ABRC: An End-to-End Rate Adaptation Scheme for Multimedia Streaming over Wireless LAN*

Wei Wang, Soung C. Liew, Jack Y. B. Lee
Department of Information Engineering
The Chinese University of Hong Kong
Shatin, N. T., Hong Kong
{wwang2, soung, yblee}@ie.cuhk.edu.hk

Abstract

The rapid growth of wireless LAN (WLAN) deployments will bring about many novel mobile applications. Among them will be real-time multimedia streaming applications running on UDP, which may interfere with current data applications running on TCP. This paper is a first attempt to investigate how to ensure the performance of these two groups of applications when they co-exist over a WLAN. Toward this end, we have designed and implemented a UDP rate adaptation scheme called Adaptive-Buffer Rate Control (ABRC) for multimedia streaming over WLAN. ABRC has two distinguishing features compared with other schemes: 1) It can achieve arbitrary bandwidth allocations between UDP and TCP in the WLAN, as opposed to previously proposed "TCP friendly" schemes, which can only achieve uniform bandwidth allocations; 2) The majority of previously proposed flexible bandwidth-allocation schemes achieve arbitrary bandwidth allocations by prioritizing and scheduling packet transmissions within network equipment (i.e., within routers, base stations, etc.). In contrast, ABRC is an end-to-end application-layer solution that does not require changes to current WLAN products, making it more readily deployable over existing networks.

Keywords – Rate control, WLAN, 802.11, Multimedia streaming, Bandwidth allocation, TCP-friendly protocols, TFRC

1. Introduction

In the last few years, wireless networks based on the IEEE 802.11 standard have been undergoing accelerated deployment in homes, enterprises, and public hotspots such as airports and hotels. The majority of the wireless LAN (WLAN) deployments use the Infrastructure Mode of 802.11, in which the terminal devices communicate with each other as well as with machines in the wired networks via an access point (AP). An AP usually has two interfaces: an 802.11 interface for communication in the WLAN, and an Ethernet interface for connection to the wired Internet infrastructure.

Currently, the prevalent applications in WLANs are TCP data applications such as web browsing, email delivery, and file downloading. Within a WLAN, these applications generate mostly downlink traffic from the AP to the wireless stations. The upstream traffic is generally much lighter. Due to the relatively low bandwidth of the 802.11 interface compared with the Ethernet interface (e.g., the maximum data rate of 802.11b is 11 Mbps while the maximum data rate of a typical Ethernet is 100 Mbps), the AP could become a significant bottleneck for downlink traffic.

Although the popular applications on WLANs today are TCP data applications, we anticipate that as WLANs become even more widespread and ubiquitous, real-time applications such as voice over IP and video streaming will also be in demand. These applications typically use UDP as the transport protocol and have stringent throughput and delay requirements. An important issue is how to satisfy the requirements of real-time applications within the WLAN and at the same time ensure harmonious co-existence with other TCP data applications.

For harmonious co-existence with TCP, the real-time applications could employ TCP-friendly protocols on top of UDP (e.g., see [1]). These protocols can ensure that the bandwidth is shared in a fair manner among TCP and UDP streams. Essentially, the bandwidth is divided evenly among all streams. When there is no competing TCP traffic, UDP streams can also fully make use of the available WLAN bandwidth. This makes it possible to support bandwidth-adaptive applications over the TCP-friendly protocols (e.g., video streaming in which the bit rate is adapted according to the bandwidth available).

TCP-friendly protocols, however, suffer from two shortcomings. First, one cannot fine-tune the proportions of bandwidths allocated to real-time applications and TCP applications in a non-uniform manner. That is, one cannot allocate more (or less) bandwidths to real-time applications than TCP applications. Second, most TCP-friendly protocols proposed to-date either attempt to mimic the behavior of TCP [2], or explicitly adjust its sending rate based on equations which model the throughput of the TCP flow under the same condition of round-trip time and packet loss rate [3]. One side effect of emulating TCP behavior is that such TCP-friendly protocols will also inherit certain shortcomings of TCP. For example, when a packet loss is detected, TCP-friendly protocols may reduce the input data rate (as in regular TCP) regardless of whether the packet loss is due to traffic

* This work is sponsored by the Areas of Excellence scheme established under the University Grant Committee of the Hong Kong Special Administrative Region, China (Project Number AoE/E-01/99).

congestion or random noise in the wireless medium. In the latter case, the input data rate should not be reduced because doing so does not solve the packet loss problem and may degrade the quality of service unnecessarily.

