Adaptive Per Hop Differentiation for End-to-End Delay Assurance in Multihop Wireless Networks

Jian Li, Zhi Li, and Prasant Mohapatra

Department of Computer Science
University of California at Davis
Davis, CA 95616.
Email: {lijian, lizhi, prasant}@cs.ucdavis.edu.

Abstract

As the rapid growth of smart hand-held devices, multihop wireless access networks have a lot of potential applications in a variety of fields in civilian and military environments. Many of these applications, such as realtime audio/video streaming, will require some form of end-to-end QoS assurance. In this paper we present an adaptive per hop differentiation (APHD) scheme towards achieving end-to-end delay assurance in multihop wireless networks. Our scheme is based on EDCA technique which is proposed in 802.11e draft. In EDCA, data packets of different priorities will use different MAC contention parameter set, which translate into different delays. Our APHD scheme extends the capability of EDCA into multihop environment by taking end-to-end delay requirement into consideration at each intermediate hop. Following a cross-layer design approach, APHD is aimed to be a distributed and localized technique. Individual nodes keep track of the channel state independently without any intercommunication overhead. Data packets carry end-to-end delay requirement along with other important information in the packet header. At an intermediate node, based on data packet's end-to-end requirement, its accumulative delay so far, and the current node's channel status, APHD smartly adjusts data

packet's priority level in order to satisfy its end-to-end delay requirement. Simulation results show that APHD scheme can provide excellent end-to-end delay assurance while achieving much higher network utilization, compared to a pure EDCA scheme.

Key words: Adaptive per hop differentiation, Cross-layer design approach, End-to-end delay assurance, Multihop wireless networks, Quality of service

1 Introduction

Multihop wireless ad hoc networks consists of a set of mobile nodes which form an ad hoc topology via dynamic wireless links. All nodes in the networks act as routers as well as end hosts. When source and destination are beyond one hop transmission range, multihop routing is used to find route from source to destination. Compared to infrastructured wireless networks (e.g. WLAN), wireless ad hoc networks are infrastructureless and have no need of centralized control. This type of networks can be fast deployed for various potential applications in the variety of fields in both civilian and military environments.

As the rapid growth of smart hand-held devices, wireless ad hoc networks are expected to extend their presence in many aspect of our life. Many of these applications, such as realtime audio/video applications, will require some form of end-to-end QoS assurance. Generally speaking, QoS support in wireless networks is a very challenging topic due to their dynamic nature [1,2]. Various techniques, from physical layer upto application layer, have been proposed to provide QoS support in wireless ad hoc networking environments [3]. In recent years, a cross-layer design approach in QoS provisioning in wireless networks has gained more and more research interest ([4–6]).

In this paper we propose an adaptive per hop differentiation (APHD) scheme towards achieving end-to-end delay assurance in multihop wireless networks. Our APHD scheme is based on EDCA technique which is proposed in emerg-

ing standard 802.11e. EDCA is designed to provide QoS differentiation in single hop wireless network environment (WLAN). It does not fit into multihop environment because it has no notion of end-to-end service guarantee. Our simulation results show pure EDCA scheme performs poorly in multihop network. APHD scheme is thus designed to extend and incorporate EDCA technique into multihop wireless network environment, aiming at provide end-to-end delay assurance for time sensitive applications. Relying on cross-layer information sharing and cooperation, APHD attempts to utilize adaptive service differentiation at each intermediate hop to achieve an end-to-end delay requirement. The simulation results show that, in both linear topology and large networks, APHD outperforms pure EDCA scheme by providing excellent end-to-end delay assurance while achieving much higher network utilization.

The rest of this paper is organized as follows. In Section 2 we discuss some background knowledge on 802.11 DCF and EDCA and the motivation of our work. Detailed design of the cross-layer architecture of our APHD scheme is presented in Section 3, followed by extensive simulation-based performance evaluation in Section 4. Related work of interest to our work is discussed in Section 5, and then the paper is concluded in Section 6.

2 Background and Motivation

In this section we will discuss background knowledge about 802.11 DCF and 802.11e EDCA. We will also discuss our motivation of APHD design.

2.1 802.11 DCF

IEEE 802.11 standard [8] defines two access modes in wireless local area network (WLAN), namely, Distributed Coordinate Function (DCF), and Point Coordinate Function (PCF). PCF is a centralized access mode and it requires

a point coordinator which is normally a part of the access point in WLAN environment. DCF does not rely on such a centralized point coordinator so it has been widely proposed for use in wireless ad hoc networks where there is no central infrastructure.

