# Improving VoIP Call Capacity over IEEE 802.11 Networks

Yeonsik Jeong[*†], Sandeep Kakumanu[*], Cheng-Lin Tsao[*], and Raghupathy Sivakumar[*]

[*]School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, Georgia
[†]Department of Multimedia System Engineering, SungKongHoe University, Seoul, Korea
Email: {ysjeong, ksandeep, cltsao, siva}@ece.gatech.edu, ysjeong@mail.skhu.ac.kr

*Abstract*— **The expected VoIP call capacity in a one hop IEEE 802.11b network with G.711 voice codec is about 85 simultaneous calls, but the actual observed capacity is only 5 calls even at the highest data rate and under zero loss conditions. In this paper we analyze the reasons behind this inferior performance of VoIP traffic. We also present algorithms at the medium access control layer to improve the observed call capacity. Finally, using ns-2 based simulations, we evaluate the algorithms and show that performance improvements of up to 300% can be achieved.**

## I. Introduction

Voice over IP (VoIP) services have been significantly gaining prominence over the last few years because of a number of impressive advantages over their traditional circuit-switched counterparts including but not limited to high bandwidth efficiency, low cost, and flexibility of using various compression strategies. Simultaneously, the use of wireless networks has also grown tremendously over the past years. Wireless LAN (WLAN) solutions, armed with better physical layer technologies, now promise high data rates of more than 100 Mbps such as IEEE 802.11n.

In this context, recent efforts have focused on marrying the potential benefits of VoIP and WLANs to provide wireless telephone services. A natural question that then arises is *how well does VoIP perform over WLAN environments*? The answer to this question is counter-intuitive. Even though the WLANs boast very high data rates and a typical VoIP call carries only 128 Kbps of bidirectional data (using G.711 voice codec), the number of VoIP calls supported by these networks is abysmally low. An IEEE 802.11b network for example can support only 5 VoIP calls even at the highest data rate of 11 Mbps, as we will show later. VoIP traffic in WLANs is characterized by its small frame sizes, and IEEE 802.11 MAC is notorious for very poor performance for small frame sizes. For small frames the overheads at the different layers of the network stack themselves pose a significant burden. Added to this IEEE 802.11 MAC protocol exhibits other distributed effects that further degrade the VoIP call capacity.

Toward the goal of improving the VoIP call capacity in wireless networks we first study the reasons behind the inferior performance of VoIP over WLANs. We identify the universal set of components that could be improved to increase VoIP call capacity over such environments. Finally, we select the three dominant components and present algorithms that improve the performance of the components, thereby increasing the VoIP call capacity. While most of the discussions in the paper apply to any of the IEEE 802.11 class of WLAN environments, we present all our discussions only in the context of IEEE 802.11b for clarity.

The contributions of this paper are thus twofold:

- We first provide testbed results for the actual VoIP call capacity in an IEEE 802.11b network; we then perform mathematical analysis for the call capacity in an IEEE 802.11 network to explain the poor performance observed, and identify three dominant components that can be exploited to improve the call capacity.
- We then present three algorithms that improve the characteristics of the three components, and thereby lead to an increase in the VoIP call capacity of IEEE 802.11 WLANs.

The rest of the paper is organized as follows: Section II presents the testbed results for actual observed VoIP call capacity, and explains the results through mathematical derivations; Section III presents three algorithms at the MAC layer that can help in improving call capacity; Section IV evaluates the algorithms, while Sections V and VI respectively present related work and the conclusions.

## II. Analysis of VoIP Call Capacity

Intuitively an 802.11b 11 Mbps channel should be able to support up to 85 G.711 VoIP calls. But in reality we observe that only meager 5 calls are supported. In this section we first give experimental results of the call capacity in an 802.11b network and then mathematically analyze the reasons for such a poor performance of VoIP traffic over an 802.11b network.

### A. Testbed Implementation

*1) Experimental Setup:* We implement a testbed to study the call capacity of an 802.11b network. The experimental setup of the testbed is shown in Fig. 1. A total of six desktops, two laptops, two routers, and one 802.11b AP are used for the testbed. All the machines run Linux operating system. The testbed consists of two domains: a wired domain and a wireless domain. To emulate multiple calls in the wireless domain we use three wireless interface cards in a single machine

This work is supported in part by the National Science Foundation under grant CNS-0519733 and the Georgia Tech Broadband Institute.

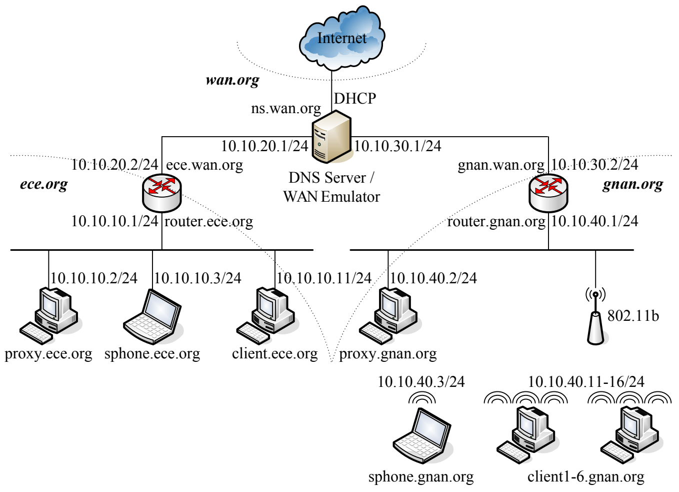


Fig. 1. Experimental setup of the testbed.

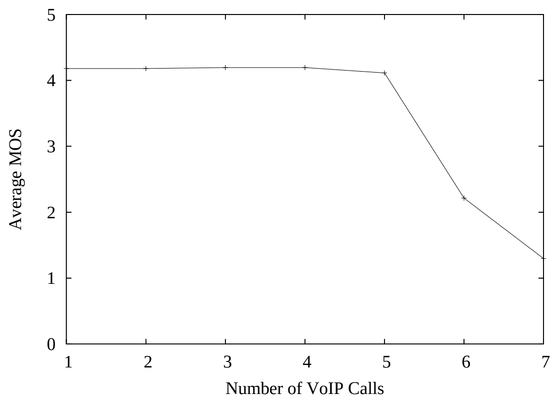


Fig. 2. Average MOS observed.

and run three virtual machines on the physical machine, each associated with a different wireless card. Two such machines have three wireless cards, respectively. The laptop connected to the AP is used as a real VoIP phone, and a VoIP call using the KPhone [1] and SIP Express Router [2] is setup between this laptop and another laptop connected to the wired domain. Other calls are emulated using bidirectional constant bit rate (CBR) traffic generated by Iperf [3]. The frame size of each UDP packet generated by Iperf is set to be 92 bytes (including 12 byte RTP header) and the data rate is set to be 73.6 Kbps.

