Dynamic Fair Queuing (DFQ): A Novel Fair Scheduler Improving Wireless Transmission over Hybrid LANs

Ce-Kuen Shieh, Yu-Ben Miao, Ming-Qi Shieh, Wen-Shyang Hwang
Institute of Electrical Engineering, National Cheng Kung University, Taiwan, R.O.C.
{ckshieh, ybmiau, mingchi }@hpds.ee.ncku.edu.tw, wshwang@cc.kuas.edu.tw

Abstract

Local area network (LAN) will be a hybrid network that includes wired and wireless links together. Nonetheless, the wired hosts always take the most bandwidth and bring about the bandwidth allocation unfairness. This problem is caused by the flow control mechanism that is dominated by the round-trip time (RTT). Among the connections on a share link, the ones with shorter RTT tend to acquire bandwidth easier and exhaust the link bandwidth eventually. The RTT of wireless transmission is much longer than wired transmission. Therefore, the protocol that implements flow control mechanism such as TCP suffers a severe low performance problem on wireless links.

This paper proposed a new scheme called DFQ that works on gateway. It prevents the wired hosts from grabbing the bandwidth of share link too much and averts the unfairness or starvation of wireless connections. The DFQ has been implemented and test on Linux OS. The experiment result shows that DFQ is practicable and makes wireless network to work with wired network seamlessly.

1. Introduction

The revolution of Internet is going faster and faster. The core network transmission materials are changing from copper wire to optical fiber. Many Internet Service Providers (ISP) already offer Fiber-to-The-Home (FTTH) service. Meanwhile, a variety of mobile computers such as mobile phones, PDAs and notebooks equipped with wireless communication devices have become more popular. New transmission techniques such as Bluetooth, GSM and satellite are joining into Internet. There will be different transmission media (wired and wireless) conjunct together in the future network. To communicate with existing network devices, these innovative devices have to follow legacy network protocols. However, those protocols were developed before the invention of new network transmission techniques. New problems arise from the integration of these devices [1][2][12][13].

One of these problems is the unfair bandwidth allocation for wired and wireless connections that adopt the protocols with flow control supported such as TCP in a hybrid network. TCP treats packet loss as network congestion based on the assumption that the wired links are reliable. However this is not the case in wireless communication. The errant reaction for packet loss in the circumstance of high bit error rate makes the suffering of wireless TCP communication performance. A number of approaches have been invented to solve this problem [3][4][5][14]. These approaches focus on dealing with packet loss and enhanceing performance of TCP connections over heterogeneous networks.

Nevertheless, it exists another problem that brings about the unfairness. The round trip time (RTT) also affects the flow control mechanism of TCP. For several connections among a link, the shorter the RTT of a connection is, the more bandwidth it tends to obtain form that link. Generally, RTT of wireless links are ten or more times greater than that of wired links, hence the wired connections will prevent the wireless connections from acquiring bandwidth. This paper proposed a mechanism named Dynamic Fair Queuing (DFQ) to solve this problem.

The DFQ is designed as a resource manager that running on the gateway of a LAN. Unlike general fairness schedulers, it doesn't allocate distinct bandwidth for every flow. Instead, it classifies network devices by RTT and prevents the starvation of bandwidth for devices which have long RTTs. By doing this, DFQ will adaptively manage resource fairly and efficiently. This paper considers that a hybrid local area network (LAN) includes wired and wireless links, so several wired and wireless devices are tested together. DFQ can identify wired and wireless devices and adjust bandwidth allocation dynamically to improve wireless transmission.

This paper is organized as follows: section 2 gives a briefing of unfairness problems among hybrid networks. And the proposed solution DFQ is depicted in section 3.

0-7695-1961-X/03 $17.00 (c) 2003 IEEE

The implementation details of DFQ are explained in section 4 and section 5 illustrates the experiment results of DFQ. Finally, a conclusion remark is given in section 6.

2. Background Overview

Typically, a hybrid LAN includes wired and wireless links. Ethernet and IEEE 802.11 represent two general standards respectively. The IEEE 802.11 defines the data-link layer protocol for data communication equipments that interconnect via the "air", like radio or infrared, in a local area network. The frequency of IEEE 802.11 is located in 900MHz and 2.4GHz called ISM (*Industrial, Scientific and Medical*) band. The ISM band license is free; every one can use this frequency band under regulated power. In other words, the ISM band gets interference easily, so some strategies [10][11] were suggested to alleviate this problem. Still, high error bit rate and long RTT are two drawbacks of wireless links. These drawbacks mislead the flow control mechanism of TCP and cause the unfair bandwidth utilization.