DCF distributes the media access task among neighboring nodes via CSMA/CA and a backoff technique. When a node has a data packet to transfer, it listens first to see whether the channel is idle or busy. If the channel is busy the node will keep waiting quietly. When the channel becomes idle and it stays in the idle state for a period of time (this required time is equal to DIFS if the previous packet was received correctly, or EIFS if the previous packet was not received correctly), the node starts its backoff timer for a random amount of time. If the backoff timer has a non-zero value already (which means it was previously in a backoff state), it does not need to select a new backoff timer value. When the backoff timer expires and the channel is (still) in idle state, the node will transmit the data packet.

In the above process the random backoff timer can minimize collisions during contentions among multiple nodes. Individual nodes calculate their own backoff timer value using the following formula:

$$backoff\_time = random() \times SlotTime, \tag{1}$$

where random() is pseudo-random integer drawn from a uniform distribution over [0, CW], where CW is called contention window and it satisfies the condition $CW\_min \leq CW \leq CW\_max$. Both CW_min/CW_max and SlotTime are determined by physical layer (PHY) characteristics. When it is the first attempt to transmit a packet, CW is assigned of CW_min value. If collision happens, CW will be increased using a binary exponential approach. In 802.11 specification, CW_min is 7 and CW_max is 255. So, when transmitting a new data packet, CW will be set to 7 initially, and then increased to 15, 31, 63, 127, and 255, respectively, in case of consecutive collisions. After each successful

transmission, CW will be reset to CW_min value.

2.2 802.11e EDCA

IEEE 802.11e [9] was proposed to supplement IEEE 802.11 MAC in order to provide service differentiation in WLAN. The drive for this supplementary draft is the great potential need of supporting realtime audio/video at home and office environments where WLAN has been widely deployed in the past few years.

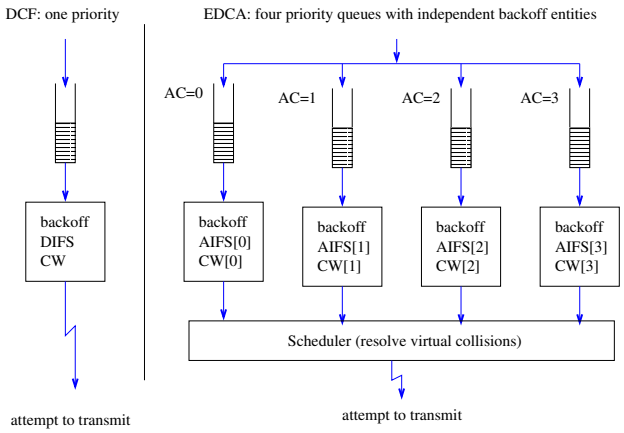

Fig. 1. IEEE 802.11e MAC structure

The 802.11e draft introduces the Hybrid Coordination Function (HCF), which defines two new MAC mechanisms namely, HCF controlled channel access (HCCA), and enhanced distributed channel access (EDCA), to replace PCF and DCF modes in 802.11. We are interested in 802.11e EDCA mode here because it does not require central administration, just like 802.11 DCF mode. The comparison between 802.11 DCF and 802.11e EDCA was illustrated in Fig. 1.

EDCA achieves service differentiation by introducing different access categories (ACs) and their associated backoff entities. Unlike in DCF, where all MAC service data units (MSDUs) are put into a single queue and thus associated with a single backoff entity, EDCA introduces four priority queues, one

for each AC, and each priority queue has its own backoff entity using different parameter set. In EDCA, the counterpart of DIFS (in DCF mode) is called Arbitration Interframe Space (AIFS). Both AIFS and the contention window size are dependent on access categories, and thus they are written in these formats: AIFS[AC], and CW[AC], where $CW\_min[AC] \leq CW[AC] \leq CW\_max[AC]$.

In a node running in EDCA mode, individual priority queues act independently of each other. Each AC priority queue competes with other priority queues at the same node as well as those priority queues in neighboring nodes. In this sense, each AC priority queue can be regarded as an independent virtual node. A higher priority queue (with a smaller AC index) will use relatively a smaller AIFS[AC] value as well as a smaller CW_min[AC] and CW_max[AC] pair. In such a manner, higher priority queue always defers less time before attempting to transmit. So, when multiple priority queues content for channel access, higher priority queue can always grab the channel first and get more chance to transmit. This is the main idea on how to achieve service differentiation in EDCA mode.

