WIRELESS FAIR QUEUING ALGORITHM FOR WINDOW-BASED LINK LEVEL RETRANSMISSION

Abdul-Rahman Elshafei and Uthman Baroudi

Computer Engineering Department
King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia
{shafei,ubaroudi}@kfupm.edu.sa

ABSTRACT

Wireless networks have unreliable channels that experience bursty and location-dependent errors. Several fair queuing algorithms have been proposed in order to provide QoS in presence of errors in a fair manner. However, most of these algorithms are unpractical as they require perfect channel predication or do not work well with the Link Layer. Wireless Fair Queuing with Retransmission (WFQ-R) algorithm was recently suggested to address these problems by penalizing flows that use wireless resources without permission in the link layer. However, the WFQ-R algorithm is based on Stop-and-Wait LLR scheme which also costs the network extensive delay and low utilization. In this paper, a new wireless fair queuing based on the WFQ-R algorithm is proposed to work with the window-based error control schemes in the link layer. The proposed algorithm has shown outstanding results compared with WFQ-R in terms of lower queuing delay, better throughput and fairly allocated resources.

Index Terms— *Wireless Fair Queuing; Wireless QoS; Wireless Packet Scheduling; Link Level Retransmission*

1. INTRODUCTION

The rapid-growth of wireless networks in recent years has sparked interest in providing the required quality of service for a wide variety of applications such as VoIP, video conferencing, WWW, ftp, telnet, etc.. Wired-networks fair queuing algorithms cannot be applied on wireless networks because wireless networks have unreliable channels that experience bursty and location-dependent errors. In order to provide quality of service in wireless networks by overcoming these type of errors, wireless fair queuing algorithms have been proposed, such as Idealized Wireless Fair-Queuing (IWFQ) [1], Server Based Fairness Approach (SBFA) [2], Channel-condition Independent Fair Queuing (CIF-Q) [3]. However these algorithms are unpractical for several reasons. First, they require perfect channel prediction. Second, these algorithms assume unrealistically ideal conditions. Third, they do not address the relation with the link layer.

Since most wireless networks adopt the Link Level Retransmission (LLR) algorithm within the link layer, Kim and Yoon suggested [4] a new wireless fair queuing algorithm that is well matched with LLR Algorithm and does not require channel prediction. The new Wireless Fair Queuing with Retransmission (WFQ-R) algorithm is able to achieve flow separation, flow compensations and maintains fairness adaptively in presence of channel errors. However, the WFQ-R algorithm only works on Stop-and-Wait error control schemes in the link layer. The WFQ-R therefore may not be practical since several link layer strategies are based on window error controls such as the Go-Back-n and Selective Repeat ARQ. Moreover, Stop-and-Wait error control scheme costs the network an extensive delay and low utilization Therefore, in this paper; we propose a new wireless fair queuing algorithm based on the WFQ-R that works with any windows-based scheme.

The remainder of the paper is organized as follows. Section2 introduces the background information on WFQ-R concepts. Section3 presents the new wireless fair queuing algorithm with windows-based link level retransmission. Simulation environment and results are presented and discussed in Section 4. Finally, section 5 concludes the paper.

2. WFQ-R ALGORITHM

In wireless networks, there are two types of fairness; data fairness and resource fairness. Data fairness is based on the amount of data received. This type of fairness guarantees that each flow receives the same amount of data. On the other hand, resource fairness provides fairness based on the amount of resources received by each flow. Resource fairness criterion is more suitable when there is a presence of errors because with data fairness an erroneous flow can exhaust almost all wireless resources. The WFQ-R algorithm is based on resource fairness. The algorithm achieves wireless fairness with the LLR algorithm by penalizing flows that use wireless resources without permission in the link layer.

