Content list

Content list .................................................................................................................. 4

List of figures .............................................................................................................. 6

List of acronyms ........................................................................................................ 7

1. Introduction ........................................................................................................... 9
   1.1. Objectives of the project .............................................................................. 9
   1.2. Abstract ....................................................................................................... 9
   1.3. Chapter overview ...................................................................................... 11

2. DRAP Model .......................................................................................................... 12
   2.1. Introduction ............................................................................................... 12
   2.2. DRAP implementation .............................................................................. 12
       2.2.1. Packet classifier (PC) ................................................................. 13
       2.2.2. Rate controller (RC) .................................................................. 14
       2.2.3. Dynamic leaky bucket ............................................................... 15
       2.2.4. Traffic shaper .............................................................................. 15
       2.2.5. Priority output queue .................................................................. 15

3. Click Modular Router .......................................................................................... 16
   3.1. Introduction ............................................................................................... 16
   3.2. Implementation language ......................................................................... 16
   3.3. Click router configuration ......................................................................... 16
       3.3.1. Declarations ................................................................................. 17
       3.3.2. Connections ................................................................................... 17
   3.4. Click elements ........................................................................................... 18
       3.4.1. Element class .................................................................................. 18
       3.4.2. Ports ............................................................................................... 18
       3.4.3. Configuration string ....................................................................... 19
       3.4.4. Method interfaces .......................................................................... 19
       3.4.5. Handlers .......................................................................................... 20
   3.5. Runtime environments ................................................................................ 20
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5.1. Kernel level</td>
<td>20</td>
</tr>
<tr>
<td>3.5.2. User level</td>
<td>20</td>
</tr>
<tr>
<td>4. Implementation of QoS mechanism</td>
<td>22</td>
</tr>
<tr>
<td>4.1. Methodology</td>
<td>22</td>
</tr>
<tr>
<td>4.1.1. Differentiated Services and IP DSCP field</td>
<td>22</td>
</tr>
<tr>
<td>4.1.2. Rate Controller problem</td>
<td>23</td>
</tr>
<tr>
<td>4.2. The input part of configuration</td>
<td>23</td>
</tr>
<tr>
<td>4.2.1. The input used for debugging the implementation</td>
<td>23</td>
</tr>
<tr>
<td>4.2.2. Original input</td>
<td>24</td>
</tr>
<tr>
<td>4.3. Packet classifying part</td>
<td>25</td>
</tr>
<tr>
<td>4.5. Output part of configuration</td>
<td>27</td>
</tr>
<tr>
<td>4.6. Description of new Click element</td>
<td>28</td>
</tr>
<tr>
<td>5. Testing part and results</td>
<td>29</td>
</tr>
<tr>
<td>5.1. Testing methodology</td>
<td>29</td>
</tr>
<tr>
<td>5.1.1. Packgen packet generator</td>
<td>29</td>
</tr>
<tr>
<td>5.1.2. Hardware and the Madwifi driver</td>
<td>30</td>
</tr>
<tr>
<td>5.1.3. Problems occurred during experiments</td>
<td>31</td>
</tr>
<tr>
<td>5.2. Results</td>
<td>32</td>
</tr>
<tr>
<td>5.2.1. Interarrival</td>
<td>32</td>
</tr>
<tr>
<td>5.2.2. End – To – End Delays</td>
<td>33</td>
</tr>
<tr>
<td>5.2.3. The throughput</td>
<td>34</td>
</tr>
<tr>
<td>6. Conclusion</td>
<td>36</td>
</tr>
<tr>
<td>List of references</td>
<td>37</td>
</tr>
<tr>
<td>Appendix</td>
<td>38</td>
</tr>
</tbody>
</table>
List of figures

Figure 1: The DRAP model .................................................. 13
Figure 2: AIMD rate controller algorithm .................................... 14
Figure 3: The dynamic leaky bucket algorithm ................................ 15
Figure 4: The declaration element ............................................. 17
Figure 5: The connections ..................................................... 17
Figure 6: IP DSCP field codes used in implementation .................... 22
Figure 7: Imitative packet generating ......................................... 23
Figure 8: The input part of the configuration ................................. 24
Figure 9: Packet classifying part .............................................. 25
Figure 10: Element class PacketProcessing ................................ 27
Figure 11: Output part of configuration ...................................... 27
Figure 12: Topology of the simulation ....................................... 29
Figure 13: Wireless cards used for implementation ......................... 30
Figure 14: Throughput for TCP without using Click (A) and using it (B) .... 31
Figure 15: The jitter of audio traffic without QoS implementation (A) and with it (B) ... 32
Figure 16: The jitter of video traffic without QoS implementation (A) and with it (B) ... 32
Figure 17: End-to-end delays for audio traffic without QoS implementation (A) and with it (B) .... 33
Figure 18: End-to-end delays for video traffic without QoS implementation (A) and with it (B) .... 33
Figure 19: The throughput for audio traffic without QoS implementation (A) and with it (B) .... 34
Figure 20: The throughput for video traffic without QoS implementation (A) and with it (B) .... 34
Figure 21: The throughput for TCP traffic without QoS implementation (A) and with it (B) .... 35
List of acronyms