2.3 Motivation for Our Design

802.11e defines four priority queues in the MAC layer and can provide service differentiation only for single hop. However, in multihop wireless networks, time sensitive applications normally require end-to-end delay assurance. How can we utilize single hop service differentiation to satisfy an end-to-end service requirement for a multihop communication session?

First, suppose we could break down the end-to-end delay requirement into smaller pieces as per hop budget. As long as each intermediate hop achieve the specified per hop delay, the end-to-end delay requirement will be satisfied.

Second, how to partition the end-to-end requirement into per hop budget?

Apparently we need to know the total budget and the hop count of route. In our design, we adopt the widely studied Dynamic Source Routing (DSR). With source routing, hop count is easy to obtain. For hop by hop routing, we may use probing technique to find out the end-to-end hop count.

Third, how to populate the per hop budget to all intermediate nodes? One possible way is to calculate the per hop budget at the sender and then piggy-back this information to intermediate nodes along the route. But this way is lack of flexibility because each node gets the same per hop budget. Moreover, the per hop budget information needs to be populated every time the route changes. We choose another approach which is more flexible by putting the requirement information in packet header. When intermediate node receives a packet, it can read this information from packet header and infer the per hop requirement for this packet.

Fourth, how to translate the per hop requirement into a MAC priority level? We need to know what is the delay for individual service classes at the current node. We propose the node monitoring technique to keep track of per class delay (PCD) at individual nodes. Based on this information, when a data packet arrives at a node, the node can smartly map its per hop budget to a proper priority level.

3 APHD Design

In this section we will present the cross-layer design of our APHD scheme. First we give a brief overview on the scheme. Then we will discuss the three major components of our APHD scheme, namely, packet header extension, node state monitoring, and per hop based priority adaption. Finally we put them together and see how packets experience adaptive service differentiation while flowing through the cross-layer architecture.

3.1 Overview

The main goal of APHD is to achieve end-to-end delay assurance for time-sensitive traffic using per hop differentiation technique.

The main idea is that, end-to-end delay requirement along with other information is carried in the packet header, and at each intermediate node, the priority of an individual packet is adjusted based its delay requirement, its delay experience and the current node's state. Because APHD bears end-to-end requirement in mind, flows choosing same class of service but traversing routes of different hop counts can enjoy fairness in term of end-to-end services.

This kind of per hop based priority adaptation technique has the following advantages:

- Localized: Priority adaptation is applied on individual hop, which is fast and flexible, compared to end-to-end flow priority adaption.
- Distributed: Individual node monitors the channel state independently. No extra intercommunication overhead.
- Efficient: Only raise priority level at places where it is needed, allowing more efficient network utilization.
- Resilient: Doesn't require resource reservation, more tolerant to route changes. No need of accurate channel capacity information.

Compared to end-to-end bandwidth reservation approach (like RSVP [10]), APHD is distributed and localized. So it is more tolerant in presence of network dynamics. We argue that end-to-end route may be broken and change fast, but the hop count between a pair of nodes is not changing so rapidly. With the hop count being stable, intermediate node can calculate fairly accurate per hop delay budget.

Compared to end-to-end priority selection, APHD is adaptive at intermediate

hop. So it is more flexible and efficient. It only raises priority level for a packet at the hop where it is needed, while using relatively low priority level when it is possible. This can help to improve network utilization.

3.2 Packet Header Extension

We extend the packet header to accommodate four necessary fields to facilitate APHD technique. The summary of important fields are shown in Table 1.

Field Name	Meaning
src	source address
dst	destination address
prio	EDCA priority for current hop
e2e_delay_req	end-to-end delay requirement
e2e_hops	end-to-end hop count
delay_so_far	accumulative delay so far
hops_so_far	number of hops traversed so far

Table 1
Important fields in APHD packet header

When sending out a packet, the sender does not specify the priority level of service. Instead, it specifies the end-to-end delay requirement *e2e_delay_req* explicitly. Along together is *e2e_hops* which is end-to-end hop count that this packet expects to traverse along the path from source to destination. These two fields are fixed when a packet travel from source to destination.

