that the FINS Framework can support legacy applications without modification and that the day-to-day performance is quite

usable. While there are limitations on which applications are supported, the reasons for this are related to the FINS Frameworks current internal performance.

3.4 Summary, Contributions and Future work

This chapter presented the architecture, design, and implementation of the FINS Framework, a new open source tool to facilitate implementation-based experiments in wireless networking research. The chapter presented the architecture and design of the FINS Framework with a focus on how this flexible architecture realized the desired features for new experimental networking tools. In addition, it discussed the implementation details of the official releases of the framework over Linux platforms, how the same implementation was successfully ported to Android platforms, and how this implementation fulfilled the requirements of the two target platforms. At the end, a thorough testing of the core performance over wired and wireless connection was provided, in addition to an extensive list of experimental scenarios that verified various functionalities of the framework (backward compatibility with legacy applications, support for cross-layer research).

The work presented in this chapter has resulted in the following publications:

- Abdallah A.S., MacKenzie A.B., DaSilva L.A., Michael M.S.,"On Software Tools and Stack Architectures for Wireless Network Experiments", IEEE Wireless Communications and Networking Conference, WCNC 2011.

- Thompson Michael S., Abdallah A.S., Reed J., MacKenzie A.B., DaSilva L.A., "The FINS Framework: An Open-Source Userspace Networking Subsystem for Linux", vol.28, no.5, pp.32-37, September-October 2014.
- Reed J., Abdallah A.S., MacKenzie A.B., DaSilva L.A., Thompson M. S., "The FINS Framework: Design and Implementation of the Flexible Internetwork Stack (FINS) Framework", IEEE Transaction on Mobile Computing (TMC), vol.15, no.2, pp.489-502, Feb. 1 2015.

A recent trend in networking research is to build experimental testbeds that integrate several forms of testing and evaluation (e.g. simulation, emulation, prototyping, platform-specific real implementation). This is achieved by integrating actual platform-dependent implementations of TCP/IP protocols, simulation environments (e.g., ns-3, ns-2, Opnet), and experimental prototyping tools. Enabling the FINS Framework to integrate with well-known network simulation environments will be a great extension step for the research community.

While the FINS Framework helps the researchers to quickly implement functional prototypes of new protocols or algorithms, the network simulator allows to abstract the testing scenarios and guarantee the repeatability of test cases. This may highly accelerate the development and evaluation process of new network solutions, before moving the prototype to field-based testing.

One of the main benefits of the integration between the FINS Framework and the

ns-3 simulator is allowing traffic from real-world Internet applications to run over ns-3 simulated models via the FINS Framework. While the transport layer may run within the FINS Framework, the network, MAC, and PHY layers can be simulated by the ns-3 simulator.

Chapter 4

Cross-layer Controller for Adaptive Video Streaming over IEEE 802.11b/g: A Heuristic Approach

This chapter presents heuristic quality control solutions for the adaptive video streaming (AVS) problem. The proposed solutions exploit some of the lessons learned on wireless bandwidth estimation and prior heuristic quality controllers from the literature. The solutions represent an effort to understand the problem dynamics when using cross-layer design before presenting more complex solutions. Our ultimate goal is to compare the achievable efficiency using such a heuristic cross-layer design and through

combining cross-layer design with more sophisticated techniques (e.g. MDP and RL) given the overhead and implementation cost associated with such techniques.

Initially, we relax some of the problem constraints. For example, we assume that the wireless link is almost symmetric—with respect to contention or retransmission at the MAC layer—so that the quality estimated by the wireless node is indicative of the quality experienced by the access point. In fixed scenarios where nodes are not mobile, this is a reasonable assumption. Our proposed controller uses a passive technique to estimate the available bandwidth for the IEEE 802.11 link based on measurements from the MAC protocol. This study has two objectives. The first objective is to investigate to what extent the proposed cross-layer passive estimation technique is efficient. The second objective is to investigate the relationship between the IEEE 802.11 physical rate adaptation algorithm (RAA) and the quality perceived by the video streaming user. This is motivated by the findings reported in [102, 103] regarding how poorly state-of-the-art RAAs (e.g. Minstrel) may perform when IEEE 802.11a/g/n links are unstable. In order to evaluate this without interference from TCP, we simulate the video traffic over a UDP connection. This is to separate the influence of the IEEE 802.11 rate adaptation algorithm (RAA) from the behavior of the TCP congestion avoidance control and guarantee that the only limit on achievable data rate is the wireless link bandwidth.

We begin by introducing our video quality controller, E2EMAC, which combines measuring the end-to-end throughput and estimating the available bandwidth of the client's wireless link. After assessing a basic version of the controller, we propose

another version that monitors the the playback buffer level with respect to an adjustable threshold. The so called ”high risk threshold” is adjusted according to the severity of the decline in the wireless link conditions. This results in a newer controller that we call the Buffer-Aware E2E-MAC (BAE2E-MAC) algorithm.

We use the ns-3 network simulator to test the E2E-MAC controller in comparison with a widely used commercial solution [63], 62], which relies on traditional E2E measurements. Then, we test the BAE2E-MAC under adaptive IEEE 802.11 physical rates—a more realistic condition—in comparison with QoE-RAHAS, a recently proposed heuristic algorithm for adaptive video streaming [82, 104].

4.1 The E2E-MAC Quality Controller

In this section, we present a heuristic cross-layer scheme which combines monitoring the E2E throughput and monitoring the contention level of the wireless last-hop. The latter is expected to provide insights into the dynamics of the wireless link, and enables the adaptation controller to respond to dynamics that are invisible to controllers that use only E2E measurements. Designing a controller that achieves such a combination is beneficial in two ways. First, the controller would respond faster than current E2E-based solutions. Second, the controller can correlate changes in the wireless link status with E2E measurements, allowing it to maintain system stability. Namely, when the wireless link dominates system behavior, the controller becomes more responsive to these

dynamics; when the wireless link is not the bottleneck, the controller maintains stable responses to the E2E conditions.

4.1.1 Proposed Approach

We use the retransmission counter as an indication of the MAC-layer contention level. Frame retransmission can be caused by collision or physical layer conditions. We monitor the retransmission counter as a passive bandwidth estimation method to avoid additional overhead. Probe-based techniques do not work well in the wireless environment as investigated thoroughly in [105].

Our heuristic algorithm is summarized as follows: We divide time into time slots called update windows UW, whose length is UW_{len} seconds. Each window is then divided into sub-slots of W seconds. For our simulation results in next section, $UW_{len} = 1$ second and $W = 0.1$ second. These values were chosen empirically based on simulation trials and data from studying different 802.11 physical rate control algorithms [106].

We calculate a linear signed weighted sum of the retry counter average over each sub-slot $W = 0.1$ second. We use higher weights for the most recent sub-slots. This calculated sum is the trend function which indicates whether the available bandwidth increases or decreases. As the retry counter average value goes down, the trend function

increases, causing higher video quality levels to be requested. We switch video quality levels based on mutually exclusive rules.

The heuristic algorithm is described as follows:

1. We calculate the average of the retry counter rc_{avg} over a sub-slot whose length is W using

$$rc_{avg} = \frac{\sum_{j=1}^{N} r_j}{N}, \tag{4.1}$$

where r_j is the retry counter value when frame j is transmitted successfully. If the maximum retry limit $Retry\_Max = 7$ is reached without success, $r_j = 2 \times Retry\_Max$. N is the total number of frames the node tries to send during the sub-frame.

2. Every update window UW, we calculate a trend α for the available bandwidth based on the calculated retry counter averages over the sub-slots using Equation 4.2. If the retry counter average is increasing, this indicates that the available bandwidth is decreasing and vice versa.

$$\alpha = \sum_{i=1}^{UW_{len}/W} (i)\,\text{sign}\left(rc_{avg_{i-1}} - rc_{avg_i}\right) \tag{4.2}$$

3. After calculating the trend from the MAC layer for each UW, the adaptation algorithm decides how often to check and request new quality level, this is called the adaptation window k seconds. Too small adaptation window may lead to

instability and create additional network overhead. The average trend over the adaptation window is given by $\epsilon = \frac{(\alpha_1+...+\alpha_k)}{k}$. Based on insights from trial runs, we have used $k = 3$ seconds for our results.

4. Concurrently at the application layer, the E2E video frames throughput over the adaptation window is given by:

$$\lambda = \frac{\sum_{i=1}^{M} S_i}{k}, \tag{4.3}$$

where S_i is the size of received video frame i in bits, and M is the total number of video frames received successfully by the application layer during the adaptation window.

5. Finally, the adaptation decision is determined in each adaptation window using Algorithm 1, where the algorithm advances using the time step $t \in \{1, 2, 3, ..., \lfloor L/k \rfloor\}$. L is the video clip total length in seconds, θ_{q_t} is the minimum bandwidth required to stream the video at quality q_t, and q_t is the video quality level chosen for step t. The range of q_t is from 1 to 5. This minimum required bandwidth is equal to the video encoding rate used to generate this quality level.

The E2E-MAC algorithm achieves a balance between risk and gain. It does not step up unless the wireless link is stable according to the first and second rules. It detects degradation in quality on wireless links faster than the E2E algorithm by utilizing the third and fourth rules, so that it steps down before rebuffering occurs. At the same time,

Algorithm 1 The E2E-MAC Adaptation Algorithm.

$Counter_{NC} = 0, t = 1$
while $t < \lfloor L/k \rfloor$ **do**
  if $(\lambda(t) > \theta_{q_t+2})$ && $((\epsilon(t) > 0)$ && $(\epsilon(t-1) > 0)$ && $(\epsilon(t-2) > 0))$ **then**
    $q_{t+1} = q_t + 2$, $Counter_{NC} = 0$
  else if $(\lambda(t) > \theta_{q_t+1})$ && $((\epsilon(t) > 0)$ && $(\epsilon(t-1) > 0))$ **then**
    $q_{t+1} = q_t + 1$, $Counter_{NC} = 0$
  else if $(\lambda(t) < \theta_{q_t-2})$ && $((\epsilon(t) < 0)$ && $(\epsilon(t-1) < 0))$ **then**
    $q_{t+1} = 1$ {request lowest level to avoid re-buffering}
  else if $(\lambda(t) < \theta_{q_t})$ || $((\epsilon(t) < 0)$ && $(\epsilon(t-1) < 0))$ **then**
    $q_{t+1} = q_t - 1${decrease gradually}
  else if $(\lambda(t) > \theta_{q_t+1})$ && $((\epsilon(t) > 0) \oplus (\epsilon(t-1) > 0))$ **then**
    $Counter_{NC} = Counter_{NC} + 1$
  else if $(Counter_{NC} > 5)$ && $(q_t \leq 3)$ **then**
    $q_{t+1} = q_t + 1$, $Counter_{NC} = 0$
  else
    $Counter_{NC} = Counter_{NC} + 1$
  end if
  $t = t + 1$
end while

the first rule makes the most of opportunities by increasing the quality by two levels when there is a significant rise in bandwidth. The $Counter_{NC}$ is used to avoid the possibility that the algorithm gets in a frozen state. It counts the iterations when the algorithm makes no decision changes due to lack of changes in the trend and/or bandwidth measurements.

4.1.2 Simulation Setup and Evaluation Results

In order to realistically evaluate the proposed algorithm, we have used actual video traces to generate video traffic within a client-server video streaming simulation that we implemented. Prior to running the simulations, we used the EvalVid tool [60] to create MPEG-4 video trace files for five different video quality levels from a single raw full

length movie file, as illustrated in Figure 4.1. We created ten short clips by cropping 5 minutes segments from the raw file of the movie *Jurassic Park*. We used a GOP size of 12 frames. The five streaming quality levels and their corresponding rates are given in Table 4.1.

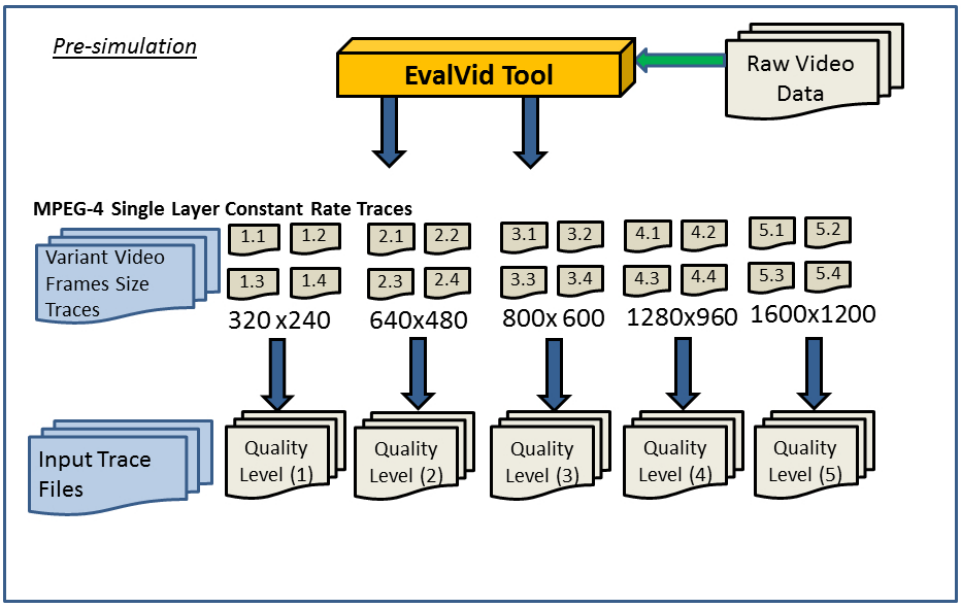


Figure 4.1: An illustration of generating video traces using the Evalvid tool.

For the simulation itself, we used the ns-3 Network Simulator [10] to implement a scenario involving two fixed wireless nodes equipped with IEEE 802.11b interfaces, an IEEE 802.11b access point, and a 100 Mbps Ethernet connection between the access point and the server node. We use adaptive video streaming client and server modules that we have developed, which are available from our GitHub repository [107]. One of the wireless nodes acts as a video streaming client while the other wireless node sends and receives background traffic when needed to test the effect of MAC-layer contention. The server node runs a video streaming server application as well as background traffic

Table 4.1: Generated video quality levels and corresponding rates.

Quality Level	Rate (Mbps)
q_1	0.3
q_2	0.7
q_3	1.5
q_4	2.5
q_5	3.5

applications. Figure 4.2 illustrates the simulation setup. The values of major simulation parameters are given in Table 4.3.

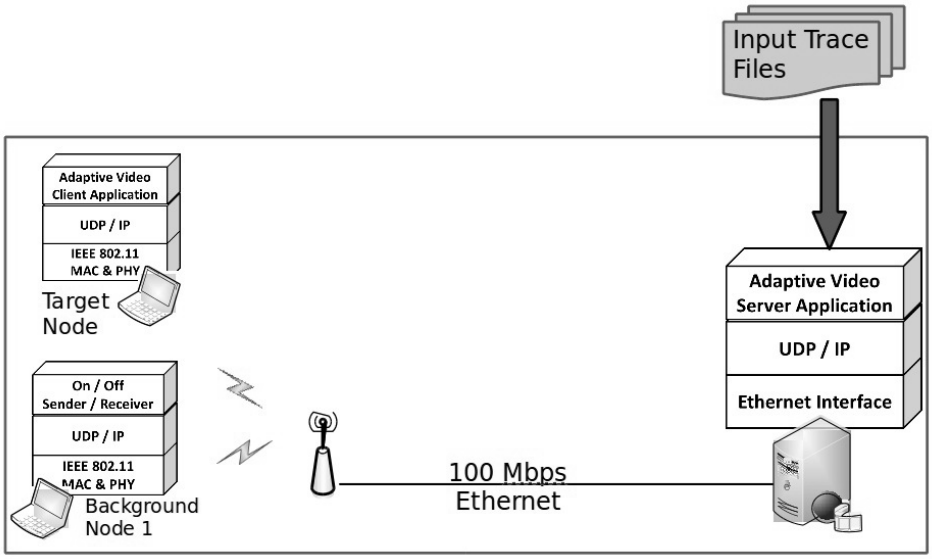


Figure 4.2: A wireless bottleneck scenario using the ns-3 simulation environment

In order to compare performance with previous work, we also implemented another controller to simulate the Akamai E2E solution as described in [63]. We use several evaluation metrics that attempt to capture user satisfaction with the received video quality and dissatisfaction with the frequency and length of re-buffering events. In

Table 4.2: The NS3 most important simulation parameters.

Parameter Name	Parameter Value
Video Clip Length	5 minutes (300 seconds)
Initial buffering delay	10 seconds
Video quality levels	5
Ethernet Link Capacity	100 Mbps
PHY Rate Manager	Constant Rates (5.5, 11) Mbps
RTS/CTS	Disabled
Short Retry limit	7
Mobility Model	Stationary (fixed location)

Table 4.3: E2E-MAC & QoE-RAHAS: A list of parameters values under ns-3 simulation.

Parameter Name	Parameter Value
UW_{len}	1 second
sub-slot W	100 millisecond
Weighted averaging function	Linear

addition, we monitor the video packets loss ratio. These metrics allow us to assess both the quality of experience (QoE) and the network utilization level. We calculate the total time the client spends streaming at each video quality level and calculate the ratio with respect to the total streaming time. The total streaming time includes both playback periods and rebuffering periods but does not include the initial buffering period. We also measure the rebuffering time and calculate the ratio with respect to the total streaming time.

For sake of simplicity, we represent the channel conditions in terms of the bit error rate (BER). We evaluate the E2E algorithm designed to emulate the Akamai algorithm d our proposed E2E-MAC algorithm under three different channel conditions: excellent, moderate, and poor corresponding to BERs of 10^{-6}, 10^{-5}, and 10^{-4}, respectively. We evaluate the performance in terms of the video frame loss ratio, the frequency of rebuffering events, and the average length of rebuffering events. For some experiments, we also provide the maximum and minimum length of rebuffering periods and the frequency of the quality switching decisions. The latter is defined as the ratio between the number of quality switching events and the maximum number of possible quality changes, which is also the total number of available video chunks for the target video content.

The proportions of streaming time spent playing each video quality level are shown in Figure 4.3. The re-buffering proportions are shown in Figure 4.4. A summary of the

video frame loss ratios are given in Table 4.4, followed by a summary of the re-buffering statistics in Table 4.5.

Under excellent conditions, the E2E-MAC algorithm tends to seek higher rates than the comparison algorithm. Unfortunately, this greedy behavior backfires sometimes. This is why the proposed E2E-MAC algorithm has a higher re-buffering ratio (6%) compared with to the E2E algorithm (3.7%) under excellent conditions, as shown in Table 4.5. In general, the E2E-MAC algorithm outperforms the E2E algorithm in terms of running higher video quality levels for significantly longer periods as shown in Figure 4.3. The E2E-MAC algorithm spends a significantly higher proportion of time (39%) running on level 4 and 5 combined than the E2E algorithm (26%). The E2E-MAC algorithm reduces the time spent running on the lowest level to only (1%). A fix to handle the observed rise in rebuffering ratio over excellent channels is discussed later in this section.

