Design and Implementation of a Virtual Quality of Service MAC Layer (VQML) for Wireless LANs

Mahbub Hassan, Kenneth Lee, Mohammad Rezvan
School of Computer Science and Engineering
The University of New South Wales
Sydney 2052, Australia

UNSW-CSE-TR0208

July 2002

THE UNIVERSITY OF
NEW SOUTH WALES

SYDNEY • AUSTRALIA

Abstract

Wireless LANs are becoming increasingly popular. While the technology offers wireless connectivity, it offers minimal or no quality of service (QoS) to multimedia applications. We propose a virtual QoS MAC layer (VQML) between MAC and networking layers to provide QoS. The proposed VQML architecture is implemented in a Linux platform and tested in an experimental wireless network test-bed in the Network Research Laboratory of UNSW. This report details the design, implementation, and experimentation of VQML.

1 Introduction

Current wireless MAC protocols offer minimal or no support for QoS applications. There are some recent initiatives in IEEE 802.11 working group to add QoS features to 802.11a and 802.11b family. This can cause cross-compatibility problems for wireless LAN (WLAN) environments. We propose an intermediate layer between MAC and networking layer (we call it VQML) to provide QoS support. Such architecture can provide seamless migration of applications between different WLANs with different MAC protocols. VQML has two salient features:

- It provides a MAC-independent QoS mechanism. If QoS support is implemented at MAC layer, each vendor may decide to implement its own specific approach.

- It is a software-based architecture. A software-based approach introduces more flexibility as the design can be easily changed and upgraded. While the same level of flexibility may be achieved by a hardware-based (e.g., FPGA and/or ASIC-based design), manipulation and/or changes can be applied in software more easily.

We have implemented VQML on Linux platform. The implementation was later tested in an experimental wireless network. Experimental results demonstrate that VQML can provide QoS to video applications over WLANs in the presence of background data traffic.

The rest of the report is organised as follows. Section 2 presents the VQML architecture. Linux implementation of VQML is described in Section 3. Section 4 explains the experimental wireless network test-bed and presents results obtained from the tests. Related work is discussed in Section 5. Finally, we provide our conclusions in Section 6.

2. VQML Architecture

Figure 1 shows the traditional protocol stack used in WLANs. In the traditional stack, Internet Protocol (IP) (layer 3) directly communicates with MAC (layer 2). In the proposed VQML architecture (see Figure 2), a VQML layer resides between IP and MAC layers, and offers software-based QoS support for networking layer. There is also a bypass connection between networking layer and MAC layer. This means that the VQML does not interrupt the operations of normal protocol stack. As regards to implementation, we have divided the project into two phases:

- In Phase-I, we deploy VQML only at wireless stations. Therefore, in Phase-I, wireless users will only be able to "send" QoS traffic across the wired backbone via WLAN access point. In other words, in Phase-I, only up-link part of the wireless link will become QoS-capable. One very useful application under such scenario is transmission of voice traffic (VOIP) from wireless stations to another user that resides outside of WLAN environment.

- In Phase-II, in addition of wireless stations, VQML (possibly with some changes) will also be deployed in access points. Therefore, down-link part of the wireless link will become QoS-capable as well. *Phase-II is outside the scope of this report.*