There are two dynamic fields in the packet header, namely, *delay_so_far* and *hops_so_far*, which account for the accumulative delay and hop count that the packet has experienced already. These two fields are initialized to be zero at

first hop. As the data packet propagates in the networks, intermediate node will update them accordingly. The dealing of *hops_so_far* is relatively simple. It is incremented by one at each hop. The treatment of *delay_so_far* is a bit more complicated: we need to have a accurate estimate and put the information in packet header before the packet really go through this hop. Details on how to measure the current hop delay for a packet will be further discussed in Section 3.3.

An alternative of accumulating *delay_so_far* at each hop would be recording *pkt_birth_time* at first hop node. With the packet birth time and the current time, an intermediate node can easily calculate *delay_so_far* for this packet. However, this is valid only if the clock of all mobile nodes are synchronized. We choose to put *delay_so_far* in packet header such that our APHD scheme does not depend on any clock synchronization assumption.

In Section 3.4 we will discuss details about how to utilize these information in per hop based priority adaption.

3.3 Node State Monitoring

In order to map a data packet's per hop budget into a proper priority level, individual nodes need to monitor channel utilization state as well as per class delay (PCD[i], where $PRIO\_min \leq i \leq PRIO\_max$).

Channel utilization: a node keeps track of channel busy time and idle time, and can calculate channel utilization percentage. This is helpful in admission control. For example, if the utilization is above some critical level over a certain time window, a node may decide that the network is overloaded and thus deny new flow request or even pick some existing flows with low priority as victims to drop. Generally speaking, admission control in multihop wireless networks need to consider the resources of the local node, the neighboring nodes within

interference range, as well as those nodes along the path. Interested readers can refer to [14–16] for further information on admission control in multihop wireless networks. In this work we assume some form of admission control is in use so the network is not overloaded at any time. Our work is focused on designing an adaption scheme which can smartly select a optimal priority level for individual data packets.

Per class delay PCD[i]: when a packet $p(i)$, where i is the priority level, is received at MAC layer of a node, it records the packet's incoming time T_a; when the packet is successfully transmitted to nexthop at time T_b, we obtain the delay that this packet experiences at this hop:

$$pkt\_delay(i) = T_b - T_a. \tag{2}$$

To smooth out the delay variance among consecutive packets of a specific priority level i, we use an exponential moving average to calculate PCD[i]:

$$PCD[i] = (1 - \alpha) * PCD[i] + \alpha * pkt\_delay(i). \tag{3}$$

The greater the α factor, the more promptly it responds to the channel state change.

Now it comes to the question: how can we update the packet header field *delay_so_far* with the current hop delay $pkt\_delay(i)$ before we actually transmit it to nexthop node? Our approach is to use an estimate of T'_b. Specifically, at MAC layer, when a packet of priority level i reaches the head of the priority queue (i.e., it gets scheduled for transmission), it will content for access to the channel with other priority queues at the same node as well as neighboring nodes. When it successfully grab the channel at time T_g, we can get an estimate:

$$T'_b = T_g + \frac{packet\_size}{transmission\_rate}. \tag{4}$$

Note that T_b' is an estimate since we omit some protocol overhead such as PLCP preamble and PLCP header that also contribute to packet transmission delay.

With the estimate time T_b', we can calculate $pkt\_delay(i)$, and add it to the *delay_so_far* field in packet header before transmitting the packet to next hop. If it turns out that the transmission fails due to collision, the packet will be rescheduled for transmission again. In such a case, the packet header need to be updated again before retransmission.

Note that in the above discussion, we focus on individual nodes' tracking of their own PCD[i] values. In reality, multiple nodes in a neighborhood may observe different delay for the same priority class. To overcome this issue, we may require neighboring nodes exchange their observed PCD[i] values. Alternatively, each node's wireless interface can operate in the promiscuous mode so as to monitor any packet transmission in the neighborhood and extract their PCD[i] from packet header. Then each local node can calculate the average PCD[i] value in a distributed way.

3.4 Per Hop Based Priority Adaptation

At each hop, a node may adjust a packet's priority level based on the *e2e_delay_req* carried in the packet header and the channel state of the current node. We need to take care of two different cases: at firs hop node and at intermediate node.

At first hop node, packet priority treatment is carried out at networks layer. This is because we need to set proper priority level for a packet before sending it down to priority queues at MAC layer. As discussed in previous section, MAC layer keeps track of per hop delay for individual priority levels. By inquiry with MAC layer, routing agent can learn the current PCD[i] information.

