Packet Fair Queueing Algorithms for Wireless Networks with Link Level Retransmission

Namgi Kim
Div. of Computer Science, Dept. of EECS
Korea Advanced Institute of Science and Technology
Daejeon, Korea
ngkim@camars.kaist.ac.kr

Hyunsoo Yoon
Div. of Computer Science, Dept. of EECS
Korea Advanced Institute of Science and Technology
Daejeon, Korea
hyoon@camars.kaist.ac.kr

Abstract— **Recently, a number of fair queueing algorithms for wireless networks have been proposed. They, however, need perfect channel prediction before transmission and rarely consider a medium access control (MAC) algorithm. In the wireless world, link level retransmission scheme is popularly used in the MAC layer for recovering channel errors. Therefore, we propose a new wireless fair queueing algorithm that works well with the link level retransmission and does not require channel prediction. Through simulation, we showed that our algorithm guarantees throughput and fairness. Also, we found that our algorithm achieves flow separation and compensation.**

Keywords - Wireless QoS; Wireless packet scheduling

I. INTRODUCTION

In recent years, with the increasing usage of these wireless data, the wireless networks are quickly becoming an integral part of the Internet. While, supporting multimedia communication applications requires the network to provide quality of service for packet flows. In wired networks, FFQ (Fluid Fair Queueing) has been a popular paradigm for providing fairness among packet flows over a shared link [1]. Besides, a number of approximation algorithms for the implementation have been proposed such as WFQ [2], SCFQ [3], SFQ [4], and WF2Q+ [5]. These algorithms for wired networks, however, cannot be applied directly to wireless networks because a wireless channel experiences location-dependent and bursty channel errors.

Recently, a number of wireless fair queueing algorithms have been proposed such as IWFQ (Idealized Wireless Fair-Queueing) [6], SBFA (Server Based Fairness Approach) [7], CIF-Q (Channel-condition Independent Fair Queueing) [8], and WFS (Wireless Fair Service) [9]. To overcome the location-dependent channel errors, these algorithms dynamically reassign channel allocation by predicting channel errors. Consequently, they need perfect channel prediction before transmission and rarely consider MAC (Medium Access Control) algorithm, but the perfect channel prediction before scheduling is very difficult in practice. Instead of channel prediction, most wireless networks adapt link level retransmission, like ARQ (Automatic Repeat reQuest), in the MAC layer for recovering channel errors. The previous wireless fair queueing algorithms, however, do not work well with this link level retransmission MAC algorithm. The WFS algorithm has proposed a specific medium access algorithm [9], but it is not commonly used in the wireless world. Therefore, in this work, we propose a wireless fair queueing algorithm that does not require channel prediction and works well with the link level retransmission, the most commonly used MAC algorithm.

The rest of the paper is organized as follows. In section II, we introduce background and motivation for this paper. In section III, we present the new wireless fair queueing algorithm with link level retransmission. Then, in section IV, we evaluate the performance of our algorithm through simulation. Lastly, in section V, we conclude the paper.

II. BACKGROUND AND MOTIVATION

A. Fairness Criteria in Wireless Networks

There are two kinds of fairness criteria in networks: data fairness and resource fairness. In wired networks, data fairness and resource fairness are generally the same. In wireless networks, however, they are not the same due to wireless channel errors. Data fairness is the fairness based on received data. It guarantees that each flow receives the same amount of data if their weights are equal. This concept, however, is inadequate for wireless networks. In wireless networks, an erroneous flow, which experiences severe channel errors, can exhaust most wireless resources and other flows may have few resources even if their channel conditions are good. On the contrary, resource fairness keeps fairness based on the amount of wireless resources used by each flow. It equally distributes scarce wireless resources to all flows. Therefore, the resource fairness is more suitable in wireless networks and we concentrate on this resource fairness rather than the data fairness.

