Incentive-Oriented Downlink Scheduling for Wireless Networks with Real-Time and Non-Real-Time Flows

I-Hong Hou
Department of ECE
Texas A&M University
College Station, TX, USA
ihou@tamu.edu

Jing Zhu and Rath Vannithamby
Intel Labs
Hillsboro, OR, USA
{jing.z.zhu, rath.vannithamby}@intel.com

***Abstract*—We study the problem of designing downlink scheduling policy for wireless networks with both real-time and non-real-time flows. Most existing scheduling policies for these networks unfairly favor one type of flows. As a result, strategic clients may lie about their types of traffic in order to get more service from the base station. To counter these strategic behaviors, a scheduling policy should be incentive compatible in the sense that clients cannot improve their performance by lying.**

We propose a novel scheduling policy that is incentive compatible. The policy can be easily implemented as a thin layer between the Network layer and the MAC layer. It can also work with a wide range of upper-layer and lower-layer protocols while providing satisfactory performance to both real-time and non-real-time flows. Simulation results show that this policy outperforms other widely used policies by large margins under a variety of scenarios.

I. Introduction

Wireless networks are increasingly used to serve both real-time and non-real-time flows. The former includes applications such as VoIP and video streaming, while the latter includes applications like file transferring and web browsing. These flows have very different service requirements. Real-time flows usually require a strict per-packet delay bound, since late packets may not be useful to the application. On the other hand, non-real-time flows do not have any stringent delay requirements, and only need to have high throughputs.

There have been a lot of studies that aim to provide satisfactory performance for both real-time and non-real-time flows in a unified framework. Uc-Rios and Lara-Rodriguez [1] have proposed a policy that jointly considers the channel states and the packet delays. Haci, Zhu, and Wang [2] have considered the tradeoff between quality of service (QoS) for real-time flows and the overall system throughput. Choi and Shin [3] have proposed an architecture that jointly considers both real-time and non-real-time flows. Jaramillo and Srikant [4] have proposed policies that achieve QoS requirements of real-time flows while provide fair resource allocation for non-real-time flows.

However, all these studies rely on clients to provide the information on whether it has a real-time flow or a non-real-time one. As different kinds of flows may receive different services, a strategic client may falsely report its type of flow in order to obtain more services. Further, these studies make various assumptions on the traffic patterns of flows and the knowledge on current channel states. These assumptions limit their applicabilities.

In this paper, we aim to design a new scheduling policy that discourages strategic clients from lying about their types of flows that can be applied to virtually any kind of systems. The intuition of our policy is analogue to the design of fastlanes in supermarkets, where customers with a small number of goods can checkout through the fastlane and enjoy faster process. Similarly, our scheduling policy provides per-packet delay guarantees to real-time flows, but reduces their throughputs. On the other hand, non-real-time flows enjoy higher throughputs, but suffer from large delays. With this design intuition, a client can better achieve its service requirement by reporting its true type of flow.

We propose a scheduling policy, namely, the joint deadline-deficit (JDD) policy, that fulfills the above vision. In addition, we show that this policy can be easily implemented as a thin layer between the Network layer and the MAC layer. The policy does not make any assumptions on, nor does it require any information from, the upper-layers and the lower-layers. Hence, it can be integrated with a wide range of upper-layers and lower-layers protocols without any modifications.

We demonstrate the performance of the JDD policy by extensive simulations under ns-2. Simulation results show that the policy is indeed incentive compatible in the sense that neither real-time flows nor non-real-time flows can benefit from lying about their true types. We also compare our policy against state-of-the-art policies under various scenarios. Simulation results show that our policy outperforms others by large margins in all tested scenarios.

The rest of the paper is organized as follows. Section II introduces our system model, formally defines the properties we aim to achieve, and discusses why current mechanisms are not suitable to achieve these properties. Section III proposes our scheduling policy and addresses some implementation issues. Section IV presents our simulation results. Finally, Section V concludes the paper.

II. Problem Overview

We consider a wireless system where a base station (BS) serves N wireless clients. Each client is associated with a downlink flow, either a real-time flow or a non-real-time one, from the BS. If a client is associated with a real-time flow, it requires that the *access network delay*, defined as the time between the time the BS receives a packet and the time the BS successfully delivers the packet to the client, of every of its packets needs to be within D. If the access network delay of a packet for a real-time flow exceeds D, the packet is no longer useful for the real-time flow. Thus, we measure the performance of real-time flows by their *timely-throughput*, defined as throughput of packets whose access network delays are no larger than D. On the other hand, non-real-time flows do not require any delay guarantees, and their performance is measured by their throughputs.