Algorithm 1 First Hop Priority Adaptation

at source node:

routing agent receives a packet p from upper layer

$perhop\_budget = \frac{e2e\_delay\_req}{e2e\_hops}$;

for (i = PRIO_max; i $\geq$ PRIO_min; i$--$)

  if (PCD[i] $\geq$ PCD_THRESHOLD[i])

    continue; /* skip this priority level */

  endif

  if (PCD[i] $\leq$ perhop_budget)

    break; /* select priority level i */

  endif

endfor

set p's priority: $prio$ = i;

The first hop adaptation algorithm is shown in 1, written in a pseudo code similar to the C language. As we can see, at first hop node, we optimistically select a low service level (i.e. with larger priority number) which fits the packet's per hop budget.

When an intermediate node receive a packet, it may be one of the two possible cases: the packet arrives earlier or later than expected. If the packet has been late already when arriving at this hop, we should select a higher priority level to speed up its transmission. If the packet has arrived earlier than expected, we say we get some flexibility time, which we called *slack*. In such a case, we can select a relatively low priority for this packet. The adaptation algorithm at intermediate hop node is shown in 2.

We would like to point out that our adaptation algorithms can lead to more efficient network utilization while attempting to satisfy the end-to-end delay requirement. At first hop node, as shown in Algorithm 1, we optimistically select a lowest priority level which can meet a packet's per hop budget. At intermediate hop nodes, as shown in Algorithm 2, we try to select a lowest

Algorithm 2 Middle Hop Priority Adaptation

at intermediate node:

MAC receives an incoming packet p

$e2e\_budget\_perhop = \frac{e2e\_delay\_req}{e2e\_hops}$;

$budget\_so\_far = e2e\_budget\_perhop \times hops\_so\_far$;

$slack = budget\_so\_far - delay\_so\_far$;

if (slack $\leq$ 0) /* being late; try to speed up */

  for (i = PRIO_min; i $\leq$ PRIO_max; i++)

    if (PCD[i] $\leq$ PCD_THRESHOLD[i])

      continue; /* select this priority level */

    endif

  endfor

else /* get flexibility; can slow down */

  $budget2go = e2e\_delay\_req - delay\_so\_far$;

  $hops2go = e2e\_hops - hops\_so\_far$;

  $budget2go\_perhop = \frac{budget2go}{hops2go}$;

  for (i = PRIO_max; i $\geq$ PRIO_min; i−−)

    if (PCD[i] $\geq$ PCD_THRESHOLD[i])

      continue; /* skip this priority level */

    endif

    if (PCD[i] $\leq$ budget2go_perhop)

      break; /* select priority level i */

    endif

  endfor

endelse

set p's priority: $prio$ = i;

satisfactory priority level whenever it is allowed (i.e., when $slack > 0$). Since lower priority backoff entities have larger contention window size, transmission collisions can be reduced when multiple flows meet at a common hop node. This reduction of collision can lead to higher throughput in addition

to canceling the side-effect of long waiting time due to larger contention window size. Moreover, since more packets try to use lower priority levels at each hop, we have more resource to speed up a packet's transmission in case its accumulated delay to this hop is more than expected so far.

In both Algorithm 1 and Algorithm 2, we utilize preset PCD_THRESHOLD[i] to prevent from overloading a priority level. Specifically, for priority level i, if the current per class delay PCD[i] exceeds the preset threshold value, we will not put more traffic load on this priority level, until PCD[i] falls back to under the threshold. The settings for PCD_THRESHOLD[i] depends on the requirement of specific applications as well as the network diameter.

We would like to point out that there is chance that Algorithm 2 might fail to find a satisfying priority level for some packets, for example, if they have exceeded their per-hop budget too much in previous hops. In some cases, even using the highest priority at a latter hop may not be able to make up for the packet's end-to-end delay requirement. For another example, when two wireless ad hoc networks merge, it may cause route change and bursty traffic fluctuation, which may cause extra delay for some packets that can not be compensated by adjusting priority level. If such cases happen, these packets could be discarded for applications like real-time streaming where packets arriving too late are just useless.

3.5 Putting It Together

The overall architecture of APHD is shown in Fig. 2. Note that the transport layer is omitted here to simplify the illustration. As we can see, the overall design of APHD follows a cross-layer approach, which has gained more and more acceptance in wireless network design in recent years.