Under moderate conditions, the The E2E-MAC algorithm outperforms the E2E algorithm with a much lower re-buffering ratio as shown in Table 4.5, and the user enjoys much longer periods running higher video quality levels as shown in Figure 4.3. The E2E algorithm runs 15% of the time on level 1, 67% on level 2 and 3 combined, and 8.3% on level 4; does not run at level 5. On the other hand, the E2E-MAC algorithm runs only 8% of time on level 1, and 56% on level 3 and 4 combined. In addition, it successfully runs 7% of time on level 5 without increasing the re-buffering overhead.

Under poor conditions, the E2E algorithm runs 46% of time on level 1, 26% on level

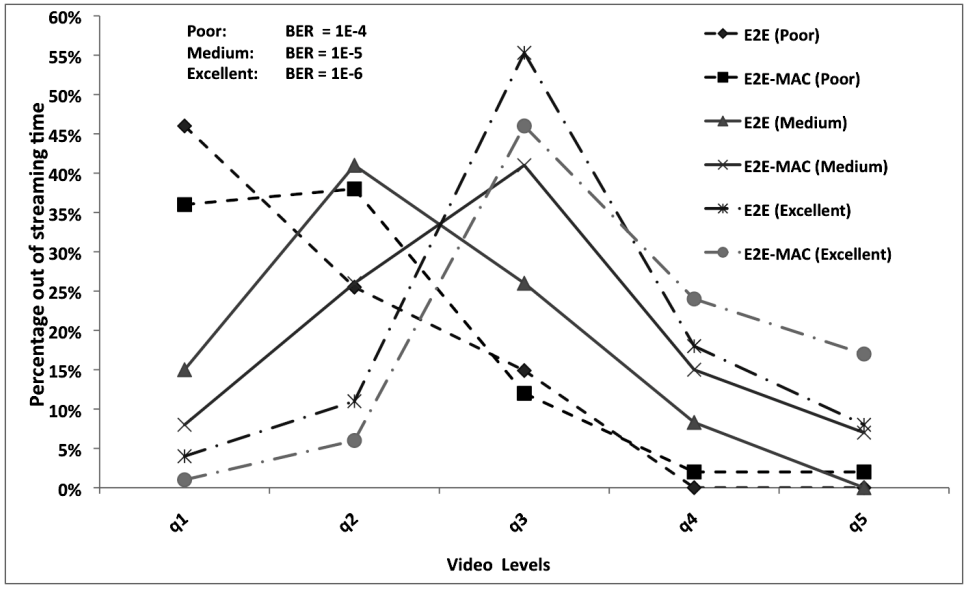


Figure 4.3: Achieved video quality levels of E2E-MAC VS E2E algorithms under different channel conditions.

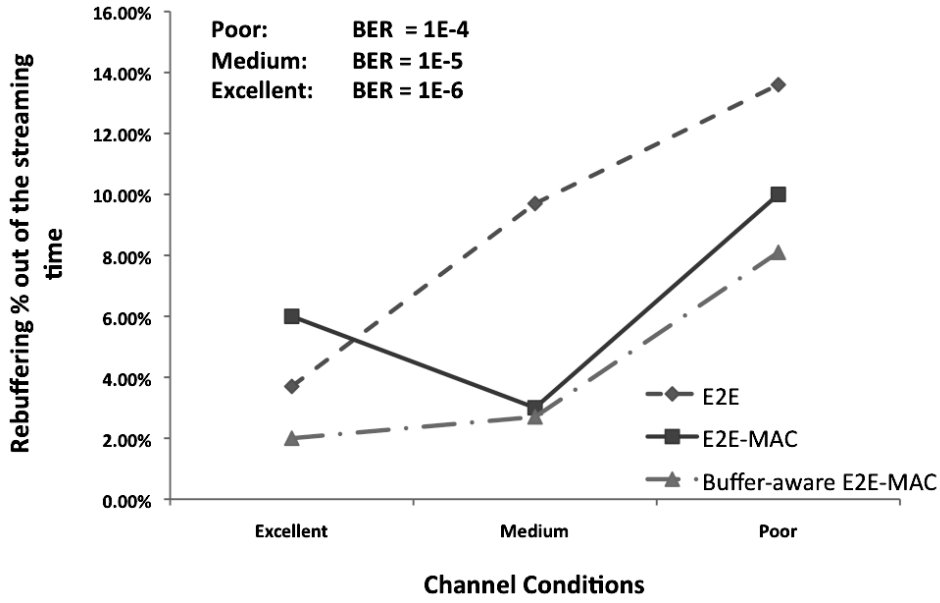


Figure 4.4: Re-buffering ratio under different conditions.

2, 15% on level 3, and 0% on levels 4 and 5. On the other hand, the E2E-MAC algorithm runs 36% of time on level 1, 38% on level 2, 12% on level 3, and 2% of time each on levels 4

and 5. We believe this is due to the algorithm's ability to use instantaneous opportunities when the wireless link improves slightly. Such micro dynamics cannot be detected by the E2E algorithm. In terms of re-buffering, which has a significant influence on user satisfaction, the E2E algorithm suffers re-buffering for 14% of the streaming time. This is because it takes more time to recover from a wrong decision. The E2E-MAC algorithm spends less time re-buffering (10%). This is a 26.4% reduction in re-buffering time under poor channel conditions.

In general, the E2E-MAC algorithm suffers more rebuffering instances than the E2E algorithm. However, the total rebuffering time is much shorter because E2E-MAC recovers more quickly due to its ability to check the wireless link conditions. For example, the average length of a rebuffering period is 4.06 seconds for the E2E algorithm over all conditions, while the average is only 2.1 seconds for the E2E-MAC algorithm.

In addition to making the rebuffering events shorter and increasing the average quality level, the video packets loss ratio is another important metric to consider as we did not use a subjective experiment to evaluate user satisfaction. Table 4.4 shows how the E2E-MAC achieves a significant reduction in the video packets loss ratio. This is a direct result of choosing the optimal quality level that keeps the required rate below the available bandwidth.

In order to avoid the increase in the rebuffering ratio under excellent channel conditions, we made a simple modification to the E2E-MAC algorithm. Instead of switching to a higher quality level guided only by the rules given in Algorithm 1, we

modified the algorithm to also check the current buffer size. The modified algorithm does not switch to a higher quality unless the buffer size is above a threshold. For initial testing purposes, we choose a threshold value of three seconds of video, the same as the length of the adaptation window k. Hence, we modified the rules in lines 4, 6, and 14 so that requested quality is not incremented unless the current playback buffer size is greater than k seconds. The far right column of Table 4.5 provides the updated rebuffering results. Figure 4.4 illustrates the rebuffering ratios under all conditions for the traditional E2E algorithm, the regular E2E-MAC, and the E2E-MAC with buffer-check. The E2E-MAC with buffer-check algorithm achieves a lower rebuffering ratio than the E2E algorithm under all testing conditions. A thorough investigation of buffer-awareness is discussed in section 4.2, where we present an enhanced algorithm called Buffer-Aware E2E-MAC (BAE2E-MAC).

Table 4.4: Video Packets Loss Ratio.

Channel conditions	E2E Algorithm	E2E-MAC Algorithm
Poor	54%	43%
Medium	26%	18%
Excellent	17%	9%

Table 4.5: Breakdown of rebuffering (underrun) and switching measurements.

	E2E			E2E-MAC			E2E-MAC with buffer-check		
	Poor	Medium	Excellent	Poor	Medium	Excellent	Poor	Medium	Excellent
Total underrun time (Seconds)	47.2	32.23	11.53	33.3	9.28	19.15	26.44	8.32	6.12
Underrun Instances	7	11	3	10	5	12	8	7	7
Average length of underrun period (Seconds)	5.83	2.65	3.7	3	1.8	1.5	3.30	1.19	0.87
Total Underrun time to streaming time ratio	13.60%	9.70%	3.70%	10%	3%	6%	8.1%	2.7%	2%
Quality switching instances	19	15	6	24	17	15	26	16	11

4.2 The Buffer-Aware E2E-MAC Quality Controller (BAE2E-MAC)

4.2.1 Proposed Approach

Inspired by the improvement in the E2E-MAC performance by adding a buffer threshold, we sought to improve this component of the algorithm, before testing it under more realistic conditions, specifically under adaptive IEEE 802.11 physical rates.

Here is a list of the modifications and additional features we have added:

- Instead of the fixed constant threshold of 3 seconds—which we used in the previous section—we propose in this section to use an adaptive threshold $bufferAdaptiveThresh$. The $bufferAdaptiveThresh$ is updated based on how long the link suffers a recession and its magnitude. Basically, we add a rule to check for how many slots the trend ϵ has been negative and to calculate the relative magnitude of this trend. Then, we assign a new value to the $bufferAdaptiveThresh$ that is a multiple of an initial allowance $initialAllowance$ plus a time period equal to a single time slot k. The current buffer individual rule is shown in Algorithm 2 in (line 3).

  The $bufferAdaptiveThresh$ is reset to the initial allowance immediately once any other rule detects an improvement in link quality. For all simulation results, the

algorithm does not run until the initial buffering period $initialBufferingPeriod$ is over.

- Instead of using the average trend over the adaptation window, given by $\epsilon_{E2E-MAC} = \frac{(\alpha_1+...+\alpha_k)}{k}$, we use an exponential weighted normalized trend that is given by:

$$\epsilon = \frac{\sum_{i=1}^{k}(normalized(\alpha_i)(\omega)^{k-i+1})}{\sum_{i=1}^{k}\omega^i}, \tag{4.4}$$

  with an ω value of 0.5.

- Since an explicit rule is now provided to check the buffer status and jump to the lowest quality level if needed, we remove the prior rule designed to avoid buffer underrun. We believe that the new buffer rule with adaptive threshold is sufficient to avoid buffer underrun.

- In addition, we modify the gradual quality decrementing rule found in the E2E-MAC algorithm to be less conservative. We only decrease the quality when the trend is negative for two consecutive slots and the throughput is less than the required bandwidth at the current slot. The new rule after modification is given in (line 10).

4.2.2 Simulation Setup and Evaluation

We test the two controllers (i.e., BAE2E-MAC, QoE-RAHAS) under two major scenarios. Under the first scenario, we create a bottleneck on the wireless link by simulating three

Algorithm 2 The Buffer-Aware E2E-MAC Controller (BAE2E-MAC).

$Counter_{NC} = 0$, $bufferSize = initialBufferingPeriod$,
$bufferAdaptiveThresh = initialAllowance$
$bufferThreatCounter = 0$
$k = 3$ and $t = 1$
while $t < \lfloor L/k \rfloor$ **do**
  if $(bufferSize < bufferAdaptiveThresh + k)$ **then**
    $q_{t+1} = 1$
    $bufferThreatCounter = +1$
    $Counter_{NC} = 0$
    $bufferAdaptiveThresh = updateBufferThreshold(...)$
  else if $(\lambda(t) > \theta_{q_t+2})$ && $((\epsilon(t) > 0)$ && $(\epsilon(t-1) > 0)$ && $(\epsilon(t-2) > 0))$ **then**
    $q_{t+1} = q_t + 2$
    $Counter_{NC} = 0$
    $bufferAdaptiveThresh = initialAllowance$
    $bufferThreatCounter = 0$
  else if $(\lambda(t) > \theta_{q_t+1})$ && $((\epsilon(t) > 0)$ && $(\epsilon(t-1) > 0))$ **then**
    $q_{t+1} = q_t + 1$
    $Counter_{NC} = 0$
    $bufferAdaptiveThresh = initialAllowance$
    $bufferThreatCounter = 0$
  else if $(\lambda(t) < \theta_{q_t})$ && $(q_t > 1)$ && $(\epsilon(t) < 0)$ && $(\epsilon(t-1) < 0)$ **then**
    $q_{t+1} = q_t - 1$ {decrease gradually}
  else if $(\lambda(t) > \theta_{q_t+1})$ && $((\epsilon(t) > 0) \oplus (\epsilon(t-1) > 0))$ **then**
    $Counter_{NC} = Counter_{NC} + 1$
  else if $(Counter_{NC} > 4)$ && $(q_t \leq 3)$ **then**
    $q_{t+1} = q_t + 1$
    $Counter_{NC} = 0$
    $bufferAdaptiveThresh = initialAllowance$
    $bufferThreatCounter = 0$
  else
    $Counter_{NC} = Counter_{NC} + 1$
  end if
  $t = t + 1$
end while

Algorithm 3 updateBufferThreshold (paramList)

1: Check paramList: $bufferSize, initialBufferingPeriod$, $bufferThreatCounter$, $initialAllowance$,$\epsilon(t), \epsilon(t-1), \epsilon(t-2)$
2: **if** $bufferSize >= initialBufferingPeriod$ **then**
3: $return(currentbufferAdaptiveThresh)$
4: **else if** $(\lambda(t) < \theta_{q_t})$ && $(bufferThreatCounter > 1)$ && $(\epsilon(t) < 0)$ && $(\epsilon(t-1) < 0)$ **then**
5: $return(2 * initialAllowance)$,
6: **else if** $(\lambda(t) < \theta_{q_t})$ && $(bufferThreatCounter > 2)$ && $(\epsilon(t) < 0)$ && $(\epsilon(t-1) < 0)$ **then**
7: $return(3 * initialAllowance)$,
8: **else if** $(\lambda(t) < \theta_{q_t})$ && $(bufferThreatCounter > 3)$ && $(\epsilon(t) < 0)$ && $(\epsilon(t-1) < 0)$ **then**
9: $return(initialBufferingPeriod)$
10: **else**
11: $return(currentbufferAdaptiveThresh)$
12: **end if**

different channel conditions: unstable, moderate, and stable. Under the second scenario, we use background traffic to create a bottleneck on the Ethernet link.

4.2.2.1 A Bottleneck over The Wireless Link

For more realistic performance evaluation of the proposed controller, we have made few modifications to the simulation setup used earlier. Instead of five quality levels only, we use seven different quality levels with more diverse bandwidth requirements, which is more appropriate to the use of IEEE 802.11g wireless standard, which has 8 different data rates, in addition to the standard rate for control frames (e.g., acknowledgments, RT/CTS). The minimum quality level has a rate requirement of 0.35 Mbps, and the maximum quality level has a rate requirement of about 6.2 Mbps as shown in Table 4.6.

Table 4.6: Generated video quality levels and corresponding rates.

Quality Level	Rate (Mbps)
q_1	0.35
q_2	0.7
q_3	1.3
q_4	1.9
q_5	2.8
q_6	4.5
q_7	6.2

It is worth mentioning that the authors in [79], where the iMinds' solutions were initially proposed, used video traces with bandwidth requirements of an average 0.91 Mbps and a variance 0.36 Mbps. We use more realistic conditions based on existing commercial VoD services, hence we use bandwidth requirements with an average of 2.54 Mbps and a variance of 3.92 Mbps.

We compare the BAE2E-MAC algorithm to the QoE-driven Rate Adaptation Heuristic for Adaptive Video Streaming (QoE-RAHAS) that is proposed by the iMinds labs in [104]. We have tried to create a fair comparison environment by replicating the the simulations used to test QoE-RAHAS in previous work. In [82, 104], the authors have shown that QoE-RAHAS outperforms Microsoft ISS Smooth Streaming (MSS) client; because of this we have chosen to compare the BAE2E-MAC only to QoE-RAHAS.

The wireless channel bandwidth in our simulation varies in accordance with the Signal-to-Interference-plus-Noise-Ratio (SINR). The SINR is simulated by keeping the

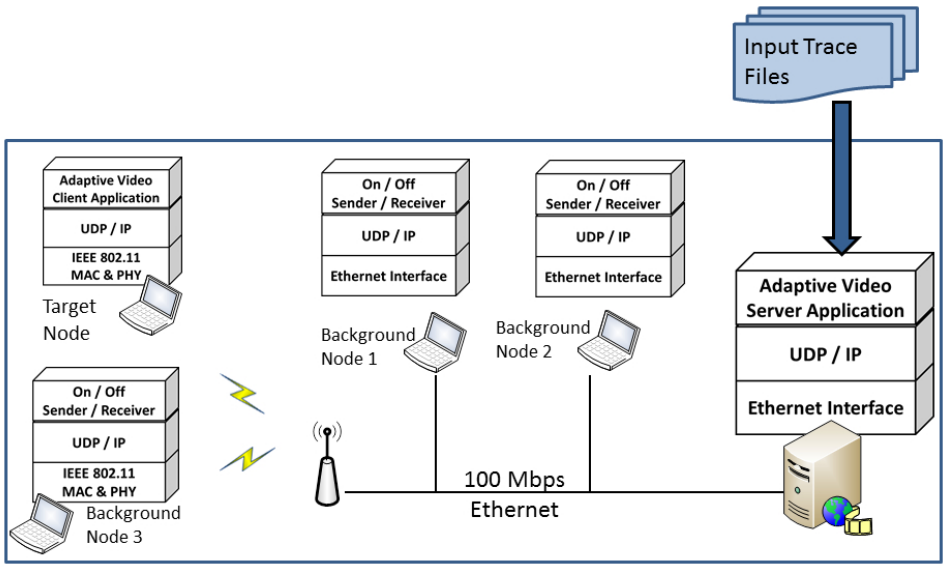


Figure 4.5: Simulation setup for testing BAE2E-MAC and QoE-RAHAS.

noise floor fixed value and assuming that the interference follows a log-normal distribution with location and scale parameters $\mu = 0$ and $\sigma = 1$, respectively. The influence of propagation loss is assumed to be normalized through adequate power control.

Our simulation setup uses an IEEE 802.11g connection with a theoretical maximum rate of 54 Mbps. The ns-3 simulator has two major models for mapping the SINR to frame success rate (FSR) over IEEE 802.11g connections, the Yans and Nist models. We have chosen to use the Nist model as recommended by the ns-3 documentation due to its ability to provide better matching results to testbed measurements. An illustration of the Nist model is shown in Figure 4.6. The frame success rates (FSR) is simulated based on a data payload of 1400 bytes, in addition to standard transport, network and MAC layers headers.

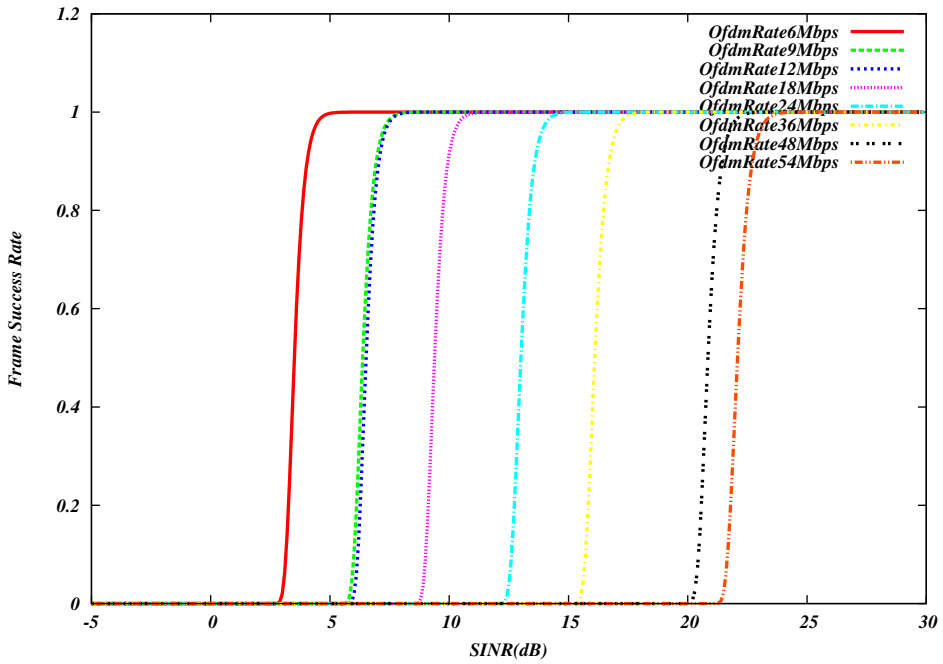


Figure 4.6: Frame success probability with respect to SINR (Nist model).