B. Network and Channel Model

We consider wireless link is a shared channel and packet scheduling is performed above the MAC layer. When a packet experiences channel errors during transmission, the packet is retransmitted in the MAC layer until the destination receives the packet correctly or the maximum number of the retransmission is reached.

This work was supported by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc) and University IT Research Center (ITrc) Project.

Packet scheduling for downlink flows is performed at a base station or an access point in centralized manner. For uplink flows, it is difficult to know current states of all mobile hosts in the base station. However, the amount of traffic is also small in proportion to that of downlink flows. So, the fairness for uplink flows is usually acquired through the contention-based channel scheduling in distributed manner. Therefore, we just deal with the fairness of downlink flows in the centralized packet control.

C. *Problems of Previous Wireless Fair Queueing Algorithms with Link Level Retransmission*

Figure 1 illustrates the problem of wireless fair queueing algorithms without considering the link level retransmission. In this example, flows 1, 2, and 3 have weights 1, 1, and 2 respectively. In an error-free model, each flow sends 2, 2, and 4 packets in two rounds. However, in a real system, the first packet of flow 1 has experienced channel error during transmission, and the packet is retransmitted in the MAC layer with three more resources. Consequently, each flow receives 2, 1, and 2 packets. Moreover, possessions of the wireless channel are 5, 1, and 2, respectively. This means that the previous prediction-based wireless fair queueing algorithms no longer provide fairness with the link level retransmission. Therefore, we propose a new wireless fair queueing algorithm that can handle this situation.

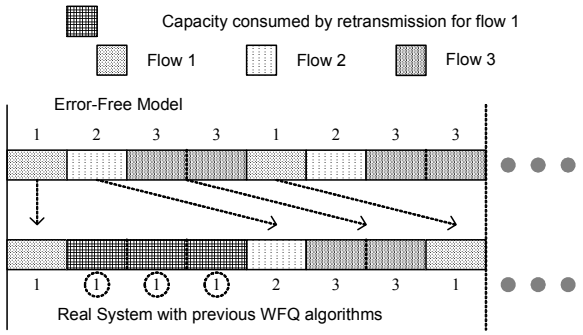


Figure 1. Unfair packet scheduling with link level retransmission

III. WIRELESS PACKET FAIR QUEUEING ALGORITHM WITH LINK LEVEL RETRANSMISSION

The basic concept of the Wireless Fair Queueing with Retransmission (WFQ-R) algorithm is that the share used for retransmission is regarded as a debt of the retransmitted flow to the others. Hence, when the retransmission occurs, the overhead, the used wireless resources in retransmission, is charged to the retransmitted flow.

The WFQ-R algorithm has two kinds of compensation types: Flow-In-Charge (FIC) and Server-In-Charge (SIC). FIC regards the entire overhead used for the retransmission as a charge of the retransmitted flow. That is, an error-prone flow should take responsibility for its own channel condition. For instance, consider a flow that has consumed two more resources for the retransmission in Figure 2. Then, in the FIC, since the flow has the entire responsibility for the two consumed resources, the flow does not use the next two resources and makes a concession to other flows in the second and third turns.

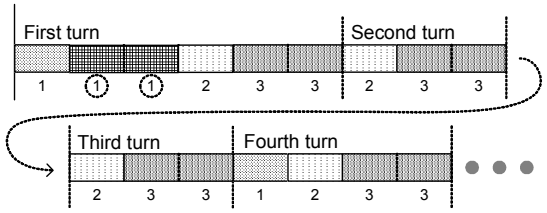


Figure 2. The example of Flow-In-Charge type

The FIC is fair with respect to allocated wireless resources. However, it may be too severe for a flow experiencing frequent errors. We propose another compensation type, SIC. For SIC, all backlogged flows are responsible for channel errors. So, the overhead is distributed over all flows and the retransmitted flow has responsibility for only a portion of the overhead in proportion to its weight. Therefore, the charging overhead of the retransmitted flow would be naturally reduced. For example, when a flow has consumed four resources for the retransmission and the weight of the flow is a quarter of the sum of the weights of all flows in Figure 3, the retransmitted flow is responsible for only one resource. Accordingly, the flow disclaims only one time in the second turn.