To allocate bandwidths in WLAN, the most direct way is to schedule packet transmissions on the AP downlink. Scheduling schemes such as Priority Queuing and Weighted Fair Queuing (WFQ) implement multiple queues for different flows or traffic classes [5][6]. However, the AP buffers in most of today's commercial products are simple FIFO queues that are shared by all traffic. We may have to live with such FIFO queues for some time to come considering the large amount of such 802.11 equipment already deployed.

To address the problems and issues outlined above, we propose in this paper an *end-to-end* UDP rate adaptation scheme called Adaptive-Buffer Rate Control (ABRC) that can achieve arbitrary bandwidth allocations between UDP and TCP at the AP. We have implemented and evaluated its performance in a real WLAN testbed. Compared with previous work, ABRC has two major distinguishing features:

1) It achieves desired bandwidth allocations in a totally end-to-end way. That is, it can be applied to the current 802.11 commercial products without any modifications on them, making it more readily deployable over existing networks. To the best of our knowledge, ABRC is the first *end-to-end* approach proposed to provide bandwidth allocations between UDP and TCP traffics in WLAN.
2) It achieves "TCP-friendliness" in a different way than the previous TCP-friendly schemes. As with these TCP-friendly schemes, ABRC can make sure that UDP streams do not hog the bandwidth at the expense of TCP connections; and when there are no competing TCP streams, UDP streams can fully make use the entire WLAN bandwidth. However, ABRC does this without trying to mimic the behavior of TCP. Consequently, it does not inherit problems of TCP in wireless networks.

2. Adaptive-Buffer Rate Control Scheme

To explain the ABRC scheme, we consider the network set-up as shown in Fig. 1. We assume that the ABRC sender is located close to the AP and connected to it through an Ethernet. An example of an ABRC sender is a video server. If the original video source is at a remote location, the server shown in Fig. 1 could be a local proxy server that obtains the video data from the original video source for feeding to the receivers within the WLAN.

The TCP sender could be a server for FTP or other TCP-based applications. Although the TCP sender is shown in Fig. 1 as being close to the WLAN and connected to it through an Ethernet, it could in general be located at a distance within the Internet.

ABRC is an application-layer protocol over UDP. In ABRC, the receiver collects data and statistics on incoming packets and feed them back to the sender at regular intervals. Based on the feedback, the sender estimates the buffer occupancies of TCP and UDP traffic in the AP downlink queue, from which the relative bandwidth usage between TCP and UDP is obtained. The sender then compares the relative bandwidth usage with the target bandwidth allocation and adjusts its sending rate accordingly in a dynamic fashion.

Specifically, the tasks performed by the ABRC scheme can be broken down as follows: 1) Estimation of TCP and UDP buffer occupancies to obtain the current relative bandwidth usage; 2) Adjust the sending rate based on 1) to achieve the desired relative bandwidth allocations.

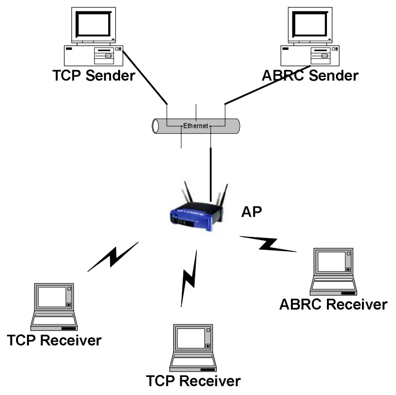


Figure 1. Network setup for ABRC scheme

2.1 Relationship between Bandwidth Usage and Buffer Occupancies

ABRC attempts to adjust the buffer occupancy of its UDP packets at the AP to achieve the desired bandwidth allocation with respect to the co-existing traffic. Denote the buffer occupancy of TCP and UDP by B_{TCP} and B_{UDP}, respectively. If the queue at the AP is FIFO, it is obvious that