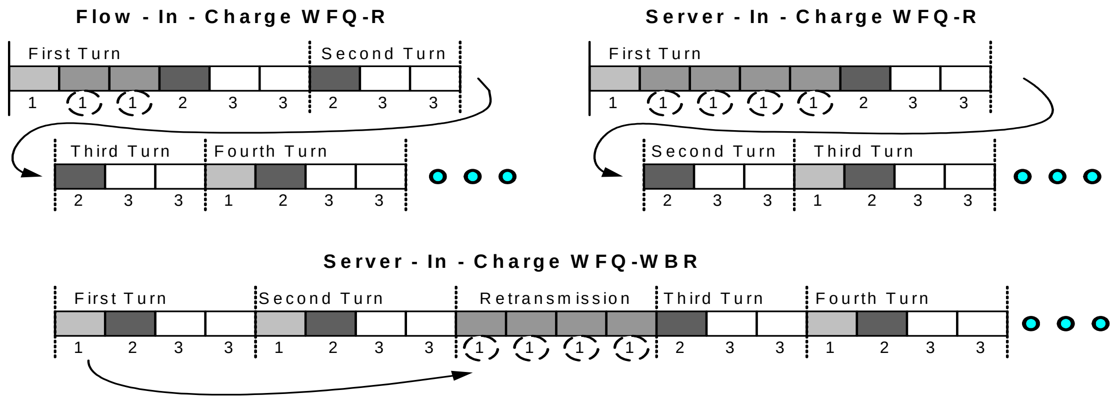

Figure 1. Illustration of FIC and SIC for both approaches. Circled flows are resources used by retransmission

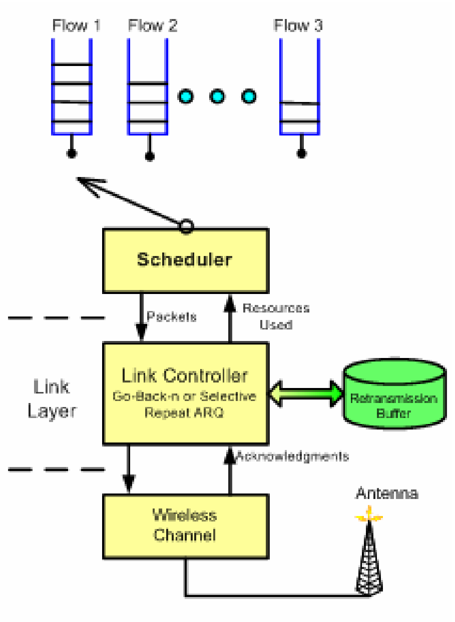

Figure 2. Wireless Channel Architecture

The main concept of the WFQ algorithm is that the resource share used for the retransmission is regarded as a debt of the retransmitted flow to other flows. The concept is also based on adopting the compensation model in which a flow receives a compensative share later when it does not get its share due to channel errors. There are two types of compensation that were proposed; flow-in-charge (FIC) and server-in-charge algorithms (SIC).

The flow-in-charge works by preventing a flow from its turn by every resource it has consumed during retransmission. However, the flow-in-charge provides only strict equity or resource fairness and may not be adequate for a flow experiencing frequent errors. On the other hand, server-in-charge provides fairness to both data and resources combined. It works by charging the overhead to all backlogged flows in a distributive manner. The retransmitted flow has responsibility for only a portion of the overhead to its weight. Consequently, server-in-charge compensation is proven to be more suitable [4] and as such we concentrate in this work on server-in-charge compensation for our new fair queuing algorithm. An example of a SIC and FIC types can illustrated in Figure 1. We can observe that for FIC, Flow 1 lost the chance to send for two consecutive turns. On the other hand, using SIC, the same flow only lost one turn which is equivalent to its weight.

2.1. Problems with WFQ-R

The main problem with the WFQ-R algorithm is that it is not compatible with windows-based LLR schemes such as Go-Back-n and Selective Repeat ARQ. The WFQ-R algorithm depends on the fact that when a packet is sent to the link-layer, the scheduler has to wait until the link layer reports the number of resources the packet used. According to the WFQ-R algorithm [4], the algorithm states that charging the flows due to retransmission must be accomplished as soon as the packet is sent. So the scheduler cannot send any further packets until this process is complete. During the wait time, the packets waiting in the queues are delayed and resources are wasted.