*2) Call Capacity Observation:* To study the maximum number of VoIP calls supported by the 802.11b AP we setup one call between the two laptops and add more calls using Iperf from different virtual machines to a fixed client on the wired domain. We observe that when the total number of calls is 4 there is no noticeable loss and delay in the network and the VoIP call does not show any significant degradation in quality. When the number of calls increases to 5 we observe that the calls start experiencing a little loss due to contention and packet drops at the queue. But the VoIP calls suffer little distortion in quality. When we further increase the number of calls to 6, the loss and delay increases and the VoIP calls become unintelligible. Fig. 2 shows how the average mean opinion score (MOS) [4] changes as the number of calls increases. The theoretical highest MOS is 4.41 due to inherent degradation that occurs when converting an actual voice conversation to a network signal and back and the minimum acceptable value is 3.6. From the graph we can conclude that the maximum number of VoIP calls that can be supported by the AP is 5. Similar testbed studies have been performed in related literature [5] [6]. Nevertheless our testbed results serve as a tested baseline for the rest of the paper.

## B. Theoretical Analysis

We now analyze the reasons for the inferior performance of VoIP over 802.11 network. For the theoretical analysis, we make the following assumptions:

- *Number of wireless nodes*: there is only one wireless node, so there is no collision.
- *Loss rate*: the link is error-free.
- *Traffic*: there are always frames available in MAC queue of AP, i.e there is a saturated traffic condition.
- *Voice codec*: G.711 voice codec is used for encoding voice. 80 byte frames are sent once every 10 ms. Hence the rate of one direction in a call is 64 Kbps.

We define several terms in order to characterize the frame rates and number of VoIP calls.

- Maximum Frame Rate (MFR) captures the frame rate that we can expect at each layer $L$.
- minimum Required Transmission Delay (mRTD) gives time taken to transmit the protocol data unit (PDU) at the corresponding layer.
- Maximum Number of VoIP Calls (MNVC) at each layer can be calculated as

$$\mathrm{MNVC}(L) = \lfloor \frac{\mathrm{MFR}(L)}{2k} \rfloor = \lfloor \frac{1}{2k \cdot \mathrm{mRTD}(L)} \rfloor, \qquad (1)$$

where $k$ is the frame rate of codec and in our analysis $k = 100$.

We now look at the capacity degradation due to each layer of the network stack.

*1) Application Layer Capacity:* The application layer capacity indicates the ideal throughput that can be achieved from a given link. If $D$ is the frame size, in bytes, generated from voice codec and $R$ is the transmission rate, in bits per second, of physical layer, we can derive the MFR for the application layer as

$$\mathrm{MFR(APP)} = \frac{R}{8D}.$$

*2) Impact of Transport and Network Layers:* The Transport and Network layers add headers to the frames and thus cause a corresponding degradation in application throughput as

$$\begin{aligned}\mathrm{mRTD(RTP)} &= \mathrm{MFR(APP)}^{-1} + \mathrm{T(RTP)}\\ \mathrm{mRTD(UDP)} &= \mathrm{mRTD(RTP)} + \mathrm{T(UDP)}\\ \mathrm{mRTD(IP)} &= \mathrm{mRTD(UDP)} + \mathrm{T(IP)},\end{aligned}$$

where $\mathrm{T}(L)$ is time taken to transmit header of the layer $L$ and can be calculated as $\frac{8 \cdot \mathrm{H}(L)}{R}$ where $\mathrm{H}(L)$ is the size of header of the layer $L$.

*3) Impact of MAC Layer:* The MAC layer adds considerable overhead to the frame including MAC header, MAC backoff time, MAC ACK, and inter-transmission times (DIFS and SIFS). We can model the mRTD at the MAC layer as

$$\begin{aligned}\mathrm{mRTD}(\mathrm{MAC}) = \mathrm{mRTD}(\mathrm{IP}) + \mathrm{T}(\mathrm{MAC}) \\ + T_{\mathrm{DIFS}} + T_{\mathrm{BO}} + T_{\mathrm{SIFS}} + T_{\mathrm{ACK}},\end{aligned}$$

where $T_{\mathrm{BO}}$ is the average backoff time when there are no other contending stations and can be approximated as $T_{\mathrm{SLOT}} \cdot (CW_{\min} - 1)/2$. $T_{\mathrm{ACK}}$ also includes transmission time ($T_{\mathrm{PHY}}$) of ACK frame. It is at the MAC layer that the maximum degradation to the call capacity occurs.

*4) Impact of Physical Layer:* Overheads of physical layer are long preamble and PLCP header, and they are always transmitted at the basic rate (1 Mbps for 802.11b). The mRTD at the PHY layer thus becomes

$$\mathrm{mRTD}(\mathrm{PHY}) = \mathrm{mRTD}(\mathrm{MAC}) + T_{\mathrm{PHY}}.$$

We observe that the maximum drop in the call capacity occurs at the MAC layer. Thus we observe that even though theoretically 85 calls are possible on an 802.11b 11 Mbps link only 5 calls are actually accommodated. (See TABLE I.)

TABLE I
MNVC AT EACH LAYER IN 802.11B WITH G.711.

| $L$ | APP | RTP | UDP | IP | MAC | PHY |
|---|---|---|---|---|---|---|
| MNVC($L$) | 85 | 74 | 68 | 57 | 6 | 5 |

## III. Algorithm Design

### A. Motivation and Overview

From the arguments in the previous section we can write an integrated equation for the MNVC in an 802.11 network as shown in Eq. 2. Analyzing this equation, we can envisage 5 different schemes to improve the call capacity by leveraging different components in the equation:

- *ACK Aggregation (AA)*: ACK aggregation refers to sending a single ACK for a block of $n$ frames. ACK aggregation results in the *reduction of* $T_{\mathrm{ACK}}$. From TABLE II we see that we can increase the call capacity by up to 2 calls.
- *Frame Aggregation (FA)*: Frame aggregation refers to fusing multiple frames destined to the same end user into a single large frame. TABLE II shows that frame aggregation gives the maximum possible improvement in terms of MNVC. However frame aggregation is associated with an increase in delay of the frames. In section III-C we present an algorithm for frame aggregation that serves as a balance between the MNVC and the expected delay. FA *decreases* $k$ in Eq. 2.
- *Link Adaptation (LA)*: Link adaptation refers to changing the transmission rate for the data frames. The 802.11b standard specifies 4 different data rates that can be used. The prevailing channel conditions affect the choice of data rate. Generally a high rate is used in conditions of low error and vice-versa. Existing rate adaptation techniques do not consider the size of frames while adapting to different rates. VoIP traffic is characterized by small frame size and hence will suffer lower frame error rates than normal traffic for the same channel conditions. Hence there is a strong motivation for a rate adaptation technique that takes into account the frame size. LA can *control* $R$ in Eq. 2 and maintain it optimal for a given channel condition.
- *Time Saving (TS)*: Time saving refers to *reducing* $T_{\mathrm{DIFS}}$ waiting time between two successive frames. From TABLE II we see that the potential benefits as a result of this technique is less than a single call. The reason for this fact is that the time interval of DIFS is very small in the order of $50\,\mu sec$ and is considerably low when compared to other MAC delays like backoffs and frame transmission times.
- *Header Compression (HC)*: Header compression refers to reducing the size of the various headers like the RTP/UDP/IP headers using the techniques either proposed in literature or otherwise. This technique also has limited potential in improving the call capacity in tune of only 0.1 calls. HC *reduces* $\sum_{L=\mathrm{RTP}}^{\mathrm{MAC}} \mathrm{H}(L)$ in Eq. 2.

In this paper we consider AA, FA and LA techniques only because they have a good potential to improve call capacity, while TS and HC provide a maximum of less than one call improvement and hence we ignore them in developing new algorithms. In the rest of this section we discuss the motivation behind each of these techniques and give detailed algorithms to realize them.

### B. ACK Aggregation (AA)

*1) Motivation:* Under saturated conditions the presence of ACK impacts the capacity of an 802.11 network. The 802.11 MAC specifies that the ACK should be sent at the basic rate. Hence even though the size of ACK is very small it eats up considerable amount of the capacity. Eq. 2 captures the impact of the ACK duration on the MNVC that can be supported. Hence one can envisage a scheme wherein ACKs are sent for every $n$ frames instead of for every frame. We define the set of $n$ frames as a block and the ACK for every block can contain information about the number of frames received correctly and the number in error. Only the lost frames can be retransmitted. TABLE II summarizes the improvements one can expect by aggregating ACKs. We observe that there is a difference between using a block size of 1 and a block size of 8 but there is not much difference between 8 and $\infty$. This is because of the presence of other terms in the denominator of Eq. 2.

Fig. 3 shows the variation in throughput (observed maximum frame rate) by varying the block sizes. We observe that smaller block sizes result in smaller delay for the frames but lower total throughput as well. A larger block size on

$$\mathrm{MNVC(PHY)} = \frac{1}{2k\left(T_{\mathrm{DIFS}} + T_{\mathrm{BO}} + T_{\mathrm{SIFS}} + T_{\mathrm{ACK}} + T_{\mathrm{PHY}} + \frac{8\left(D+\sum_{L=\mathrm{RTP}}^{\mathrm{MAC}} \mathrm{H}(L)\right)}{R}\right)}. \tag{2}$$

TABLE II
ANALYSIS OF MNVC(PHY) IMPROVEMENT USING DIFFERENT SCHEMES.

| AA | | FA | | LA | | TS | | HC | |
|---|---|---|---|---|---|---|---|---|---|
| # of ACK | MNVC | $k$ (/s) | MNVC | $R$ (Mbps) | MNVC | $T_{\mathrm{DIFS}}$ ($\mu$s) | MNVC | $\sum \mathrm{H}(L)$ (Bytes) | MNVC |
| 1 | 5.1 | 100 | 5.1 | 11 | 5.1 | 50 | 5.1 | 74 | 5.1 |
| 1/2 | 6.1 | 50 | 9.7 | 5.5 | 4.6 | 10 | 5.3 | 48 | 5.2 |
| 1/4 | 6.7 | 25 | 17.4 | 2 | 3.4 | 0 | 5.4 | | |
| 1/8 | 7.1 | 12.5 | 28.9 | 1 | 2.4 | | | | |
| 0 | 7.5 | 6.25 | 43.2 | | | | | | |

the other hand has larger delay for the frames but higher overall throughput. This is because when there is an error in a large block size scenario, it takes longer time to retransmit. However the better overall throughput given by a larger block size comes from the higher capacity due to smaller number of ACKs.

Even though the destination delays sending ACK to the source until all the frames belonging to one block are received, the destination can reduce the play buffer delay by sending the uncorrupted frame up to the upper layer whenever it receives a frame.

*2) Adaptive ACK Aggregation Algorithm:* Based on the above observation that there are cross over points we can think of an algorithm that changes the block size adaptively. We simulate this scheme as a 2.5 layer solution in between the MAC layer and the interface queue. We assume that there is no preset block size and we send a block ACK request from the source once all frames in the present block are sent. Upon receiving the block ACK request, the destination responds with a block ACK containing the required information. The source then starts a new block and retransmits the required frames of the previous block and continues with newer frames. Because of the presence of a block ACK request we can change the size of block at will. We implement a simple adaptive scheme

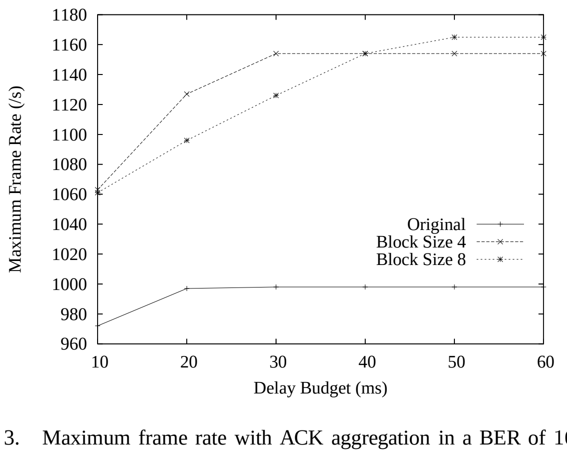


Fig. 3. Maximum frame rate with ACK aggregation in a BER of $10^{-5}$.

where we increase the block size upon receiving a block ACK with all successes and reduce the size on receiving a block ACK with even a single data loss. The detailed algorithm is shown in Fig. 4. The adaptive algorithm chooses the right block size depending on the number of losses in the current block. Thus the adaptive algorithm should give a performance that is the best of both worlds i.e it should have a good delay characteristic as well as high saturated capacity.