QoS  Quality of Service
Wi-fi  Wireless Fidelity
DRAP  Dual-Queue Rate-Controlled Access Point
LAN  Local Areas Network
WLAN  Wireless Local Area Network
OFDM  Orthogonal Frequency-Division Multiplexing
PDA  Personal Digital Assistant
IP  Internet Protocol
RTP  Real Time Protocol
RTCP  Real Time Control Protocol
RT  Real Time
NRT  Non Real Time
PC  Packet Classifier
BE  Best Effort
RC  Rate Controller
MAC  Medium Access Control
ACK  Acknowledgment Code
AIMD  Additive Increase, Multiplicative Decrease
DGCRA  Dynamic Generic Cell Rate Algorithm
ITU  International Telecommunication Union
ATM  Asynchronous Transfer Mode
BSD  Berkeley Software Distribution
STL  Standard Template Libraries
DCF  Distributed Coordination Function
FIFO  First In, First Out
ASCII  American Standard Code for Information Interchange
TCP  Transmission Control Protocol
DSCP  Differentiated Services Code Point
DiffServ  Differentiated Services
ToS  Type of Service
CU  Currently Unused
ecn  Explicit Congestion Notification
PHB  Per-Hop-Behavior
EF  Express Forwarding
AF  Assured Forwarding
SADDR  Source Address
SPORT  Source Port
DADDR  Destination Address
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPORT</td>
<td>Destination Port</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>LLC</td>
<td>Logical Link Control</td>
</tr>
<tr>
<td>BSSID</td>
<td>Basic Service Set IDentifier</td>
</tr>
<tr>
<td>PCI</td>
<td>Protocol Control Information</td>
</tr>
</tbody>
</table>
1. Introduction

1.1. Objectives of the project

The main objective of the project is implementing a Linux wi-fi hotspot with QoS support. To do this without changing the Linux Kernel driver new software architecture for building flexible and configurable routers Click Modular Router is used for implementation [1]. Project is based on a proposal found in the paper of Rubén González and Llorenç Cerda introducing DRAP model [2]. The implementation code is divided into modular elements written in C++ programming language combined by the flexible Click configuration language. The router can be executed in Linux user level or kernel level as well. The experiments has been prosecuted using Packgen 0.2 traffic generator [3].

1.2. Abstract

Recently there has been a great proliferation of public wireless local area networks (WLAN’s) everywhere. Public places like hospitals, airports, coffee shops, university campus, hotels, etc., are propitious environments to deploy them offering to the general public high speed communication services with or without paying fees.

A wireless LAN or WLAN is a wireless local area network, which is the linking of two or more computers without using wires. WLAN utilizes spread-spectrum or OFDM (802.11a) modulation technology based on radio waves to enable communication between devices in a limited area, also known as the basic service set. This gives users the mobility to move around within a broad coverage area and still be connected to the network [4].

The throughput characteristics and performance of Wireless Local Area Networks (WLANs) make them ideally suited as a networking platform for hotspots. Hotspots are venues that offer Wi-Fi access. The public can use their laptop, PDA, Dual-mode phones, Nintendo DS or PlayStation
Portable to access the Internet. Of the estimated 150 million laptops, 14 million PDAs, and other emerging Wi-Fi devices sold per year for the last few years, most include the Wi-Fi feature [5].

Those facts are related with an increasing trend to use videoconferencing. A videoconference is a set of interactive telecommunication technologies which allow two or more locations to interact via two-way video and audio transmissions simultaneously. It has also been called visual collaboration and is a type of groupware. It differs from videophone in that it is designed to serve a conference rather than individuals. Videoconferencing over IP seems to be replacement of phone calls and etc. In hotspots, an access point plays a switch role in the sense it forwards packets among nodes inside the WLAN and also to these wireless nodes from outside the WLAN [6]. This causes a bottleneck at the access point degrading the downlink performance in such a way that is very difficult to provide Quality of Service (QoS) to multimedia contents, for instance, videoconferencing.