TCP is a protocol that has been operating on Internet for a long time. It is one kind of connection-oriented and provides reliable connections. Basically, the TCP sender and the TCP receiver handshake during the data transmission time to guarantee the correctness of packet delivery. In order to control the flow rate, TCP brings in sliding windows concept, and contents two windows implements. One is the *sliding window* which is dominated by receiver. It keeps tracking the sequence number of transmitted TCP packets and adjusts its size as receiver's desire. This window size specifies the maximum amount of data that sender can send out at a time before receiving the acknowledgement (the ACK packet) from receiver. Ideally, this size is equal to effective bandwidth multiplied by RTT.

The other window is the *congestion window (cwnd)* which is dominated by sender. It specifies the number of packets that sender can send out at one time interval to control the sending rate of packets and to evade the network congestion. There are two phases of this window: slow start and congestion avoidance. When a new TCP connection starts, the congestion window size is initialized to be one packet. The congestion window will be increased by one for every acknowledged packet. This phase is called "slow start" in which congestion window is increased in exponential.

Eventually, the congestion window will reach the limitation of network bandwidth, and then intervening routers have to drip packets. To deal with packets loss, a slow start threshold size (*ssthresh*) is set. When *cwnd* is greater than this threshold, TCP changes from slow start to congestion avoidance phase. In this phase, *cwnd* is incremented by 1/*cwnd* for each acknowledged packet. The growth curve of *cwnd* will become smoother to avoid high packet drop rate [6].

If there is any packet loss during the transmission, TCP will assume that the packet is dropped because of network congestion. In response, sender will decrease its *cwnd* by half and reduces packet-sending rate. If *cwnd* drops down below *ssthresh*, it changes to slow start mode again. Note that the *ssthresh* may also be dynamically changed on the subject of RTT.

Given that the *cwnd* grows as sender receive ACK from receiver, RTT has an effect on the growing speed of *cwnd*. Figure 1 shows the influence of RTT on *cwnd*. The long RTT is 3 times of the short RTT. Both *ssthresh* are set to be 32.

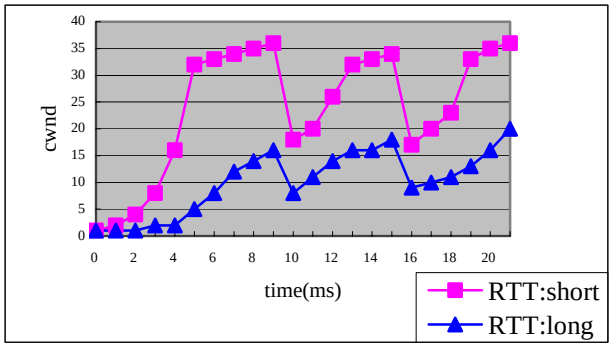

Figure 1. The effect of RTT to TCP cwnd

For the short RTT connection, it increases its *cwnd* more rapidly than long RTT connection does. Therefore, short RTT connections always exhaust bandwidth rapidly and lead unfairness to long RTT connections. As a hybrid network is considered, it could be worse than ever. Typically, the RTT in wireless link is ten times than that in wired link (Wired RTT 300 μs, Wireless RTT 3*m*s). It goes without saying that wireless connections will suffer bandwidth starvation when competing with wired connections.

3. DFQ scheduling scheme

To provide a better quality of service for wireless connection, it is necessary to interpose between wired and wireless connections. In this thesis, we propose a new fair scheduler which called Dynamic Fair Queuing (DFQ) which works on the gateway to solve the unfairness problem over hybrid network.

Figure 2 illustrates where the DFQ installed in the gateway.

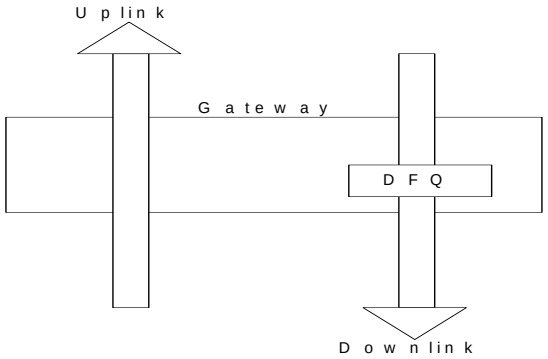

Figure 2. The place that DFQ installed in gateway

The DFQ is installed on the downlink of the gateway. It is because that most end-users play as clients in Internet and the downlink traffic is always more significant than uplink traffic. The main goal of DFQ is to allocate a reasonable bandwidth for each host. Figure 3 shows the model of DFQ.