## C. Frame Aggregation (FA)

*1) Motivation:* The MAC throughput in an 802.11 network is limited by the size of the frame. Fig. 5 shows the maximum throughput observed in a one hop single flow scenario using an error-free 802.11b 11 Mbps link (with RTS/CTS mechanism suppressed). We observe that for a small frame size, of say 92 bytes (the size of a G.711 encoded VoIP frame with RTP header), the maximum throughput is about 931 Kbps, which means that we can get only 6.3 calls from such a network. The reason for this inferior performance is because of the various overheads involved, like the PLCP header, DIFS and SIFS intervals and random backoff. These have been explained in considerable detail in section II. The effect of the overheads decrease as the frame size increases.

An obvious scheme to improve the call capacity will be to aggregate frames. That said, it is a non trivial problem as to how one should go about aggregating frames. A direct consequence of aggregating frames is increased delay. However, there is a delay budget that can be accommodated by the voice codecs and it varies between codecs. Towards this goal of achieving a balance between the number of frames that can be aggregated and the delay budget, we propose an intelligent scheme which scales the number of voice calls that can be accommodated as the delay budget increases. With a delay budget of 100 ms we can get as many as 25 calls at the same time. We also show that such a scheme can also perform very well in the presence of other background traffic.

*2) Enhanced Piggybacking:* We now explain the algorithm in Fig. 6 used for frame aggregation. The core idea behind the algorithm is that we should aggregate frames only when required. Specifically, we aggregate only those frames that

```
Variables:
$currentBlockSize$: Current block size,
$nextBlcokSize$: Next block size,
$recv[i]$: Array of received frames in block maintained at receiver,
$sent[i]$: Array of sent frames in block,
$pkt$: Packet (data, block ACK request, or block ACK),
Functions:
tx($pkt, dst$) // send $pkt$ to $dst$ and make backup in buffer
rx($pkt$) {
    receive frame $pkt$
    update $recv[i]$
} // receive frame from source
send_block_ack_req($dst$) // send block ACK request to $dst$
recv_block_ack_req() {
    create block ACK frame
    fill information about $recv[i]$ in block ACK
} // receive block ack request
send_block_ack($src$) // send block ACK to $src$
recv_block_ack() // receive block ACK
retransmit_lost() // retransmit all lost frames
```

```
Adaptive ACK Aggregation Algorithm:
For each block
At source:
$currentBlockSize = nextBlockSize$
for $i = 0$ to $currentBlockSize$
    tx($sent[i], dst$)
send_block_ack_req($dst$)
recv_block_ack()
if all frames in block received successfully
    $nextBlockSize = nextBlockSize + 1$ // up to 8
else
    $nextBlockSize = nextBlockSize - 1$ // down to 2
retransmit_lost()
At destination:
while $pkt$ is not block ACK request frame
    rx($pkt$)
recv_block_ack_req()
send_block_ack($src$)
receive lost frames
```

Fig. 4. AA: Adaptive ACK aggregation algorithm.

have already been received by the underlying queue that precedes the MAC layer in the network stack. The intuition behind such a scheme is the observation that a number of frames are being dropped at the AP because of the queue getting filled up when the number of calls increased beyond 5. Frames are aggregated just before transmitting on the physical layer after all the waiting for DIFS, SIFS and/or backoff times at the MAC layer. We limit the maximum size of the frame that is transmitted to 2304 bytes (maximum allowed frame size in 802.11 standard).

In a purely downstream environment (only AP to client communication) the number of flows that can be accommodated increases as the delay budget allowed increases. This saturates beyond a point because of the maximum frame size limitation. In the upstream case however we do not observe a phenomenal increase in capacity. This can be explained as follows. Each client has only one flow (100 frames per second in our case) and it competes with other nodes for a transmission slot. By the time a client gets a slot to transmit it has typically only one or two frames to aggregate. The AP on the other hand has a number of flows but still contends as one node in network. Hence there will be a number of frames for each flow when it gets to transmit.

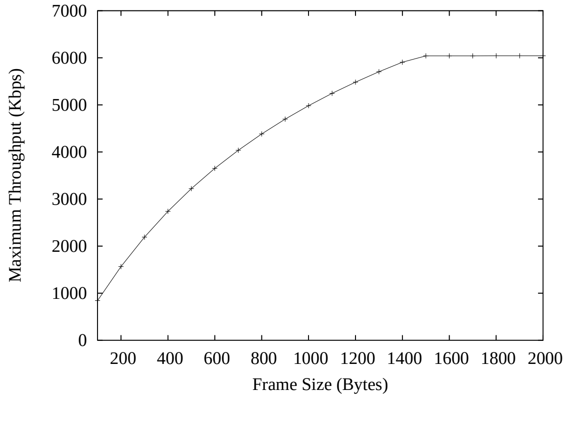


Fig. 5. Effect of frame size on maximum throughput.

To correct this imbalance between downstream and upstream flows we propose an enhancement to the piggybacking algorithm. We make modifications at both AP and client sides. At the AP we increase the buffer size of the queue to hold up to 500 frames. At the client side we maintain a variable $prevNumAggregate$ that holds the number of frames aggregated in the previously received frame from the AP. When the client gets a chance to transmit it checks if it has frames at least equal to that variable. If yes, it goes ahead to transmit the aggregated frame. However if it does not have enough frames it goes into a random backoff without increasing the contention window size.

We do not increase the contention window size because we do not want to wait for a very long time because of the delay budget. Since each client now waits for the same number of frames as in downstream it will contend for the channel at large time intervals. The AP on the other hand will contend for each of its client one after the other. Thus the contentions are now balanced between the upstream and downstream nodes depending on the amount of traffic. We refer to this scheme as *enhanced piggybacking*. We study the behavior of this scheme using ns-2 simulations and the results are discussed in the next section.

## D. Link Adaptation (LA)

*1) Motivation:* It is intuitive that the MNVC at a higher rate would be larger than at a lower rate. However this is true only when we assume that the link is error-free. When the signal-to-noise ratio (SNR) of the channel is low, it would be advisable to transmit at a lower rate. This is because for a given SNR the loss for a lower rate transmission is lower than at a higher rate.