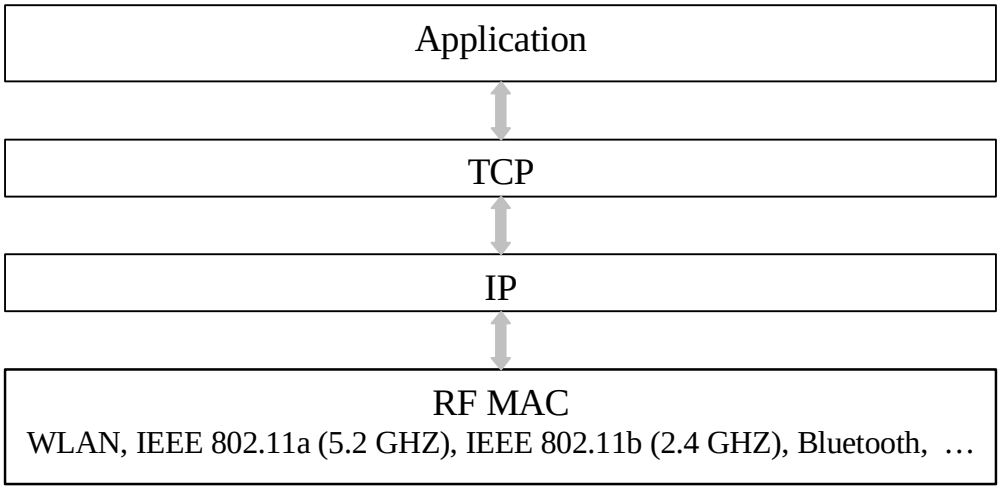

Figure 1: A typical protocol stack for wireless LAN

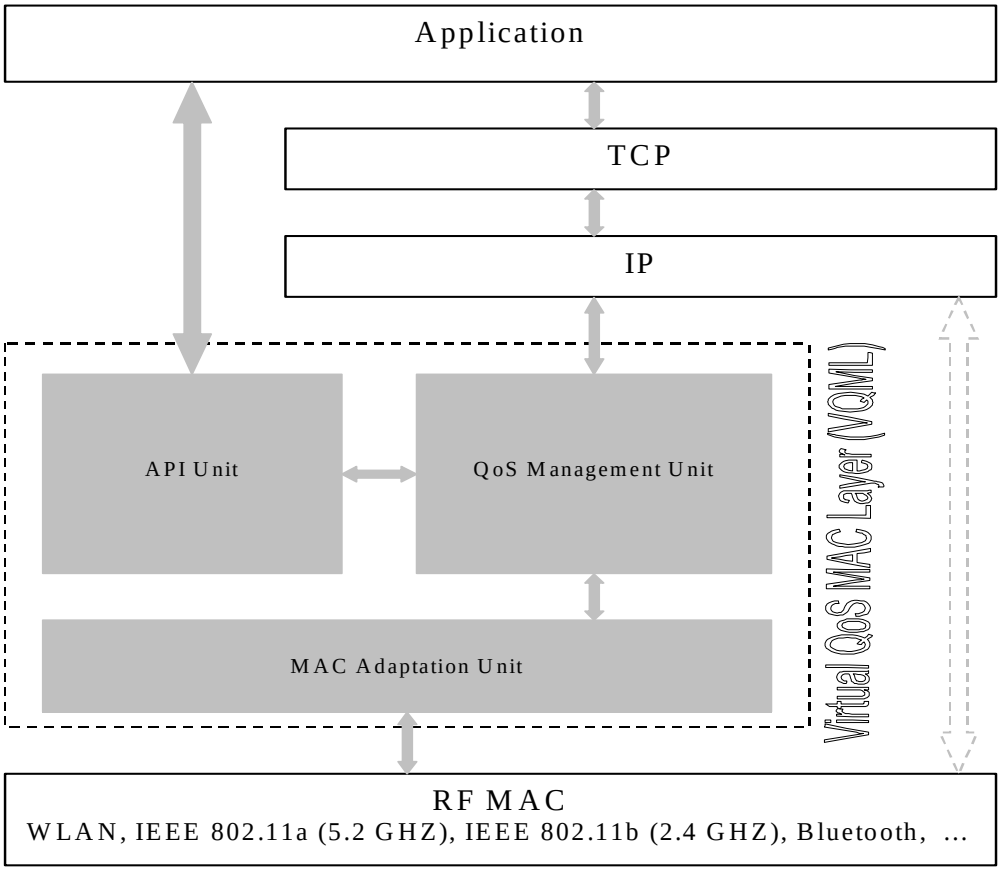

Figure 2: VQML resides between MAC and layer 3 to provide QoS for Wireless LAN

VQML consists of three units, QoS management, application programming interface, and MAC adaptation. Each unit handles a specific set of tasks. These units are further described in the following subsections.

2.1 QoS Management Unit

This unit is the heart of the proposed VQML architecture. It handles all QoS- specific tasks. Other units are designed to support the operations of this unit. It is designed to be fully programmable system calls provided by API unit. Figure 3.3 details the internal architecture of this unit.

QoS management unit itself is made up of a number of units. The Packet Classifying Unit (PCU) receives network layer packets (i.e., IP packets). The main task of this unit is to map *"user-level*", "*application-level*", or "*flow-level"* priority to "*access-level*" priority. Because wireless stations operate in a shared medium, there is no mean of defining a QoS "route". While it is possible to compute an end-to-end QoS route in ATM or IP networks, in WLANs the concept of QoS can only be interpreted as *"access"* priority. This is because in a WLAN environment, stations compete to gain access to the shared wireless channel. Access priority is inherently defined for each individual packet (or individual frame). Therefore, VQML can smoothly operate even if a specific application or flow like a MPEG-2 stream generates packets with different priorities. PCU can potentially look at ToS field (or similar field) in IP header and assign a priority level to the packet. Based on the assigned priority, PCU then stores the packet in one of the queues in the queue pool.

As Figure 3 shows, PCU is connected to a parameter bank. This bank provides different settings for parameters that control the operation PCU and queue pool. For example, PCU may be forced to look at other fields than ToS or the number of access priority levels and number of queues can be changed. These changes can happen via system calls provided by API unit from application/user layers and are stored in this bank. Each time VQML resets and starts working, it can access this bank to tune its behaviour. The Transmission Control Unit (TCU) selects the packets from different queues based on a specific algorithm. Note that a variety of different scheduling algorithms can be implemented. It would be wise to experiment with a broad range of scheduling algorithms. Again, because TCU is connected to an algorithm bank, multiple algorithms can be implemented and stored simultaneously.

2.2 Application Programming Interface (API) Unit

This unit provides application level interface so that the functionality of VQML can be easily changed or tuned. It can also be used for performance monitoring and diagnostic purposes. The importance of API unit can be described as follows.

- At experimental level, it provides a flexible way to experiment with different algorithms and parameter settings that control the operation of VQML. The data collected from these experiments can be used to find optimal working range for parameters and also efficient algorithms. Without API unit, each new feature/algorithm has to be tightly hard-coded into the source code and new system should be built.