Next, we discuss our problem formulation. The BS assigns a *weight* to each client. The weights of clients with real-time flows are $\alpha < 1$, while those of clients with non-real-time flows are 1. The BS aims to impose a scheduling policy where all packets delivered for real-time flows have access network delays less than D, and the amount of time that the BS spends on a client is proportional to its weight. The intuition is that the BS provides per-packet delay guarantees for real-time flows. On the other hand, in order to prevent non-real-time flows from falsely declaring itself as real-time ones, the BS also aims to make non-real-time flows receive more service than real-time ones.

It is well known that, if the channel capacity of each client is static, a policy achieves *weighted proportional fairness* among all clients by making each client receive an amount of service time proportional to its weight. That is, let w_n be the weight of client n, and r_n be the throughput, or timely-throughput, of n if it is associated with a non-real-time flow, or a real-time flow, respectively. A desirable policy can then maximize $\sum_n w_n \log r_n$. In addition to weighted proportional fairness, a desirable scheduling policy should also satisfy the following properties:

1) *Incentive Compatibility:* The scheduling policy should be able to provide incentives for clients to honestly reveal whether it has a real-time flow or a non-real-time one. Recall that the performance of a real-time flow, and a non-real-time one, is measured by its timely-throughput, and throughput, respectively. Hence, we can achieve incentive compatibility if the timely-throughput of a flow is maximized when it declares itself as a real-time flow, and its throughput is maximized when it declares itself as a non-real-time flow.
2) *Versatility:* The scheduling policy needs to be compatible with a wide range of upper-layer and lower-layer protocols. For example, some flows may generate constant-bit-rate (CBR) traffic, while others may employ some congestion control policies, such as TCP or TCP-Friendly Rate Control protocols [5], to generate traffic. On the other hand, the BS may transmit all packets using a fixed modulation, or it may employ rate adaptation mechanism and choose suitable modulations for different clients. It may also suffer from interference and contention from adjacent wireless devices. The proposed policy needs to provide satisfactory performance under all these environments.
3) *Deadline Awareness:* The scheduling policy needs to keep track of the delay requirements of all packets, so as to ensure the access network delays of all real-time flow packets are no larger than D.
4) *Work Conservation:* The scheduling policy always schedules a packet for transmission as long as there is one available for transmission.

We now discuss why current scheduling policies are not suitable to provide the aforementioned properties. The most common approaches for improving delay performance for real-time flows are DiffServ [6], IntServ [7], and their variations. In a nutshell, these approaches classify flows according to their respective delay requirements, and then provide more services to real-time flows than non-real-time ones in the hope of improving the delay performance of real-time flows. This approach is also adapted by IEEE 802.11e [9], where clients with real-time flows have higher chance of accessing the wireless channel than clients with non-real-time flows. However, these approaches are still unaware of the actual deadline of each packet. Moreover, as they provide more services to real-time flows than non-real-time ones, non-real-time flows may falsely declare themselves as real-time flows in order to obtain more services. Hence, these approaches cannot guarantee incentive compatibility.

Our goal is to design a scheduling under which the amount of time a client receives is proportional to its weight, and hence achieves weighted proportional fairness among clients. Two approaches to achieve this are weighted round robin [10] and weighted proportional fair scheduling [11]. However, neither policies are aware of packet deadlines. Further, as we will show in Section IV, simulation results demonstrate that the weighted round robin results in much worse performance than our proposed policy.

On the other hand, weighted proportional fair scheduling makes scheduling decisions based on the instant channel quality of each client. Obtaining the precise and instant channel information requires frequent channel probing and the ability of precise channel estimation. Moreover, in some networks, such as highly mobile networks and networks that suffer from the interference of nearby devices, it may be infeasible to obtain such information. Hence, it is not always practical to implement a weighted proportional fair scheduling policy.

III. Design and Implementation of the Joint Deadline-Deficit Policy

In this section, we propose a novel scheduling policy, the *joint deadline-deficit* (JDD) policy, that aims to achieve weighted proportional fairness among clients and provides the desirable properties, including deadline awareness, work conservation, incentive compatibility, and versatility.