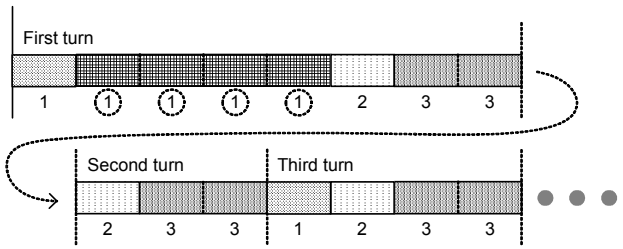


Figure 3. The example of Server-In-Charge type

A. *Algorithm Description*

TABLE I. TERMINOLOGIES FOR WFQ-R ALGORITHM

v_i	: virtual time of flow i
lag_i	: the difference between the service that flow i should receive in a reference error-free packet system and the service it has received in the real system lagging if positive, leading if negative, and in-sync otherwise
A	: the set of active flows
α	: minimal fraction of service retained by any leading flow
s_i	: normalized amount of service actually received by a leading flow i since its became leading
c_i	: normalized amount of compensation service received by a lagging flow i
r_i	: the rate of flow i

The full WFQ-R algorithm is shown in Figure 4 and parameter definitions are shown in Table I. The basic framework and notations follow that of CIF-Q algorithm [8].

When a packet is received, it is enqueued at a buffer in **on receiving** function. Then, the server picks up a packet from the buffer and sends it in **on sending** function. If a flow has no more packets to send, the flow leaves scheduling in leave function. In the **on sending** function, after a packet is selected to send, the packet is sent by **send_pkt()** function. The **send_pkt()** function gets the packet from the buffer and adjusts degrees of flows as leading or lagging. Then, it calls **send_and_charge()** function, in which the packet is actually sent through MAC layer by **send()** function. The **send()** function returns the overhead used by the link level retransmission. After that, the charging overhead, for which retransmitted flow takes responsibility, is calculated by **charge()** function depending on compensation type. Lastly, the charging overhead is distributed over all backlogged flows.

```
1  on session i receiving packet p:
2     if (i ∉ A)
3        v_i ← max (v_i, min_{k∈A}{v_k});
4        lag_i ← 0;
5        A ← A ∪ {i}; /* mark flow active */
6     enqueue(queue_i, p);
7
8  on sending current packet: /* get next packet to send */
9     i ← min_{v_i} {i ∈ A};
10    if (lag_i ≥ 0 or (lag_i < 0 and s_i ≤ αv_i))
11       /* flow i non-leading or leading with degradation */
12       send_pkt(i, i); /* flow i served through v_i selection */
13    else /* flow i is leading and not allowed to send */
14       /* select lagging flow j to compensate */
15       j ← min_{c_k} {k ∈ A | lag_k > 0};
16       if (j exists)
17          send_pkt(j, i); /* serve flow j but charge to i */
18          if (i ≠ j and empty(queue_j) and lag_j ≥ 0)
19             leave(j); /* j becomes inactive */
20       else /* there is no lagging flow */
21          send_pkt(i, i); /* serve given back to flow i */
22    if (empty(queue_i) and lag_i ≥ 0)
23       leave(i); /* i becomes inactive */
24
25 send_pkt(j, i) /* serve flow j but charge to i */
26    p ← dequeue(queue_j);
27    v_i ← v_i + p.length / r_i;
28    if (i = j and lag_i < 0 and s_i ≤ αv_i)
29       /* flow i is leading and served through v_i selection */
30       s_i ← s_i + p.length / r_i;
31    if (i ≠ j)
32       lag_j ← lag_j − p.length; /* flow j has gain extra service */
33       if (lag_j > 0)          /* j continues to be lagging */
34          c_j ← c_j + p.length / r_j;
35       if (lag_j + p.length ≥ 0 and lag_j < 0)
36          /* j just becomes leading */
37          s_j ← αv_j;
38       lag_i ← lag_i + p.length; /* flow i has lost service */
39       if (lag_i − p.length ≤ 0 and lag_i > 0)