Quality of service, or QoS can be explained as a set of quality requirements on the collective behavior of one or more objects. In the fields of packet-switched networks and computer networking, the traffic engineering term Quality of Service refers to control mechanisms that can provide different priority to different users or data flows, or guarantee a certain level of performance to a data flow in accordance with requests from the application program. Quality of Service guarantees are important if the network capacity is limited, especially for real-time streaming multimedia applications, for example voice over IP and IP-TV, since these often require fixed bit rate and may be delay sensitive. A network or protocol that supports Quality of Service may agree on a traffic contract with the application software and reserve capacity in the network nodes during a session establishment phase. During the session it may monitor the achieved level of performance, for example the data rate and delay, and dynamically control scheduling priorities in the network nodes. It may release the reserved capacity during a tear down phase. A best-effort network or service does not support Quality of Service. In computer networking, a good QoS may mean advanced QoS mechanisms, or high probability that the network is able to provide the requested level of performance [7].
1.3. Chapter overview

Chapter 2 introduces the DRAP model, this project is based on. Introduction of the Click Modular Router which was used for implementation is given in Chapter 3. Chapter 4 describes implementation of the quality of service mechanism for a hotspot. All of the testing work is described in Chapter 5 and the conclusion and final words is in Chapter 6.
2. DRAP Model

2.1. Introduction

The Dual-Queue Rate-Controlled Access Point (DRAP) was proposed by Rubén González and Llorenç Cerda' from Computer Architecture Department (Technical University of Catalonia) [2]. They introduced a mechanism that may be collocated at the access point to provide QoS to videoconferencing flows in the downlink direction. They assumed that packets belonging to videoconferencing flows can be classified into its audio and video components. A mechanism was proposed, where video and best effort traffic are rate limited.

2.2. DRAP implementation

Rubén González and Llorenç Cerda' focused on videoconferencing applications designed for Internet as gnomemeeting [8] during their implementation work. These applications use RTP/RTCP to manage audio and video flows. RTCP provides feedback about the quality (delay and losses) of these flows. This information is used to automatically adjust the parameters of the audio and video codecs, such that they fit the available bandwidth. The audio flow has stronger constraints, since a minimum bandwidth is required for a reasonable quality. The video flow is more adaptive, e.g. adjusting the frame rate and image resolution. In case of scarce bandwidth, it is possible to only send a still image, or even close the video session. Therefore, they proposed a dual queue strategy where the video and best effort traffic are rate controlled, where one queue (RT) is for real time traffic (audio and video), and the other one (NRT) is for best effort traffic. The RT queue is shorter and has full priority over the NRT queue. Achievement was called Dual-Queue Rate-Controlled Access Point (DRAP):
2.2.1. Packet classifier (PC)

Packets coming from the upper layer (IP) are classified by the PC. If the incoming packet has been marked as Best-Effort (BE), then it is forwarded to the Traffic Shaper. Otherwise, it is a Real-Time (RT) packet. They assumed that RT packets can be further classified as audio and video packets. Video packets are forwarded to a Dynamic Leaky Bucket, and audio packets are directly forwarded to the RT queue.
2.2.2. Rate controller (RC)

To regulate both the Best-Effort and the video traffic a Rate Controller was used. Every node implements it to independently regulate these traffic flows. The rate controller provides the departure rate of the Traffic Shaper and the Dynamic Leaky Bucket blocks based on the packet delay information measured at the MAC layer. Every transmitting node knows the beginning and the ending time of its packet transmissions. Hence, packet delay is easily calculated as the difference between packet sending time and the correspondent ACK reception time. To calculate the departure time of the best-effort traffic shaper, and the limit for video rate at the dynamic leaky bucket, they used an AIMD (Additive Increase, Multiplicative Decrease) algorithm:

```
Procedure update shaping rate
{ /* Called every T seconds */ if (n > 0) }
then
  s ⇔ s × (1 − r/100)
else
  s ⇔ s + c
end if
if ((s − a) > a × g/100) then
  s ⇔ a × (1 + g/100)
end if
```