We first define the concept of *deficit*. As discussed in Section II, the amount of time that the BS spends transmitting packets for a client should be proportional to its weight. As the weight of client n is w_n, then client n should receive a portion of $\frac{w_n}{\sum_m w_m}$ time from the BS. The deficit of client n then reflects the difference between the amount of time it *should* receive, and the amount of time it *actually* receive from the BS. It can be formally defined as follows:

Definition 1: The deficit of client n at time t is denoted by $d_n(t)$, and is defined iteratively as follows:

1) $d_n(t_0) = 0$, where t_0 is the time when client n joins the system.
2) If no clients join or leave the system between times (t_1, t_2), then $d_n(t_2) = d_n(t_1) + \frac{w_n}{\sum_{m\in M} w_m}(t_2 - t_1) - \{$Amount of time that the BS spends on n during $(t_1, t_2)\}$, where M is the set of clients present in the system between times (t_1, t_2). When computing the amount of time that the BS spends on n, we include all possible overheads such as times needed for channel contention, retransmissions, and ACKs.

We now describe the JDD policy. It is designed to be a thin layer between the Network layer and the MAC layer, and provides two functions, *enqueue()* and *dequeue()*. Fig. 1 illustrates the implementation of the JDD policy. When a packet arrives at the BS, the Network layer calls *enqueue()*. The JDD policy then marks the deadline of the packet, and puts it into the buffer. If the packet belongs to a real-time flow, its deadline is set to be (current time)$+D$. If the packet belongs to a non-real-time flow, its deadline is set to be a large value. In our implementation, we set its deadline to be (current time)+10 seconds, since the timeout value of TCP is usually on the order of seconds.

The core mechanism of the JDD policy is the *dequeue()* function, which chooses a packet for the MAC and PHY layers to transmit when being called by the MAC layer. Our policy for the *dequeue()* function is very simple.

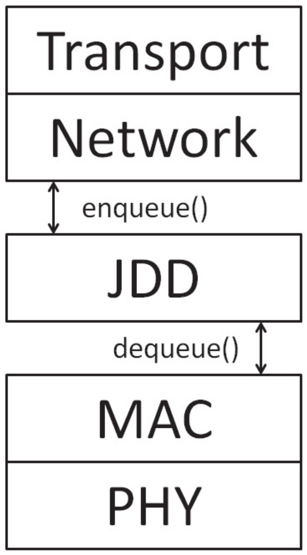


Fig. 1: Architecture of the JDD policy

Whenever it is being called, it first drops all packets whose deadlines have passed. It then picks and transmits the packet who has the earliest deadlines among those whose flows have positive deficits. In the case that all flows have negative deficits, the JDD policy picks and transmits the packet with the earliest deadline. The following example illustrates the *dequeue()* function of the JDD policy.

Example 1: Consider a system with three clients, A, B, and C. Clients A and B are associated with real-time flows, while C is associated with a non-real-time flow. Suppose at time t, each client has two packets in the buffer. The deadlines of the packets of A are $t+2$ and $t+5$, those of B are $t+1$ and $t+3$, and those of C are $t+100$ and $t+101$, respectively. At time t, the deficits of the three clients are $d_A(t) = 0.1$, $d_B(t) = 0.5$, and $d_C(t) = 3$, respectively. Finally, we set $\alpha = 0.5$, and hence $w_A = w_B = 0.5$, and $w_C = 1$.

When the *dequeue()* function is called at time t, all three clients have positive deficits, and hence the JDD policy simply selects the packet that has the earliest deadline, which is the first packet of B. Assume that it takes 1 time unit to transmit the packet. After the transmission is completed, we can update the deficits of clients as $d_A(t+1) = 0.1+1\times\frac{0.5}{2} = 0.35$, $d_B(t+1) = 0.5+1\times\frac{0.5}{2}-1 = -0.25$, and $d_C(t+1) = 3+1\times\frac{1}{2} = 3.5$. At time $t+1$, since $d_B(t+1) < 0$, the JDD policy picks the packet with the earliest deadline among those belong to A and C, which is the first packet of A. Assume that it takes 3 time units to transmit the packet. Now, at time $t+4$, we have $d_A(t+4) = 0.35+3\times\frac{0.5}{2}-3 = -1.9$, $d_B(t+1) = -0.25+3\times\frac{0.5}{2} = 0.5$, and $d_C(t+1) = 3.5+3\times\frac{1}{2} = 5$. When the *dequeue()* function is called at time $t+4$, the second packet of B has expired and needs to be dropped. Thus, even though B has a positive deficit, it cannot be scheduled. The JDD policy then transmits the first packet of C, as the deficit of A is negative. □