We simulate the BAE2E-MAC controller and the QoE-RAHAS under adaptive PHY rates. In our experiments, we deploy two well known rate adaptation algorithms (RAAs), the Minstrel algorithm [100] and the receiver-based autorate (RBAR) algorithm (a.k.a. Ideal RAA). We use Minstrel as the actual rate adaptation algorithm run over the simulated access point (AP) and stations (STAs), while the RBAR runs in the background for comparison.

We have chosen the RBAR and Minstrel because they represent the two categories of rate adaptation algorithms, PHY-based algorithms and link-layer-based algorithms, respectively. While the former assumes the possibility of sending instantaneous feedback regarding the received SINR values from receivers to transmitters using an out-of-band mechanism, the latter does not depend on PHY-layer measurements.

Instead, Minstrel relies on MAC-layer, specifically the transmission statistics for each available transmission rate. Minstrel has been widely adopted, becoming the main RAA deployed in the Linux kernel and used by many wireless device vendors. However, some studies have been skeptical about its performance under unstable channel conditions, specifically claiming that other algorithms outperform minstrel when channel conditions are not stable [103]. Generally, it has been found that Minstrel performs fairly well when channel conditions are improving (by responding quickly enough), but that it struggles when channel conditions are deteriorating (taking relatively longer time to respond).

The most important ns-3 simulation parameters are listed in Table 4.7. To create a fair comparison environment for other controllers, we have used their suggested initial buffering period of 10 seconds. However, we provide a sample of results when a shorter initial buffering period is used. Viewer surveys suggest that much shorter initial buffering is associated with better QoE. An initial buffering period of 10 seconds is too long according to these surveys. The values used for the major parameters of BAE2E-MAC and QoE-RAHAS are presented separately in Table 4.8.

We report our results after running three different levels of channel stability. The channel stability is described in terms of the granularity of changes made to the signal to interference plus noise ratio (SINR). For the least stable channel, changes occur every 200 ms or less; for the moderate channel, SINR changes every 200 to 500 ms; and the most stable channel has SINR changes every 500 ms or longer. The least stable channel is the

Table 4.7: The ns-3 most important simulation parameters.

Parameter Name	Parameter Value
Video Clip Length	5 minutes (300 seconds)
Video chunk Length	2 seconds
Initial buffering delay	10 seconds
Video quality levels	7
Ethernet Link Capacity	100 Mbps
PHY Rate Manager	Minstrel
Available Phy Data Rates	(6, 9, 12, 18, 24, 36, 48, 54) Mbps
RTS/CTS	Disabled
Short Retry limit	7
Mobility Model	None

Table 4.8: BAE2E-MAC: A list of parameters values under ns-3 simulation.

Parameter Name	Parameter Value
BAE2E-MAC	
UW_{len}	0.5 s
sub-slot W	50 ms
Decision Time Window k	2 s
Weighted averaging function	Exponential
QoE-RAHAS	
$DecisionWindow$	Same length of a single chunk of video (i.e., 2 s) second
W	50 ms
sub-slot k	2 s

most challenging, and while the most stable channel is more predictable, even with the existence of background traffic.

For an accurate comparison between QoE-RAHAS and our proposed controller (BAE2E-MAC), we evaluate both in terms of the same quality of experience (QoE)-centered metrics used in [79, 80] to evaluate QoE-RAHAS and some reinforcement-learning-based controllers as well. We will use the same metric to evaluate our proposed solution in Chapter 5.

The metric is called estimated mean opinion score (estimated-MOS). It is a quantitative version of the well-known and widely-used subjective MOS metric. In [108, 109], the authors presented two models to estimate the MOS. The two models can be combined into a single model, which associates a few quantitative satisfaction components with the estimated QoE perceived by the user. These components are the frequency of buffer underrun events, the average length of buffer underrun events, and the mean and standard deviation of the achieved video quality. The metric is given by a linear combination of components as follows:

$$MOS_{est} = \max(5.67\mu - 6.72\sigma - 4.95\phi + 0.17, 0), \tag{4.5}$$

where μ and σ are the quality level average and standard deviation respectively, and ϕ is associated with the rebuffering frequency and the average freeze period. To calculate ϕ, we use:

$$\phi = \frac{7}{8}\max\left(\frac{ln(F_{freq})}{6} + 1, 0\right) + \frac{1}{8}\left(\frac{(\min(FT_{avg}, 15))}{15}\right), \tag{4.6}$$

where F_{freq} and FT_{avg} are the frequency of rebuffering events and the average length of rebuffering period.

4.2.2.2 A Bottleneck over The Ethernet Link

In this scenario, we have background traffic that hits the Ethernet link according to specific patterns. The first pattern is fixed traffic, the second is a sinusoidal function, the third is a step function, and the fourth is a burst traffic that follows a random distribution (i.e., Poisson). The point is to verify that our proposed technique is not only suitable for the cases where the bottleneck is the wireless link.

4.2.3 Results & Analysis

We summarize our results from comparing the two controllers (BAE2E-MAC and QoE-RAHAS). We start with the first scenario, where the wireless link is the bottleneck before we follows with the second scenario, where the Ethernet link is the bottleneck.

4.2.3.1 A Bottleneck over The Wireless Link

We test the bottleneck over the wireless link under the three channel conditions that we defined earlier: the unstable, moderate, and stable conditions. We have run each scenario a number of times to obtain confidence in the results.

For the unstable channel, our main finding is that the performance of quality controllers are mainly dominated by poor performance of the PHY rate adaptation algorithm (Minstrel). Minstrel performance was disappointing under unstable conditions, which confirms the claims in [103].

Minstrel was not able to recover quickly enough or to follow rapid changes in the channel conditions (especially when the channel conditions improve), hence it was unable to utilize short periods in which conditions were improved; as a result, Minstrel had poor average performance. Instead, Minstrel provided a smooth low rate, which dropped for short intervals to a very low rate. We believe that a carefully chosen constant rate (using empirical-based calibration) may have provided better throughput under such conditions. Figure 4.7 shows

Figure 4.7, shows the average performance of Minstrel compared to the RBAR algorithm. The multi-rate retry chain (MRRC) mechanism implemented by Minstrel does not work efficiently enough under unstable channel conditions. In contrast, the RBAR was able to exploit short periods of good conditions, and pushing for higher rates in such intervals and achieving overall better performance.

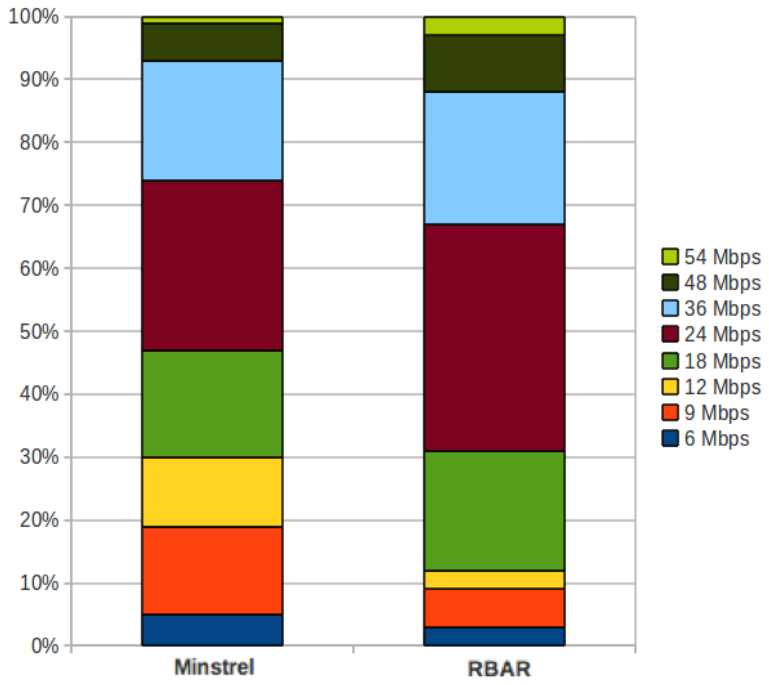


Figure 4.7: Rate selection breakdown under unstable channel.

Figure 4.8 shows the available bandwidth versus the achievable throughput according to Minstrel, in addition to the estimated bandwidth perceived by each video quality controller. Looking at Figure 4.8, which shows 6 seconds of the 300 seconds simulated video stream, we see how the available bandwidth estimated by the two video controllers is pushed down by Minstrel. Although the available bandwidth changes quickly, it has been on average above the upper limit that Minstrel has been able to reach. Although Minstrel does not give space for the controllers to make a big difference, we see that BAE2E-MAC keeps trying to achieve better performance, indicated by the updates it makes to its estimated bandwidth value. On the other hand, QoE-RAHAS continues following a safe constant low level policy. However, QoE-RAHAS did not respond to rate collapses, that occurred few times and exposed the

playback buffer to underrun possibilities. This explains the lower average estimated-MOS achieved by QoE-RAHAS as shown in Figure 4.9.

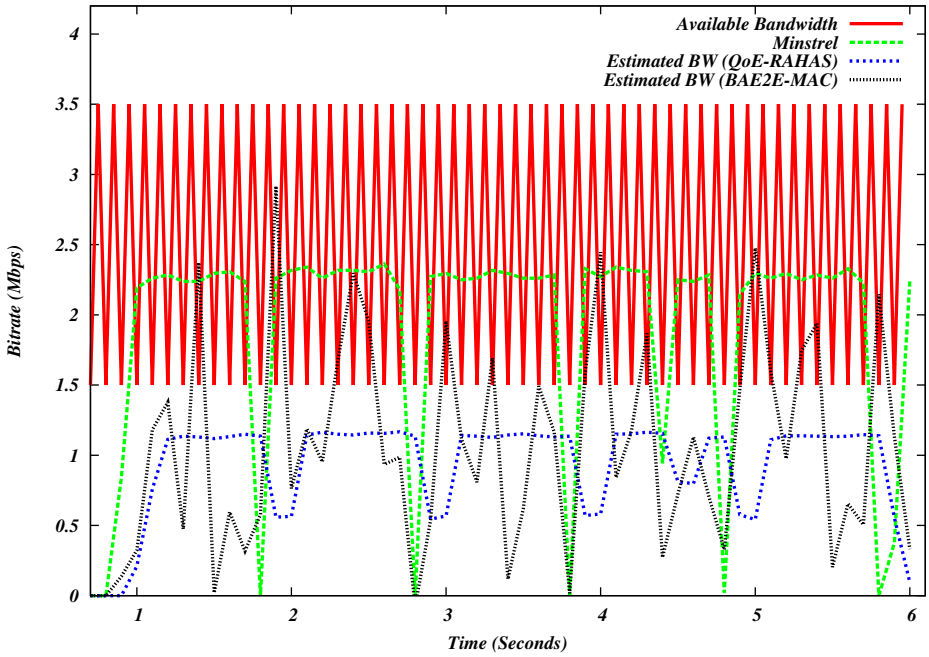


Figure 4.8: A snapshot of performance comparison between BA2E2E-MAC and QoE-RAHAS [unstable channel].

Figure 4.9 also shows that our proposed algorithm performs better than QoE-RAHAS under unstable channel conditions when using a shorter initial buffering period. This is a great advantage, as previously mentioned in the literature survey, since viewer feedback is highly negative about longer initial buffering periods. While most research work assumes acceptable initial buffering of 10 seconds, surveys show that an initial delay longer than few seconds results in many viewers that quit watching the target content.

For the second type of channel conditions (under moderate conditions), we notice

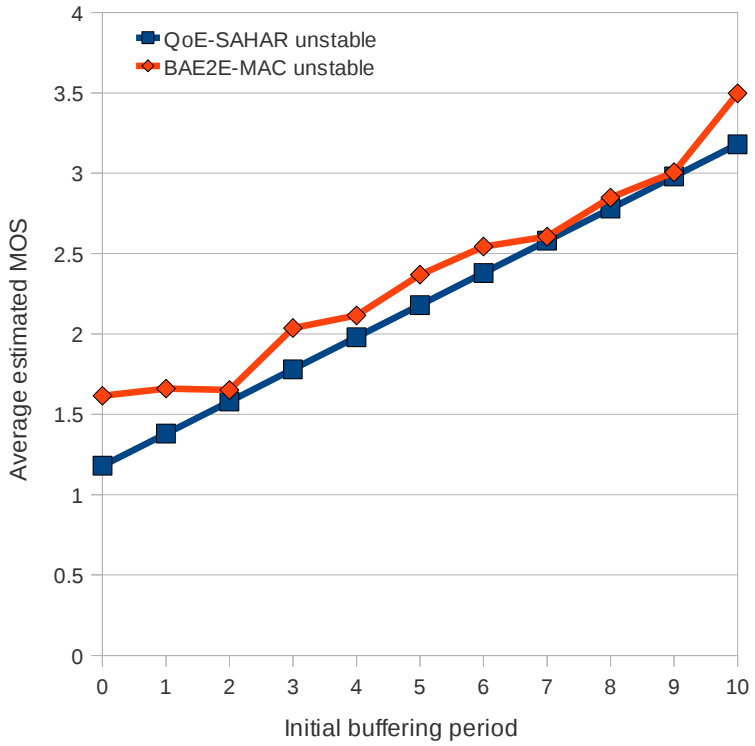


Figure 4.9: Initial buffer period vs estimated MOS with an unstable channel.

that the gap in performance between Minstrel and RBAR starts to shrink though Minstrel still reduces the freedom available to the video quality controllers. In Figure 4.10, RBAR managed to increase the percent of time spent on the highest possible rates while the Minstrel did not.

Since Minstrel started to provide a higher rate, the quality controllers started to move within a wider range. We observe in Figure 4.11 the onset of some over-estimation events. Also, the ability of the BAE2E-MAC controller to exploit short improvements in the available bandwidth begins to become more obvious.

Figure 4.12 shows the influence of the initial buffering period on the two controllers

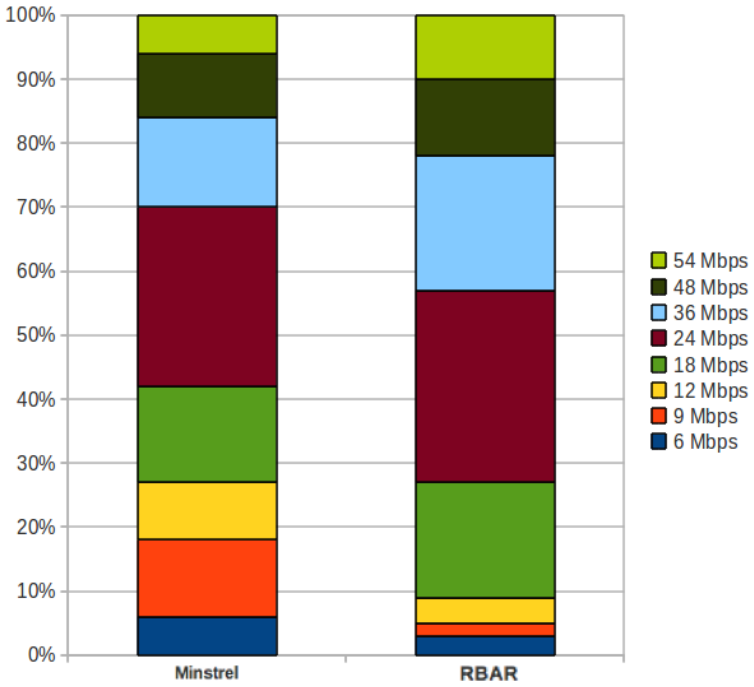


Figure 4.10: Rate selection breakdown under moderate channel conditions.

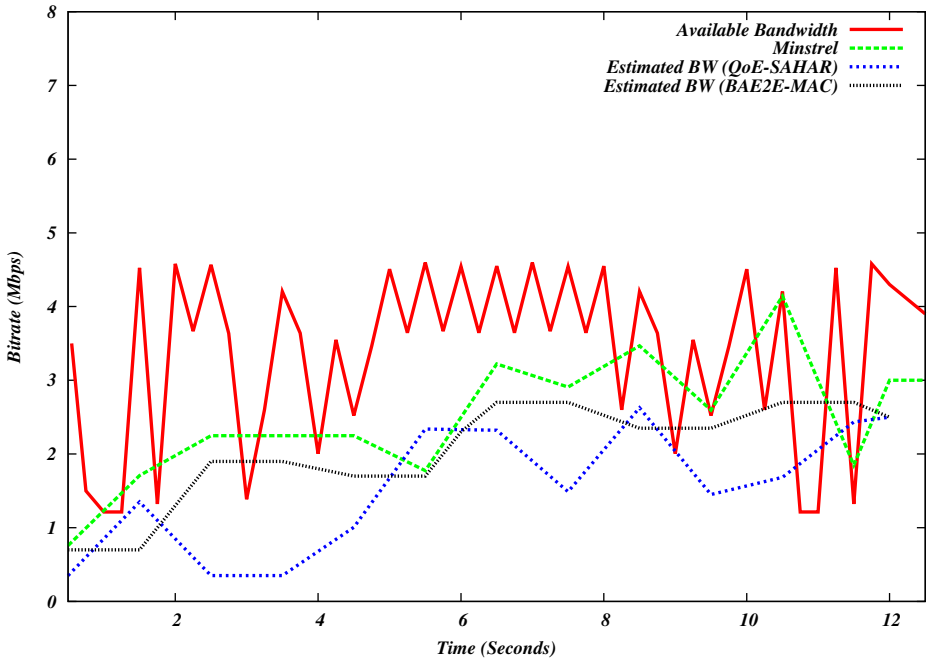


Figure 4.11: A snapshot of performance comparison between BA2E2E-MAC and QoE [moderate channel].

performance. Clearly, our proposed controller performs better than QoE-RAHAS under shorter initial buffering periods again under moderate channel conditions.

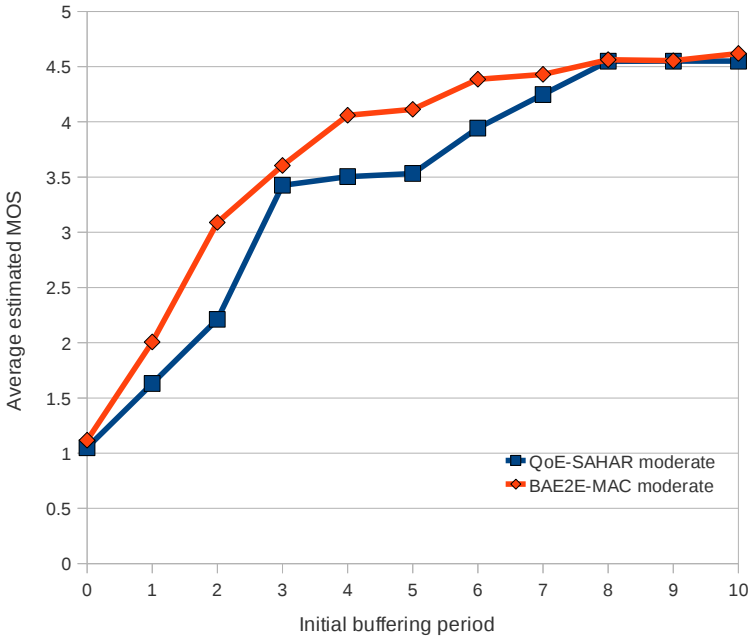


Figure 4.12: Initial buffer period vs estimated MOS under moderate channel conditions.