$$\lambda_{UDP} / \lambda_{TCP} = B_{UDP} / B_{TCP} \tag{1}$$

where λ_{TCP} and λ_{UDP} are the throughputs (or output rates) of TCP and UDP at the queue, respectively. Denote the desired ratio of bandwidth allocation by

$$k_T = \lambda_{UDP} / \lambda_{TCP} \tag{2}$$

If we knew the value of B_{TCP}, then the target buffer occupancy for UDP traffic needed to achieve the desired bandwidth-allocation ratio is

$$B_T = k_T \, B_{TCP} \tag{3}$$

In practice, due to the dynamic nature of TCP operation, the value of B_{TCP} will change over time. A key ingredient in ABRC is a method for tracking such change so that the target buffer occupancy for UDP, B_T, can be adjusted accordingly to maintain the desired k_T. This is detailed in the next two subsections.

2.2 Estimation of Buffer Occupancy

To estimate the UDP buffer occupancy the sender adds a sequence number to every packet sent. In the ith feedback packet, the receiver provides the sequence number $Seq_{recv}[i]$ of the most recently received packet. Upon receiving the feedback, the sender estimates the UDP buffer occupancy at the end of ith feedback interval by

$$B_{UDP}[i] = Seq_{send}[i] - Seq_{recv}[i] \tag{4}$$

where $Seq_{send}[i]$ is the most recent packet sent out by the sender when the ith feedback packet is received. Note that (4) is only an approximate upper bound because when the sender receives the feedback, the current Seq_{send} could be larger than the Seq_{send} when Seq_{recv} was measured on the receiver. The inaccuracy of (4) is caused by the transmission time of the feedback packet. (the transmission time as defined here includes all the overhead incurred by the 802.11 protocol, such as backoff, physical preamble and InterFrame Space, etc) From our experiments, we have observed that the error in (4) is only one or two packets.

To estimate the TCP buffer occupancy, ABRC makes use of packet inter-arrival times registered by the receiver. Suppose that the AP queue were filled by UDP packets only, the inter-arrival time for two consecutive UDP packets observed by the receiver would be the transmission time of one UDP packet, denoted as T_{UDP} ,where T_{UDP} also includes the various 802.11 protocol overheads.

When the AP queue has a mix of UDP and TCP packets, the *average* inter-arrival time of the two consecutive UDP packets observed by the receiver, T_{mix}, will be larger than T_{UDP}. T_{mix} is related to the ratio of TCP and UDP buffer occupancies.

At the end of the ith feedback interval, if that the buffer occupancy of TCP is twice that of UDP (i.e., $B_{TCP}[i] = 2B_{UDP}[i]$), then the average inter-arrival time of UDP packets in the ith feedback interval, $T_{mix}[i]$ should be the transmission time of a UDP packet plus two transmission times of a TCP_DATA packet plus α times the transmission time of one TCP_ACK packet. Generally, $\alpha = 1$ or 2, depending on whether the TCP sends a TCP_ACK for every packet or every two packets. The reason for adding the transmission time of TCP_ACK as part of $T_{mix}[i]$ is that the wireless channel is shared by all wireless stations. Therefore, the sending of upstream TCP_ACKs will also increase the delay of the downstream packets. If we ignore the transmission time of TCP_ACK, since it is relatively small compared with TCP_DATA, then approximately $T_{mix}[i]$ has a direct relationship with the buffer occupancies as follows:

$$T_{mix}[i] = T_{UDP} + k[i]T_{TCP} \tag{5}$$

where $k[i] = B_{TCP}[i]/B_{UDP}[i]$ is the ratio between the TCP and UDP buffer occupancies for the ith feedback interval. If the TCP has the same packet length of UDP, that is, $T_{TCP} = T_{UDP}$, then[1]