We now discuss why the JDD policy satisfies the desirable properties shown in Section II. It is obvious that the JDD policy is aware of packet deadlines and is work conserving. Since the JDD policy only schedules clients with positive deficits, non-real-time flows will be served more frequently than real-time flows. Hence, non-real-time flows do not have the incentive to pretend to be real-time ones. On the other hand, as shown in the previous example, since deadlines of the packets of non-real-time flows are large, they are usually served after those of real-time ones. As a result, non-real-time flows may suffer from long delay and small timely-throughput. Therefore, real-time flows cannot improve their performance by falsely declaring themselves to be non-real-time ones.

Finally, the JDD policy is versatile and can work with a wide range of upper layers and lower layers protocols. The JDD policy does not make any assumptions on the upper layers, and hence can work with flows with arbitrary traffic patterns. As for the MAC and PHY layers, the JDD policy only needs the information on the amount of time taken to transmit a packet. However, this information is not always available. We counter this difficulty by estimating the time taken to transmit a packet by the duration between two consecutive function calls for *dequeue()*. Algorithm 1 describes the details of *dequeue()*.

Algorithm 1 dequeue

```
1: Global variables: $preTime, preScheduled, [d_n], [w_n]$
2: $curTime \leftarrow$ current time
3: $totWeight \leftarrow \sum_n w_n$
4: $m \leftarrow preScheduled$
5: $d_m \leftarrow d_m - (curTime - preTime)$
6: for $n$ do
7:     $d_n \leftarrow d_n + \frac{w_n}{totWeight}(curTime - preTime)$
8: end for
9: Drop all expired packets
10: $pkt \leftarrow$ the packet with the earliest deadline among
    those whose destination $n$ has $d_n > 0$
11: if $pkt = \phi$ then
12:     $pkt \leftarrow$ the packet with the earliest deadline
13: end if
14: $preTime \leftarrow curTime$
15: $preScheduled \leftarrow$ the destination of $pkt$
16: return $pkt$
```

IV. Simulation Results

We present our simulation results in this section. We have implemented the JDD policy in ns-2 as a thin layer between the Network layer and the MAC layer. In our implementation, the JDD policy inherits the `Queue` class, and hence it can be integrated with other layers easily without any modifications to the overall architecture. In all our simulations, we use the `Shadowing` module to model signal attenuation and random packet loss, and we assume that all real-time flows require the access network delays of all packets to be no larger than $100ms$.

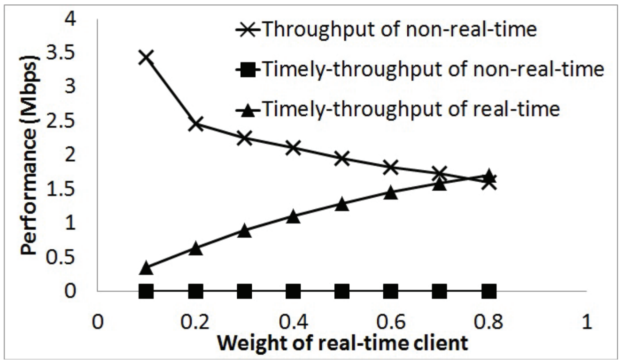


Fig. 2: Throughputs and Timely-throughputs of both non-real-time and real-time flows, respectively.

A. Incentive Compatibility

As discussed in Section II, an important property that we aim to achieve is incentive compatibility. That is, we want to make sure real-time clients have no incentives to pretend to be non-real-time clients, and vice versa. In this section, we first evaluate whether the JDD policy achieves this property.

We consider a system with 10 clients, 5 of them are associated with real-time flows, while the other 5 are associated with non-real-time flows. In order to make the comparison fair, we place all clients $400m$ away from the BS, so that all clients have the same channel qualities. We assume that all flows are generated by the TCP protocol, and the BS uses IEEE 802.11 as its MAC protocol. We set the weight of real-time flow to be α, and evaluate the performance by varying the value of α between $[0.1, 0.8]$. We measure the performance by three metrics: the total throughput of non-real-time flows, the total timely-throughput of non-real-time flows, and the total timely-throughput of real-time flows. We record the access network delays of all delivered packets, and hence we can compute the total timely-throughputs of non-real-time flows. Also note that we do not include the total throughput of real-time flows. Since the JDD policy is deadline aware and never transmits a packet of real-time flow that has exceeded its deadline, the total throughput is exactly the same as the total timely-throughput for real-time flows.