40          /* i just becomes lagging */
41          c_i = max (c_i, min_{k∈A}{c_k | lag_k > 0});
42    send_and_charge(p, j, i);
43       /* send pkt with retransmission and charge overhead */

44 send_and_charge(p, j, i); /* send pkt p with retransmission */
45                          /* and charge overhead */
46    used_ret ← send(p);    /* send pkt p through MAC layer and */
47       /* return used wireless resources due to retransmission */
48    if (used_ret ≤ 0 or A − {j} = ∅) /* if no retransmission or */
49       return;                  /* no other flows, then return */
50    charged ← charge(used_ret, j); /* charging overhead depending */
51                                 /* on compensation type */
52    if (i = j and lag_j < 0) /* flow i is leading and served through v_i */
53       s_j ← s_j + charged / r_j;
54    else /* in-sync or lagging in i = j, or i ≠ j */
55       lag_j ← lag_j − charged; /* flow j has gain extra service */
56       if (lag_j > 0)          /* j continues to be lagging */
57          c_j ← c_j + charged / r_j;
58       if (lag_j + p.length ≥ 0 and lag_j < 0) /* j just becomes leading */
59          s_j ← αv_j;
60       for (l ∈ A − {j}) /* other flows ditributively get share */
61          lag_l_before ← lag_l;
62          lag_l ← lag_l + charged × η / Σ_{k∈A−{j}} r_k;
63          if (lag_l_before ≤ 0 and lag_l > 0)
64             /* i just becomes lagging */
65             c_l = max (c_l, min_{k∈A}{c_k | lag_k > 0});
66
67 charge(used_ret, j) /* calculate amount of charging overhead */
68    switch (COMPENSATION_TYPE)
69       case FLOW_IN_CHARGE :
70          /* entire overhead is charged to the retransmitted flow */
71          return used_ret × (1 − r_j / Σ_{k∈A} r_k);
72       case SERVER_IN_CHARGE :
73          /* overhead is distributively charged to flows in the server */
74          return used_ret × (1 − r_j / Σ_{k∈A} r_k) × r_j / Σ_{k∈A} r_k;
75
76 leave(i) /* flow i leaves */
77    A ← A \ {i};
78    for (j ∈ A) /* update lags of all active flow */
79       if (lag_j ≤ 0 and lag_j + lag_i × r_j / Σ_{k∈A} r_k > 0)
80          /* j just becomes lagging */
81          c_i ← max (c_i, min_{k∈A}{c_k | lag_k > 0});
82       lag_j ← lag_j + lag_i × r_j / Σ_{k∈A} r_k;
83       if (∃j ∈ A s.t. empty(queue_i) ∧ lag_i ≥ 0)
84          leave(j);
```

Figure 4. WFQ-R Algorithm

More detail descriptions of functions are as follow:

- **on receiving**: when a flow *i* becomes backlogged and active, its virtual time v_i is initialized to the maximum of its virtual time and the minimum virtual time among other active flows (line 3). Then, its lag is initialized to zero (line 4).
- **on sending**: The algorithm selects the active flow *i* with the minimum virtual time for service (line 9). If that flow is not leading or leading but did not get minimal fraction of service, then the packet at its queue is transmitted (line 10-12). However, if the flow is leading with getting more than minimal service, we search for the lagging flow *j* with the minimum c_j (line 15). If there is such a flow *j*, the packet at its queue is transmitted, instead of the packet of flow *i* (line 16-19). Otherwise, transmit the packet of flow *i* originally (line 20-21).
- **send_pkt()**: After the packet to transmit is decided, the virtual time of flow *i*, v_i, is advanced as $v_i + p.length / r_i$ where r_i is the rate of flow *i* (line 27). Then, parameters are adjusted depending on the flow's