Under stable channel conditions, the gap between RBAR and Minstrel is quite small as shown in Figure 4.13, and a wide range of quality levels become potentially achievable for the video quality controllers. Unsurprisingly, the gap between the two controllers disappeared because the channel is too stable to allow BAE2E-MAC to show better adaptation than an end-to-end controller, except at moments when the available bandwidth collapses suddenly due to events on the wireless link as illustrated in Figure 4.14.

When it comes to performance under different initial buffering periods, BAE2E-MAC still outperforms QoE-RAHAS, especially for initial buffering periods of less than 6

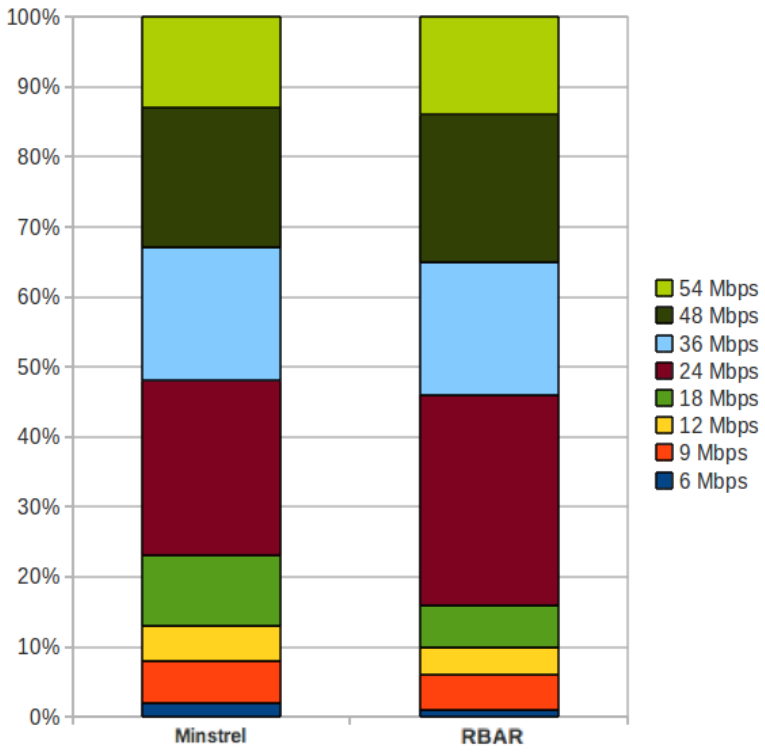


Figure 4.13: Rate selection breakdown under stable channel conditions.

seconds. Once the initial delay becomes 6 seconds or longer, the two algorithms perform similarly. This is clearly shown in Figure 4.15.

we have noticed that BAE2E-MAC often requests a higher quality then reverses the decision almost immediately in the next decision slot. This is due to a possible misinterpretation of the improvement trend under specific circumstances. For example, a decrease in the retry counter was noted by BAE2E-MAC after Minstrel responded to a frame failure by lowering the PHY rate. While the probability of frame success has increased, the achievable throughput may have decreased dramatically. The BAE2E-MAC rules tend to assume that every improvement in the trend ϵ indicates potential higher throughput until measuring the throughput proves the opposite. This

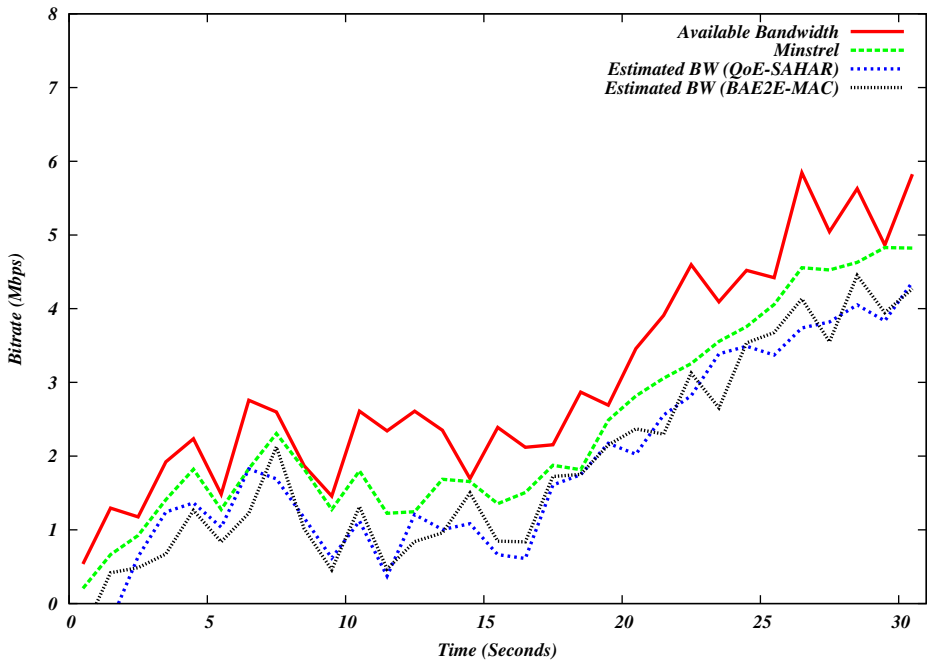


Figure 4.14: A snapshot of performance comparison between BA2E2E-MAC and QoE-RAHAS under stable channel conditions.

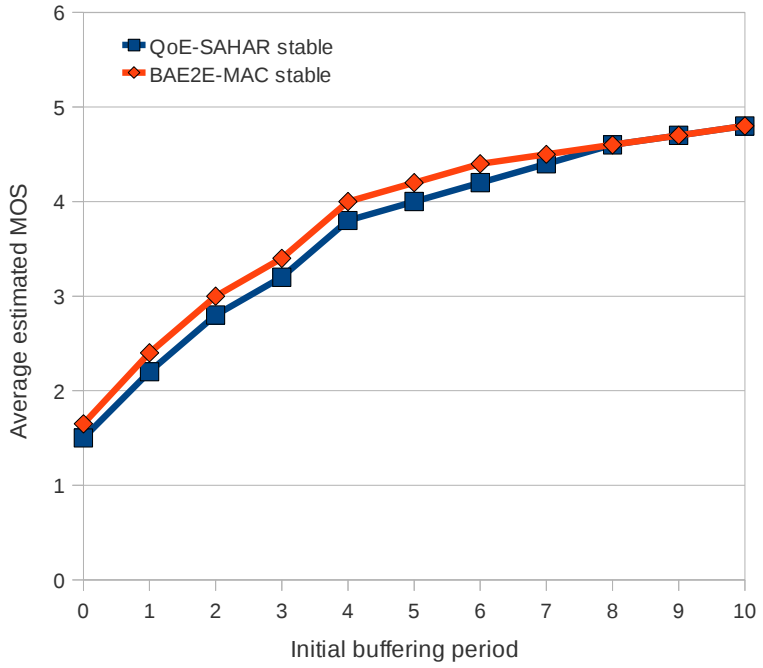


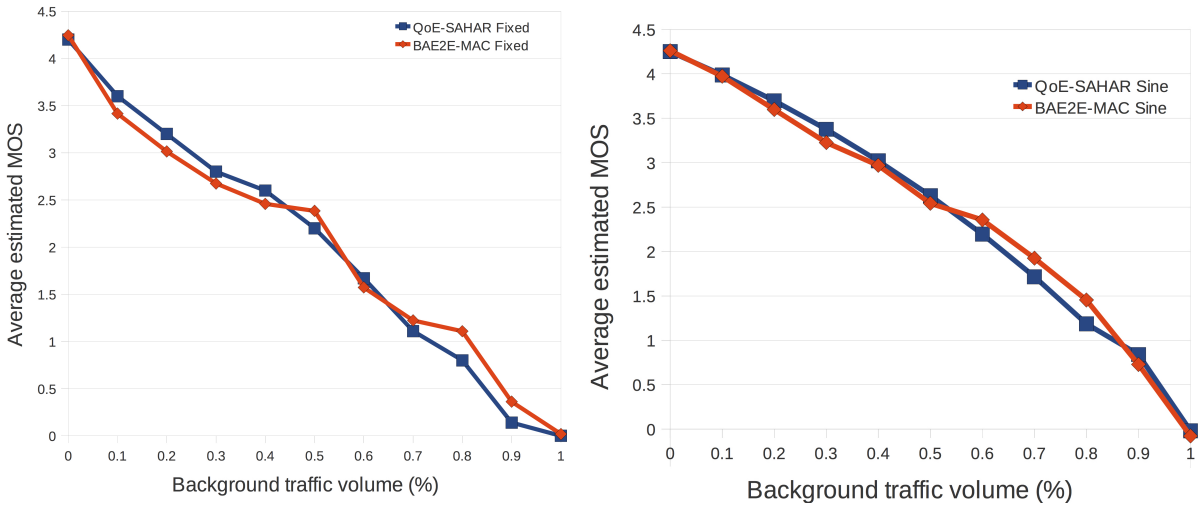
Figure 4.15: Initial buffer period vs estimated MOS under stable channel conditions.

can be rapidly addressed by checking the throughput $\lambda(t)$ at the next slot, which guarantees the controller does not ask for much higher quality and to reverse if needed.

Since BAE2E-MAC aims to monitor the wireless link, but it suffers from a single interpretation of the changes in the retry counter, we suggest adding a new feature to the BA2E2E controller by making it aware of Minstrel statistics and its interpretations that are related to how the MRRC mechanism works. A prototype of this modification is implemented using the FINS Framework in Section 5.5. Also, we apply this learned lesson in designing the reward function for the RL-based solution, which we present in Section 5.4.

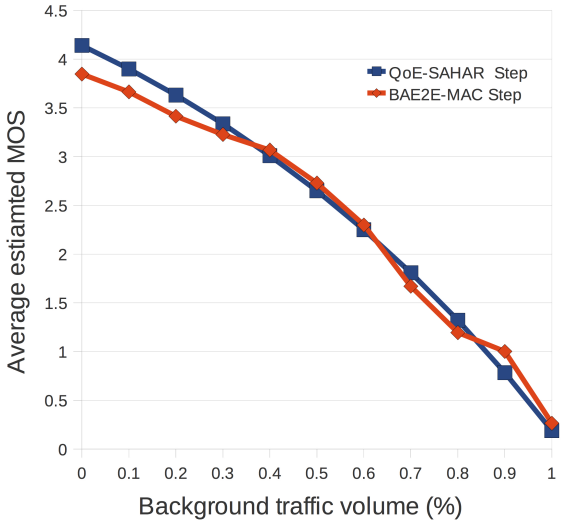
4.2.3.2 A Bottleneck over The Ethernet Link

The main finding of the experiments with a bottleneck over the Ethernet link is that our proposed controller (BAE2E-MAC) performs as well as QoE-RAHAS. Although QoE-RAHAS provides better results when it comes to random burst traffic in the background (the Poisson scenario), we have observed that this is highly dependent on the initial buffering period. All the results in Figure 4.16 assume an initial buffering of 10 seconds. If the initial buffering period is decreased, the performance of QoE-RAHAS also decreases, similarly to what we saw in the wireless bottleneck scenario. In experiments with initial buffering below 4 seconds, BAE2E-MAC outperforms QoE-RAHAS under all background configurations (i.e., fixed, sine, step, random).

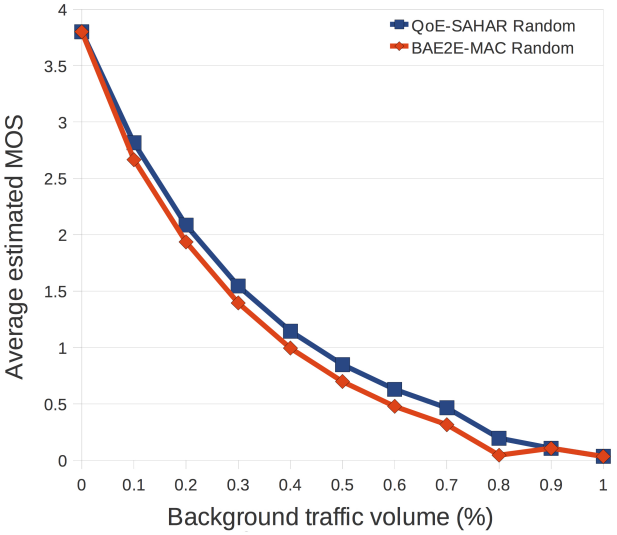


(a) Fixed pattern

(b) Sine function



(c) Step function



(d) Random burst (Poisson)

Figure 4.16: Performance of BAE2E-MAC vs QoE-RAHAS under different background traffic patterns.

It is also clear from Figure 4.16 that as the background traffic increases with respect to the total available bandwidth, BAE2E-MAC starts to slightly outperform QoE-RAHAS. There are situations in which the video link is being starved where BAE2E-MAC provides a 22% improvement in the QoE in comparison with QoE-RAHAS, specifically when background traffic exceeds 60% of the available bandwidth.

4.3 Conclusion

In this chapter, we have presented a cross-layer algorithm, for adaptive video streaming over WiFi access network. We presented two version of the algorithm, E2E-MAC, which does not monitor the playback buffer and BAE2E-MAC, which does not permit the buffer size to drop below an adaptive threshold.

In the first part of this chapter, we have shown how the E2E-MAC algorithm considers both wireless link conditions and E2E measurements. The algorithm is evaluated using a set of ns-3 simulation modules that we developed. The proposed algorithm outperforms the traditional E2E-based technique under various channel conditions with a slight increase in re-buffering overhead under excellent wireless channels. The rebuffering occurs due to the algorithm's greedy behavior in seeking higher video quality. This problem is mitigated by incorporating the buffer status into the algorithm.

Noting that a simple rule to monitor the buffer size improve performance, we

created a more sophisticated version of the algorithm exploiting an efficient heuristic to monitor buffer size, BAE2E-MAC. We compared BAE2E-MAC with QoE-RAHAS, which is among the latest heuristic algorithms presented by the research community.

The comparison results under a wide range of conditions have verified the efficiency of the BAE2E-MAC and its ability to perform as well as QoE-RAHAS in general. Our proposed solution outperformed QoE-RAHAS in all scenarios under shorter initial buffering delays, which is a significant advantage for streaming video viewers.

Given the promising results acquired by the BASE2E-MAC algorithm, we recommend cross-layer solutions to address the challenges of adaptive video streaming over wireless networks (e.g., WiFi, WiMAX, LTE). We believe an optimal controller can be designed by jointly optimizing bandwidth estimation across the transport, MAC, and physical layers. In the next chapter, we design an optimal cross-layer-based controller using the Markov decision processes framework and reinforcement learning (RL).

Chapter 5

Adaptive Video Streaming over IEEE 802.11: A Learning-based Approach

In this chapter we propose a framework to solve the HTTP-based AVS problem by deploying a well known mathematical optimization tool, the Markov Decision Processes (MDP). We start with a short introduction to the MDP, model-based solution techniques, and model-free approximate solution methods (e.g., reinforcement learning (RL)).

In the second section, we formulate the HTTP-based AVS problem as an Markov decision process, defining the system system state model and the reward function.

In section three, we solve the proposed system model using an offline algorithm, assuming a predefined transition probability model for the dynamics of the HAVS process

over wireless. We illustrate and analyze the solution for the system using an offline policy iteration algorithm.

In section four, the system is solved using reinforcement learning where no transition probability model exists. Instead, we run an ns-3 simulation where the system states and reward functions are learned by an RL agent. Evaluation results are presented and discussed.

Section five presents the prototype we developed, using the FINS Framework, for the RL agent-based solution under real world experimental setup.

Finally, we conclude by highlighting our most significant findings.

5.1 Introduction to Markov Decision Processes and Reinforcement Learning

Markov decision processes (MDPs) are a mathematical optimization framework that aim to optimize sequential decision-making processes under uncertainty [110]. The framework has been deployed in several fields under various names such as optimal control, stochastic dynamic programming, and stochastic control. The two classic categories of the problems that are addressed by the framework are usually called finite-horizon and infinite-horizon MDP. A widely used variant of the framework is called partially observable Markov decision processes (POMDPs), which are used when

observations of system states may be erroneous or where system states cannot be directly monitored at all (in which case we monitor associated phenomena).

According to [110, 111], a standard stochastic MDP model is defined by a tuple

$$(X, A, A(.), f, R, \alpha) ,$$

where X is the system state space, and A is the control actions space. For each $x \in X$, there is a finite set of feasible control actions $A(x)$. Given an initial state x_0, the state transition law is given by a function f that governs the system transition from state $x_t = i$ to state $x_{t+1} = j$ such that $x_{t+1} = f(x_t, a_t, z_{t+1})$, where $x_t \in X$, $a_t \in A(x_t)$ and z_{t+1} belongs to some random distribution Z. This random process Z captures the stochastic dynamics of the system such that z_{t+1} represents the system's random behavior in the time period from t to $t + 1$.

In a discrete context (i.e., discrete state space, discrete action space), the system can be described with the 5-tuple $(S, A, P.(\cdot, \cdot), R.(\cdot, \cdot), \gamma)$ where:

- S is the set of finite system states,
- A is the set of possible actions, the expression $A(s)$, if used, refers to the subset of actions available at the state (s) if different from the global A,
- P is the transition probability matrix, $S \times A \times S \to [0, 1]$, which defines a transition distribution over the next state in terms of the current state and action,
- R is the reward $S \times A \to R$, and

- $\gamma \in [0, 1)$ is the discount factor, which represents the difference in importance between future rewards and present rewards.

A solution to an MDP is an optimal decision policy $\pi^* : S \to A$ that maximizes an objective function. Several objective functions can be used such as expected discounted reward, average reward per stage, and expected total reward. A typical objective function of an MDP model is the expected reward-to-go function $R(s_k, a_k)$, which is usually discounted with a *factor* γ.

As MDPs can be divided into continuous and discrete classes, they can be also divided into finite and infinite horizon. In this chapter, we are interested in discrete time, infinite horizon ($H < \infty$), discrete state space, and discrete action space MDPs. In such problems, the reward-to-go value for system state $s \in S$ at stage k, when executing a policy $\pi(s_k)$ is given by:

$$V^{\pi}(s_k) = \sum_{s_{k+1} \in S} P_{\pi(s_k)}(s_k, s_{k+1}) \left(R_{\pi(s_k)}(s_k, s_{k+1}) + \gamma V^{\pi}(s_{k+1}) \right), \tag{5.1}$$

where $P_{\pi(s_k)}(s_k, s_{k+1})$ is the probability to move from state s_k to state s_{k+1} after applying action a that is generated by the policy π such that $a = \pi(s_k)$.