Simulation results are shown in Fig. 2. We can observe two important behaviors. First, the timely-throughput of non-real-time flows is always close to zero for all α. Therefore, a real-time client will suffer from serious performance degradation if it false declare itself as a non-real-time one. Second, the throughput of non-real-time flows is larger than that of real-time flows when $\alpha \leq 0.7$. When $\alpha = 0.8$, the throughput of real-time flows

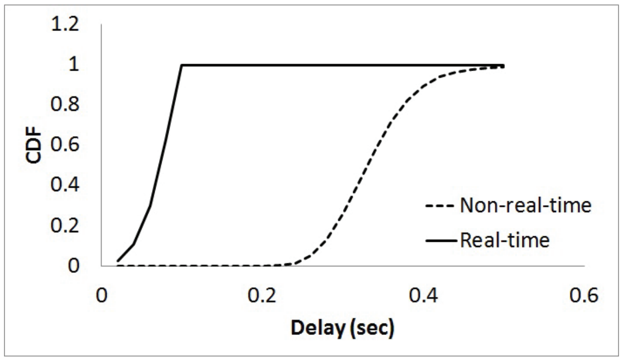


Fig. 3: Cumulative distribution functions of packet delays

is actually slightly higher than that of non-real-time ones. This is due to the behaviors of TCP. The traffic generated by TCP is affected by both packet loss rate and round-trip time. Since real-time flows have smaller delays, and hence smaller round-trip times, they may generate more traffic than non-real-time ones when α is close to 1. This suggests that the value of α should be chosen to be a reasonable small number to ensure that the throughput of non-real-time flow is higher than that of real-time flow.

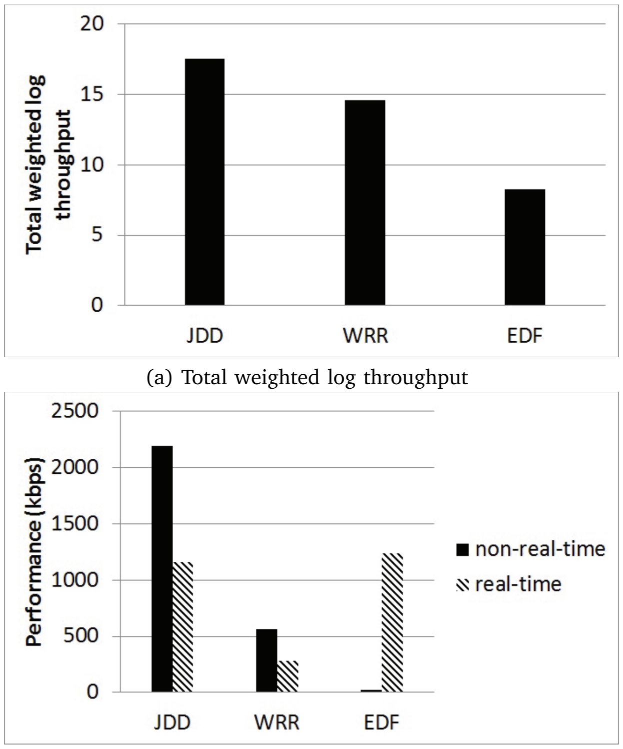
We also want to better understand the delay performance of both real-time and non-real-time flows. We set $\alpha = 0.5$ and record the access network delays of all delivered packets. We then plot the cumulative distribution functions of packet delays for both real-time and non-real-time flows. The results are shown in Fig. 3. It can be shown that the access network delays of all packets for real-time flows are below $100ms$, as the JDD policy is deadline aware. Moreover, the access network delays of packets for non-real-time ones are much larger. In fact, about 74% of packets for non-real-time flows have access network delays larger than $300ms$.

In sum, with a suitable value of α, the JDD policy ensures that non-real-time flows enjoy high throughputs at the expense of large delays, while real-time flows achieve small delays by sacrificing throughputs. Hence, the JDD policy is incentive compatible.