situation (line 28-41). If flow i is leading but gets services due to graceful degradation, s_i is updated to s_i. + $p.length$ / r_i (line 28-30). If flow j is served but charge to i ($i \neq j$), then the flow j has gain extra service and flow i has lost service (line 31-41). As a result, lag_j is updated to lag_j - $p.length$ (line 32), and lag_i is updated to lag_i + $p.length$ (line 38). c_i, s_i, and c_i are also adjusted depending on flow i and j's situations (line 33-37, 39-41). After that, the selected packet is passed to **send_and_charge()** function (line 42-43).

- **send_and_charge()**: In this function, the packet is actually transmitted through the MAC layer and the effect of the link level retransmission is reflected to our wireless fair queueing algorithm. The **send()** function transmits the packet and returns the amount of used wireless resources, $used_{ret}$, due to the link level retransmission in the MAC layer (line 46-47). If there is no retransmission ($used_{ret} \leq 0$) or no other active flow ($A - \{j\} = \emptyset$), then finish the procedure and return to first step of algorithm (line 48-49). Otherwise, the flow j has gain extra service by link level retransmission and it takes responsibility for that. flow j's charging overhead, $charged$, is calculated through **charge()** function with $used_{ret}$. (line 50-51). The $charged$ is the amount of resources for which the retransmitted flow has to be responsible. At that time, if flow j is leading, s_j is updated to s_j. + $charged$ / r_j (line 52-53). Otherwise, lag_j is reduced to lag_j - $charged$ (line 55) and c_i and s_i, are updated depending on flow j's situation (line 56-59). Because the other flows are deprived of their share due to the flow j's retransmission, they have to get more shares later. Thus, the other flows's lags, lag_l, are increased in proportion to their weights (line 60-65). Hence, lag_l is updated to $lag_l + charged \times r_l / \sum_{k \in A-\{j\}} r_k$ (line 62). Consequently, the retransmitted flow becomes leading, and the other flows become lagging. Next time, the lagging flow has higher priority to get wireless channel than the leading flow.

- **charge()**: this function calculates and returns charging overhead caused by link level retransmission. The charging overhead, $charged$, is calculated based on the amount of wireless resources used due to link level retransmission, $used_{ret}$. There are two kinds of types: FIC and SIC. For FIC, the retransmitted flow j has to take responsibility for the entire resources used by link level retransmission. So, FIC returns the amount of other flows' resources, which are used in retransmission without permission (line 71). On the other hand, SIC distributes a charge of used resources to all backlogged flows. So, SIC returns only a portion of the amount of used resources, which has to be charged by the flow in proportion to its weight (line 74).

- **leave**: when a lagging flow i becomes unbacklogged and wants to leave, its positive lag_i is proportionally distributed among all the remaining active flows j such that each lag_j is updated to $lag_j + lag_i \cdot r_j / \sum_{k \in A} r_k$ (line 78-84).

IV. SIMULATION

In this section, we present results from simulation to demonstrate the fairness properties of WFQ-R algorithm. The following performance measures are used to evaluate the algorithm:

- **Allocated resources**: the amount of wireless resources used by a flow. It directly represents the resource fairness.
- **Queueing Delay**: experienced delay in a queue.
- **Goodput**: the actual amount of data received at a destination. It represents the data fairness. However, the goodput is a second metric as compared to the allocated resources because the data fairness cannot be sustained strictly in wireless networks, as mentioned before.