TABLE III shows a quick look at possible MNVC at different rates. The results are obtained using ns-2 simulations. We observe that in an error-free condition 11 Mbps will give

```
Variables:
src: Source of frame (can be AP or client),
dst: Destination of frame (can be AP or client),
self: Variable to say whether the device is AP or client,
pkt: Frame,
prevNumAggregate: Counter at client which stores number
    of frames aggregated in the previous frame from AP,
maxFrameSize: Maximum frame size,
numAggregate: Number of frames aggregated,
Functions:
tx(pkt, dst) {
    wait until slot to transmit
    find all frames in queue to same destination
    aggregate as many frames as possible until maxFrameSize
    if self is client
        if numAggregate is less than prevNumAggregate
            backoff
    send aggregated frame
} // transmit pkt to dst
rx(pkt) {
    receive aggregated frame
    if self is client
        update prevNumAggregate
    split aggregated frame and send up
} // receive pkt
Enhanced Piggybacking Algorithm:
At source:
    pkt = head of line frame at queue
    tx(pkt, dst)
    recv_ack() // receive 802.11 ACK
At destination:
    rx(pkt)
    send_ack() // send 802.11 ACK
```

Fig. 6. FA: Enhanced piggybacking algorithm.

the highest MNVC but with an SNR of 5 dB we see that 2 Mbps gives higher MNVC. Hence 2 Mbps would be the ideal rate at 5 dB SNR. Similarly for other SNR there is also a rate at which we get the maximum benefit. We call this rate the ideal rate, which induces the need of rate adaptation technique.

Several schemes have been proposed in literature and some like the ARF [7] have been implemented in practice to adapt the transmission rate depending on the prevailing channel conditions. But all these schemes either adapt to the number of frames in error or use SNR measured at the receiver and communicated to the sender via CTS frame. These schemes do not consider the frame size for making the decision on the data rate.

There is an inverse relation between SNR and bit error rate (BER) which directly translates to frame error rate (FER) depending on the frame size. Hence for a given BER the FER of a small frame might be acceptable whereas the FER for a large frame size might be unacceptable. Fig. 7 illustrates this fact. We observe that for an SNR of 6 dB, a data rate of 11 Mbps might be acceptable for small frame size but unacceptable for large frames. This gives us a motivation for a size aware rate adaptation technique.

ARF is a popular rate adaptation technique that is widely deployed in many existing WLANs. It is a simple but robust technique that chooses the rate depending on the reception or failure of the successive ACKs for the data frames. Briefly the ARF algorithm works as follows: For every ten consecutive ACKs correctly received the rate is increased to the next higher allowed value and upon two consecutive ACK failures the rate is reduced to the immediate lower allowed rate. As is evident from the algorithm this technique does not take into account the frame size during rate adaptation. It is a known fact that ARF does not perform very well under rapidly changing channel conditions because it depends on ten successful ACKs to increase the rate. This problem of ARF worsens when the frames are of varying sizes. VoIP frames are characterized by small frame sizes and when they are interspersed with large frame sized background traffic the ARF will suffer in the sense that frames will be sent at wrong rates.

Fig. 8 shows the number of frames sent at wrong rate of both VoIP and large background traffic frames as a function of background traffic. By wrong rate we mean that the frame is sent either at a higher or lower rate than ideal. We observe that as the amount of background traffic increases the number of frames sent at wrong rate increases. Based on these observations we propose a new rate adaptation scheme that is both size aware and channel aware. We call this scheme Size-

TABLE III
MNVC(PHY) AT A GIVEN SNR.

| Rate (Mbps) | MNVC | | | | |
|---|---|---|---|---|---|
| | No Error | 10dB | 5dB | 3dB | 2dB |
| 11 | 5.2 | 5.2 | 0.8 | 0 | 0 |
| 5.5 | 4.7 | 4.7 | 2.4 | 0.2 | 0 |
| 2 | 3.5 | 3.5 | 3.5 | 1.6 | 0 |
| 1 | 2.5 | 2.5 | 2.5 | 2.5 | 2.5 |
| ARF | - | 5.2 | 3.9 | 3.6 | 2.3 |

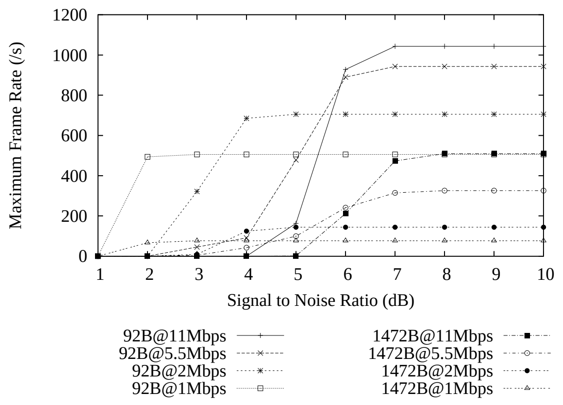


Fig. 7. Maximum frame rate vs. SNR with different frame sizes.

```
Variables:
sTh[i] and fTh: Array of success threshold per frame size i and failure threshold,
linkRate[i]: The chosen data rate per frame size i,
success[i]: The number of consecutive success per frame size i,
failure[i]: The number of consecutive failure per frame size i,
retry: The number of retransmission,
probe[i]: The boolean array of probe state per frame size i,
SARF Algorithm:
M: maxFrameSize/scaleFactor,
C = currentFrameSize/scaleFactor
while retry < maxRetry
    if current transmission succeeds
        retry = 0
        for i = 0 to C
            success[i] = success[i] + 1
            probe[i] = false
        for i = 0 to M
            failure[i] = 0
            if success[i] ≥ sTh[i]
                success[i] = 0
                probe[i] = true
                increase linkRate[i] to the next higher rate
        break
    else // current transmission fails
        retry = retry + 1
        for i = C to M
            failure[i] = failure[i] + 1
        for i = 0 to M
            success[i] = 0
        if retry = 1 and probe[C] = true
            probe[C] = false
            decrease linkRate[C] to the next lower rate
        else if retry ≠ 1 and retry%fTh = 0
            for i = C to M
                if failure[i] ≥ fTh
                    decrease linkRate[i] to the next lower rate
```

Fig. 9. LA: SARF algorithm.

aware Auto Rate Fallback (SARF) as it is based on ARF.

*2) SARF Algorithm:* The core idea behind SARF is that if a small frame is in error then there is a high probability of error for a large frame as well. Similarly when a large frame is successful, there is a very high probability of success for small frames as well. The exact algorithm is shown in Fig. 9.

The source maintains two counters (success counter and failure counter) for different frame sizes. These counters are updated (based on the previous conjecture) upon reception or failure of ACKs. Specifically upon a successful reception of ACK for a frame of size $C$ all the success counters for frame sizes less than $C$ are updated. Similarly when the ACK for a frame of size $C$ gets lost all the failure counters for frame sizes above $C$ are updated. When these individual counters reach preset threshold values the rate for the corresponding frame size is modified (either increased or decreased).