B. Performance Comparison and Versatility Evaluations

In this section, we compare the JDD policy against two other widely used policies. One policy is the weighted round robin (WRR) policy, and the other is the earliest deadline first (EDF) policy, where the BS always transmits the packet with the earliest deadline. In order to make fair comparisons, we modify the WRR policy so that it is deadline aware and drops expired packets.

We consider a system with 5 real-time clients and 5 non-real-time ones. The distances between clients and the BS are evenly spaced between $80m$ and $710m$. In all simulations, we set $\alpha = 0.5$. We compare the performance by $\sum_n w_n \log r_n$, where w_n is the weight of n, and r_n is the throughput or timely-throughput of n, depending on whether n is a non-real-time client or a real-time one.



(a) Total weighted log throughput

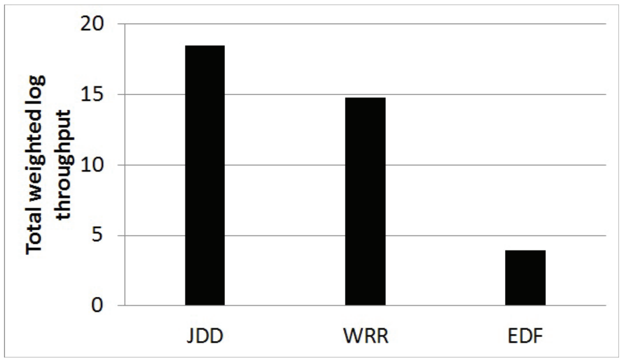
(b) Performance of both non-real-time and real-time flows

Fig. 4: Performance comparison when all flows generate traffic using TCP

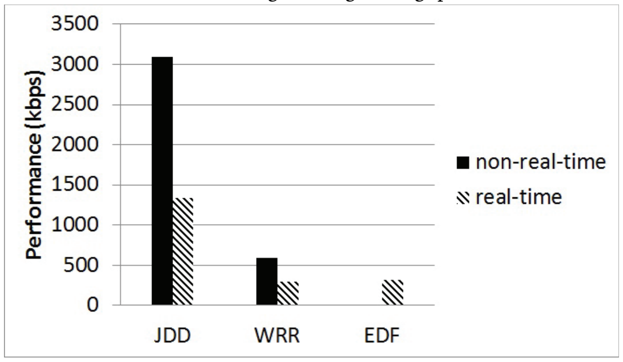
We choose $\sum_n w_n \log r_n$ as the performance metric because a policy achieves weighted proportional fairness by maximizing $\sum_n w_n \log r_n$. In addition, we also compare the total throughput of non-real-time flows and the total timely-throughput of real-time flows.

We consider two different congestion control protocols in the upper-layer, TCP and the TCP-Friendly Rate Control protocol. Simulation results for these two protocols are shown in Fig. 4 and Fig. 5, respectively. It can be shown that the JDD policy outperforms the other two policies by large margins. The WRR policy achieves poor performance because it is unaware of the channel qualities and spends too much time on clients with poor channel qualities. On the other hand, although the JDD policy does not assume any knowledge on the channel qualities, it indirectly observes the channel qualities by estimating the time taken for each transmission, and then integrates this information in the scheduling decisions. Hence, the JDD policy achieves much better performance. Finally, the EDF policy focuses on real-time flows and inevitably starves non-real-time flows. In addition to poor performance, the EDF policy is not incentive compatible as non-real-time flows can benefit from pretending to be real-time ones.

Finally, we demonstrate that the JDD policy can work under various kinds of channels. We add another pair of sender and receiver into the system. They are $30m$ from the BS, and hence the BS now suffers from channel con-