A. Simulation Environments

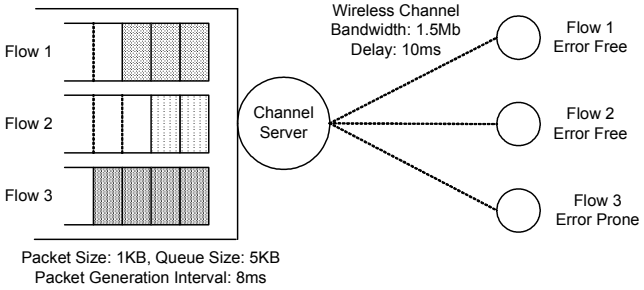


Figure 5. Simulation Topology

For simulation, we used an ns simulator [10]. The simulation topology is shown in Figure 5. There are three flows in the simulation: two error-free flows and one error-prone flow. The retransmission probability of flows 1 and 2 are zero, and that of flow 3 is 0.2. All flows have the same weight. The detail properties of each flow are shown in Table II. If a packet experiences the link level retransmission, four more resources are used for that. We simulate CIF-Q algorithm [8] and our algorithms with the link level retransmission MAC algorithm. The CIF-Q is for evaluating the performance of prediction-based wireless fair queueing algorithms without channel prediction. As mentioned before, most wireless systems adapts the link level retransmission MAC algorithm, instead of the perfect channel prediction.

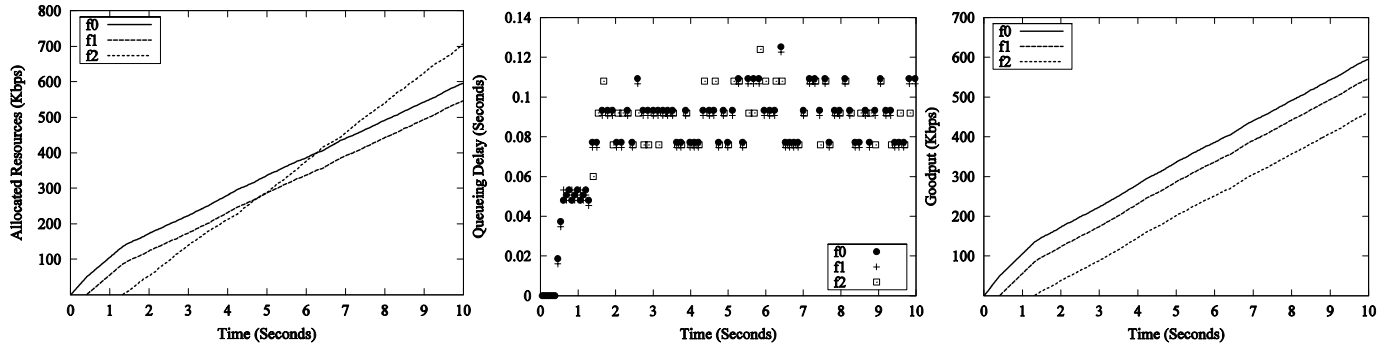
TABLE II. PROPERTIES OF FLOWS IN THE SIMULATION

	flow1	flow2	flow3
Weight	1	1	1
Retrans. Prob.	0	0	0.2
Start Time	0 s	0.4 s	1.3 s
Stop Time	10 s	10 s	10 s
Packet Size (KBytes)	1KB	1KB	1KB
Queue Size (KBytes)	5KB	5KB	5KB
Packet Gen. Interval	8 ms	8 ms	8 ms