$$T_{mix}[i] = (1 + k[i])T_{UDP} \tag{6}$$

In the implementation, we can simply use the minimum inter-arrival time observed by the receiver as an estimation of T_{UDP}, counting on the fact that occasionally, two UDP packets will be transmitted in succession even when there are TCP packets in the AP queue. So when the receiver measures $T_{mix}[i]$, the TCP buffer occupancy of the ith feedback interval can be estimated by

$$B_{TCP}[i] = \frac{T_{mix}[i] - T_{UDP}}{T_{UDP}} B_{UDP}[i] \tag{7}$$

2.3 Sending Rate Adjustment

Equation (7) provides us with a way to track the change in TCP buffer occupancy. The ABRC sender can then dynamically adapt its sending rate to maintain a certain ratio k_T between UDP and TCP buffer occupancies. The sending rate adjustment is a two-step process. The first step is to set up a target UDP buffer occupancy B_T to follow the changes of the TCP buffer occupancy so that the ratio between the UDP and TCP buffer occupancies can be maintained at or zoomed into k_T (i.e., to achieve equality in (3)). The second step is to adjust the sending rate to let the UDP buffer occupancy narrow into B_T upon receiving the feedback from the receiver.

For the first step, the sender can estimate the current TCP buffer occupancy $B_{TCP}[i]$ on receiving the ith feedback packet based on (7). When it is time to adjust the target buffer occupancy, the B_T can be set to $B_T = k_T B_{TCP}[i]$, according to (3). We may want to make B_T adjustment

[1] Note that strictly speaking the assumption of equal length for UDP and TCP packets is not necessary. If we define buffer occupancies in terms of numbers of bytes (or amounts of work) rather than packets, equation (6) will remain valid even for UDP and TCP packets of different size.

interval several times the feedback interval so as to leave time for the ABRC to achieve the previously set B_T.

For the second step, besides sequence number and inter-arrival time, the receiver also measures the UDP throughput over a feedback interval t by counting the number of packets received during the t seconds. Let BW_i denote the throughput (in packets/second) of the ith feedback interval. Note that BW_i is related to $T_{mix}[i]$ by $BW_i = 1/T_{mix}[i]$.

When the sender receives the ith feedback packet, it estimates the bandwidth of the $(i+1)th$ feedback interval as BW^*_{i+1}. Similar to the bandwidth estimations of many other network protocols, BW^*_{i+1} can be simply set to BW_i; or alternatively, a window average or an exponential average on BW_i, BW_{i-1}, …, can be used as the estimate, with the averaging/smoothing period set according to the responsiveness desired and the bandwidth fluctuations observed.

From the feedback, the sender also derives the current UDP buffer occupancy $B_{UDP}[i]$ according to (4). For the $(i+1)th$ feedback interval, we expect the number of UDP packets leaving the AP queue to be $BW^*_{i+1} * t$. Let S_{i+1} be the sending rate during the $(i+1)th$ interval. Then, the number of UDP packets entering the AP queue during the $(i+1)th$ interval is $S_{i+1} * t$.

The change in buffer occupancy is $S_{i+1} * t - BW^*_{i+1} * t$. The target change in buffer occupancy is $B_T - B_{UDP}[i]$. Equating these two terms and rearranging, we have

$$S_{i+1} = (B_T + BW^*_{i+1} * t - B_{UDP}[i])/t \qquad (8)$$

3. Experimental Evaluation

To evaluate the performance of the ABRC scheme, we have implemented and tested it in a real network configured as in Fig. 1. The TCP sender and receivers are FTP server and clients.

The ABRC implementation uses a feedback interval of one second (i.e., $t = 1$) and B_T adjustment interval of three seconds. For bandwidth estimate of the next interval BW^*_{i+1}, we simply let $BW^*_{i+1} = BW_i$, the bandwidth registered by the receiver for the current feedback interval. This makes the ABRC scheme somewhat aggressive in response to bandwidth changes.