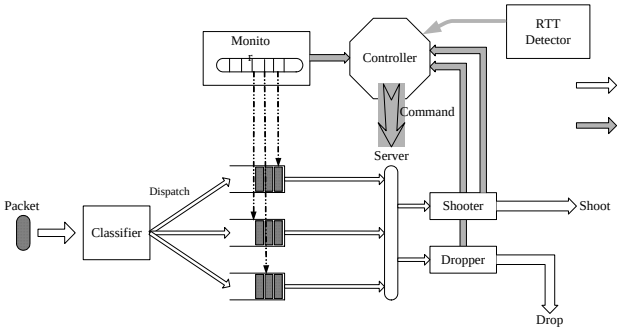

Figure 3. DFQ architecture

These components are explained as follows:

Classifier: The Classifier classifies each packet and dispatches it to a proper queue according to the destination address of packet.

Monitor: The Monitor statistics bandwidth usage of each queue and reports to the Controller periodically.

Controller: Controller is the soul of DFQ. It commands Server to shoot or drop packets. The controller makes decision according to the information that other components report.

Server: The Server decides the service order of queues and shoots or drops packets according to the Controller's command.

RTT detector: The function of RTT detector is to identify the host type that is wired or wireless device. It sends broadcast ping as a probe periodically. Every host in this LAN will reply with a ping acknowledgment. RTT detector can distinguish between wired and wireless links by the collected RTT.

Shooter: It injects packets into the network and reports the length of packets to the Controller.

Dropper: The Dropper drops packets and reports to the Controller.

There are two flows in figure 3. One is packet flow that shows packet path in DFQ. There are four stages, 1.classifier 2.queue 3.sever 4.dropper or shooter, which every packet will come over in DFQ. The other flow is control flow that shows relationship between each component.

When a packet arrived, the Classifier dispatches the packet to the proper queue. The Monitor will watch arrival rate of each queue and report the queue information to the Controller. Every queue is linked like a circle. The Server takes packet from each queue by means of a round robin principle. The Controller will make a decision and tell Server to shoot or drop the packet. The Dropper and Shooter inform the Controller if they drop or shoot any packet. The Controller gathers all necessary information from other components and makes decision by these data.

The algorithm for the Controller to make decision is quite simple. For each packet of wired connection, the controller drops it with the probability P_d.

$$P_d = 1 - \frac{B_{wireless}}{B_{wired}} \times \frac{RTT_{wireless}}{RTT_{wired}}$$

Where $B_{wireless}$ and B_{wired} represent the total allocated bandwidth for wireless and wired connection correspondingly. They are observed and updated by the Monitor at regular intervals. The $RTT_{wireless}$ and RTT_{wired} are the round trip time discovered by the RTT detector. We assume that the receiving buffers are the same for wired and wireless devices. Hence DFQ considers the bandwidth inversely proportional to RTT.

4. Implementation of DFQ

This paper considers the implementation of DFQ on Linux. Linux is an open source operating system thus suites for experimental testing. Figure 4 explains the network architecture in Linux.