The optimal reward-to-go value function when executing the optimal policy $\pi^*(s)$ particularly is given by:

$$V^*(s_k) = \max_{a \in A(s)} \left\{ \sum_{s_{k+1} \in S} P_a(s, s_{k+1}) \left(R_a(s, s_{k+1}) + \gamma V^*(s_{k+1}) \right) \right\}. \tag{5.2}$$

Similarly, the optimal policy $\pi^*(s)$ is found by solving

$$\pi^*(s_k) = \arg\max_{a \in A(s)} \left\{ \sum_{s_{k+1} \in S} P_a(s, s_{k+1}) \left(R_a(s, s_{k+1}) + \gamma V^*(s_{k+1}) \right) \right\}. \tag{5.3}$$

For a model-based solution where $P(s_k, s_{k+1})$ is given or can be initialized then updated at run time (aka, online model-based prediction), a solution can be acquired through dynamic programming algorithms or approximate dynamic programming techniques (e.g., linear programming). The former are of two classes: policy iteration algorithms or a value iteration algorithms. Both classes of algorithms basically repeat two steps known as value-maximization and value-update until the changes to the value function $V(s)$ is below a convergence threshold.

According to [110] and [111] it is less computationally demanding to use policy-iteration methods when the action space has lower dimension than the state space (as in the HTTP-based AVS problem). Therefore, we use a policy-iteration algorithm to solve the model proposed in the next section.

When a model does not exist, an approach called reinforcement learning (RL) is used instead. In such scenario, since the transition probability distribution is not available, a learning algorithm runs to learn how to make optimal decisions. The tutorial in [112] is a great reference for full details. RL relies on repeatedly updating the state-value function $V(s)$ according to an iterative temporal difference learning rule

where:

$$newValue = oldValue + \alpha(targetValue - oldValue), \tag{5.4}$$

where $\alpha \in [0, 1]$ is the learning rate.

There are several algorithms that apply the RL concept such as Q-learning, SARSA, and Dyna-Q. SARSA stands for State-Action-Reward-State-Action. The authors in [79] proposed a Q-learning agent to solve the HAVS problem, and we will investigate in (Section 5.4) the use of SARSA to address a similar problem. . Below we provide a summary of the two algorithms, their similarities, and their differences. Generally, the algorithms differ in handling the tradeoff between exploration and exploitation.

The Q-learning algorithm applies Bellman's theorem to the value function $V(s)$. It relies on solving Bellman's equation for Q:

$$Q^*(s_k, a_k) = R(s_k, a_k) + \gamma \mathbb{E}\left[\max_a Q^*(s_{k+1}, a)\right]. \tag{5.5}$$

Hence, we iteratively update the Q value (aka, the action-value function) as:

$$Q(s_k, a_k) \leftarrow Q(s_k, a_k) + \alpha_i \left[r_k + \gamma \max_a Q(s_{k+1}, a) - Q(s_k, a_k)\right]. \tag{5.6}$$

Q-learning is an off-policy algorithm which means it learns by following a path that does not necessarily resemble the policy the algorithm is executing. Due to this off-policy learning, Q-learning is expected to have higher exploration rate than on-policy learning

algorithms. Algorithm 4 provides the Q-learning algorithm steps in detail, T is the set of terminal states, if found.

Algorithm 4 The Q-learning algorithm.

repeat
  Initialize variables, s, a
  while $s \notin T$ **do**
    execute action a
    observe reward r and next state s_{k+1}
    update the action-value function,
    $Q(s_k, a_k) \leftarrow Q(s_k, a_k) + \alpha_i \left[r + \gamma \max_{a_{k+1}} Q(s_{k+1}, a_{k+1}) - Q(s_k, a_k)\right]$
    choose action a_{k+1}
    $s = s_{k+1}, a = a_{k+1}$
  end while
  decay the learning rate α.
until convergence condition is satisfied

On the other hand, with the SARSA algorithm (steps provided in Algorithm 5) updates do not occur only on the state-action-state trajectory, but rather an extended to the state-action-reward-state-action trajectory $(s_k, a_k, r_k, s_{k+1}, a_{k+1})$. When SARSA is given a state-action pair (s_k, a_k), it simulates the action over the state, acquires the reward r_k then updates the system state to s_{k+1}, before it uses the current optimal policy to generate the next action a_{k+1}. It is worth mentioning that the current optimal policy is calculated based on the current Q values, which are updated as:

$$Q(s_k, a_k) \leftarrow Q(s_k, a_k) + \alpha_i \left[r_k + \gamma Q(s_{k+1}, a_{k+1}) - Q(s_k, a_k)\right]. \tag{5.7}$$

Algorithm 5 The SARSA reinforcement learning algorithm.

repeat
  Initialize variables, s, a
  while $s \notin T$ **do**
    execute action a
    observe reward r and next state s_{k+1}
    choose random action a_{k+1} with probability ϵ
    or an action $a_{k+1} = \pi(s_{k+1})$ with probability $1 - \epsilon$
    update the action-value function,
    $Q(s_k, a_k) \leftarrow Q(s_k, a_k) + \alpha_i \left[r + \gamma Q(s_{k+1}, a_{k+1}) - Q(s_k, a_k) \right]$
    $\pi(s) \in \arg\max_{a_{k+2}} Q(s, a_{k+2})$
    $s = s_{k+1}, a = a_{k+1}$
  end while
  decay the learning rate α.
until convergence condition is satisfied

5.2 System Model using Markov Decision Process (MDP)

Using the simulation records for the channel simulations from the previous chapter, we build a finite state Markov Chain (FSMC) for the wireless channel as shown in Figure 5.1, in a similar approach to the widely used discrete model in [113]. The model is formed based on the range of the SINR values from the Nist simulation model and integrated with the Minstrel multi-rate retry behavior so that we end up with a Minstrel-SINR interaction model whose output is the frame success rate for any combinational state of the Minstrel available rates and the SINR values. The transition matrix we have estimated matches the Minstrel FSMC-based behavior described by the authors in [114].

In order to run a full TCP throughput FSMC model over the wireless link whose model built above, we integrate the TCP stack model proposed in [115, 116] after replacing the Markov chain of the automatic rate fallback (ARF) adaptation algorithm

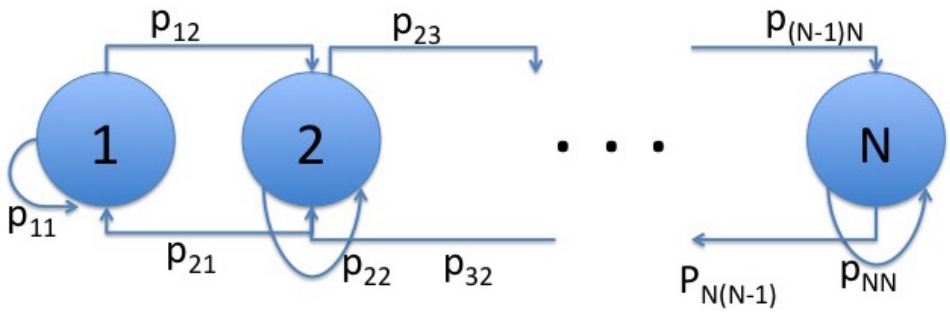


Figure 5.1: Finite state Markov chain model for channel SINR with N discrete states.

with the Minstrel Markov chain. This results in a full TCP-DCF-Minstrel interaction model. The channel stability here is captured by the probability $p_{ii} = 1 - p$, which is the probability of the SINR model to stay at the same state value at the next slot or to switch to another state with a probability p. As p increases as the channel becomes less stable.

Given the MDP framework introduced earlier, we can formulate the adaptive video streaming (AVS) problem as follows. We assume an on-demand streaming video clip of length L seconds, and a client controller that implements a sequential decision process in stages with length l_{seg} seconds each, equal to the video chunks. At the beginning of each stage $k \in \{0, 1, ..., K\}$, where $K = L/l_{seg}$, the client application decides whether to continue operating at the same quality level $q_k = q_{k-1}$ or switch to different quality $q_k \in A$. The actions available belongs to a discrete space given by $A = \{1, ..., M\}$, where M is the highest quality level. We use the same seven quality levels and bandwidth requirements as in the previous chapter.

While we agree with Turck et al. [79] on most of the components that contribute to the user QoE, we disagree with the equal, linear weighting given to each of these

components in their design of the system states and reward function. Turck et al. built their design on the following principles:

- Buffer filling is the most important criteria, so it is the only criterion that generates positive rewards.
- Buffer underrun is the worst state, so it results in a large negative reward.
- Any deviation from the maximum quality is a loss and all deviations are treated on a linear scale, assuming that the difference in the value between any two quality levels is a unit step, which does not align with research findings about the human visual perception and how it contributes to the video streaming quality of experience (QoE) [87]. Research findings suggest much more complicated relationships—the *estimated MOS* quantitative model is the most reliable available model.
- Finally, the reward is given by a linear combination of three equally-weighted components, which are the quality level, the switching behavior, and the buffering-filling.

Motivated by the lessons learned from the heuristics in Chapter 4 —especially the interactions related to cross-layer dynamics—and the potential to enhance the previous work by Turck et al., we modify the design they followed for their system states and its reward function by taking into account the following design considerations:

- the components of quality, switching, and buffer-filling do not contribute equally to

the reward function and the relative weighting of these components should not be static over the entire course of the video streaming session;

- rebuffering must be penalized but not so much that it causes the controller to be biased toward filling the buffer as fast as possible using low quality content;
- a sudden drop in quality level from the top levels to the lowest level is the worst kind of switching with respect to the visual perceptual component of the QoE, hence it should be penalized significantly with respect to the rest of possible switching penalties;
- while stability is desired and fluctuating decision is not, we aim to maximize the utilization of the available bandwidth. Hence, we only punish for consecutive switching decisions that result from over estimating available bandwidth if the target buffer growth was not achieved; and
- expanding the system states to include cross-layer information, specifically the estimated TCP congestion window and the Minstrel rate.

In terms of the system state, we present a three dimensional, cross-layer-based system state. These components are the buffer growth, the expected TCP window size, and the Minstrel rate reported by MRCC. We assume an implementation of such model will have cross-layer access to the calculate and read the last two components. Table 5.1 provides a summary of system states. Notice that the B_{max} is actually the channel capacity or the Minstrel rate used in the same time slot. But for simplicity of calculations,

Table 5.1: System state variable.

State component	Range	Levels
buffer growth	$[0 - B_{max}]$ sec	$\frac{B_{max}}{l_{seg}} + 1$
TCP expected window	$[0 - CW_{max}]$ bps	$M + 1$
Minstrel rate	$6, 9, 12, 18, 24, 36, 48, 56$	8

we discretized the range of rates up to the maximum (54 Mbps). The solving algorithm is then modified to check only the applicable subset at runtime. This will reduce some complexity, which is needed because the proposed cross-layer-based state space is larger than the state space used in [79].

In terms of the reward function, we adopt the view in [109] that the three main components of QoE are buffer underrun events, quality switching behavior, and playback quality level. However, our reward model differs from the reward in [79] because we do not use equal, static weights. Therefore we combine the three components using an adaptive weighted model as follows:

$$R_k = \beta R_k^{quality} + \zeta R_k^{switching} + \phi R_k^{buffering}, \tag{5.8}$$

while ϕ is an adaptive weight that is given by

$$\phi = 1 - \frac{B_k}{B_{max}}, \tag{5.9}$$

and

$$\beta + \zeta + \phi = 1. \tag{5.10}$$

Concurrently, each of the three reward components are given as follows:

$$R_k^{quality} = \frac{q_k - M}{M}, \tag{5.11}$$

$$R_k^{buffering} = \begin{cases} \frac{B_k - B_{k-1}}{B_{max}} & \text{if } B_k = 0 \\ -1 & \text{if } 0 < B_k < B_{alarm}, \end{cases} \tag{5.12}$$

and

$$R_k^{switching} = \begin{cases} \frac{q_k - q_{k-1}}{M} & \text{if } q_k - q_{k-1} < 1 - M \\ \frac{-2(M-1)}{M} & \text{if } q_k - q_{k-1} > 1 - M \\ \frac{-2(q_k - q_{k-1})}{M} & \text{if } q_k - q_{k-1} < 0, q_{k-1} - q_{k-2} > 0. \end{cases} \tag{5.13}$$

5.3 Offline Solution

A Monte Carlo simulation was developed using the aforementioned system model and run over the cross-layer TCP-DCF-Minstrel model to generate the transition probability matrix for the MDP model. The resulting model was then solved by an enhanced policy

iteration algorithm which updates state-value function several times per epoch. A portion of the implementation relies heavily on the MDP toolbox [117].

Policy iteration algorithms consist of two stages. The first stage is called policy evaluation, where a policy is picked and applied to the system. The second stage is called policy improvement, in which the optimal policy is selected after the value functions have been updated. The algorithm keeps iterating until the maximum number of iterations is reached or until the residual value falls under the predefined convergence threshold.

We present evaluation results for the offline solution in terms of the achieved QoE under a range of channel conditions (expressed in terms of the channel state transition probability p) and in terms of the efficiency ratio (with respect to an ideal controller that already knows the entire future sequence of system states).

We start by investigating the performance under different value of the transition probability. We setup the model to assume the buffer already has 10 seconds of video content before the policy iteration algorithm starts.

Figure 5.2 shows the average estimated-MOS achieved after the policy-iteration algorithm reached convergence, under various combinations of transition probability and discount factor. We can observe that as the channel gets more unstable,placing less value on the future achieves higher average MOS. This is due to the fact that weighting future rewards more heavily is not appropriate when the channel changes frequently with $p = 0.5$. A conservative discount factor of 0.5 provides better performance under

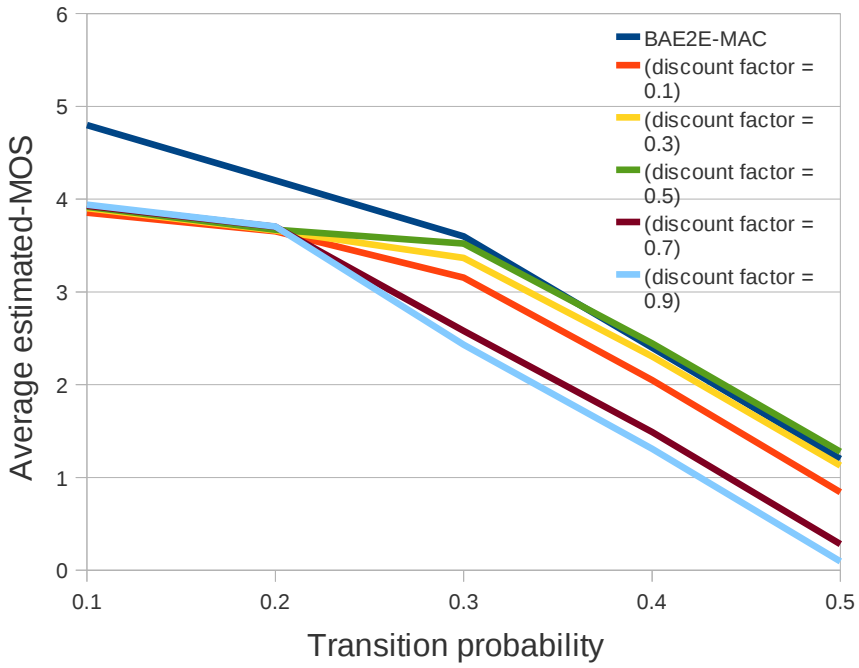


Figure 5.2: Average estimated-MOS under different channel transition probability p.

such high transitional probability as it balances between the immediate reward and future gain.

Figure 5.3 illustrates the performance of the policy iteration algorithm relative to an ideal controller which has access to the future under different values for the transition probability. As the channel changes becomes more frequently, the achieved performance ratio clearly decreases in a non-linear relationship. As the residual values that are given by the x-axis decrease as the learning efficiency increases in general.

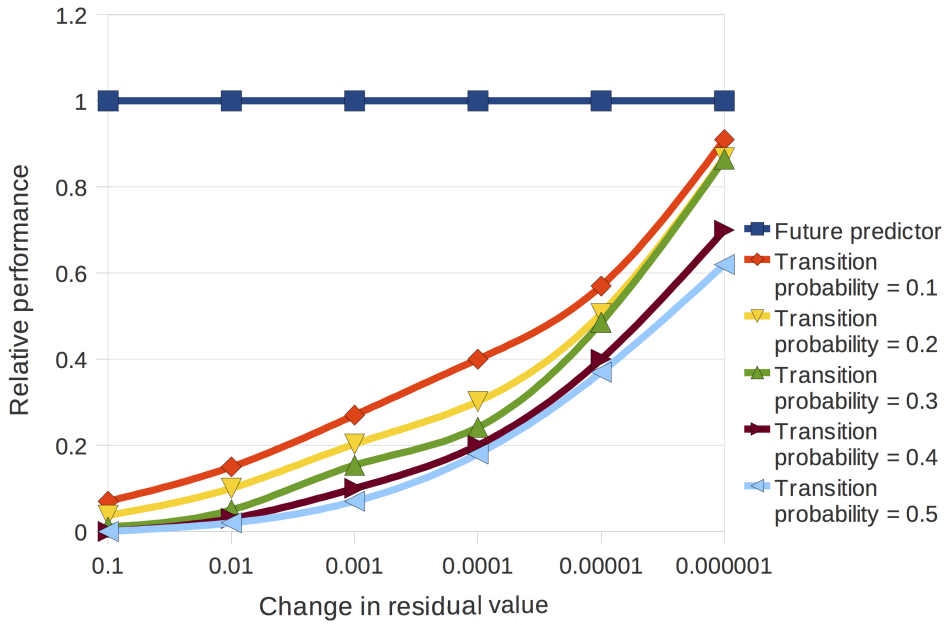


Figure 5.3: The performance climbing rate of the policy iteration algorithm.

5.4 Online Solution using Reinforcement Learning (RL)

In this section, we present our reinforcement learning solution for the aforementioned system model. We implement a quality controller using the on-policy SARSA algorithm, which was explained earlier in this chapter. We compare the SARSA controller with the Q-learning controller of Turck et al. using the same ns-3 simulation setup that we presented in Chapter 4.

The two RL-based controllers are developed in C++ classes and integrated into the video client ns-3 module. The first controller uses the Q-learning algorithm and the system model developed by Turck et al. [79], whereas second controller uses the SARSA algorithm and the system model we proposed in section 5.2.

We begin by comparing the two agents with the optimal performance achieved by an ideal controller in terms of performance ratio. The ideal controller is made aware of the simulated events before they occur, hence it can plan the entire future path. Figure 5.4 shows the performance ratio averaged over simulation runs that cover three channel conditions (unstable, moderate, and stable). It is clear that Q-learning performs on average better than SARSA until a point where Q-learning climbing rate starts to slow down and SARSA outperforms it. This aligns with expectations as the more exploring-oriented controller (Q-learning) tends to give better results when the training period is still below a certain level, until the behavior of the Q-learning gets dominated by its well-known disadvantage, named *jumping of the cliff* (a.k.a. aggressive exploration). Q-learning as an off-policy RL algorithm is more aggressive in exploring, which explains the need of Turck et al. to have extremely large punishment values for buffer underrun to prevent the algorithm from taking aggressive actions that lead to buffer underun events.