- At operational level, API can be used to fine-tune performance of VQML based on specific

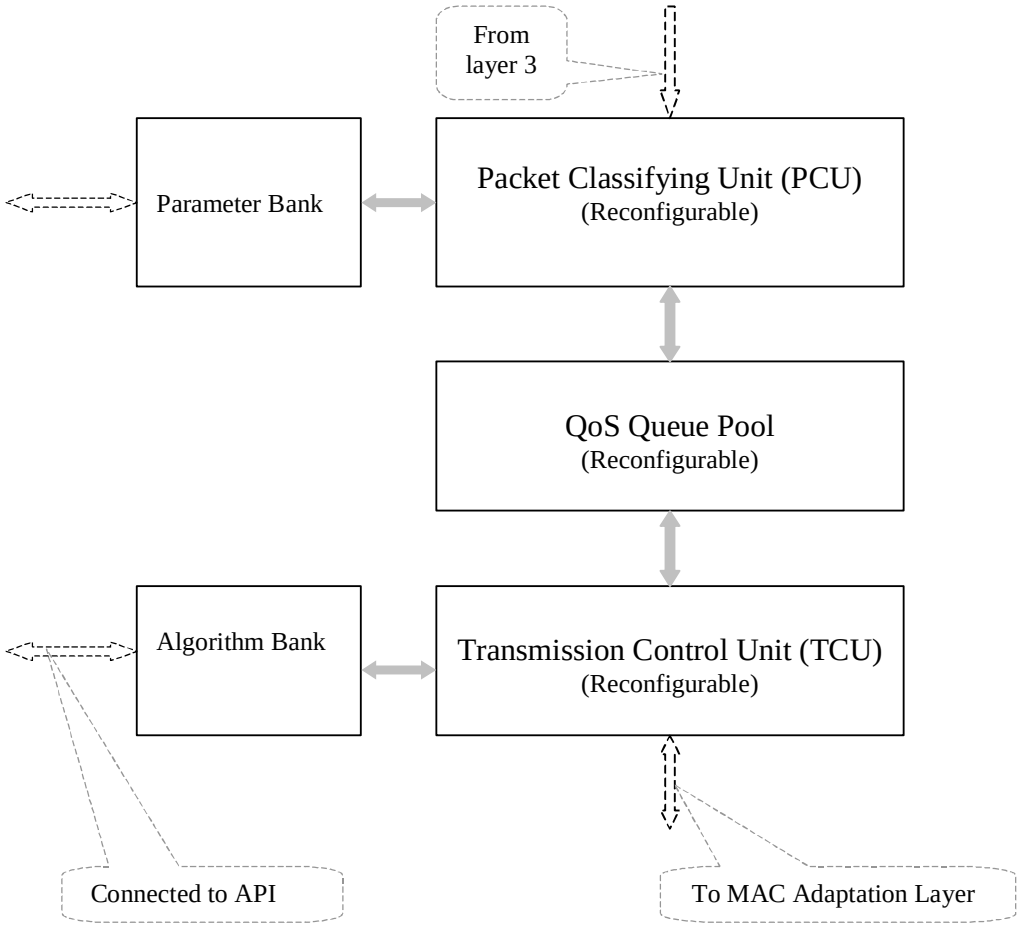

Figure 3: Internal architecture of QoS management unit

requirements of user/network.

- Because API system calls will be visible at application level, it may be feasible to establish application-level QoS awareness. However, the real merit and practicality of this issue needs more investigation and is not the purpose of VQML.

API unit provides a set of application-level system calls for tuning and changing different operations of VQML. A user interface may be suitable to let system users to tune different aspects of VQML. Some tunable aspects of VQML can be outlined as follows:

1. IP header look up. PCU (refer to Section 2.1) needs to look at some fields in the incoming IP packets to interpret their required QoS and accordingly assign an access priority to the packets. Such QoS information can be carried in different parts of IP header (e.g., ToS). The PCU can be tuned so that it can look for QoS information at different places in the IP header. This gives the system more flexibility.

2. Number of priorities. PCU classifies the incoming IP packets into multiple "levels" or "access priority". The number of these priorities (and the number of corresponding priority queues) is tunable.

3. Packet scheduling algorithm. TCU (refer to Section 2.1) is responsible for transmission of QoS IP packets and uses an algorithm to transmit the packets on wireless channel. Such algorithm may be implemented in many different ways. Therefore, the rapid and easy implementation of new algorithms in TCU seems an attractive idea. API will provide a system call by which one of the available algorithms is selected.

2.3 MAC Adaptation Unit

VQML is a cross-platform protocol architecture. This means the network layer is connected to VQML and is not aware of underlying MAC protocol. On the other hand, VQML itself has to deal with and support different wireless MAC protocols. This mandates that some low-level components of VQML should be re-written, each time the whole architecture is ported on a new MAC layer. The MAC adaptation unit (MAU) is designed to handle this issue smoothly.

Figure 4 shows the details of MAU. It is made of two units; unified portable interface (UPI) and device driver/chip interface (DDI). The UPI provides a seamless interface for communication and hides the details of MAC protocol. While the interface remains fixed, the internal implementation of UPI is subject to change as we port the architecture on top of different MAC protocols. DDI is in direct contact with MAC chip (via device driver and/or hardware ports on MAC chip). This unit should be completely re-written for each MAC protocol.