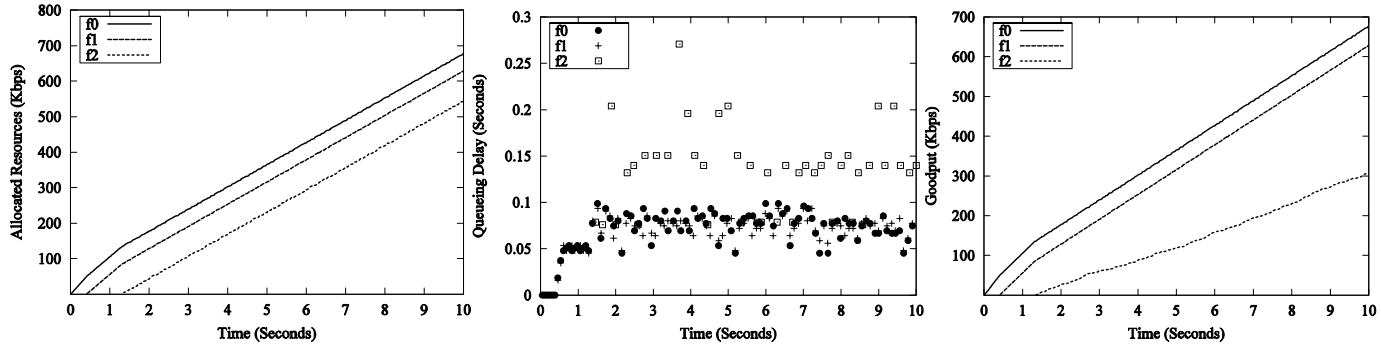
B. Simulation Results

The simulation results are shown in Figure 6 and Table III. In Figure 6-(a) and Table III, we can find that the CIF-Q algorithm with the LLR (CIF-Q_LLR) does not provide fairness properly. It assigns too many wireless resources to error-prone flow. Additionally, the delay of the error-free flow is also increased due to the error-prone flow. The CIF-Q_LLR seems to keep goodput fairness, but this is the result from lopsided sacrifice of error-free flows and, moreover, it cannot be maintained when the error-prone flow suffers severe channel errors.

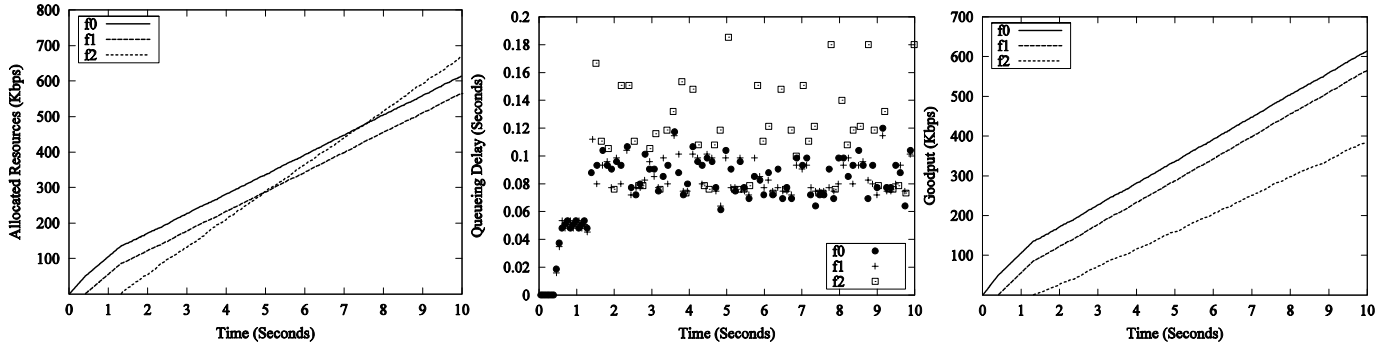
Our WFQ-R algorithms, however, fairly distribute wireless channel resources to each flow. The WFQ-R_FIC gives the same amount of resources among flows precisely (shown in Figure 6-(b)). The WFQ-R_SIC gives slightly more resources to the error-prone flow to compensate error recovery (shown in Figure 6-(c)). In addition, the WFQ-R_SIC separates the delay of error-free and error-prone flows completely. While, the WFQ-R_SIC separates the delay smoothly.