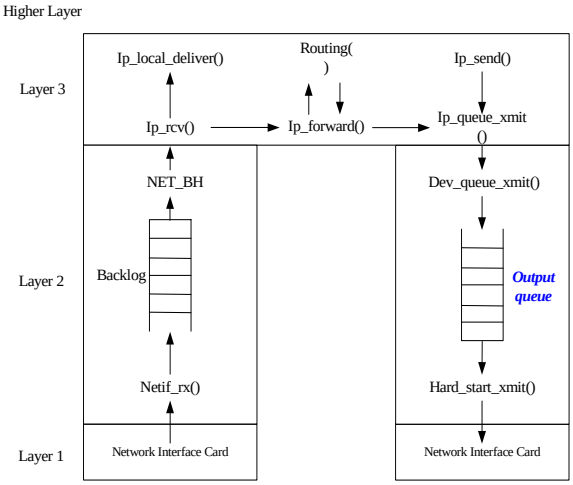

Figure 4. Linux network architecture

When the network interface card receives a packet, it triggers a hardware interrupt and OS invokes the interrupt service routine that will enqueue the packet to backlog queue in Linux kernel. The Linux kernel will check the backlog queue periodically. If there are any packets in the backlog queue, they will be brought to upper layers. In the case of an IP packet, the kernel will pass it to *Ip_rcv* routine. The *Ip_rcv* routine determines the packet should be sent to upper layer or to *Ip_forward* routine according to this packet is for local host or for other host, respectively. The *Ip_forward* routine looks up the routing table and determines the output network interface that this packet should go and pass it to *Ip_queue_xmit* routine. The *Ip_queue_xmit* builds IP header for the packet and transfer it to *Dev_queue_xmit* routine. The *Dev_queue_xmit* routine is one of Linux core network procedure. In this procedure, the Linux kernel offers an interface to add different schedulers called queuing discipline. The default output queue, FIFO queue, can be replaced by other user-specified queuing discipline. Finally, this packet is sent to the network device queue and the network interface card will deliver it to the next hop.

Mainly, DFQ invokes *Dev_queue_xmit* routine to add its redefined queuing discipline. In Linux kernel, there are some queuing discipline residing in the directory *net/sched/*. The Linux queuing discipline is a nested structure. There are three kinds of traffic control elements that form building blocks:

- Queuing discipline
- Classes
- Filter / Policer

The default queuing discipline is FIFO queue in Linux.

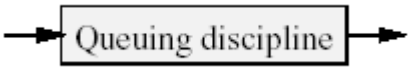

Figure 5. FIFO queuing discipline

If queuing discipline needs to carry out more complex functions, extra classifiers can be added and attached with another queuing discipline. Every building block can contain other building blocks. Figure 6 shows a more complex queuing discipline.

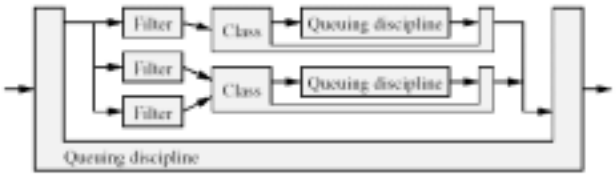

Figure 6. A queuing discipline contain classifier and other queuing discipline

The queuing discipline operation and classifier structure are defined in *include/net/pkt_sched.h*. The queuing discipline should provide the following set of functions to control operation:

enqueue、*dequeue*、*requeue*、*drop*、*init*、*change*、*reset*、*destroy*、*dump*

The classifier should provide the following set of functions to control operation:

graft、*leaf*、*get*、*put*、*change*、*delete*、*walk;*

The detail definitions of these operations are described in [7].

To add a new queuing discipline into Linux kernel, the user-specified functions for queuing discipline and classifier should be declared in a *struct Qdisc_ops*. Routin *register_qdisc* is invoked to register the *struct Qdisc_ops* and the *unregister_qdisc* is for removing a queuing discipline from the kernel [8].

To operate the DFQ, we also need a program named *tc*. The *tc* program can manipulate (add, del or change etc…) the kernel traffic control elements. It is one part of iproute2 project [9].

5. Experiment Results

5.1. Experimental Platform

Figure 7 displays our experiment platform. The *Mouse* is the gateway where DFQ is implemented in. There are 2 wired hosts *Dragon* and *Rabbit* and 3 wireless hosts *MS1*, *MS2* and *MS3* that form a hybrid network. These hosts act as clients and the rest of 3 hosts *Bull*, *Snake* and *Tiger* act as the servers.

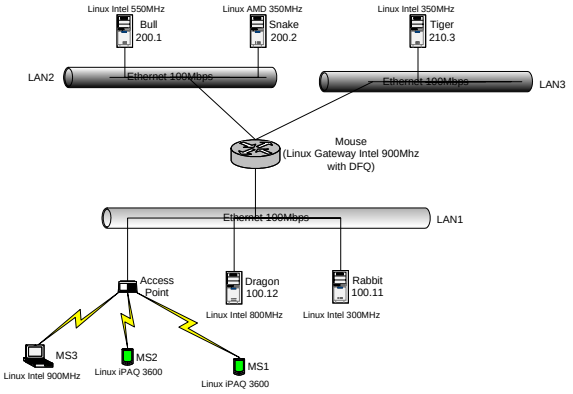

Figure 7. The experiment platform

The platform contains three 100MBps LANs that are connected by a gateway running Linux OS with kernel 2.4.17. The wireless devices are IEEE 802.11b network interface cards with 11Mbps maximum data transmission rate and 7.5Mbps efficient data transmission rate. There are two kinds of mobile stations: one is laptop pc (MS3); the other two are iPAQ PDAs (MS1 and MS2). We use the *pttcp* tool to track the bandwidth allocation. The *pttcp* records the bandwidth usage of every connection periodically and the time interval we set here is 5 seconds.