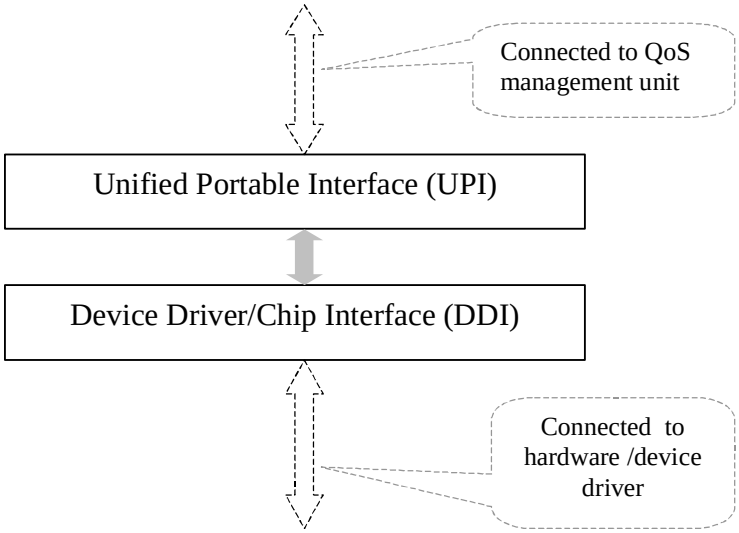

Figure 4: Internal architecture of MAC adaptation Unit

3 Linux Implementation

We have implemented VQML in the traffic control (TC) framework of Linux operating system. We briefly describe the TC framework before explaining VQML implementation.

3.1 Traffic Control

TC in Linux provides a range of control functions for packets transmitted from the Linux networking code to the device driver. TC allows to classify packets, queuing of packets in multiple queues according to a given priority, and scheduling of packets from different queues with different rates. These features are very useful in implementing VQML. Implementation of VQML in the TC framework is described below.

3.2 Implementation of VQML units

Implementation of VQML requires modifications to both kernel and user-space components of TC. The user-space program of TC is a part of the *iproute* package. It is used to manipulate individual traffic control elements. This includes manipulating parameters for the queuing disciplines, filters and traffic classes. It forms the basis of the API unit for VQML, it provides a flexible way to control the operation of VQML.

The QoS management unit and MAC adaptation unit of VQML are both integrated into the Linux kernel. The modularised nature of Linux means modules can be dynamically inserted and removed from the kernel and thereby ensures a flexible and extendable architecture.

The kernel code of TC resides mainly in the directory *net/sched* of the Linux kernel source tree. Major components of QoS management unit are implemented here. Transmission control unit (TCU) selects the packets from different queues based on a specific queuing discipline.

The packet classifying unit (PCU) is implemented in the traffic control framework as filters and classes. There are a number of ways to classify packets:

- RSVP – using 5 tuples and RSVP control messages.
- u32 – matching any field in the IP header
- route – using routing table
- fw/ipchains/iptables – using netfilter/iptables or ipchains to mark packets to a flowid.

The communication between traffic control elements in user-space and in the kernel is implemented using the *rtnetlink* mechanism (rtnetlink is based on netlink mechanism).

The VQML framework is designed to be flexible and completely extensible. The PCU and the TCU in the QoS management unit are designed to be reconfigurable and upgradeable. TCU can be reconfigured via the API unit. The tc interface enables users to select a suitable queuing discipline from the scheduling algorithm bank as required. PCU can be reconfigured via the tc interface.

The new queuing discipline and packet classifier must be implemented according to the interface as specified in *include/linux/pkt_sched.h* and *include/linux/pkt_cls.h*. In addition, it also requires modifications to the user-space tc program. A new parsing file is written which details how the parameters are parsed from the tc interface. The file is named as *q_{new_queuing_discipline_name}.c*.

In addition, the new scheduling algorithm is implemented in a file called *sch_{new_queuing_disciplin_name}.c*. Each new instance of queuing discipline must provide the following set of functions to control its operation (see *struct Qdisc_ops in include/net/pkt_sced.h*):

- enqueue - enqueues a packet with the queuing discipline.
- dequeue – returns the next packet eligible for sending.
- requeue – puts a packet back into the queue after dequeuing it with dequeue. This differs from enqueue in that the packet should be queued at exactly the place from which it was removed.
- drop – drops one packet from the queue.
- init – initialises and configures the queuing discipline.
- change – change the configuration of a queuing discipline.
- reset – returns the queuing discipline to its initial state. All queues are cleared, timers are stopped, etc.
- destroy – removes a queuing discipline.
- dump – return diagnostic data used for maintenance.

The Inter-Frame Gap (IFG) scheduling algorithm is implemented to demonstrate the extensibility of the VQML architecture. The algorithm basically works by controlling the sending rate of each outgoing interface. It essentially controls and allocates the duration of the inter-frame gap depending on the priority of the traffic. Higher priority traffic is assigned a smaller gap.