The interactions between ABRC sender and receiver are summarized below:

1) For each feedback interval over one second, the receiver measures the average throughput of the ABRC stream, collects the sequence number of the most recently received packet. This information is fed back to the sender at the end of the one second.
2) After receiving a feedback packet, the sender estimates the UDP buffer occupancy $B_{UDP}[i]$ and the average bandwidth of the next second BW^*_{i+1}.
3) Every three seconds, the sender estimates the TCP buffer occupancy B_{TCP}. It then adjusts the B_T to $k_T * B_{TCP}$, where k_T is the desired bandwidth allocation. If the new computed B_T is below a threshold 5 (i.e., $B_T < 5$), let $B_T = 5$. This is to deal with the case when there is no competing TCP traffic in the AP. By maintaining a minimum B_T, ABRC scheme can make full use of the wireless channel while others are not using it.
4) Every second, adjust the sending rate according to (8).

3.1 Bandwidth Allocation Results

The first set of our experiments explores whether ABRC allocates bandwidths according to the targets. Table 1 shows the results with one ABRC connection and one FTP connection. Table 2 shows the results with one ABRC connection and two FTP connections. We see that for most of the cases ABRC can achieve the desired bandwidth allocation rather well.

Table 1. One ABRC coexists with one TCP connection

Bandwidth allocation factor k_T	ABRC throughput (Mbps)	TCP throughput (Mbps)
1	2.53	2.58
2	3.79	1.66
3	4.20	1.36

Table 2. One ABRC coexists with two TCP connections

Bandwidth allocation factor k_T	ABRC throughput (Mbps)	TCP 1 throughput (Mbps)	TCP 2 throughput (Mbps)	Total TCP throughput (Mbps)
1	2.53	1.53	1.16	2.69
2	3.74	0.85	0.87	1.72
0.5	1.48	2.05	1.72	3.77

We would like to point out that ABRC treats all TCP traffic as a whole regardless of how many actual TCP connections there really are. As a result, ABRC will allocate bandwidth to the UDP stream with respect to the total bandwidth of all other TCP streams rather than with respect the bandwidth of a particular TCP stream. As a matter of fact, in general it allocates bandwidth to the ABRC stream with respect to sum total of all other traffic, including other UDP streams as well, if any. Thus, the value of k_T is the ratio of the bandwidth allocated to the ABRC stream with respect to aggregate bandwidth of all other streams.

3.2 Comparison with a TCP-friendly Congestion Control Scheme

In our second set of experiments, we compare the dynamic behaviors of ABRC with the well-known TCP-friendly Rate Control (TFRC) scheme [3]. TFRC is an equation-based TCP-friendly congestion control mechanism which adjusts its sending rate as a function of the measured rate of loss events, where a *loss event* consists of one or more packets dropped within a single round-trip time. The reader is referred to [3] for details on TFRC. The TFRC implementation used in our experiments was downloaded from its official website [4] with some modifications on the receiver to record the throughputs over time.

We first examine the operations of TFRC and ABRC in isolation. Figure 2 shows the protocol behaviors of TFRC and ABRC when each of them is the only traffic stream within the WLAN. The bandwidths recorded over time are the numbers of bits received by the receiver over successive one-second intervals.

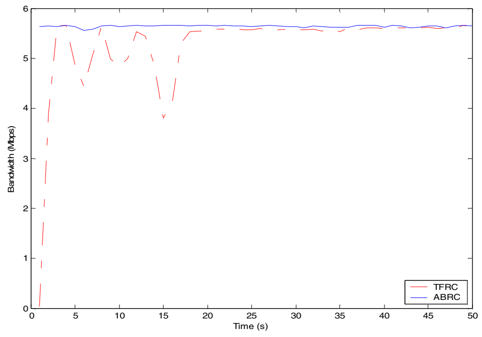
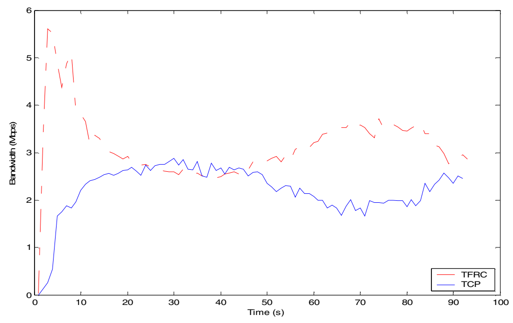