(a) CIF-Q_LLR algorithm



(b) WFQ-R_FIC algorithm



(c) WFQ-R_SIC algorithm

Figure 6. Simulation Results

According to the simulation results, we can find that our WFQ-R algorithms keep well the fairness with the link level retransmission MAC algorithm. The difference between FIC and SIC is a trade-off between the flow separation and compensation [11]. The flow separation is a property that an error-free flow should not be impacted at all by other flows. Strict flow separation, however, brings about the starvation of a frequent error-prone flow due to the shortage of compensation resources. Thus, we proposed two compensation types, FIC and SIC, to coordinate flow separation and compensation appropriately. In other words, the FIC rigidly keeps resource fairness and the SIC gives a little more resources to error-prone flows for data fairness. Through the WFQ-R algorithms, we can also adaptively achieve the separation among flows in delay and throughput depending on each channel condition.

TABLE III. AVERAGE VALUE OF SIMULATION RESULTS

	flow1	**flow2**	**flow3**
Allocated Resources (Kbps)	wireless channel resources used by each flow		
CIF-Q_LLR	423.908	424.828	649.195
WFQ-R_FIC	499.31	500.23	499.31
WFQ-R_SIC	441.379	442.299	614.253
Goodput (Kbps)	actual amount of received data		
CIF-Q_LLR	423.908	424.828	422.989
WFQ-R_FIC	499.31	500.23	282.299
WFQ-R_SIC	441.379	442.299	353.103
QueueDelay (ms)	queueing delay in a buffer		
CIF-Q_LLR	91.06	89.334	92.209
WFQ-R_FIC	75.564	74.501	140.93
WFQ-R_SIC	85.616	85.19	111.785

V. CONCLUSION

We proposed a new wireless fair queueing algorithm. It achieves wireless fairness with the link level retransmission by penalizing flows using resources without permission. In addition, our algorithm does not need channel prediction for guaranteeing fairness. Through simulation, we proved that our algorithm guarantees throughput and fairness. Furthermore, we found that our algorithm achieves flow separation and compensation.

REFERENCES

[1] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," in proceeding of ACM SIGCOM'89, 1989.

[2] A. Parekh and R. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case," IEEE/ACM Transactions on Networking, Vol. 1, No. 3, Jun. 1993.

[3] S. Golestani, "A Self-Clocked Fair Queueing Scheme for Broadband Applications," in proceeding of ACM SIGCOMM'94, Toronto, Canada, Jun. 1994.

[4] P. Goyal, H. M. Vin, and H. Cheng, "Start-Time Fair Queueing: A scheduling Algorithm for Integrated Services Packet Switching Networks," IEEE/ACM Transactions on Networking, Vol. 5, No. 5, Oct. 1997.

[5] J. Bennett and H. Zhang, "Hierarchical Packet Fair Queueing Algorithms," IEEE/ACM Transactions on Networking, Vol. 5, No. 5, Oct. 1997.

[6] S. Lu, V. Bharghavan, and R. Srikant, "Fair Scheduling in Wireless Packet Networks," IEEE/ACM Transactions on Networking, Vol. 7, No. 4, Aug, 1999.

[7] P. Ramanathan and P. Agrawal, "Adapting Packet Fair Queueing Algorithms to Wireless Networks," in proceeding of ACM MOBICOM'98, Dallas, USA, Oct. 1998.

[8] T. S. E. Ng, I. Stoica and H. Zhang, "Packet Fair Queueing Algorithms for Wireless Networks with Location-Dependent Errors," in proceeding of IEEE INFOCOM'98, 1998.

[9] S. Lu, T. Nandagopal and V. Bharghavan, "Design and Analysis of an Algorithm for Fair Service in Error-Prone Wireless Channels," ACM Wireless Networks Journal, Vol. 6, No. 4, pp. 232-343, 2000.

[10] http://www.isi.edu/nsnam/ns/

[11] V. Bharghavan, S. Lu, and T. Nandagopal, "Fair Queuing in Wireless Networks: Issues and Approaches," IEEE Personal Communications, Feb. 1999.