In contrast to the WFQ-R algorithm, the proposed wireless fair queuing with windows-based retransmission (WFQ-WBR), the scheduler can send several successive packets without waiting for resources consumed during transmission from the LLR.

3. WIRELESS FAIR QUEUING ALGORITHM WITH WINDOW-BASED RETRANSMISSION

3.1. Network Model

In this section, we present our Wireless Fair Queuing with Windows-Based Retransmission (WFQ-WBR) algorithm. The architecture of the network model is depicted in Figure 2 where a cell has a base station or an access point with a shared wireless channel. The scheduler server exists in a communication layer above the link layer. It selects a packet for transmission from on the active flows based on the wireless fair queuing algorithm and sends it to the link layer. The scheduler continues to send packets to the link layer from each flow in turn according to their weights until one of the three conditions are met:

1. There no more packets to send from all flows (no active flows)
2. The maximum windows size N has reached for packet transmission in the link layer and thus the scheduler has to wait until an acknowledgement has been received in the link layer for the previous N packets sent.
3. When the link layer receives the amount of resources consumed by a flow during the retransmission of one of the packets.

When the third condition happens, the scheduler calculates the amount of extra resources (*R*) a flow used during retransmission and then calculates the amount of compensation needed to be served to other flows based on their weights as follows. It finds first how much should be charged:

$$Ch\arg e = R \times \left(1 - \frac{wieght(j)}{\sum_{k \in active} weight(k)} \right) \times \left(\frac{wieght(j)}{\sum_{k \in active} weight(k)} \right) \quad (1)$$

Then, it distributes the charge of used resources to all backlogged flows. Charge is done in proportional to its weight.

$$lag\,(A) + = Ch\arg e \times \left(\frac{wieght\,(A)}{\sum_{k \in active} weight\,(k)} \right) \quad (2)$$

Then, the algorithm computes the difference between the amount of service should have received and the services that the flow actually received. If lagging is positive, then the flow received less service. If lagging is negative, then the flow received more service. If lagging is 0, then the flow is in sync.

For example, assume there are three flows and the third flow has twice the weight. The link layer has a selective-repeat ARQ protocol with window size of 8 packets. In the first turn, each flow sends packets according to their weights. Flow 1 experiences errors within transmission during the first turn, but the negative acknowledgment has not yet been received. The scheduler continues to select and send packets normally until a negative acknowledgment is received. When flow 1 has used four extra resources from the retransmission and the weight of the flow is a quarter of the sum of the weights of all flows as in Figure 2, the retransmitted flow is responsible for only one resource. Accordingly, the flow disclaims only one resource in the third turn.

3.2. Algorithm Description

The algorithm is divided into several functions. The full algorithm WFQ-WBR algorithm is shown in Table 1.

We define the following variables for the algorithm:

- **serviceRecieved(i):** normalized amount of service received by a leading flow i
- **lagging(i):** indicates the difference between the amount of service should have received and the services that the flow actually received. If lagging is positive, then the flow received less service. If lagging is negative, then the flow received more service. If lagging is 0, then the flow is in sync.
- **Compensation(i):** the normalized amount of compensation service received by a lagging flow i
- **resourcesRecieved(i):** total amount of resources that a flow consumed
- **weight(i):** rate of flow
- α: fraction of service retained by leading flow
- **packetCounter(i):** the number of outstanding packets (unacknowledged packets < windows size)

A detailed description of each function is as follows:

1. **On Receiving:**
   When a packet is received it is added to queue. If flow i is backlogged, then the resources received for i should at least the equal to the minimum resources received for each other active flow
2. **Deactivate:**
   Removes a flow i from scheduling when there is no more packets. This flow i has lagging(i) that is proportionally distributed among all active flows
3. **On sending:**
   Selects a flow i with minimum virtual time. If flow i is lagging then it is transmitted. If flow i is leading but did not receive the minimum service, then it is transmitted. If flow i is leading and it already received its minimum service, then search for lagging flow j with minimum compensation. If found transmit j otherwise i. If however the number of outstanding packets reached maximum, i.e. the window size, then the packets will not be sent until an acknowledgement is received. The selected packet is sent to sendPAcket function
4. **send Packet():**
   Gets packets from buffer and adjusts lagging for each flow. Then sends the packet through the link-layer. ResourcesRecieved for transmitting flow is incremented by the amount of resources consumed by the packet depending on the ratio of the packet length and the rate of flow. If flow i is leading, then servicesRecieved(i) is also incremented by the packet/rate ratio. On the other hand, if flow i is lagging, then the lagging(i) is decremented by the packet/rate

ratio. The function also saves the values of i and j for each packet p sent, since acknowledgements or negative acknowledgments may be received out of order in a selective repeat request ARQ scheme.

Table 1 WFQ-WBR Algorithm

```
On receiving packet p from flow i:
  If flow i not active
    resourcesRecieved_i ← max(resourcesRecieved_i, min{v_k})
    lagging(i) =0;
    flow i becomes active;
  add p to queue;

On sending current packet:
  find active i with minimum resourcesRecieved
  while(packetCounter(i) < windowSize(i))
      find another active i with minimum resourcesRecieved
      if (new i does not exits)  return;
  if (i is lagging or leading with graceful degradation)
    sendPacket(i,i);
    packetCounter(i)++;
  else
    j ← flow with minimum compensation service
    if(j exists and packetCounter(j) < windowSize(j))
      sendPacket(i,j);
       packetCounter(j)++;
      if (j buffer is empty)  deactivate j
    else sendPacket(i,i); packetCounter(i)++;
  if( queue i empty) deactivate i;
sendPacket (j,i) :
  remove p from queue
  resourcesRecieved(i) += length(p) / weight(i);
  if( i = j)
    lagging(j)  -= length(p);
    if(lagging(j) > 0)  // still lagging
      compensation(j) = length(p) / weight(j);
   if(lagging(j) + length(p) ≥ 0 while j is lagging )
      // Graceful degradation of service
      serviceRecieved(j) ← α*resourcesRecieved(j)
   lagging (i) += length(p);
   if(lagging(i) – length(p) ≤ 0 and i became leading)
         compensation (i) = max(compensation (i), minimum
          compensation of all other active lagging flows)
  send p to MAC layer;
  store values of i , j temp in buffer until link layer responds
```

```
On receiving packet p from flow i:
  If flow i not active
    resourcesRecieved_i ← max(resourcesRecieved_i, min{v_k})
    lagging(i) =0;
    flow i becomes active;
  add p to queue;
On sending current packet:
  find active i with minimum resourcesRecieved
  while(packetCounter(i) < windowSize(i))
      find another active i with minimum resourcesRecieved
      if (new i does not exits)  return;
  if (i is lagging or leading with graceful degradation)
    sendPacket(i,i);
    packetCounter(i)++;
  else
    j ← flow with minimum compensation service
    if(j exists and packetCounter(j) < windowSize(j))
      sendPacket(i,j);
       packetCounter(j)++;
      if (j buffer is empty)  deactivate j
    else sendPacket(i,i); packetCounter(i)++;
  if( queue i empty) deactivate i;
sendPacket (j,i) :
  remove p from queue
  resourcesRecieved(i) += length(p) / weight(i);
  if( i = j)
    lagging(j)  -= length(p);
    if(lagging(j) > 0)  // still lagging
      compensation(j) = length(p) / weight(j);
   if(lagging(j) + length(p) ≥ 0 while j is lagging )
      // Graceful degradation of service
      serviceRecieved(j) ← α*resourcesRecieved(j)
   lagging (i) += length(p);
   if(lagging(i) – length(p) ≤ 0 and i became leading)
         compensation (i) = max(compensation (i), minimum
          compensation of all other active lagging flows)
  send p to MAC layer;
  store values of i , j temp in buffer until link layer responds
```

```
on status usedResources for p from link layer:
retrieve corresponding i and j of packet
packetCounter(j)--;
      for all active flows A
charged = usedResources
  $\times\left(1-\frac{wieght(j)}{\sum_{k\in active} weight(k)}\right)\times\left(\frac{wieght(j)}{\sum_{k\in active} weight(k)}\right)$
  if( i = j and i is leading)
    servicesRecieved += charged/weight(j);
  else
     lagging(j) -= charged;
     if( j is still lagging)
       compensation(j) += charged/weight(j);
     if( lagging (j) + length(p) ≥ 0 and j became leading)
       // Graceful degradation of service
     serviceRecieved(j) ← α*resourcesRecieved(j)
       lagging (A) += charged $\times\left(\frac{wieght(A)}{\sum_{k\in active} weight(k)}\right)$
  //distribute the shares for compensation
       if(A just became lagging)
compensation (A)= max(compensation (A),
         minimum compensation of all other active lagging flows)

deactivating i
  for all active flow f
        lagging(f)+= lagging(i) $\times\left(\frac{wieght(f)}{\sum_{k\in active} weight(k)}\right)$
        if(f has become lagging)
           compensation (f)= max(compensation (f), minimum
            compensation of all other active lagging flows)
```