4 Experiments

In this section, we detail the experiments carried out in our networking lab to test the performance of the VQML implementation described in the previous section. The objective of these experiments is to demonstrate the effectiveness of VQML in providing QoS to multimedia traffic over WLANs. More specifically, we carry out our tests with a streaming video to assess VQML's performance in video-on-demand scenario.

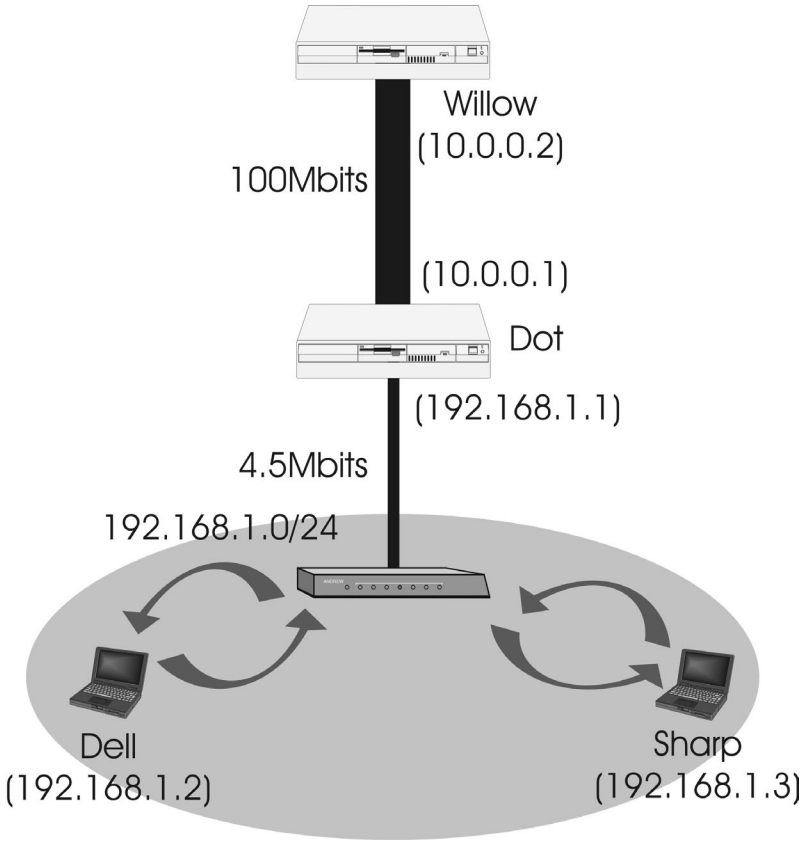

Figure 5: Wireless Test Bed Configuration

4.1 Experimental Test-Bed

Figure 5 shows the wireless network test bed used for all experiments. There are four PCs, Dell, Sharp, Dot and Willow. Dot is the wireless subnet gateway; it has a 100Mbps Ethernet connection to Willow and an Ethernet connection to the wireless access point. Dell and Sharp are mobile laptops with wireless cards[1]. All machines are installed with Linux 2.4 kernels. VQML is installed on both Dell and Sharp and thus allowing the outgoing traffic to be shaped and policed. Dot and Willow are Pentium III 600Mhz and 450Mhz respectively with at least 128MB of Ram. Dell is a Pentium III 800Mhz laptop with 256MB of RAM. Sharp is a Pentium III 600Mhz laptop with 128MB of RAM.

[1] Wireless cards are Lucent Orinoco 11Mbit Wavelan Silver

Table 1: Properties of the experimental video clip.

File Properties
File name: crush.mpg
File size: 61490kbytes
Format: MPEG2
Video Properties
Frame width: 352 Pixels
Frame height: 240 Pixels
Encoded frame rate: 29.97 Frames/s
Bitrate: 2Mbit/sec
Audio Properties
Audio bit rate: 224 kbit/s
Audio coding: MPEG Audio Layer 2
Sampling Freq: 44.1 kHz

Table 1 shows important properties of the video clip used for the video streaming application. The video player used to conduct the video streaming experiments is called MPlayer. It is a movie player for Linux. It plays most MPEG, VOB, AVI, VIVO, ASF/WMV, QT/MOV, FLI, NuppelVideo, yuv4mpeg, FILM, RoQ, OGG and some RealMedia files, supported by many native, XAnim, and Win32 DLL codecs. It supports a wide range of output drivers. It works with X11, Xv, DGA, OpenGL, SVGAlib, fbdev, AAlib, SDL, VESA, and some lowlevel card-specific drivers (for Matrox, 3Dfx and Radeon).

Due the limited number of video streaming servers available on Linux (in particular, there is no MPEG2 open source video streaming server freely available at the time of this project), all experiments were conducted by streaming the video clip to the player via web server. Apache was used for the purpose of this demonstration. Its primary job is to transmit the file (video clip) across the network upon receiving the GET request from the client. Thus, the web server is unaware of the encoding of the video file.

To give higher priority to video traffic, we have prioritised all traffic which goes through the http port (i.e. port 80). Best effort traffic is generated using Netperf via different ports (other than port 80). TCP was primarily used as the transport protocol in all our experiments. RTP/RTCP which normally runs over UDP was not used for this demonstration. In addition, the video clip was encoded prior to the streaming as the web server doesn't encode the video stream in real time.