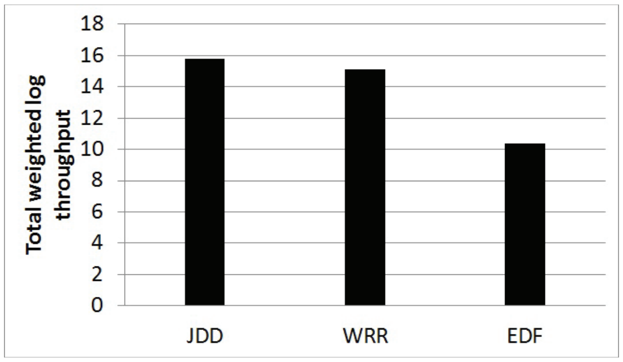


(a) Total weighted log throughput

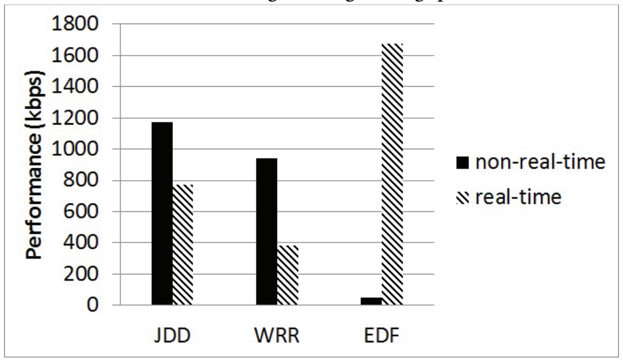


(b) Performance of both non-real-time and real-time flows

Fig. 5: Performance comparison when all flows generate traffic using the TCP-Friendly Rate Control protocol



(a) Total weighted log throughput



(b) Performance of both non-real-time and real-time flows

Fig. 6: Performance comparison when there are interfering networks present

tention and transmission collisions. We place the clients so that their distances to the BS are evenly distributed between $70m$ and $610m$. We assume that all flows use TCP as the congestion control protocol. Simulation results are shown in Fig. 6. It can be observed that the JDD policy outperforms the other two. These results suggest that the JDD policy is versatile and can provide satisfactory performance under various upper-layers and lower-layers scenarios.

V. Conclusions

We have proposed a novel downlink scheduling policy, the JDD policy, for wireless systems where real-time flows and non-real-time flows coexist. The JDD policy satisfies the important properties including incentive compatibility, versatility, deadline awareness, and work conservation. In particular, the JDD policy makes non-real-time flows have high throughputs in the exchange of large delays, while real-time users enjoy the per-packet delay guarantees by sacrificing throughputs. Further, the JDD policy is easily implementable, and can work with a wide range of upper-layers and lower-layers protocols. Simulation results have demonstrated that the JDD policy outperforms other policies under various scenarios.

References

[1] C. Uc-Rios and D. Lara-Rodriguez, "An efficient scheduler for real and non-real time services maximizing satisfied users in wireless networks," in *Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–5, 2011.

[2] H. Haci, H. Zhu, and J. Wang, "Novel scheduling for a mixture of real-time and non-real-time traffic," in *Proceedings of IEEE Globecom*, 2012.

[3] S. Choi and K. Shin, "A unified wireless lan architecture for real-time and non-real-time communication services," *IEEE/ACM Transactions on Networking*, vol. 8, pp. 44–59, Feb. 2000.

[4] J. Jaramillo and R. Srikant, "Optimal scheduling for fair resource allocation in ad hoc networks with elastic and inelastic traffic," *Networking, IEEE/ACM Transactions on*, vol. 19, pp. 1125–1136, Aug. 2011.

[5] J. Widmer, R. Denda, and M. Mauve, "A survey on tcp-friendly congestion control," *Network, IEEE*, vol. 15, pp. 28–37, May 2001.

[6] Cisco, "Diffserv: The scalable end-to-end quality of service model," 2005.

[7] Y. Bernet, P. Ford, R. Yavatkar, F. Baker, L. Zhang, M. Speer, R. Braden, B. Davie, J. Wroclawski, and E. Felstaine, "A framework for integrated services operation over Diffserv networks," 2000.

[8] Tranzeo Wireless Technologies, "WiMAX QoS classes," 2010.

[9] S. Mangold, S. Choi, P. May, O. Klein, G. Hiertz, and L. Stibor, "IEEE 802.11e wireless LAN for quality of service," in *Proc. of European Wireless*, pp. 32–39, 2002.

[10] L. Le, E. Hossain, and A. Alfa, "Service differentiation in multi-rate wireless networks with weighted round-robin scheduling and arq-based error control," *IEEE Transactions on Communications*, vol. 54, pp. 208–215, Feb. 2010.

[11] H. Kim, K. Kim, Y. Han, and S. Yun, "A proportional fair scheduling for multicarrier transmission systems," in *Proc. of IEEE Vehicular Technology Conference (VTC2004-Fall)*, vol. 1, pp. 409 – 413, sept. 2004.