The greater foresight of the SARSA algorithm and our more balanced reward function improve the performance in a stable way, which is more resilient under all channel conditions. At a point of the training curve that is far beyond realistic scenarios, the two controllers meet again.

Next, we compare the two methods in terms of the average QoE achieved by each of them. The ns-3 simulation setup gave us the flexibility of testing the performance under multiple well-known TCP flavors, where each one had its own congestion

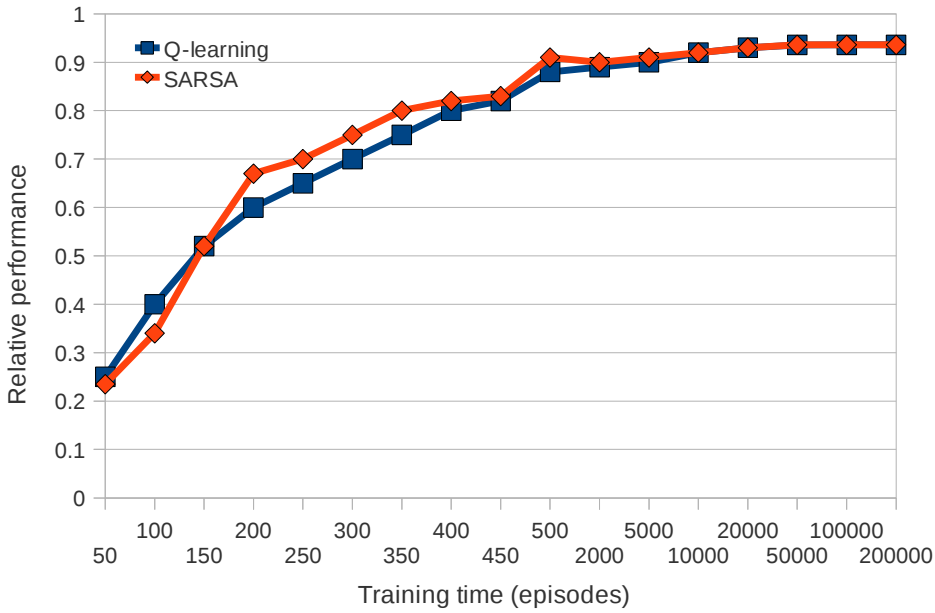
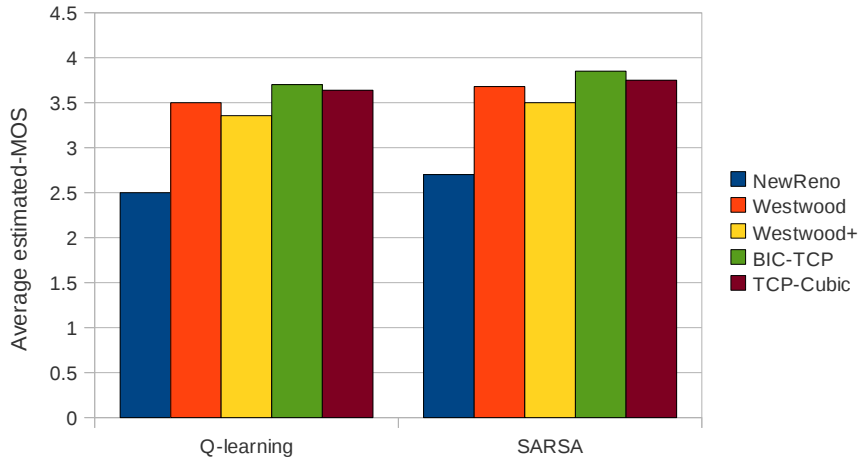


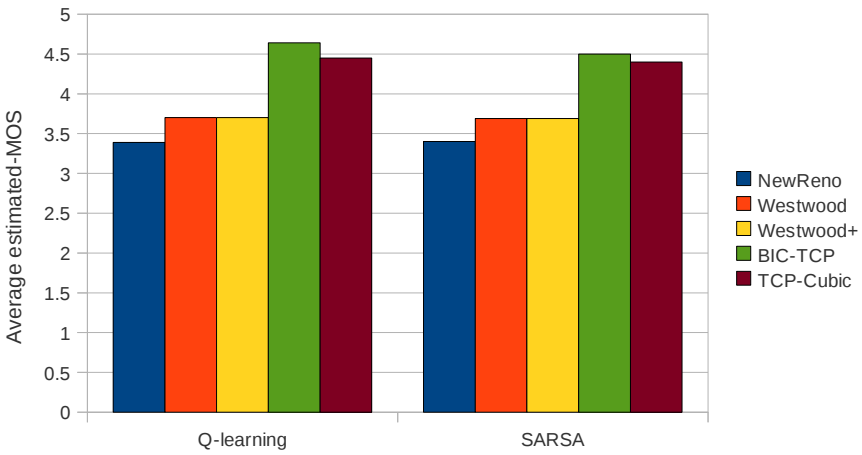
Figure 5.4: The influence of the training period on the relative performance of the RL-based controllers.

avoidance. The analysis of section 5.3 only dealt with TCP-Reno congestion avoidance due to the difficulty associated with developing an analytical models for each different version of TCP. Here we report results of using ns-3 simulation models of various TCP flavors including TCP-NewReno, TCP-Westwood [118] , TCP-Westwood+, BIC-TCP, and TCP-Cubic [119] under three different channel conditions (unstable, moderate, stable) as shown in Figures 5.5a, 5.5b, and 5.5c.

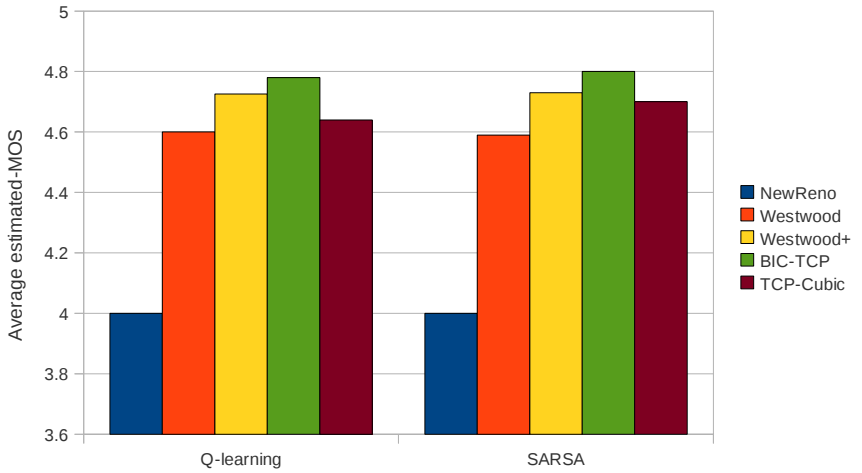
The most important observation in terms of TCP flavors is how BIC-TCP and TCP-Westwood provide the highest average performance. By looking at snapshots from the TCP flows, where each flow represents downloading a single video chunk amid the adaptive video streaming session, we can better interpret these results. Figures 5.6 and 5.7 provide examples of downloading video chunks under stable and unstable channels,



(a) Unstable channel



(b) Moderate channel



(c) Stable channel

Figure 5.5: Performance of Q-learning and SARSA under different channel conditions using various TCP flavors.

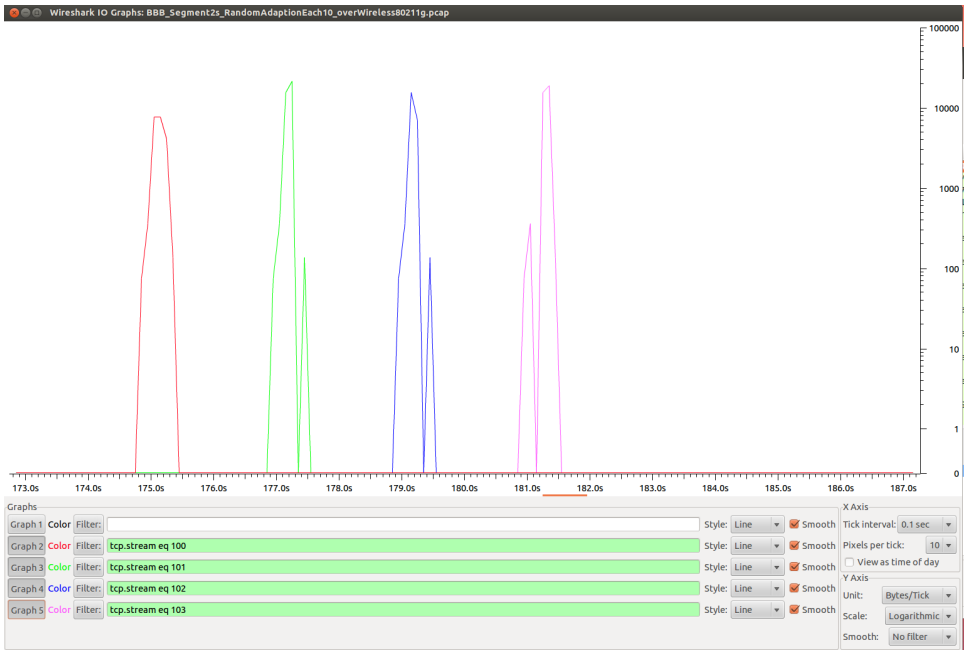


Figure 5.6: Instantaneous throughput of TCP connections while downloading video chunks (stable channel).

respectively. The curves show the progress and collapse of the TCP flows' throughput as captured using Wireshark [120] from a real world video streaming session that was run over the Internet through an IEEE 802.11g connection, using the DASH standard QtPlayer and the LibDash API [121].

Every time the throughput collapses, it is an advantage if the TCP congestion window can grow quickly to reach a download rate corresponding to the rates currently achievable on the wireless link by Minstel. It is preferred to download atarget video chunk completely before starting a newer chunk. Establishing a new TCP connection and start the download of a newer chunk while the earlier chunk is not yet done is not preferred by most video applications and browsers. If a deadline to send the HTTP GET request is reached, though, the controller may proceed to send the request for the next

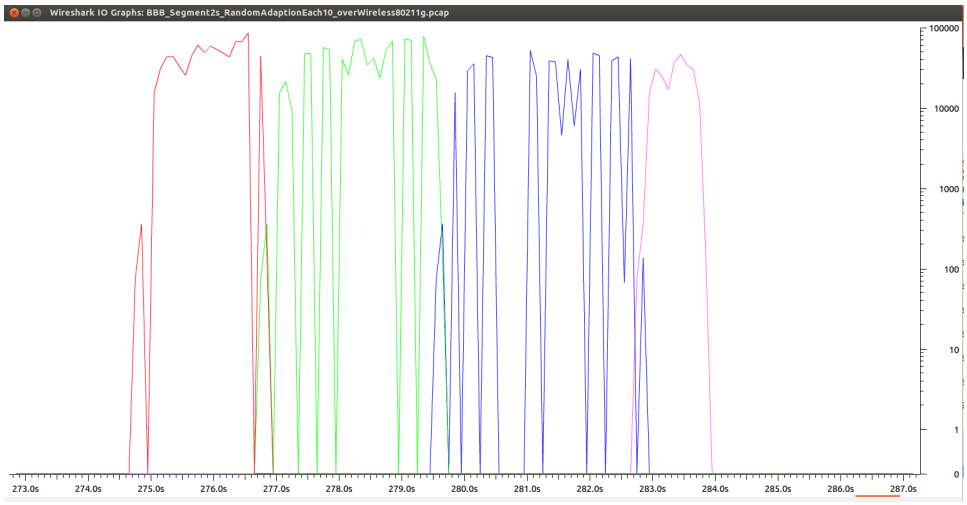


Figure 5.7: Instantaneous throughput of TCP connections while downloading video chunks (unstable channel).

chunk. This results in an overlapping time window where the two connections may compete for link bandwidth as seen in Figure 5.7, where the chunks overlapped because the download time of the earlier chunk was prolonged. Although the two flows being downloaded concurrently will not create contention on the link, downloading chunks concurrently may still reduce the QoE.

TCP-Westwood and BIC-TCP have the most aggressive congestion window growth functions, allowing to utilize the available wireless bandwidth most efficiently. Westwood has a bandwidth estimation algorithm that works well with wireless link, where the congestion window grows based on the arrival rate of the ACKs, not just on ACK-counting or timing out. Westwood+ decreases the aggressiveness rate of Westwood by dealing with the issue of ACKs compression when ACKs arrives very close to each other in time. However, it is clear that the adaptive video streaming

performs better under the original Westwood, as well as the original BIC-TCP before it was modified to become TCP-Cubic.

In terms of comparison between Q-learning and SARSA, under stable conditions, SARSA controller performs slightly better than Q-learning under Westwood+, BIC-TCP, and TCP-Cubic, whereas they perform almost equally for the rest of TCP flavors. It is clear that this changes under unstable conditions, with SARSA consistently outperforming Q-learning. This is most likely due to excessive exploration by Q-learning using its off-policy mechanism. Unfortunately, the channel conditions is too bad to explore, hence the cost of the *jumping the cliff* property is very high. On moderate channel, their performance is almost equal except a slight difference for the Q-learning performance over BIC-TCP.

In addition to aforementioned evaluation matters, we gathered more data to investigate the benefits of using reinforcement learning instead of relying on a heuristic approach. Our goal is to determine how much gain is made by using reinforcement learning and when it makes sense to use.

The performance of the two heuristic algorithms (QoE-RAHAS and BAE2E-MAC) and the two RL-based controllers (Q-learning and SARSA) are evaluated with respect to an imaginary controller that is aware of the entire sequence of simulation events before they occur. The same setup was run using all aforementioned TCP flavors, but we report here the results under TCP-Cubic only as other flavors were similar. Figure 5.8 shows the relative performance as a function of the learning time, which is represented by the number of training episodes the RL-based controllers required. The performance values

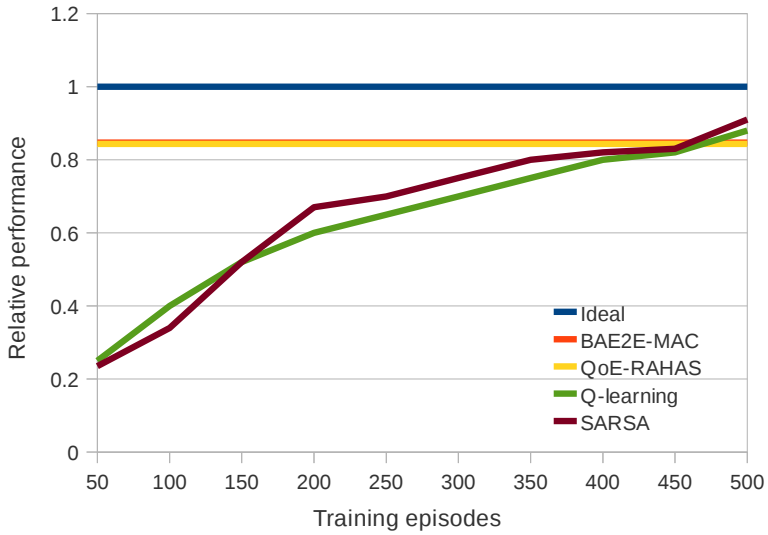


Figure 5.8: Average relative performance over all channel conditions.

are averaged over three channel conditions and with equal number of training episodes spent on each channel condition scenario.

The curve illustrates how the Q-learning achieves faster development earlier due to its tendency for wilder exploration until it reaches an intersection point with the SARSA. The progress rate of the latter becomes clearly higher after the intersection point until it gets closer to the optimal. Both RL agents exceed the performance of heuristic controllers after enough training. The question is how much training we can afford. According to this large number of required training episodes, it is obvious that the controllers need to be trained under simulation or emulation testbeds before they are integrated into actual commercial solutions.

Looking at the curve in Figure 5.8, the most trained SARSA controller achieved less than 10% improvement in the average performance when comparted to the highest

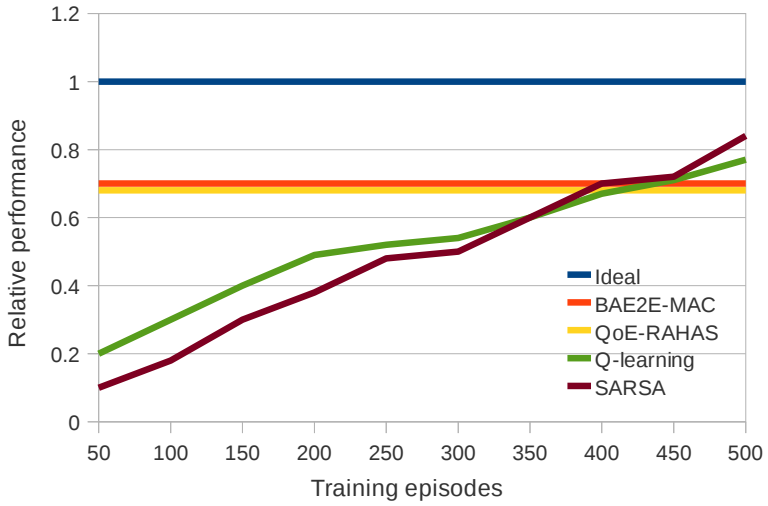


Figure 5.9: Average relative performance over [unstable channel] conditions.

heuristic performance. This changes dramatically when looking at the curves in Figure 5.9, which illustrates similar comparison but for only unstable channel conditions scenarios. The RL-based controllers made much greater difference for unstable channels. The SARSA provides almost 20% performance improvement and the Q-learning achieved 10% better performance than the best heuristic. This suggests that under unstable channel conditions, the RL-based controllers makes a significant difference that is worth the expected runtime computational requirements and the offline training cost.

A futuristic video controller may combine heuristic and learning controllers by alternating between them according to the channel conditions, However, with the extraordinary growth in device computational capabilities, we do not think that the computational cost of the RL algorithm is that significant even for mobile devices, especially as offline training before installation would be the best approach to adopt.

5.5 System Prototype using the FINS Framework

In this section we present our effort to deploy a system prototype of the two RL-based controllers in a real-world experimental setup. We use the DASH standard testing servers provided by the ITEC group [122]. The laptop that runs the video client application connects to the ITEC benchmarking server (143.205.176.132) from the Virginia Tech campus network through a Netgear N300 IEEE 802.11b/g/n wireless router access point. The latter configured to run as IEEE802.11g AP for the entire time. We create background download traffic using another laptop with the same specifications. The background traffic is a large file (multiple Gigabytes) downloading over an FTP session.

The client video application runs on top of the FINS Framework stack on a Dell laptop (Inspiron 1525), which is equipped with Intel Core2 Duo processor, 3MB RAM, and a Broadcom IEEE802.11g wireless network card. The block diagram of the software stack is illustrated in Figure 5.10.

Using the FINS Framework provides us with an access to poll the TCP protocol status through the real time manager (RTM) module and to use the Broadcom device driver (which was modified by us) to interact with the mac80211 kernel module to poll the Minstrel status.