Figure 2. Dynamic bandwidth usage of TFRC and ABRC when there is no other competing traffic

Similar to TCP, TFRC also has the slow start and congestion control phases. As can be seen in Fig. 2, after being started, TFRC needs more than 20 seconds to zoom to full utilization of the WLAN bandwidth. This is due to its design principle of not aggressively seeking out available bandwidth. The time needed for achieving the equilibrium is rather long, considering that the RTT is rather short in this experiment. Usually, RTT is no more than several hundreds of milliseconds.

In contrast, ABRC makes full use of the WLAN bandwidth right from the start of the experiment. This is attributed to the operation of ABRC in which it tries to keep to AP queue nonempty at all time so that the AP always has some packets to transmit.



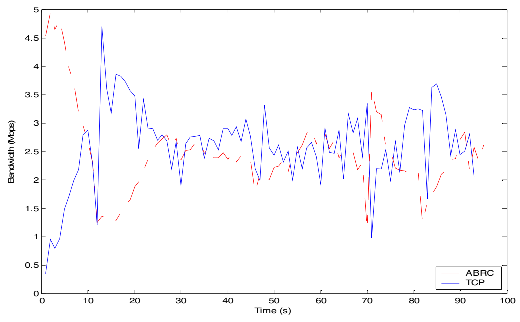


Figure 3. Dynamic bandwidth usage of TFRC /ABRC versus bandwidth usage of competing TCP traffic

Figure 3 shows the bandwidth-usage evolutions for TFRC and ABRC when there is one competing TCP connection. Here the bandwidth allocation factor k_T of ABRC is set to one so as to make ABRC comparable with TFRC.

The RTT within WLAN is quite small so that TCP's sending rate is limited by the advertised window rather than by the congestion window. That is, the maximum value of advertised window (64 Kbytes for the socket API we used) is quickly reached while the congestion window is still increasing. TFRC does not incorporate an advertised window limit. It only tries to emulate throughput limitation of TCP caused by congestion window limit. Thus, even after TCP has reached its advertised window limit, TFRC continues to increase its bandwidth usage until it causes buffer overflow at the AP.

Before the buffer overflows and after the advertised window limit is reached (between 45 sec and 75 sec in Fig. 3), TFRC can grab more bandwidth than TCP. The throughput of TCP can be estimated by Max Advertised Window divided by RTT. Throughput of TCP decreases because Max Advertised Window remains constant while RTT increases – RTT increases because TFRC continues to flood the AP buffer and causes its total occupancy to continue to increase. After 75 sec, AP buffer overflow occurs, and both packets from TCP and TFRC start to be dropped. Upon detection of lost packets, TRFC immediately reduces its rate. Although TCP also decreases its congestion window upon packet loss detection, the decreased congestion window is still larger than the advertised window. Meanwhile, the RTT has decreased because TFRC has already decreased its sending rate. Thus, the throughput of TCP increases (between 75 sec and 90 sec). As can be seen in Fig. 3, for a sustained period of time (between 50 sec and 95 sec), the bandwidth usage is not allocated in a uniform and "fair" manner, contrary to the design objective of TFRC.

In contrast, as illustrated in the right part of Fig. 3, ABRC tracks and follows the changes of TCP much more closely. Although there are still fluctuations, the periods are much smaller (or frequency components much higher). The implication is as follows. Take a video server as an example. Suppose that VBR (variable bit rate) encoding is used to adapt the video bit rate to the sending rate. Unless very large buffers are used at the encoder and the decoder (with the consequence of higher delays at the encoder and decoder), one cannot smooth the fluctuations in bandwidth usage of TFRC because the fluctuations have very low frequency components. As a result, the video quality tends to vary periodically over time, despite there is only one competing TCP connection. This variation of video quality is self-induced rather than due to changes in network activity.