5. **On status:**

When the link layer responds to the upper layer with the amount of used resources during retransmission. Then retrieve the corresponding flow i and j for the status report received. packetCounter is decremented since it has been acknowledged. Also calculates the overhead in which the retransmitted flow is responsible. Then distribute the charge of used resources to all backlogged flows. Charge is done in proportional to its weight.

4. SIMULATION EXPERIMENTS

4.1. Simulation Environment

In this section, we represent the results of the simulation experiments to demonstrate the fairness of the WFQ-WBR algorithm using the server-in-charge approach. To evaluate the algorithm the following metrics are measured:

1. Allocated resources: the amount of wireless resources consumed by a flow
2. Queuing delay: experienced delay in a queue
3. Received data: the actual amount of data at a receiver

For simulation we used C++ programming to model and simulate the algorithm. For a fair comparison with the WFQ-R simulation results presented in [4], we use the simulation parameters which can be summarized as follows: There are three transmission flows; flow1, flow2, flow3. The probability of retransmission for flow1 is 0, flow2 is 0 and flow3 is 0.2. The packet transmission starts for flow1 at time 0, flow2 at 0.4s and for flow3 at 1.3s. The packet inter-arrival time for each flow is exponentially distributed with mean of 8 ms. Each flow has a 5KB queue separately. The packet size is fixed at 1KB and the weight of each flow is 1000. The flows share a wireless channel with a 1.5Mbps bandwidth. If a packet is retransmitted in the link layer, four more resources are consumed by the link level retransmission. The total simulation time was 10s.

The error control scheme used in the link layer is selective repeat request ARQ with maximum window-size of 6. The time delay for the link layer to respond with amount of used resources a delivered packet consumed is investigated under different values. Unfortunately, due to the limited space, we are only going to present the system performance under 4.667 ms.

4.2. Simulation Results

To compare the efficiency of the WFQ-WBR to the WFQ-R algorithm, we added a fixed delay that represents the time period which the link layer responds to the upper layer with the status report after a packet is transmitted. In our simulations we used different delay parameters at different maximum window sizes to demonstrate the performance of the WFQ-WBR compared to WFQ-R. We represent the LLR response time by *d* and the maximum window size by *W*. The results show the amount of allocated resources, queuing delay and received data served to each flow.

From Figure 3, we can observe that the WFQ-R SIC separates the delay smoothly and gives slightly more resources to the error-prone flow to compensate for error recovery. However, the allocated resources for WFQ-WBR appears to be slightly less in proportion compared to the results obtained for WFQ-R. This is due to presence of outstanding packets within the windows frame that were not yet acknowledged and therefore the link layer has not yet reported the number of consumed resources obtained by these packets.