4.3 Experiment 1: Video without VQML

The aim of this experiment is to demonstrate that the required bandwidth for the privileged flow cannot be maintained in the standard WLANs without VQML.

The video stream is established from Dell to Willow via gateway Dot. Whilst video is running, best effort traffic is also generated from Sharp to Willow. Since the external link has a greater capacity than the wireless medium, any degradation in the video quality would be strictly due to the wireless medium being congested and not the external link. In this experiment, VQML is not implemented, and hence we do not adjust inter-frame gaps of video and best effort flows.

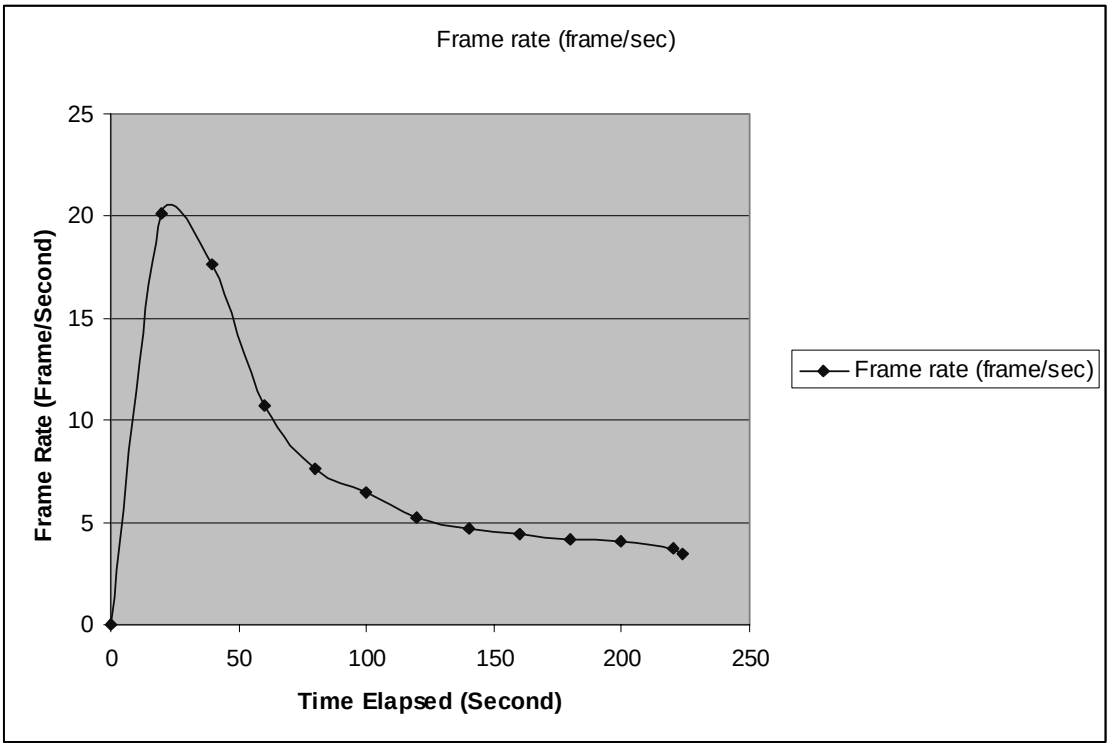

Figure 6: The video playback frame rate as time elapsed (without VQML)