The adaptive algorithm is similar to ARF i.e upon successive ACKs received correctly the rate is increased to the next level and upon consecutive ACK failures the rate is reduced. The threshold values are chosen based on some heuristic methods. We observe that even though the MNVC does not improve drastically for SARF the number of frames sent at wrong rate is significantly less than that of ARF. The packets sent at wrong rate may incur either large delay due to lower rate than ideal or much loss due to higher rate than ideal.

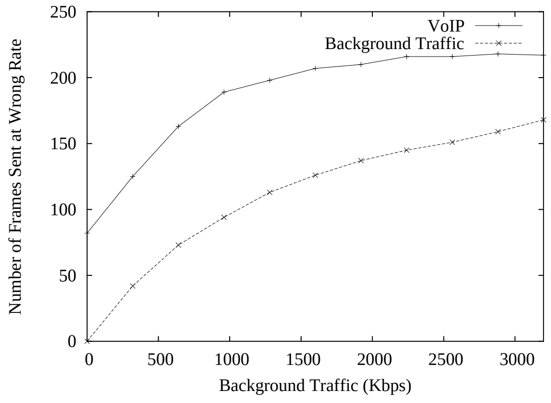


Fig. 8. Impact of heterogeneous frame sizes in ARF.

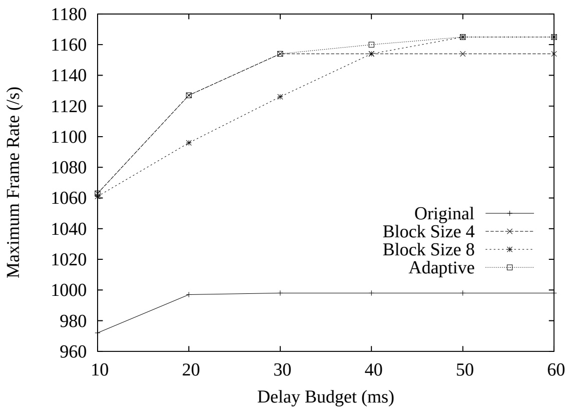
## IV. Performance Evaluation

We use the open source ns-2 [8] for all our simulations. Unless otherwise stated bi-directional CBR traffic with a frame size of 92 bytes is used for modeling the VoIP traffic. The frame rate of traffic is set at 100 frames per second in either direction i.e AP to client and client to AP. Each call originates from a wired node and has two wired links and one final 802.11b wireless link in an infrastructure mode. In scenarios where background traffic is required we use another CBR traffic with a frame size of 1472 bytes. The rate of the background traffic is varied according to the need.

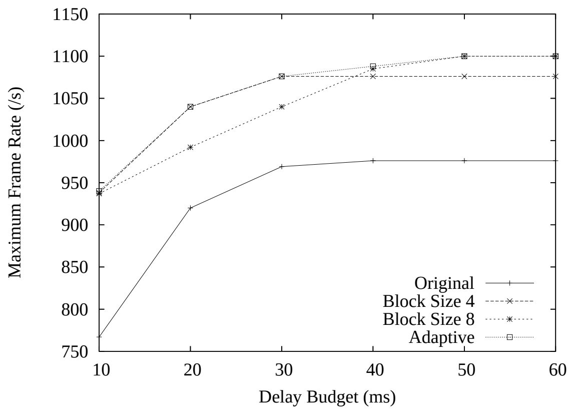
### A. Metrics Studied

We use three metrics to evaluate the performance improvement when our proposed algorithm applied. In all the simulations we assume the delay budget as follows: A call is assumed to be successful if less than 2% of its frames are lost. Losses in a call occur either because they are received after the delay budget or because they are dropped due to BER.

- *Maximum Frame Rate (MFR)*: MFR is defined as the number of frames which can be received maximally per second within a given delay budget. This is a fine grained metric used to study the improvements in the order of one call or lesser in AA and LA.
- *Maximum Number of VoIP Calls (MNVC)*: This is defined as the MFR divided by the number of frames per second



(a) MFR vs. Delay Budget with a BER of $10^{-5}$



(b) MFR vs. Delay Budget with a BER of $10^{-4}$

Fig. 10. Simulation results of AA.

for each call ($200$ in our case). The MNVC is calculated by increasing the number of calls from one and stopping when the loss experienced by any call is more than 2%. This metric is used to study improvements greater than one call in FA.

- *Number of Frames Sent at Wrong Rate*: This metric is used to study the LA techniques. The number of frames sent at wrong rate gives an estimate of inferior performance to an ideal rate adaptation scheme that magically sends frames at the exact rate depending on the channel conditions at the receiver.

MFR and MNVC are the same as those at the PHY layer in Section II-B except that they consider the delay budget.

### B. ACK Aggregation (AA)

To study the performance of the ACK aggregation scheme we setup bi-directional CBR flows between wired node and client node in the wireless domain. We use MFR as a fine grained metric to study the improvement of this scheme.

Fig. 10(a) and Fig. 10(b) show the performance results of the ACK aggregation algorithm at different BERs. The capacity improvement due to the aggregation of ACKs is evident from the saturated frame rate at higher delay budget. The saturation throughput is higher for a larger block size. This is because for a larger block size smaller number of ACKs are sent thus making more room for data frames. However when a frame gets lost it will take longer time to get retransmitted if the block size is large. This delay gets worse when the BER is higher. Thus we see that for a BER of $10^{-4}$ the block size 8 performs significantly poorer than block size 4 for low delay budgets. The adaptive algorithm chooses the right block size depending on the number of losses in the current block. Thus the adaptive algorithm gives performance that is the best of both worlds. It has both better delay characteristics as well as better saturated frame rate.

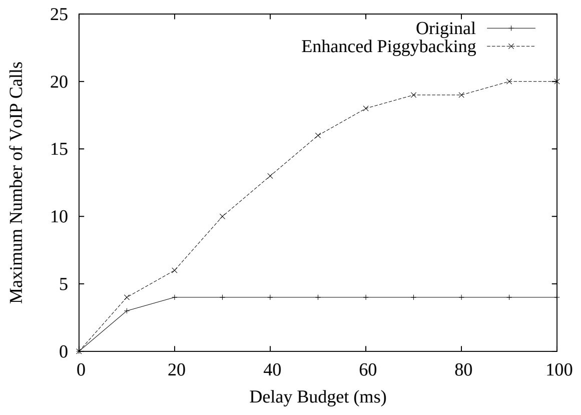
### C. Frame Aggregation (FA)

To simulate the number of VoIP calls using the enhanced piggybacking scheme we assume an erroneous environment with a BER of $10^{-5}$. We setup a number of bi-directional CBR flows with a rate of 100 frames per second in either direction. To study the number of VoIP calls for a given delay budget we increase the number of such bi-directional flows to different clients and stop when the loss at the given delay is more than 2% of the frames. Losses occur due to both erroneous link and frames exceeding the delay budget.