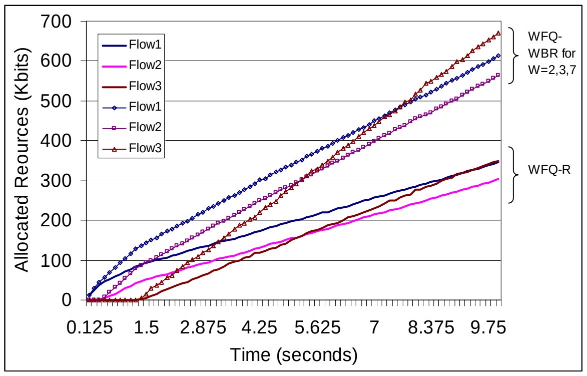

Figure 3. Comparison between WFQ-WBR and WFQ-R allocated resources when LLR response delay is 4.667 **ms**

On the other hand, Figure 4 shows the significant improvements obtained in the throughput of the system using a window-based queuing algorithm compared to WFQ-R. Furthermore, considering the queuing delay, we can see the significant drop in this metric under the proposed algorithm as shown in Figure 5 and Figure 6. Nevertheless, Flow 3 suffers more delay than Flow 1 & 2 as Flow 3 is the only erroneous flow in the system and therefore, it will not be allowed to send data over some turns as explained above. As the window size increases, the queuing delay decreases.

Furthermore, as shown in figures 3 and 4, the results for the amount data received and allocated resources with different window sizes are the same. Since the total response of the system is 10ms (LLR delay + service time), the maximum number of windows slots that can be utilized is only 2 regardless of the window size. Significant effects of the system throughput at different window sizes will become apparent as we increase the LLR response time. Figure 7 shows the difference in the amount of data received for each flow with window sizes of 2 and 7 when the LLR response time is 14.67 ms. For this particular delay time the data received for WFQ-R does not exceed 170 packets for all flows.

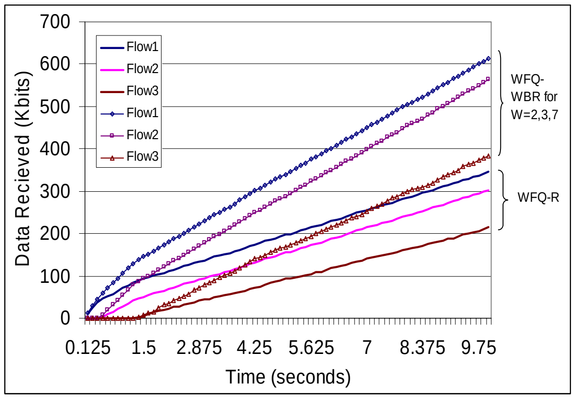

Figure 4: Comparison between WFQ-WBR and WFQ-R received data when LLR response delay is 4.667 ms

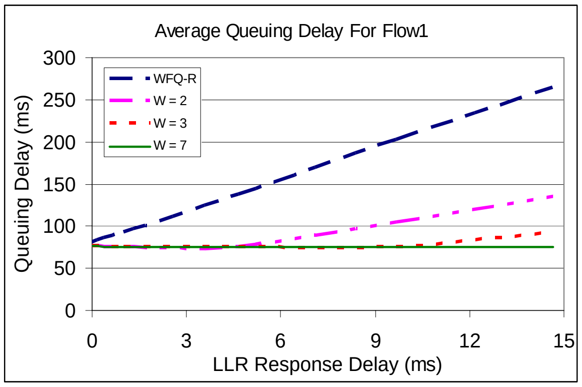

Figure 5 Average Queuing delay for Flow1 & 2 at different LLR response times and different windows sizes

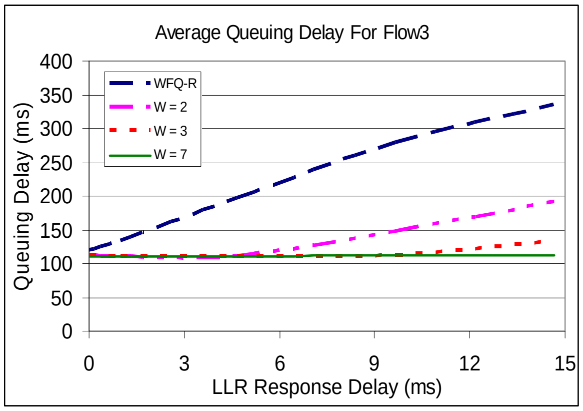

Figure 6 Average Queuing delay for Flow3 at different LLR response times and different windows sizes