5.2. Effectiveness of DFQ

In order to verify the effectiveness of DFQ, we measured the throughput of wireless connection while the wired connection acts.

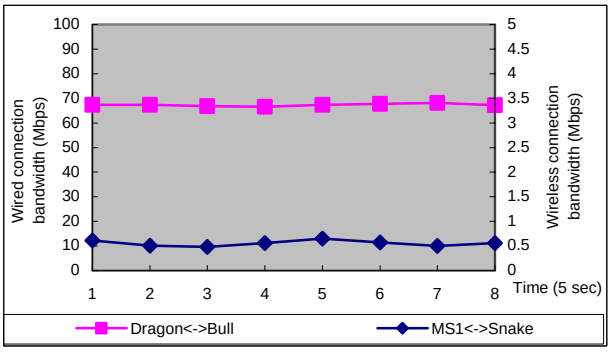

Figure 8. 1Wired connection & 1Wireless connection (Without DFQ)

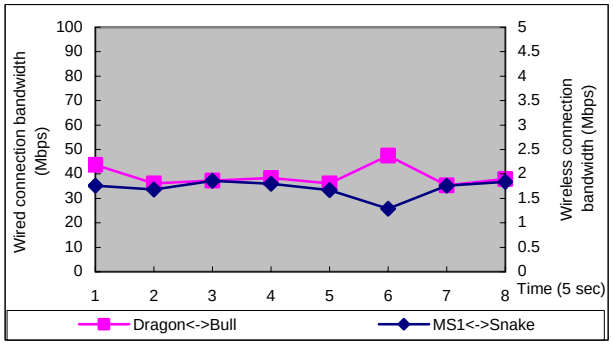

Figure 9. 1 Wired connection & 1Wireless connection (With DFQ)

Firstly, we test one wired connection and one wireless connection which are *Bull* to *Dragon* and *Snake* to *MS1*, correspondingly. As figure 8 shows, the wired host grabs most of the bandwidth. The wireless connection just gets about 0.55Mbps bandwidth even though its maximum bandwidth of TCP flow is about 4Mbps theoretically. Figure 9 demonstrates the effectiveness of DFQ. The wired bandwidth is decreased to 40Mbps and the wireless bandwidth is pulled up to 1.8Mbps. The bandwidth of wireless connection has about 300% improvement while the wired connection can still maintain 40% bandwidth allocation.

Secondly, we test two wired connections and two wireless connections. We placed these two servers (*Snake* and *Tiger*) at different LAN segments to verify the fairness with different sources.

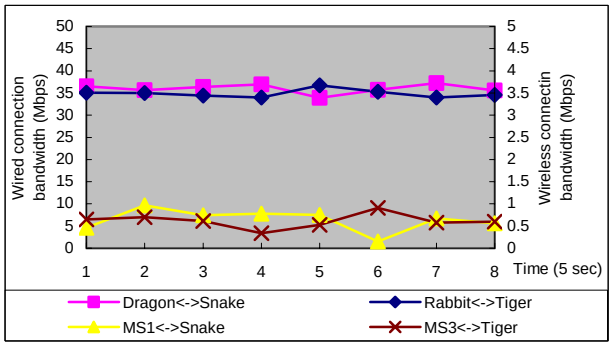

Figure 10. 2 Wired connections & 2Wireless connections (Without DFQ)

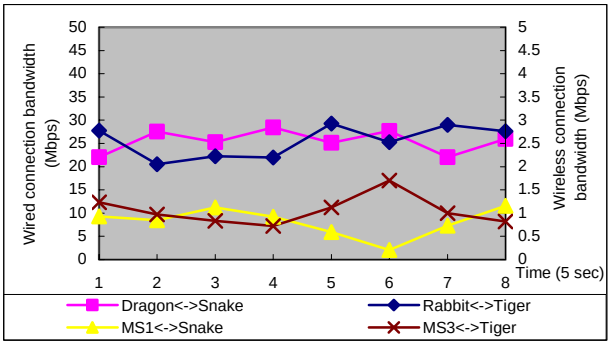

Figure 11. 2 Wired connections & 2Wireless connections (With DFQ)

Figure 10 indicates that wired connection obtains most bandwidth as well. Figure 11 illustrates that DFQ successfully improves the wireless transmission and does not induce any unfairness. Note that at the time 6, we close the connection of MS 1 and start a new TCP session for it. The congestion window of MS 1 will be set to one packet at that time. By doing this, we can observe the change of bandwidth allocation between wired and wireless connections. The results show that the total bandwidth of wireless connections still retains at a certain level. Due to the lack of space, the scenario of multiple wired and wireless connections is not shown here. Their experimental results are consistent with experiment 2. The total bandwidth of wired connections is about 40Mbps while the total bandwidth of wireless connections is about 2Mbps. The fairness among connections is sustained in these cases.