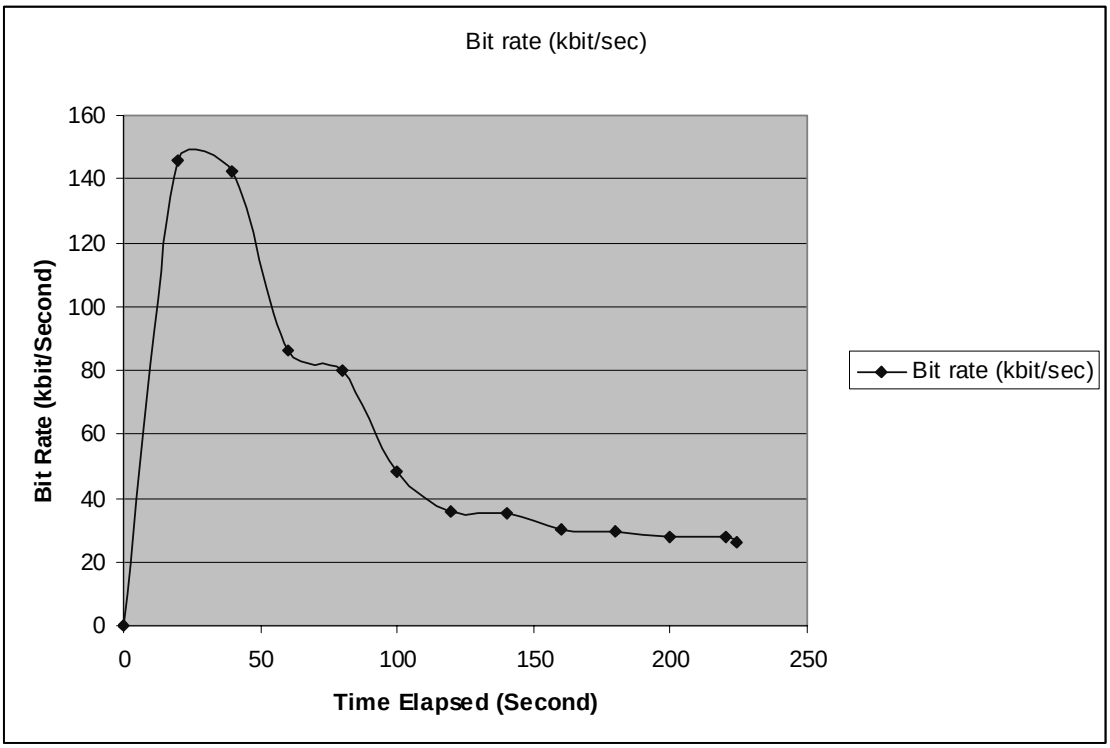

Figure 7: The audio playback bit rate as time elapsed (without VQML)

First we examined the perceived QoS of the video. We focused on picture detail, video smoothness, and audio quality. We found that video and audio streams were out of lip-sync and the video was quite "jerky". Second, we recorded video playback frame rates and audio playback bit rates as time elapsed. Video frame rates and audio bit rates are shown in Figures 6 and 7, respectively. As we can see, video frame rates and audio bit rates drop drastically as file transfer traffic contends for wireless bandwidth.

4.4 Experiment 2: Video with VQML

The objective of this experiment is to demonstrate that the required bandwidth for the video can be maintained in the wireless medium with the use of the VQML protocol. We have the same experimental set-up as in the previous experiment, except VQML selects a smaller (50% less) inter-frame gap for the video traffic.

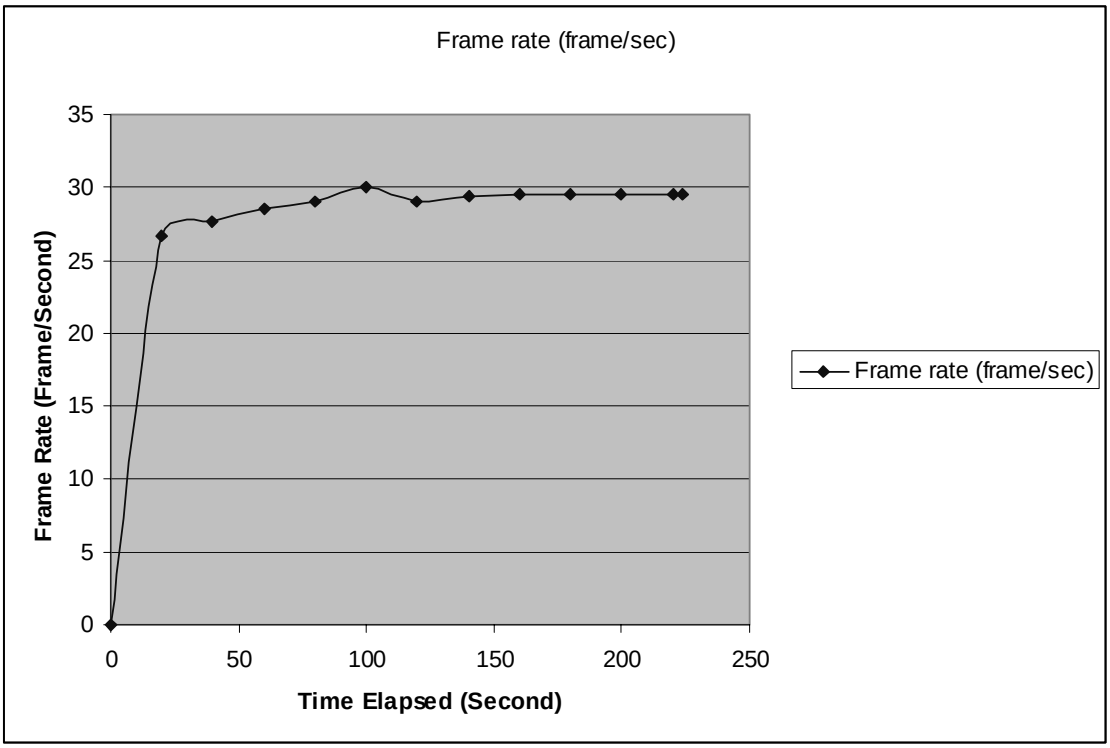

Figure 8: The video playback frame rate as time elapsed (with VQML)

This time, the perceived quality of video was excellent. The video and audio streams were lip synchronised and there were no "jerkiness" during playback. Figures 8 and 9 shows the video frame rates and audio bit rates. As can be seen, we were able to maintain the original frame and bit rates at playback.