As both the RL-based controllers are already implemented in C++ for the ns-3 simulator, we have successfully ported them to run from within the QtPlayer by making

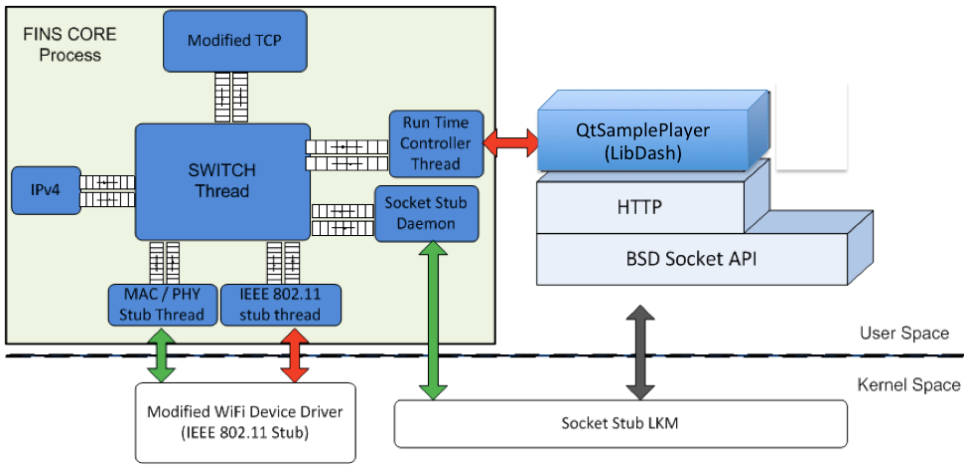


Figure 5.10: The node stack using DASH standard and the FINS Framework.

calls to the DASH API provided by the LibDash library. The controllers implement the DASH standard interface for adaptation logic.

In order to boost the performance of the controller and avoid long runs of the experimental setup before gathering results, we have initialized the controller with a copy of the agent memory from the simulation environment after the latter was trained for 200 episodes. The 200 episodes point is a midpoint on the performance curve, this avoids picking a controller that is already over-trained to match the simulation environment.

Then the experimental setup has been run for about 200 more episodes so that we compare the performance. We run the 200 episodes using a 10 minute clip of "Big Buck Bunny" with 480p resolution. The ITEC benchmarking server provides a multi-presentation description (MPD) file for this clip that supports 13 different quality levels whose bandwidth requirements vary from 101 Kbps to 4.49 Mbps. The video

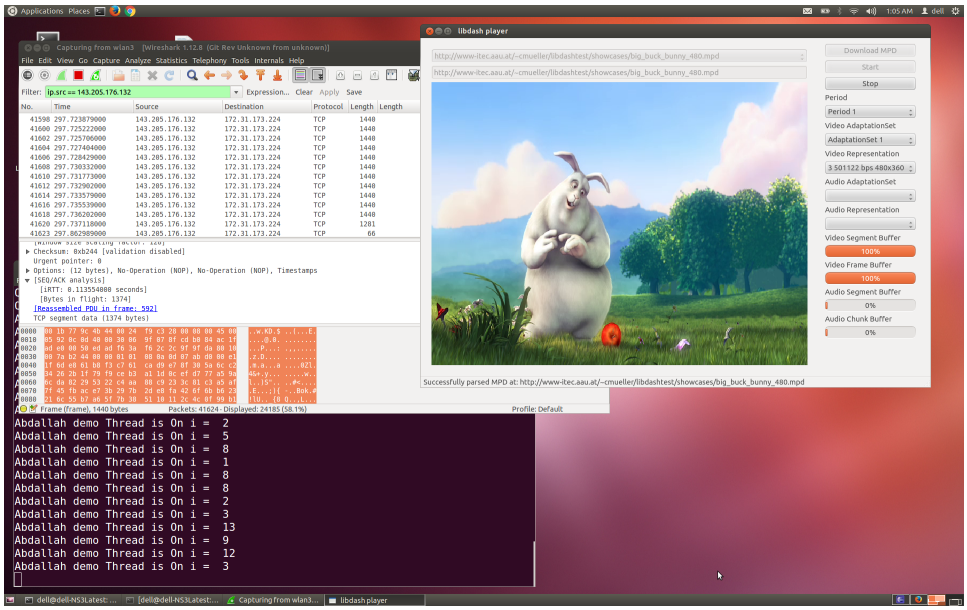


Figure 5.11: Streaming Big Buck Bunny using SARSA-based controller over the FINS Framework.

content is divided into 2s segments. A snapshot of the system while running can be seen in Figure 5.11.

The video window is shown in the upper right and the quality level streamed appears in the terminal window on the left lower part. For the purposes of traffic analysis, Wireshark is used to capture packets for later analysis of TCP behavior. The TCP module in the FINS Framework implements the Reno flavor.

Figure 5.12 shows the average performance of the RL-based controllers over the FINS Framework versus the average performance of TCP-Reno runs under the simulation environment. A drop in the performance is noticed but is expected due to differences between the simulation wireless link model (Nist) and the real world wireless behavior. However, we notice that SARSA still does better on average.

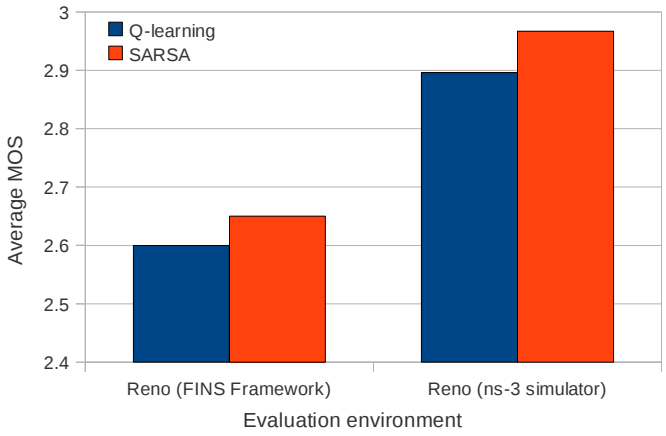


Figure 5.12: Average MOS achieved by Q-learning and SARSA controllers over the FINS Framework vs the ns-3 simulator [TCP Reno].

There are three factors that may contribute to the drop in performance between the simulation and the experiment: The latency-related implementation issues with the FINS Framework's TCP module, the fact that the RL-based controllers might have needed longer training under the experimental setup, and the overhead of the FINS Framework internal communication. We think the training period is the most sensitive factor.

This simple prototype shows that the use of RL-based controllers is practical for HTTP-based adaptive video streaming sessions. It also illustrates the flexibility provided by the FINS Framework, which has lowered the barrier for this experiment. It allowed us to reuse the modules described in Chapter 3 to implement the aforementioned experimental setup and run desired experiments with much less effort that would be required without the FINS Framework.

5.6 Conclusion & Future work

In this chapter, we presented our new system model to formulate the HTTP-based AVS problem as a Markov decision process (MDP). We presented an offline solution using a dynamic programming policy iteration algorithm under a Matlab-based Monte-Carlo like environment. Then, we presented an online solution using reinforcement learning (RL), specifically using the SARSA online-policy algorithm. The proposed solution was compared to related work that used the Q-learning algorithm. The SARSA controllers showed better performance on average with a performance improvement that reached 20% compared to the highest performance achieved by heuristic algorithms in some scenarios, which is double the improvement achieved by Q-learning.

In addition, we presented a prototype, using the FINS Framework, to deploy the RL-based controllers in an actual real-world AVS setup, which involved using the latest commercial standard tools. The setup showed the potential of integrating RL-based controllers into video players and the flexibility provided by the FINS Framework for research purposes.

The work in this chapter opens the road to some future directions. Here are some highlights on these lessons and future directions:

The design of the reward function makes a great difference when dealing with a problem with complex dynamics. Adaptive video streaming has such dynamics due to significant cross-layer interactions (wireless physical layer, MAC layer DCF, routing

TCP, and the quality controller's actions). Our design for the system states and the reward function, in addition to the choice of the SARSA algorithm, made a noticeable performance difference, especially under unstable conditions. The Q-learning algorithm used by Turck et al. did not perform as well due to the tendency to follow aggressive exploration, which causes more and longer buffer underrun events. Attempting to address this issue with the Q-learning algorithm by severely punishing buffer under runs in the reward function results in a bias toward filling the buffer, whatever the quality.

Our results and previous results from [79] confirm that the main challenge for the RL-based controller is the required learning time before it performs on a level comparable to existing solutions. Although we have shown how to significantly reduce the learning time by copying the agent status from a learning environment to a deployment environment, we believe more can be done by combining modeling and learning, which is known as model-based learning. We believe more investigation of this approach using the state of the art algorithms, published in the field of optimistic planning [123], can greatly shorten the learning time and improve the performance. This is another interesting future direction to investigate.

Chapter 6

Conclusions

This dissertation addressed two related research problems in wireless networks. The first problem is the lack of experimental software tools to support implementation-based studies in networking. The second problem is the performance of the quality selection process in the context of HTTP-based adaptive video streaming in a wireless environment and its influence on the user quality of experience (QoE). The two problems are strongly related since most prior solutions for video streaming suffer from a lack of thorough evaluation under realistic networking conditions. We addressed the former problem through the creation of a new experimental networking tool, the FINS Framework. After proposing new HAVS controllers to address the later problem, we evaluated those controllers using both simulation and implementation using the FINS Framework. Using both types of evaluation provided higher fidelity for our results, and

included not only implementing our own solutions, but also implementing and testing prior solutions published by others, for comparison purposes.

This chapter summarizes our contributions and highlights some important future directions to pursue.

6.1 Summary and Contributions

The contributions presented in this dissertation can be briefly summarized as follows:

We designed and implemented the Flexible Internetwork Stack (FINS) Framework, a new experimental software tool that lowers barriers for implementation-based studies in networking. The tool fulfills the needs of the research community in terms of reconfigurability, flexibility, reusability, and low implementation cost, and it supports open source platforms including Linux and Android. The FINS Framework is a useful tool for those interested in implementation-based evaluation in place of or to supplement simulation and emulation techniques. The tool keeps the development cycle within user-space avoiding the high cost and challenges associated with kernel programming. The architecture and implementation methods that we adopted support numerous research use-cases including cross-layer designs, context-aware applications and protocols, clean-slate designs, cognitive networking experiments, cognitive network nodes, and dynamic spectrum access scenarios.

We designed, simulated, and implemented three new client-based quality controllers for HTTP-based adaptive video streaming (HAVS) over WiFi access networks. In the first controller, we designed a cross-layer control algorithm called the E2E-MAC, which combines estimation of the available wireless bandwidth and measurement of end-to-end throughput to choose the best video quality level to request from the server. In the second controller, we designed a cross-layer buffer-aware control algorithm called the BAE2E-MAC, which combines buffer status awareness, estimation of available wireless bandwidth, and measurement of end-to-end throughput to choose the best video quality level to request from the server. The results showed that our proposed controller performed better than the most efficient prior solution which relied on end-to-end measurements only. Under unstable channels, our proposed controller outperformed the prior solution for the entire range of initial buffering periods recording a performance improvement up to 33% in scenarios with much shorter initial buffering.

For the third controller, we used machine learning to design a near-optimal quality controller for HAVS. After formulating the problem as a Markov decision process (MDP), we applied a model-based solution for scenarios in which the system model is known, and then presented a model-free, learning-based solution using the SARSA reinforcement learning algorithm. The proposed controller outperformed a prior solution, which adopted the Q-learning algorithm instead. Our careful choice of learning algorithm and system design (system state and reward function) enabled the proposed controller to achieve a 20% performance improvement with respect the best heuristic

controller, which is double the performance improvement achieved by the Q-learning-based prior solution. The evaluation results were confirmed under two evaluation environments, specifically the ns-3 simulator and an implementation-based experimental setup, where we used the FINS Framework to implement a prototype for the proposed controller and its counterpart from prior published work. The implementation-based setup used a standard adaptive video streaming application and its associated library, which are compliant with the DASH ISO standard. As a result of this experimental evaluation, our proposed solution has already been prototyped for further development and possible commercial deployment.

Although it may seem expected —after all prior work found in the literature —to achieve better performance and QoE by using cross-layer design, this dissertation presented more than just additional performance gain. We not only proposed an evaluation for a cross-layer solution using a simulation environment, but also implemented a full prototype, integrated it into well-known standard software library, and connected it to a standard video server supported by the DASH Industry Forum (DASHIF). This successful prototype shows that it is time to move cross-layer design and machine-learning-based approaches from the research field to actual deployment. It is time to take the concept of cognitive networking from simulation environments to real world deployment as well.

6.2 Future Work

The results from the previous two chapters gave us insights not only into the performance gains achievable through cross-layer designs when combined with machine learning approaches, but also into the feasibility and low cost of deploying such approaches when using the right software tools. Given these insights, here are some short term research topics that should be pursued motivated by the findings of this dissertation:

- The combination of simulation tools and experimental testbeds is a trend in networking research, and we believe that a combination of the FINS Framework and the ns-3 network simulator would create a powerful and useful tool. Introducing cross-layer solutions into the ns-3 simulator could break its current careful software design. By integrating the FINS Framework concept of modules , we could allow cross-layer approaches in ns-3, without breaking the careful modularization of the simulator. Of course, this would also permit a rapid progression from initial simulation, to prototype experimental evaluation, and back to simulated investigation of network scaling.

- The pros and cons of deploying cross-layer solutions also need to be reevaluated. Industry has not been enthusiastic about deploying cross-layer solutions due to perceived compatibility issues that may result from violating the stack model. The open source community, on the other hand, has implemented some cross-layer

solutions that improve efficiency without breaking compatibility. For example, reconfiguring TCP parameters has been possible since Linux kernel version 2.6. We think it is the time to review legacy anti-cross-layer policies. Given that the adaptive video streaming already suffers from a lack of incompatibility at the application layer, adding optional cross-layer support to the DASH standard (which has been developed to unify adaptive video streaming standards) could be an elegant solution to retain backward compatibility and support futuristic scenarios as well.

- Although we primarily addressed HAVS over WiFi access networks, there are other wireless technologies that are widely deployed, specifically cellular services. Although there are significant differences between WiFi and cellular networks, our proposed approach could be adapted to LTE and other cellular standards.

- From QoE point of view, choosing the initial buffering period is critical because of the strong correlation between initial buffering time and viewer tendency to stop watching the video content. A radical cross-layer solution to accelerate the download of the initial video chunks, could dramatically improve user satisfaction.

- Finally, the performance improvement achieved by SARSA encourages us to investigate how other RL algorithms may perform in the video streaming context. Different learning paradigms and algorithms, such as model-based learning and model-based planning-learning methods (e.g., Dyna, Dyna-Q), may enable further

improvements. In addition, variations on the MDP framework can be used to design new system models for the HAVS problem.

Bibliography

[1] M. S. Thompson, A. E. Hilal, A. S. Abdallah, L. A. DaSilva, and A. B. MacKenzie, "The MANIAC Challenge: Exploring MANETs through competition," in *Proceedings of the International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt)*, 2010, pp. 443 –452.

[2] V. Srivastava, A. B. Hilal, M. S. Thompson, J. N. Chattha, A. B. MacKenzie, and L. A. DaSilva, "Characterizing mobile ad hoc networks: The MANIAC challenge experiment," in *Proceedings of the ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*, 2008, pp. 65–72.

[3] CISCO, "CISCO visual networking index: Forecast and methodology, 2011–2016," Tech. Rep., 2011.

[4] ——, "CISCO visual networking index: Forecast and methodology, 2014–2019," Tech. Rep., 2014.

[5] A. Zambelli, "IIS smooth streaming technical overview," Microsoft Coorporation, Tech. Rep., 2009.

[6] Apple, "HTTP live streaming (HLS) draft to IETF," Tech. Rep., October 2012.

[7] "Real-time messaging protocol (RTMP) specification," Technical Report, April 2009.

[8] A. S. Abdallah, A. B. MacKenzie, L. A. DaSilva, and M. S. Thompson, "On software tools and stack architectures for wireless network experiments," in *IEEE Wireless Communications and Networking Conference (WCNC)*, March 2011.

[9] "NS2 Simulator." [Online]. Available: http://www.isi.edu/nsnam/ns/

[10] "The ns-3 Simulator," July 2013. [Online]. Available: http://www.nsnam.org/documentation/

[11] "OPNet Simulator." [Online]. Available: http://www.opnet.com/solutions/network_rd/modeler.html

[12] S. Kurkowski, T. Camp, and M. Colagrosso, "MANET simulation studies: The incredibles," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 9, pp. 50–61, 2005.

[13] W. Kiess and M. Mauve, "A survey on real-world implementations of mobile ad-hoc networks," *Ad Hoc Networks*, vol. 5, no. 3, pp. 324–339, 2007.

[14] P. De, A. Raniwala, S. Sharma, and T. Chiueh, "Design considerations for a multihop wireless network testbed," *IEEE Communications Magazine*, vol. 43, no. 10, pp. 102–109, Oct. 2005.

[15] V. Lenders, J. Wagner, S. Heimlicher, M. May, and B. Plattner, "An empirical study of the impact of mobility on link failures in an 802.11 ad hoc network," *IEEE Wireless Communications*, vol. 15, no. 6, pp. 16–21, December 2008.

[16] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "GENI: A federated testbed for innovative network experiments," *Computer Networks*, vol. 61, pp. 5 – 23, 2014.

[17] G. Bouabene, C. Jelger, and C. Tschudin, "Virtual network stacks," in *PRESTO Sigcomm Workshop*, 2008.

[18] H. Hassan, R. Eltarras, and M. Eltoweissy, "Towards a framework for evolvable network design," in *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*. SpringerLink, 2009, pp. 390–401.

[19] V. Gazis, E. Patouni, N. Alonistioti, and L. Merakos, "A survey of dynamically adaptable protocol stacks," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 1, pp. 3 –23, 2010.

[20] R. W. Thomas, D. H. Friend, L. A. Dasilva, and A. B. Mackenzie, "Cognitive networks: Adaptation and learning to achieve end-to-end performance objectives," *IEEE Communications Magazine*, vol. 44, no. 12, pp. 51 –57, dec. 2006.

[21] N. C. Hutchinson and L. L. Peterson, "The X-Kernel: An architecture for implementing network protocols," *IEEE Transactions on Software Engineering*, vol. 17, pp. 64–76, 1991.

[22] P. Sutton, L. E. Doyle, and K. E. Nolan, "A reconfigurable platform for cognitive networks," in *1st International Conference on Cognitive Radio Oriented Wireless Networks and Communications, (CrownCom)*, 2006, pp. 1–5.

[23] S. P. Aaron Beach, Mike Gartrell and R. Han, "X-Layer: An experimental implementation of a cross-layer network protocol stack for wireless sensor networks," Department of Computer Science University of Colorado at Boulder, Tech. Rep., December 2008.

[24] "OpenOnload," July 2010. [Online]. Available: http://www.openonload.org/

[25] G. Conti, Marco; Maselli and G. Turi, "Design of a flexible cross-layer interface for Ad Hoc networks," *Challenges in Ad Hoc Networking*, vol. 197, pp. 189–198, 2006.

[26] "CLICK Router," January 2010. [Online]. Available: http://read.cs.ucla.edu/click/

[27] "Protolib," January 2010. [Online]. Available: http://cs.itd.nrl.navy.mil/work/protolib/index.php

[28] S. G. Georg Kunz, Olaf Landsiedel and K. Wehrle, "Protocol factory: Reuse for network experimentation," in *6th USENIX Symposium on Networked Systems Design and Implementation*, 2009.