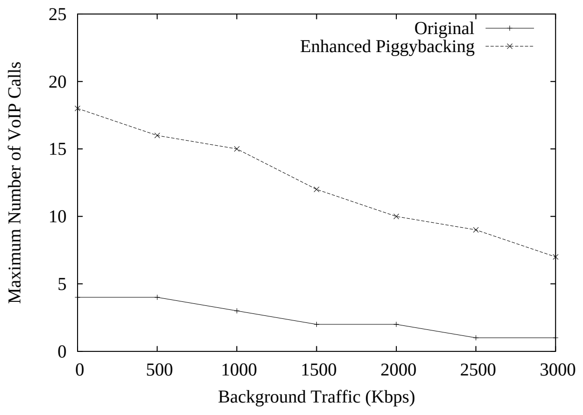
Fig. 11(a) shows the MNVC that can be accommodated for a given delay budget. We compare the enhanced piggybacking scheme with the original data-ACK scheme. The original scheme peaks at a maximum of 4 calls, which is one smaller than the analysis result due to error condition and delay budget. However the enhanced piggybacking scheme gives up to 20 calls for a delay budget of 100 ms and 18 calls for a delay budget of 60 ms. Note that 60 ms is an acceptable wireless delay for good quality of VoIP.

The enhanced piggybacking scheme is able to scale with increasing delay budget. One drawback of the enhanced piggybacking scheme is that as the number of calls increases the delay for every user increases. A suitable scheme at the AP should be employed to limit the number of calls based on a given delay budget. A possible QoS policy can be applied if we can measure the delay for each frame for the leg of transmission from AP to the far end client. This delay information can be used to set the delay budget of the wireless leg of the call. This will be part of our future work.

A typical AP serves VoIP traffic as well as normal data to other clients. We study the effect of background traffic on the call capacity. Fig. 11(b) shows the performance of both schemes in the presence of background traffic. We introduce a single source of background traffic and increase its rate. It is intuitive to expect that the number of calls that can be supported in the presence of background traffic reduces with increasing rate of the background traffic. With a background traffic of 3000 Kbps an original scheme can support only 1 call while the enhanced piggybacking scheme can support up to 7 calls with a delay budget of 60 ms. This shows that our scheme can be applied in the presence of heavy background



(a) MNVC vs. Delay Budget



(b) MNVC vs. Background Traffi c

Fig. 11. Simulation results of FA.

traffic as well.

### D. Link Adaptation (LA)

We use a single VoIP call and a single background CBR source with a large frame size (1472 bytes) to study the efficacy of SARF. We compare the performance of SARF with ARF. We assume a hidden Markov channel model [9] to simulate varying channel SNR conditions.

Fig. 12(a) shows the MNVC for ARF and SARF for varying background traffic. We observe that both SARF and ARF perform similarly. This is because both ARF and SARF are based on similar principles of tracking the channel condition based on success or failure of ACKs. However when we look at the number of frames sent at wrong rate in Fig. 12(b) SARF sends a much smaller number of frames at wrong rate than ARF. ARF does not change its rate based on frame sizes. Hence successful small frames unnecessarily increase the rate of large frames and large frame losses unnecessarily reduce the rate of small frames. SARF also performs well for other QoS metrics like delay budget but we do not present those details due to lack of space.

## V. Related Work

We divide the previous work into two categories: VoIP capacity improvement approaches and the schemes related to our individual algorithms.

### A. VoIP Capacity Improvement Approaches

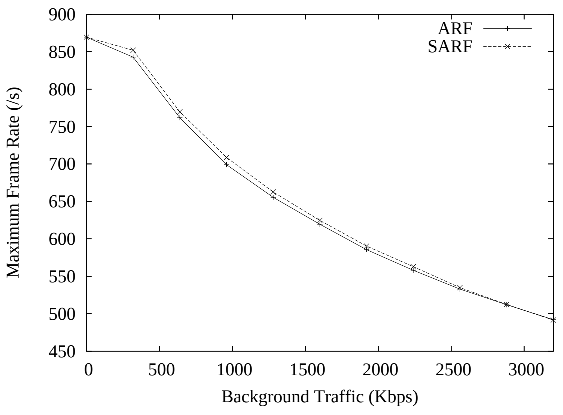
With a growing importance of VoIP and popularity of wireless networks, several research works have been proposed to address and improve the performance of VoIP over wireless networks. In [5] the authors analyzed the maximum number of VoIP calls over IEEE 802.11b networks using a variety of different voice codecs and verified their simulation results with testbed implementations. In [6] the authors propose multiplexing-multicast and priority-queuing schemes to improve the call capacity by 100%. However multicast frames have to be sent at the rate that all the clients can receive them. This may lead to some clients with a good link condition being unnecessarily being penalized. In [10] the authors evaluate the capacity of VoIP over 802.11b under a variety of channel conditions and propose a mechanism to select the frame size of voice codec adaptively given delay budget. [11] proposes the use of dual queue of 802.11 MAC to differentiate and prioritize VoIP traffic to other best effort traffics. However simply prioritizing the VoIP traffic is not sufficient because the problems due to small size of VoIP frames still exist.

### B. Related Schemes

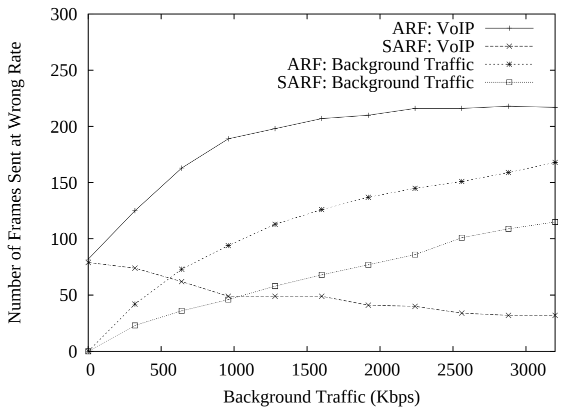
[12] and [13] evaluate the performance of IEEE 802.11e block ACK scheme for both basic access and RTS/CTS under error-free and noisy channels, respectively. The block ACK mechanism in the IEEE 802.11e standard requires that all frames in a block are sent with a spacing of SIFS between them. This means that if we do not have a frame to send within a SIFS time interval, the block ACK mechanism cannot be used. Our scheme on the other hand poses no such requirements. Using the Block ACK request mechanism we can change to send a Block ACK request for only the frames we have sent.