Although not shown here due to limit space, our experimental results beyond 90 sec on the left part of Fig. 3 exhibit continuous long-term oscillatory behavior. The average throughput of TFRC is higher than that of TCP. TFRC in fact fails to achieve uniform bandwidth allocations with TCP within the WLAN.

With ABRC, the oscillations have much smaller periods. As a result, a smaller buffer can be used at the encoder and decoder to smooth out the fluctuations, and the video quality can be maintained at a more consistent level than that with TFRC. In addition, the targeted uniform bandwidth allocation in this case is achieved after averaging out the oscillations of ABRC and TCP.

4. Discussions

Although the competing traffic in this paper is TCP, the derivation in Section 2 is independent of the particulars of TCP. Therefore, ABRC is not limited to the case of UDP coexisting with TCP. ABRC can also be used when there are other co-existing traffic types, including other ABRC streams.

When there are more than one ABRC streams, additional constraints need to be met to maintain a stable system. For a particular ABRC stream i, let us define $\beta_i = k_T/(k_T+1)$, where β_i is the proportion of the total bandwidth in the WLAN allocated to the ith ABRC stream. For example, if k_T is one, then the ABRC stream will be allocated 50% of the total bandwidth.

If there are n ABRC connections competing with several TCP connections, for system stability, $\sum_{i=1}^{n}\beta_i < 1$. The remaining bandwidth proportion $1-\sum_{i=1}^{n}\beta_i$ is fairly shared by TCP connections.

From the analysis in the previous section, it is clear that ABRC will not have the shortcomings of TCP and TCP-friendly protocols in wireless networks, such as degraded throughput due to the inability to distinguish between packet losses due to traffic congestion and random bit error. This is because unlike TFRC, ABRC does not adjust its sending rate based on detection of lost packets, and therefore lost packets due to random wireless noise will not cause it to decrease its sending rate. Our experiment results also validate this. Due to the page limit, we will present such results in a separate paper.

5. Conclusions

In this paper, we have presented an end-to-end rate control protocol called ABRC to achieve any desired bandwidth allocations between UDP and TCP traffic in the WLAN. The design of ABRC is based on an analytical framework. To test its performance in practice, we have implemented ABRC in a WLAN testbed. The experiment results validate the ability of ABRC to achieve good bandwidth-allocation performance as intended.

We have also compared the performance of ABRC with a typical TCP-friendly rate control protocol named TFRC. Our results show that ABRC does not have a number of the shortcomings of TFRC in wireless networks, such as imprecise bandwidth allocation, long-term oscillatory behavior, and unnecessary throughput degradation due to random wireless noise.

Last but not least, unlike many previous schemes proposed in the literature, ABRC achieves bandwidth allocations in a totally end-to-end way so that it can be applied to the current installed base of commercial WLAN products without any modifications on them. As an application-layer protocol, it is readily deployable over the existing Internet infrastructure.

References

[1] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the Internet," *Networking, IEEE/ACM Trans. on*, Vol. 7, Aug. 1999, pp. 458-472

[2] R. Rejaie, M. Handley and D. Estrin, "RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the Internet," *Proc. INFOCOM'99* , vol. 3, Mar. 1999, pp. 1337-1345

[3] S. Floyd et al., "Equation-based congestion controlfor unicast applications," *Proc. ACM SIGCOMM*, Aug. 2000, pp. 43-56.

[4] http://www.icir.org/tfrc/

[5] Law, K. L. E., "The bandwidth guaranteed prioritized queuing and its implementations," *Proc. GLOBECOM'97,* vol. 3, Nov. 1997, pp. 1445-1449.

[6] M. Katevenis, S. Sidiropoulos and C. Courcoubetis, "Weighted round-robin cell multiplexing in a general-purpose ATM switch chip," *IEEE JSAC*, Oct. 1991.

[7] Y. R. Yang, Nin Sik Kim and S. S. Lam, "Transient behaviors of TCP-friendly congestion control protocols," *Proc. INFOCOM'01*, vol. 3, Apr. 2001, pp. 1716-1725.