Figure 2: AIMD rate controller algorithm

The variable \( n \) counts the packets with delays higher than the delay threshold \( d \). The variable \( s \) is the rate limit computed by the algorithm. If one or more packets have been delayed more than \( d \), then \( s \) is reduced (multiplicative decrease by \( r \) percent). If packets delay is below \( d \), then \( s \) is increased gradually (additive increase with \( c \) kbps increment). If the actual transmission rate is considerably higher than the rate-limit \( s \), then a BE source might transmit an uncontrolled burst affecting RT packets. To avoid it, if the difference between actual transmission rate and the rate-limit is higher than \( g\% \) of the actual rate, then the AIMD algorithm adjusts the shaping rate to be \( g\% \) above the actual transmission rate.
2.2.3. Dynamic leaky bucket

Since video traffic cannot be delayed, this module limits the video rate by discarding excessive traffic. The rate is time varying and adjusted by the RC. Dynamic leaky bucket was implemented using the Dynamic Generic Cell Rate Algorithm (DGCRA) defined by ITU for ATM Networks:

\[

c_k = l_{vst} + L/V_k \\
y_k = c_k - a_k \\
\text{if } (y_k \leq \tau_u) \text{ then} \\
l_{vst} = \max(c_k, a_k) \\
\text{else} \\
\text{Drop} \\
\text{end if}
\]

Figure 3: The dynamic leaky bucket algorithm

\( V_k \) is the rate in bps computed by the RC at packet \( k \) arrival, and \( L \) is the size of the packet in bits. Upon arrival of packet \( k \) at time \( a_k \), the algorithm computes the theoretical arrival time \( c_k \), and the deviation \( y_k \) from it. If \( y_k \) is higher than a tolerance \( \tau_u \), the packet is discarded.

2.2.4. Traffic shaper

The PC forces best-effort packets to go through the traffic shaper. The shaper stores arriving packets in a queue with the service rate fixed by RC.

2.2.5. Priority output queue

At the output of the mechanism they allocate a dual priority queue: One Real-Time (RT) queue and another one for Best-Effort (NRT queue). Audio packets go directly from the PC to the RT queue. The video packets arrive to the same RT queue but once they have passed through the dynamic leaky bucket. The best-effort packets are allocated in the NRT queue once they are rate conformed at the shaper stage. Finally, the RT queue is always served first, thus, best effort packets are not dispatched to the MAC layer meanwhile the RT queue is not empty.
3. Click Modular Router

3.1. Introduction

The Click modular router is an open source flow-based router that can be run either in user space or in kernel mode on Linux or BSD machines. Offering an extensible toolkit for writing packet processors, is the general idea of Click. Evidently, this idea is different from most router software as most firms want to receive a fee for their knowledge and work. Because of this motivation, changing parts of the protocol or adding parts to the protocol is not allowed either. Those routers usually only permit the network administrator to change some settings, but that’s it.

Click router is assembled from packet processing modules called elements. Individual elements implement simple router functions like packet classification, queuing, scheduling, and interfacing with network devices. For example there are elements in Click that accept packets from a network interface (the FromDevice element), that store packets in a buffer (the Queue element), that discards all the incoming packets (the Discard element) and so on. All the Click elements can be found at the Click Modular Router web page [9].

3.2. Implementation language

In view of the fact that Click should be able to run in kernel mode, Click is implemented in a 'limited' C++ language. It is forbidden to include classes from the standard template libraries (STL) because of this kernel-feature Click has. Luckily, Click provides its own implementation of frequently used STL classes with a similar interface as the STL classes.

3.3. Click router configuration

The Click language describes the configuration of a Click router. It has two main directives: declarations declare new elements and connections connect those elements together. Click router
configuration consists of a list of processing elements which are directed by a graph. This graph describes the possible flow a packet has. This configuration is saved on file and parsed by the Click router to initialize all elements and their interconnection.

3.3.1. Declarations

The declaration of elements looks like this:

\[
\text{name :: class (config); } \quad \text{(e.g. output ::ToDevice(ath0);)}
\]

Figure 4: The declaration element

This declares an output element that has the same type as the ToDevice() element and listens on the interface ath0 which is a configuration string. The configuration strings are arguments that are passed to an element for initializing the state of the element.

3.3.2. Connections

The edges, called connections of the elements like:

\[
\text{name1 [port1] -> [port2] name2; } \quad \text{(e.g. IPClassifier[0] -> [0] output();)}
\]

Figure 5: The connections

They represent possible paths that packets may travel. Here the two names are the names of previously declared elements, and the two ports are nonnegative integers. Example says that IPClassifier’s output port 0 should be connected to output’s input port 0. Two connections can be strung together into a single statement if the output element of one is the same as the input element of the other.
3.4. Click elements

Elements are the building blocks of a router that have packet processing capabilities. The Click architecture is centered on the element. Each element is a software component representing a unit of router processing. Elements perform conceptually simple computations, such as decrementing an IP packet’s time-to-live field, rather than large, complex computations, such as IP routing. They generally examine or modify packets in some way. Packets are the particles of network data that routers process. At run time, elements pass packets to one another over links called connections.