[14] and [15] propose several frame aggregation mechanisms to be used in IEEE 802.11n standard. However their algorithms do not consider the delay budget in AP and perform aggregation of as maximum number of frames as possible in AP, while our proposed FA algorithm aggregates the frames bidirectionally in both AP and client for a given delay budget.

There are several works dealing with rate adaptation like [7][16][17]. We have looked at [7] in Section III-D in considerable detail. RBAR [16] is a feed-back based rate adaptation scheme that uses the RTS/CTS mechanism to convey current channel conditions at the receiver to the sender. But such a scheme would require SNR measurements and this might be prohibitive. In [17] the authors propose an adaptive algorithm based on ARF with an assumption that ARF does not react well to highly varying or static wireless channel.



(a) MFR vs. Background Traffic



(b) Number of Frames Sent at Wrong Rate vs. Background Traffic

Fig. 12. Simulation results of LA.

## VI. Concluding Remark and Future Work

In this paper we analyze the reasons behind the inferior performance of VoIP traffic over wireless LANs. We setup an experimental testbed to serve as a testbed baseline and to justify our analysis. While doing so we propose three algorithms at the MAC layer to improve the observed call capacity namely ACK Aggregation (AA), Frame Aggregation (FA), and Link Adaptation (LA). Extensive ns-2 simulations are performed to show the efficacy of these algorithms. We find that FA can provide capacity improvements in the order of up to 300%.

Although our analysis and simulation results are performed on G.711 codec, those results can hold true in other codecs, for example GSM 6.10 codec. We show the preliminary analysis and simulation results in 802.11b network with GSM 6.10 codec in TABLE IV and Fig. 13.

TABLE IV
MNVC at each layer in 802.11b with GSM 6.10.

| $L$ | APP | RTP | UDP | IP | MAC | PHY |
|---|---|---|---|---|---|---|
| MNVC($L$) | 416 | 305 | 259 | 188 | 13 | 10 |

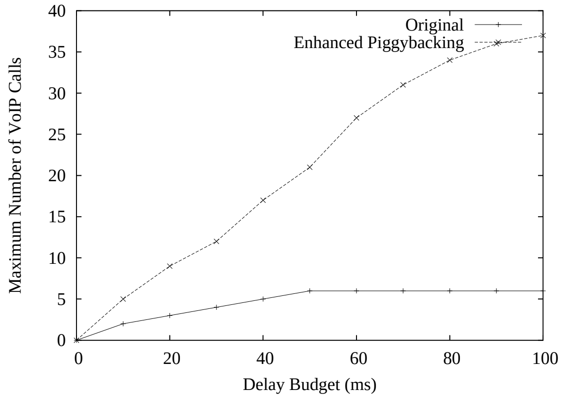


Fig. 13. Simulation result of FA for GSM 6.10.

As part of our future work we will complete the analysis and simulation for a variety of codecs and IEEE 802.11 variants. We also plan to implement both software only and standard compliant solution of the proposed algorithms in the real testbed.

## References

[1] KPhone. [Online]. Available: http://www.wirlab.net/kphone/
[2] SIP Express Router. [Online]. Available: http://www.iptel.org/ser/
[3] Iperf. [Online]. Available: http://dast.nlanr.net/Projects/Iperf/
[4] A. P. Markopoulou, F. A. Tobagi, and M. J. Karam, "Assessment of VoIP quality over Internet backbones," in *Proc. IEEE Infocom'02*, June 2002, pp. 150–159.
[5] S. Garg, , and M. Kappes, "On the throughput of 802.11b networks for VoIP," Avaya Labs Research, NJ, Tech. Rep. ALR-2002-012, Mar. 2002.
[6] W. Wang, S. C. Liew, and V. O. K. Li, "Solutions to performance problems in VoIP over a 802.11 wireless LAN," *IEEE Trans. Veh. Technol.*, vol. 54, no. 1, pp. 366–384, Jan. 2005.
[7] A. Kamerman and L. Monteban, "WaveLAN-II: A high-performance wireless LAN for the unlicensed band," *Bell Labs Technical Journal*, vol. 2, no. 3, pp. 118–133, Aug. 1997.
[8] The network simulator: ns-2. [Online]. Available: http://www.isi.edu/nsnam/ns/
[9] W. Turin and R. V. Nobelen, "Hidden Markov models for fading channels," *IEEE J. Select. Areas Commun.*, vol. 16, no. 12, pp. 1809–1817, Dec. 1998.
[10] D. P. Hole and F. A. Tobagi, "Capacity of an IEEE 802.11b wireless LAN supporting VoIP," in *Proc. IEEE ICC'04*, June 2004, pp. 196–201.
[11] J. Yu, S. Choi, and J. Lee, "Enhancement of VoIP over IEEE 802.11 WLAN via dual queue strategy," in *Proc. IEEE ICC'04*, June 2004, pp. 3706–3711.
[12] I. Tinnirello and S. Choi, "Efficiency analysis of burst transmissions with block ACK in contention-based 802.11e WLANs," in *Proc. IEEE ICC'05*, May 2005, pp. 3455–3460.
[13] T. Li, Q. Ni, T. Turletti, and Y. Xiao, "Performance analysis of the IEEE 802.11e block ACK scheme in a noisy channel," in *Proc. IEEE BroadNets'05*, Oct. 2005, pp. 551–557.
[14] Y. Xiao, "Packing mechanisms for the IEEE 802.11n wireless LANs," in *Proc. IEEE Globecom'04*, Nov. 2004, pp. 3275–3279.
[15] Y. Kim, S. Choi, K. Jang, and H. Hwang, "Throughput enhancement of IEEE 802.11 WLAN via frame aggregation," in *Proc. IEEE VTC'04-Fall*, Sept. 2004, pp. 3030–3034.
[16] G. Holland, N. Vaidya, and P. Bahl, "A rate-adaptive MAC protocol for multi-hop wireless networks," in *Proc. ACM MobiCom'01*, July 2001, pp. 236–250.
[17] M. Lacage, M. H. Manshaei, and T. Turletti, "IEEE 802.11 rate adaptation: A practical approach," in *Proc. ACM MSWiM'04*, Oct. 2004, pp. 126–134.