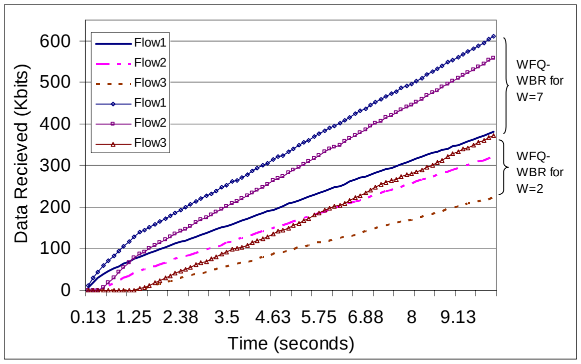

Figure 7 Received data comparison between windows sizes 2 and 7 when LLR response delay is 14.667 ms

5. CONCLUSIONS

We presented a new wireless fair queuing algorithm with windows-based retransmission and its role in providing quality of service in error-prone wireless networks. The algorithm is based on the WFQ-R suggested earlier by Kim and Yoon and modified to work with link layer that uses window-based schemes for error control such as Go-Back-n and Selective repeat ARQ. Through simulation, we have shown the significant improvement that can be obtained in term of high throughput and very low queuing delay. However, we also found that our WFQ-WBR algorithm had little effect on the fairness performance in comparison to the WFQ-R, but generally, WFQ-WBR is able to maintain fairness adaptively in presence of errors.

6. ACKNOWLEDGMENT

This work is supported by King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia. Abdul-Rahman Elshafei would like to thank Mohamad al-Maliki and Dr. AbdulHafid Bouhraoua for their valuable suggestions.

7. REFERENCES

[1] S. Lu, V. Bharghavan and R. Srikant, Fair scheduling in wireless packet networks, IEEE/ACM Transactions on Networking 7 (1999) (4).

[2] P. Ramanathan, P. Agrawal, Adapting packet fair queueing algorithms to wireless networks, in: proceeding of ACM MOBICOM'98, Dallas, USA, Oct. 1998.

[3] T.S.E. Ng, I. Stoica, H. Zhang, Packet fair queueing algorithms for wireless networks with location-dependent errors, in: Proceeding of IEEE INFOCOM'98, 1998.

[4] Namgi Kim, Hyunsoo Yoon: "Wireless packet fair queueing algorithms with link level retransmission." *Computer Communications* 28(7): 713-725 (2005).

[5] P. Lin, B. Benssou, Q. L. Ding, and K. C. Chua, "CS-WFQ: awireless fair scheduling algorithm for error-prone wireless channels,"in Proceedings of Computer Communications and Networks, 2000, pp.276–281

[6] H. Luo and S. Lu, A Topology-Independent Fair Queueing Model in Ad Hoc Wireless Networks Proc. IEEE Int'l Conf. Network Protocols, 2000

[7] Yung Yi, Yongho Seok, Taekyoung Kwon, Yanghee Choi and Junseok Park, "W2F2Q : Packet Fair Queuing in Wireless Packet Network," ACM WoWMoM 2000, Boston, Massachusetts, USA, August, 2000.

[8] A.K.F Khattab and K.M.F Elsayed, "Channel-quality dependent earliest deadline due fair scheduling schemes for wireless multimedia networks", *7th ACM Intl. symp. on modeling, analysis and simulation of wireless and mobile systems*, pp. 31-28, Venice, Italy, 2004

[9] Vijay Raghunathan, Saurabh Ganeriwal, Curt Schurgers, Mani B. Srivastava, "Energy Efficient Wireless Packet Scheduling and Fair Queuing," ACM Transactions on Embedded Computing Systems, Vol.3, No.1, pp. 431-447, February 2004.