[29] O. Landsiedel, S. Kunz, Georg, and W. Klaus, "A virtual platform for network experimentation," in *Proc. of the ACM Workshop on Virtualized Infrastructure Systems and Architectures*, 2009, pp. 45–52.

[30] A. Herms and D. Mahrenholz, "Unified development and deployment of network protocols," in *MESH NETWORKING Workshop*, Budapest, Hungary, July 2005.

[31] M. Sooriyabandara, T. Farnham, and Wellens, "Unified link layer API: A generic and open API to manage wireless media access," *Computer Communications*, vol. 31, no. 5, pp. 962–979, March 2008.

[32] G. Noubir, W. Qian, B. Thapa, and Y. Wang, "Experimentation-oriented platform for development and evaluation of MANET cross-layer protocols," *Ad Hoc Networks*, vol. 7, no. 2, pp. 443 – 459, 2009.

[33] L. Sánchez, J. Lanza, and L. Muñoz, "Experimental assessment of a cross-layer solution for TCP/IP traffic optimization on heterogeneous personal networking environments," *Personal Wireless Communications*, pp. 284–296, 2006.

[34] H. Aïache, V. Conan, L. Lebrun, J. Leguay, S. Rousseau, and D. Thoumin, "XIAN automated management and nano-protocol to design cross-layer metrics for ad hoc networking," in *NETWORKING Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*, 2008, pp. 1–13.

[35] H. Lundgren, D. Lundberg, J. Nielsen, E. Nordstrom, and C. Tschudin, "A large-scale testbed for reproducible ad hoc protocol evaluations," in *IEEE Wireless Communications and Networking Conference, 2002.*, vol. 1, 2002, pp. 412–418 vol.1.

[36] R. Lent, "A testbed validation tool for MANET implementations," in *IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2005, pp. 381–388.

[37] H. Kazemi, G. C. Hadjichristofi, and L. A. DaSilva, "MMAN - A monitor for mobile ad hoc networks: Design, implementation and experimental evaluation," in *Proc. of the ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH)*, 2008.

[38] E. Weingärtner, C. Terwelp, and K. Wehrle, "ProMoX: A protocol stack monitoring framework," Aachen Uni., Germany, Tech. Rep., 2009.

[39] "SolarFlare," July 2010. [Online]. Available: http://www.solarflare.com/index.php

[40] L. Doyle, P. Sutton, K. Nolan, J. Lotze, B. Ozgul, T. Rondeau, S. Fahmy, H. Lahlou, and L. DaSilva, "Experiences from the iris testbed in dynamic spectrum access and cognitive radio experimentation," in *IEEE Symposium on New Frontiers in Dynamic Spectrum*, April 2010, pp. 1–8.

[41] "Android," April 2011. [Online]. Available: http://www.android.com/

[42] M. Zec, "Implementing a clonable network stack in the FreeBSD kernel," in *Proceedings of the USENIX 2003 Annual Technical Conference*, 2003, pp. 137–150.

[43] IETF, *RFC 2326, Real Time Streaming Protocol (RTSP)*, 1998. [Online]. Available: https://www.ietf.org/rfc/rfc2326.txt

[44] "IETF RFC 3550 - Real Time Control Protocol (RTCP)," July 2003.

[45] L. De Cicco, S. Mascolo, and C. Abdallah, "An experimental evaluation of Akamai adaptive video streaming over HSDPA networks," in *Proc. of IEEE International Symposium on Computer-Aided Control System Design (CACSD)*, Sep 2011, pp. 13 – 18.

[46] D. Nguyen, L. Tran, H. Le, N. P. Ngoc, and T. C. Thang, "An evaluation of segment duration effects in HTTP adaptive streaming over mobile networks," in *National Foundation for Science and Technology Development Conference on Information and Computer Science (NICS)*, Sept. 2015, pp. 248–253.

[47] I. Sodagar, "The MPEG-DASH standard for multimedia streaming over the internet," *IEEE MultiMedia*, vol. 18, no. 4, pp. 62 –67, April 2011.

[48] "International Standards Organization (ISO)," April 2012. [Online]. Available: http://www.iso.org/iso/home.html

[49] Conviva, "How consumers judge their viewing experience," conviva.com, Tech. Rep., 2015.

[50] S. S. Krishnan and R. K. Sitaraman, "Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs," in *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*. ACM, 2012, pp. 211–224.

[51] "YouTube," March 2012. [Online]. Available: www.youtube.com

[52] B. Sardar and D. Saha, "A survey of TCP enhancements for last-hop wireless networks," *IEEE Communications Surveys Tutorials*, vol. 8, no. 3, pp. 20 –34, 2006.

[53] K.-C. Leung and V. Li, "Transmission control protocol (TCP) in wireless networks: Issues, approaches, and challenges," *IEEE Communications Surveys Tutorials,*, vol. 8, no. 4, pp. 64 –79, 2006.

[54] X. Chen, H. Zhai, J. Wang, and Y. Fang, "TCP performance over mobile ad hoc networks," *Canadian Journal of Electrical and Computer Engineering,*, vol. 29, no. 1, pp. 129 –134, jan.-april 2004.

[55] E. Setton, T. Yoo, X. Zhu, A. Goldsmith, and B. Girod, "Cross-layer design of ad hoc networks for real-time video streaming," *IEEE Wireless Communications Magazine*, vol. 12, pp. 59–65, 2005.

[56] X. Zhu and B. Girod, "Video streaming over wireless networks," in *15th European Signal Processing Conference,*. IEEE, 2007, pp. 1462–1466.

[57] M. Nikoupour, A. Nikoupour, and M. Dehghan, "A cross-layer framework for video streaming over wireless ad-hoc networks," in *Third International Conference on Digital Information Management (ICDIM),*, Nov. 2008, pp. 340 –345.

[58] Y. Shan and A. Zakhor, "Cross layer techniques for adaptive video streaming over wireless networks," in *Proc. of IEEE International Conference on Multimedia and Expo,*, vol. 1, 2002, pp. 277 – 280.

[59] A. Moid and A. O. Fapojuwo, "A cross-layer framework for efficient streaming of H.264 video over IEEE 802.11 networks," *J. Comp. Sys., Netw., and Comm.*, vol. 2009, pp. 4:1–4:13, Jan. 2009. [Online]. Available: http://dx.doi.org/10.1155/2009/682813

[60] J. Klaue, B. Rathke, and A. Wolisz, "Evalvid - a framework for video transmission and quality evaluation," in *In Proc. of the 13th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, 2003, pp. 255–272.

[61] A. Moid and A. Fapojuwo, "Test-bed implementation of a cross-layer framework for video streaming over IEEE 802.11 ad-hoc wireless network," in *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications,*, sept. 2009, pp. 2965 –2969.

[62] T. Kim and M. H. Ammar, "Receiver buffer requirement for video streaming over tcp," in *Visual Communications and Image Processing*. SPIE, 2006.

[63] L. De Cicco, S. Mascolo, and C. Abdallah, "An experimental evaluation of Akamai adaptive video streaming," in *USAB, Special Session Interactive Multimedia Applications*. Springer-Verlag, November 2010, pp. 447–464.

[64] X. Qiu, H. Liu, D. Li, S. Zhang, D. Ghosal, and B. Mukherjee, "Optimizing HTTP-based adaptive video streaming for wireless access networks," in *IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT),*, Oct. 2010, pp. 838 –845.

[65] "QualNet Simulator," March 2012. [Online]. Available: http://www.scalable-networks.com/

[66] V. Srivastava and M. Motani, "Cross-layer design: A survey and the road ahead," *IEEE Communications Magazine,*, vol. 43, no. 12, pp. 112–119, Dec. 2005.

[67] F. Fu and M. van der Schaar, "A new theoretic framework for cross-layer optimization with message exchanges," in *IEEE INFOCOM Workshops*, April 2008, pp. 1–4.

[68] ——, "A new systematic framework for autonomous cross-layer optimization," *IEEE Transactions on Vehicular Technology*, vol. 58, no. 4, pp. 1887–1903, May 2009.

[69] Y. Zhang, F. Fu, and M. van der Schaar, "On-line learning and optimization for wireless video transmission," *IEEE Transactions on Signal Processing*, vol. 58, no. 6, pp. 3108–3124, June 2010.

[70] N. Mastronarde and M. van der Schaar, "A new approach to cross-layer optimization of multimedia systems," in *IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP)*, March 2010, pp. 2310–2313.

[71] F. Fu and M. van der Schaar, "Structural Solutions for Dynamic Scheduling in Wireless Multimedia Transmission," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 5, pp. 727–739, May 2012.

[72] H.-P. Shiang and M. van der Schaar, "A quality-centric TCP-friendly congestion control for multimedia transmission," *IEEE Transactions on Multimedia*, vol. 14, no. 3, pp. 896–909, June 2012.

[73] N. Mastronarde, F. Verde, D. Darsena, A. Scaglione, and M. van der Schaar, "Transmitting Important Bits and Sailing High Radio Waves: A Decentralized Cross-Layer Approach to Cooperative Video Transmission," *IEEE Journal on Selected Areas in Communications*, vol. 30, no. 9, pp. 1597–1604, October 2012.

[74] O. Habachi, H.-P. Shiang, M. van der Schaar, and Y. Hayel, "Online learning based congestion control for adaptive multimedia transmission," *IEEE Transactions on Signal Processing*, vol. 61, no. 6, pp. 1460–1469, March 2013.

[75] Y. Xiao and M. van der Schaar, "Optimal Foresighted Multi-User Wireless Video," *IEEE Journal of Selected Topics in Signal Processing*, vol. 9, no. 1, pp. 89–101, Feb 2015.

[76] D. Palomar and M. Chiang, "A tutorial on decomposition methods for network utility maximization," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 8, pp. 1439–1451, Aug 2006.

[77] Y. Zhang, F. Fu, and M. van der Schaar, "Online learning for wireless video transmission with limited information," in *International Packet Video Workshop*, May 2009, pp. 1–10.

[78] S. Petrangeli, M. Claeys, S. Latre, J. Famaey, and F. De Turck, "A multi-agent Q-Learning-based framework for achieving fairness in HTTP Adaptive Streaming," in *IEEE Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–9.

[79] M. Claeys, S. Latre, J. Famaey, and F. De Turck, "Design and Evaluation of a Self-Learning HTTP Adaptive Video Streaming Client," *IEEE Communications Letters*, vol. 18, no. 4, pp. 716–719, April 2014.

[80] J. van der Hooft, S. Petrangeli, M. Claeys, J. Famaey, and F. De Turck, "A learning-based algorithm for improved bandwidth-awareness of adaptive streaming clients," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, pp. 131–138.

[81] M. Claeys, S. Latr, J. Famaey, T. Wu, W. V. Leekwijck, and F. D. Turck, "Design and optimisation of a (FA)Q-learning-based HTTP adaptive streaming client," *Connection Science*, vol. 26, no. 1, pp. 25–43, 2014. [Online]. Available: http://dx.doi.org/10.1080/09540091.2014.885273

[82] S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen, and F. De Turck, "Network-based dynamic prioritization of http adaptive streams to avoid video freezes," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, pp. 1242–1248.

[83] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[84] "DASH Industry Forum," Online, January 2016. [Online]. Available: http://dashif.org/

[85] M. Tokic and G. Palm, "Value-Difference Based Exploration: Adaptive Control between Epsilon-Greedy and Softmax," in *Advances in Artificial Intelligence*, ser. Lecture Notes in Computer Science, J. Bach and S. Edelkamp, Eds. Springer Berlin Heidelberg, 2011, vol. 7006, pp. 335–346. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-24455-1_33

[86] C. W. Chen and F. Zhengyong, "Video over IEEE802.11 wireless LAN: A brief survey," *China Communications*, vol. 10, no. 5, pp. 1–19, May 2013.

[87] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hobfeld, and P. Tran-Gia, "A survey on quality of experience of HTTP adaptive streaming," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 469–492, 2015.

[88] P. Juluri, V. Tamarapalli, and D. Medhi, "Measurement of Quality of Experience of Video-on-Demand Services: A Survey," *IEEE Communications Surveys & Tutorials*, 2015.

[89] P. Casas, A. D'Alconzo, P. Fiadino, A. Bar, A. Finamore, and T. Zseby, "When YouTube Does not Work –Analysis of QoE-Relevant Degradation in Google CDN Traffic," *IEEE Transactions on Network and Service Management*, vol. 11, no. 4, pp. 441–457, Dec 2014.

[90] P. Casas, P. Fiadino, A. Sackl, and A. D'Alconzo, "YouTube in the move: Understanding the performance of YouTube in cellular networks," in *IFIP Wireless Days (WD)*, Nov 2014, pp. 1–6.

[91] P. Casas, A. D'Alconzo, P. Fiadino, A. Bar, and A. Finamore, "On the analysis of QoE-based performance degradation in youtube traffic," in *International Conference on Network and Service Management (CNSM)*, Nov 2014, pp. 1–9.

[92] X. Che, B. Ip, and L. Lin, "A Survey of Current YouTube Video Characteristics," *IEEE MultiMedia Magazine*, vol. 22, no. 2, pp. 56–63, Apr 2015.

[93] F. Wamser, M. Seufert, P. Casas, R. Irmer, P. Tran-Gia, and R. Schatz, "YoMoApp: A tool for analyzing QoE of YouTube HTTP adaptive streaming in mobile networks," in *European Conference on Networks and Communications (EuCNC)*, June 2015, pp. 239–243.

[94] J. Reed, A. Abdallah, M. Thompson, A. MacKenzie, and L. DaSilva, "The fins framework: Design and implementation of the flexible internetwork stack (fins) framework," *Mobile Computing, IEEE Transactions on*, vol. 15, no. 2, pp. 489–502, Feb 2016.

[95] A. S. Abdallah, M. D. Horvath, M. S. Thompson, A. B. MacKenzie, and L. A. DaSilva, "Facilitating experimental networking research with the fins framework," in *Proceedings of the 6th ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*, ser. WiNTECH '11. New York, NY, USA: ACM, 2011, pp. 103–104. [Online]. Available: http://doi.acm.org/10.1145/2030718.2030745

[96] M. Thompson, A. Abdallah, J. Reed, A. MacKenzie, and L. DaSilva, "The fins framework: an open source userspace networking subsystem for linux," *Network, IEEE*, vol. 28, no. 5, pp. 32–37, September 2014.

[97] R. Thomas, L. DaSilva, and A. MacKenzie, "Cognitive networks," in *Proc. IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN)*, Nov 2005, pp. 352–360.

[98] "TCPDUMP/LIBPCAP Public Repository." [Online]. Available: http://www.tcpdump.org/

[99] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Trans. Comput. Syst.*, vol. 18, pp. 263–297, August 2000.

[100] W. Yin, K. Bialkowski, J. Indulska, and P. Hu, "Evaluations of MadWifi MAC layer rate control mechanisms," in *International Workshop on Quality of Service (IWQoS),*, June 2010, pp. 1–9.

[101] "Skype." [Online]. Available: http://www.skype.com

[102] D. Xia, J. Hart, and Q. Fu, "On the performance of rate control algorithm Minstrel," in *IEEE International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC),*, Sept 2012, pp. 406–412.

[103] ——, "Evaluation of the Minstrel rate adaptation algorithm in IEEE 802.11g WLANs," in *IEEE International Conference on Communications (ICC),*, June 2013, pp. 2223–2228.

[104] S. Petrangeli, J. Famaey, M. Claeys, and F. De Turck, "A qoe-driven rate adaptation heuristic for enhanced adaptive video streaming," iMinds Labs, Tech. Rep.

[105] D. Gupta, D. Wu, P. Mohapatra, and C.-N. Chuah, "Experimental comparison of bandwidth estimation tools for wireless mesh networks," in *IEEE INFOCOM,*, April 2009, pp. 2891 –2895.

[106] M. Mirza, P. Barford, X. Zhu, S. Banerjee, and M. Blodgett, "Fingerprinting 802.11 rate adaption algorithms," in *INFOCOM, 2011 Proceedings IEEE*, April 2011, pp. 1161–1169.

[107] "Ns-3 Simulation Modules for Adaptive Video Streaming," July 2013. [Online]. Available: https://github.com/abdo5520/AVS-NS3

[108] J. De Vriendt, D. De Vleeschauwer, and D. Robinson, "Model for estimating QoE of video delivered using HTTP adaptive streaming," in *IFIP/IEEE International Symposium on Integrated Network Management (IM 2013),*, May 2013, pp. 1288–1293.

[109] R. Mok, E. Chan, and R. Chang, "Measuring the quality of experience of HTTP video streaming," in *IFIP/IEEE International Symposium on Integrated Network Management (IM),*, May 2011, pp. 485–492.

[110] D. P. De Farias, "The linear programming approach to approximate dynamic programming: Theory and application," Ph.D. dissertation, Stanford University, 2002.

[111] I. Chang, M. C. Fu, J. Hu, and S. I. Marcus, *Simulation-based Algorithms for Markov Decision Processes (Communications and Control Engineering)*. Springer Verlag, London, 2007.

[112] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An introduction*. Cambridge Univ Press, 2011.

[113] H. S. Wang and N. Moayeri, "Finite-state Markov channel-a useful model for radio communication channels," *IEEE Transactions on Vehicular Technology*, vol. 44, no. 1, pp. 163–171, 1995.

[114] I. Kim and Y.-T. Kim, "Realistic modeling of IEEE 802.11 WLAN considering rate adaptation and multi-rate retry," *IEEE Transactions on Consumer Electronics,*, vol. 57, no. 4, pp. 1496–1504, November 2011.

[115] J. Choi, K. Park, and C.-k. Kim, "Cross-layer analysis of rate adaptation, DCF and TCP in multi-rate WLANs," in *IEEE INFOCOM,*. IEEE, 2007, pp. 1055–1063.

[116] ——, "Analysis of cross-layer interaction in multirate 802.11 WLANs," *IEEE Transactions on Mobile Computing,*, vol. 8, no. 5, pp. 682–693, 2009.

[117] I. Chadès, G. Chapron, M.-J. Cros, F. Garcia, and R. Sabbadin, "Mdptoolbox: a multi-platform toolbox to solve stochastic dynamic programming problems," *Ecography*, vol. 37, no. 9, pp. 916–920, 2014.

[118] S. Gangadhar, T. A. N. Nguyen, G. Umapathi, and J. P. Sterbenz, "TCP Westwood (+) protocol implementation in ns-3," in *Proceedings of the International Conference on Simulation Tools and Techniques, ICST*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2013, pp. 167–175.

[119] B. Levasseur, M. Claypool, and R. Kinicki, "A TCP CUBIC implementation in ns-3," in *Proceedings of the Workshop on ns-3,*. ACM, 2014.

[120] "Wireshark," June 2011. [Online]. Available: http://www.wireshark.org/

[121] C. Mueller, S. Lederer, J. Poecher, and C. Timmerer, "Demo paper: Libdash - An open source software library for the MPEG-DASH standard," in *IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, July 2013, pp. 1–2.

[122] "The ITEC project." [Online]. Available: http://www-itec.uni-klu.ac.at/dash/

[123] L. Busoniu and R. Munos, "Optimistic planning for Markov decision processes," in *International Conference on Artificial Intelligence and Statistics, AISTATS*, vol. 22, 2012, pp. 182–189.