5.3. Trade-off of DFQ

The penalty of DFQ protecting wireless connection bandwidth is the degradation of total link bandwidth utilization. However, consider a scenario that a wireless connection is trying to deliver a 1 Mbytes mail when wired connection is transferring a 100 Mbytes file. The ordinary transmission time is 11.4 and 12.5 seconds for wired and wireless connections, respectively. While with DFQ, the transmission time becomes 13.1 and 4.4 seconds, as the link bandwidth utilization decrease from 65% to 62%. The transmission time of wireless connection is reduced significantly whereas the influence of wired connection and link bandwidth utilization is minor. It is especially true when most wireless connections deliver small bulks of data.

6. Conclusion and Future Work

This paper proposed a new scheme called DFQ that improves the wireless communication. It is implemented on gateway to protect wireless connections from wired connections and solve the unfairness problem between them. The experiment results show that DFQ is feasible and make the IEEE 802.11 network working with Ethernet seamless. In the future, the DFQ will be extended to collaborate with other proposed mechanisms [3][4][5][14] over heterogeneous network. Meanwhile, the authors are studying in offering a fair bandwidth allocation for wireless connection with more efficient link bandwidth. Different bandwidth allocation strategies can be applied and are still an open issue.

Acknowledgement: This project is sponsored by NSC-91-2213-E-006-054.

Reference

[1]. Fu-Ming Tsou, Hong-Bin Chiou, Zsehong Tasi, *"WDFQ : An Efficient Traffic Scheduler with Fair Bandwidth Sharing for Wireless Multimedia Services,"* IEICE TRANS. Communications JANUARY 2000

[2]. Songwu Lu, Vaduvur Bharghavan, R. Srikant, *"Fair Scheduling in Wireless Packet Networks,"* IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 7 NO. 4, Auguest 1999

[3]. Jian-Hao and Kwan L. Yeung,*"FDA: A Novel Base Station Flow Control Scheme for TCP over Heterogeneous Networkings,"* IEEE INFOCOM 2001

[4]. H. Balakrishnan, S. Seshan, and R.H. Katz, *"Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks."* ACM Wireless Networks, Vol.1(4), December 1995

[5]. E. Ayanoglu, S. Paul, et.al, *"A Link-Layer Protocol for Wireless Networks,"* ACM/Baltzer Wireless Networks Journal, Vol.1, Page 47-60, February 1995

[6]. V. Jacobson, *"Congestion Avoidance and Control."* In Proc. ACM SIGCOMM 88, August 1988

[7]. Werner Almesberger, *"Linux Network Traffic Control -Implementation Overview,"* April 23, 1999

[8]. Saravanan Radhakrishnan, *"Linux-Advanced Networking Overview Version1,"* August 22, 1999

[9]. Bert Hubert, Gregory Maxwell, *"Linux Advanced Routing & Traffic Control Howto,"* December 17, 2001

[10]. Christine Fragouli, Vijay Sivaraman, Mani B. Srivastava, *"Controlled Multimedia Wireless Link Sharing via Enhanced Class-Based Queuing with Channel-State-Dependent Packet Scheduling,"* IEEE 1998.

[11]. Pravin Bhagwat, Partha Bhattacharya, Arvind Krishna and Satish K. Tripathi, *"Using channel state dependent packet scheduling to improve TCP throughput over wireless LANs,"* Wireless Networks 3 (1997) Page 91-102

[12]. Z. D. Shelby et al, *"Wireless IPv6 Networking – WINE,"* IST Mobile communicate, Summit 2000, Galway, Ireland, Oct. 2000,Page 575-80.

[13]. WINE Project http://www.vtt.fi/ele/projects/wine

[14]. Claudio Casetti, Saverio Mascolo, *"TCP Westwood: Congestion Window Control Using Bandwidth Estimation,"* IEEE 2001.