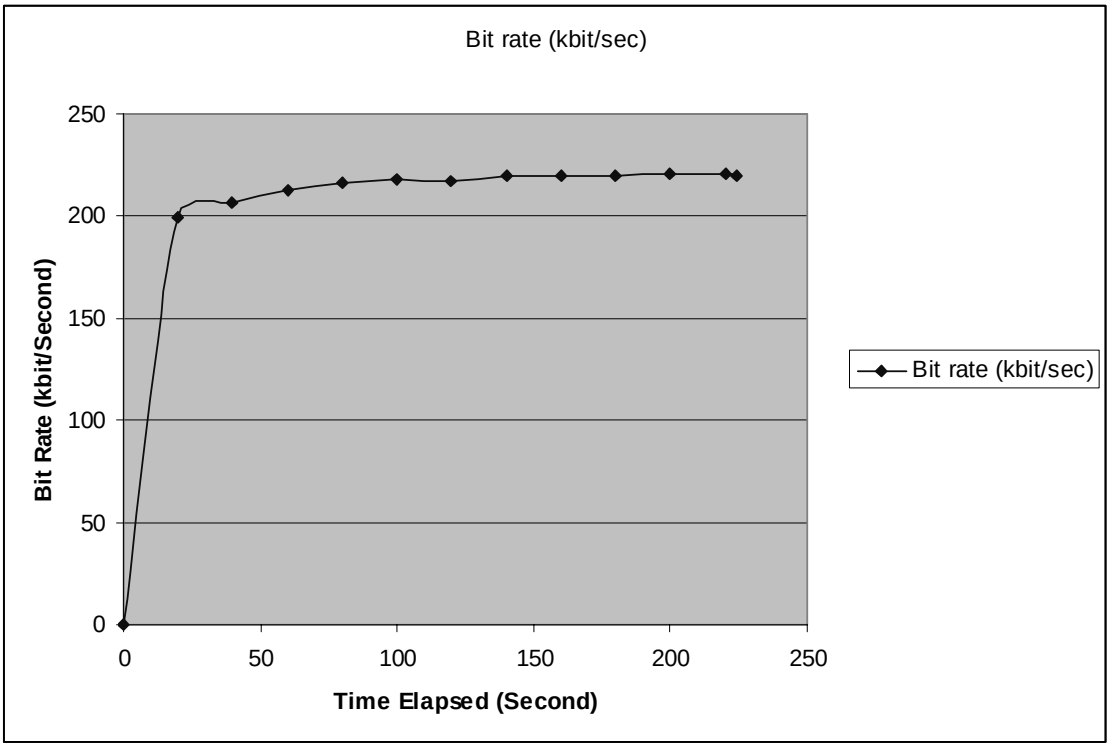

Figure 9: The audio playback bit rate as time elapsed (with VQML)

5 Related Work

Veres et al. [1] proposed two distributed estimation algorithms to provide service differentiation for delay sensitive and best-effort traffic. The Virtual MAC (VMAC) algorithm works by passively monitoring the radio channel and estimates locally achievable service levels. The Virtual MAC estimates key MAC level statistics related to service quality such as delay, delay variation, packet collision and packet loss. The Virtual Source (VS) algorithm, on the other hand, utilises the Virtual MAC to estimate application level service quality. The Virtual Source allows application parameters to be tuned in response to dynamic channel conditions based on "virtual delay curves". We provide a comparison of VMAC with VQML:

- VQML is a software-based intermediate layer between MAC and networking layer to provide QoS support. This contrasts to VMAC which essentially modifies the MAC algorithm in order to form a fully distributed wireless differentiated services network.
- Distributed virtual algorithms require a distributed, differentiated service capable MAC which implies that it requires modification to the wireless card's device driver. This contrasts to VQML, which does not require any modification to the device drivers, and hence VQML provides a cross-platform protocol architecture which interoperates with any wireless MAC protocol.
- VQML provides a seamless interface for communication and hides the details of MAC protocol. Thus, as we port the architecture on top of different MAC protocols, only two units require changes. They are, namely, the UPI and DDI. This contrasts to VMAC which mandates significant changes to the device driver in order to adapt to different wireless MAC protocols.
- Because VMAC was implemented within the device driver of the wireless device, it can not be seamlessly ported to other wireless devices. Thus, the algorithm must be completely re-written for each wireless device. VQML, on the other hand, is less restricted in this regard as it is not aware of

the underlying MAC protocol. Hence, it provides a seamless interface for communication and can easily be ported to different wireless devices.

- There is an advantage, however, of implementing the algorithm in the MAC layer. VMAC is capable of capturing all "overheard" layer two transmissions (e.g., CTS, RTS, ACK packets, even with CRC errors). Hence, it is capable of correlating these data into useful traffic statistics and thus providing service differentiation for delay sensitive and best-effort traffic. VQML, on the other hand, lacks these functionalities. It cannot estimate the availability and the utilisation of network resources since it does not collect traffic statistics from the network.

6 Conclusion

We have designed, implemented, and tested a virtual MAC layer, called VQML, to provide MAC independent QoS to multimedia applications over wireless LANs. The tests were carried out for video streaming applications. Our results show that VQML can achieve good QoS for the video application in the presence of background file transfer traffic.

Acknowledgement

The project was funded by a UNSW University Research Support Program (URSP) grant, 2001.

References

[1] A. Veres, A. Campbell, M. Barry, L.H. Sun, "Supporting Service Differentiation in Wireless Packet Networks using Distributed Control," IEEE JSAC, Vol. 19, No. 10, 2001.