The element is the most important user-visible abstraction in Click. They control every aspect of router packet processing. Every property of a router configuration is specified either through the choice of elements or through their arrangement. Device handling, routing table lookups, queuing, counting, and so forth are all implemented by elements. Inside a running router, elements are represented as C++ objects that maintain private state, and connections are pointers to elements. Elements have five important properties: element class, ports, configuration strings, method interfaces and handlers.

3.4.1. Element class

Each element belongs to an element class that determines the element’s behavior. An element’s class specifies which code to execute when the element processes a packet. For example the code in an element class determines how many ports elements of that class will have, what handlers they will support, and how they will process packets. A packet transfer from one element to the next is implemented with a single virtual function call.

3.4.2. Ports

Each element also has input and output ports, which serve as the endpoints for packet transfers. Every connection links an output port on one element to an input port on another. Only ports of the same kind can be connected together. For example, a push port cannot be connected with a pull port. An element can have zero or more of each kind of port. Different ports can have different semantics; for
example, many elements emit normal packets on their first output port and erroneous packets on their second. The number of ports provided by an element may be fixed, or it may depend on the element’s configuration string or how many ports were used by the configuration. Ports may be push, pull, or agnostic. Click supports two packet transfer mechanisms, called push and pull processing. On a push connection, packets start at the source element and are passed downstream to the destination element. On a pull connection, in contrast, the destination element initiates packet transfer: it asks the source element to return a packet, or a null pointer if no packet is available. The type of a connection is determined by the ports at its endpoints. Each port in a running router is either push or pull. Connections between two push ports are push, and connections between two pull ports are pull; connections between a push port and pull port are illegal. Elements set their ports’ types as the router is initialized. They may also create agnostic ports, which behave as push when connected to push ports, and pull, when connected to pull ports.

3.4.3. Configuration string

The optional configuration string contains additional arguments passed on the element at router initialization time. For many element classes, configuration strings define per-element state and fine-tune element behavior, much as constructor arguments do for objects. Lexically, a configuration string is a list of arguments separated by commas. Most configuration arguments fit into one of a small set of data types: IP addresses, for example, or integers, or lists of IP addresses.

3.4.4. Method interfaces

Each element exports methods that other elements may access. This set of methods is grouped into method interfaces. Every element supports at least the base method interface which contains, for example, methods for transferring packets. Elements can define and implement arbitrary other interfaces in top of this. For example, Click’s Queue element, which implements a FIFO packet queue, exports an interface that reports its current length.
3.4.5. Handlers

Handlers are methods that are exported to the user, rather than to other elements in the router configuration. They support simple, text based read/write semantics, as opposed to fully general method call semantics. For example, the Queue element mentioned above has a handler that reports its current length as a decimal ASCII string. In the Linux kernel driver, handlers appear as files in the dynamic / proc file system. At user level, handlers may be accessed via a TCP-IP-based protocol.

3.5. Runtime environments

A click configuration can either run in kernel level, in user level or in the simulator. Configurations for the three different environments differ slightly, mostly with respect to packet sources and sinks.

3.5.1. Kernel level

To run in kernel level the kernel has to be patched in order to allow click elements written in C++ to be compiled and executed inside a kernel module. Thereafter the compiled module includes the click elements written in C++ and can be installed by insmod or more comfortable by click-install, which mounts the click file system and can receive as a parameter the configuration file of the router. To change the router configuration once the kernel module is loaded the click proc like file system can be used. This file system is also used to communicate with the running router through the installed handlers. For writing elements all usual kernel level limitations apply, especially no use of floating point numbers and of user level libraries is permitted. In kernel level the click module replaces the Linux networking stack. Packets have to be explicitly handed to the system or packet sniffers. If not, these packets will only be seen by the click code running inside kernel level.

3.5.2. User level

Code compiled for user level click can be easily executed without patching the kernel. A configuration file is passed as parameter when loading click. Click gets the packets from the Linux system and other registered packet sniffers or user level programs can see the packets as well, even if
they are discarded inside the click configuration. Interaction with handlers can for example take place over a socket by using a ControlSocket element which establishes a socket listening on the specified port and calling the desired handler inside click when a matching petition is received. In this project user level was used.