The core component of APHD architecture is Node State Monitoring function,

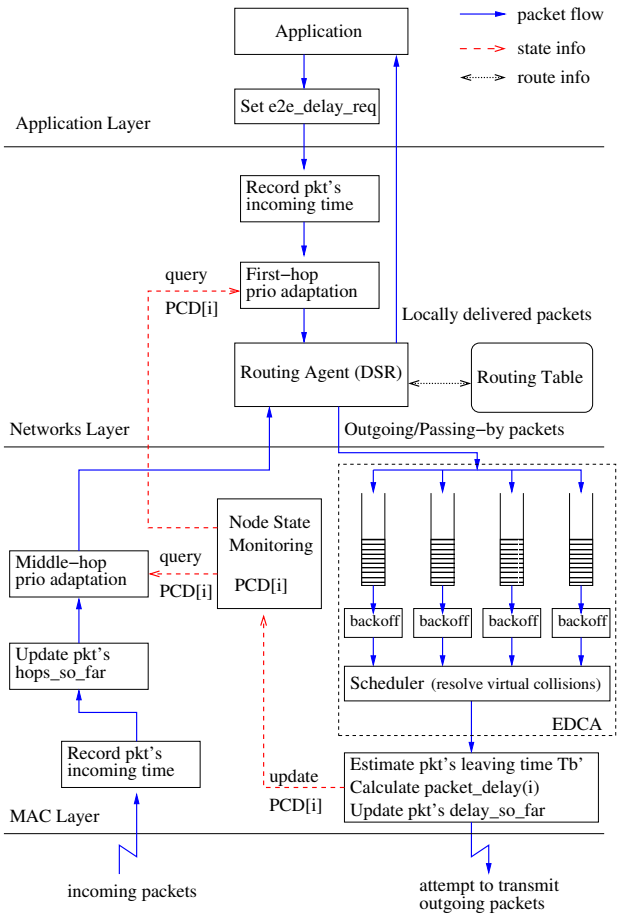

Fig. 2. APHD's overall architecture

which mainly sits in the MAC layer. It keeps track of per class delay (PCD[i]) by measuring the incoming and leaving times of packets of different priority level. The PCD[i] information is shared with the priority adaption functions across networks layer as well as MAC layer.

Let's trace a packet's flowing path in this architecture. When MAC receives an incoming packet, it first records its incoming time T_a. The packet header's $hops\_so\_far$ field will be increased by one hop. Then, by inquiring PCD[i] and packet header information, the packet's MAC priority level is adjusted using Algorithm 2. For locally-generated packets, the application layer specifies an end-to-end delay requirement. At network layer, by inquiring PCD[i] information from MAC Layer, the packet's end-to-end requirement is mapped to a proper MAC priority level based on Algorithm 1. Now, both locally generated

packets and incoming packets are handed to the Routing Agent (DSR). The output from DSR will be divided into local drop traffic and outgoing traffic. Some packets are for local applications and thus handed up to the application layer. Other packets are outgoing traffic and thus handed down to different MAC queues based on their priority level. Per hop differentiation takes effect at this step due to EDCA mechanism, and higher priority packets are expected to have a shorter per hop delay. When a packet is actually scheduled for transmission, we estimate its final bit's leaving time, update packet header $delay\_so\_far$ field as well as the node state (PCD[i]) accordingly.

The above process is repeated at each intermediate hop, and service delay assurance can be achieved from an end-to-end point of view. As we can see from Figure 2, the adaptive service differentiation process requires information sharing as well as collaborative actions from MAC layer, network layer and application layer. This embodies the cross-layer design philosophy of our APHD architecture.

4 Performance Evaluation

We have done extensive simulations using NS-2 (version 2.26) [25] to evaluate the performance of APHD scheme. DSR codebase is ported from an older version NS-2.1b8a. EDCA codebase is ported from TKN's implementation [26,27]. In 802.11e draft, eight priority levels and four access categories (ACs) are defined and there is a mapping relation between a priority level and an AC. Here we do not distinguish this difference and use the terms AC and priority level interchangeably. PHY and MAC parameter settings are summarized in Tables 2 and 3, respectively. Note this PHY setting is corresponding to a link capacity of 11 Mbps. A node's transmission range is 250 meters while the carrier-sensing range is 550 meters, both of which are default settings in NS-2 simulator.

Parameter	Value
SlotTime	20 μs
CCATime	15 μs
RxTxTurnaroundTime	5 μs
SIFSTime	10 μs
PreambleLength	144 bits
PLCPHeaderLength	48 bits
PLCPDataRate	1 *Mbits/s*
MaxPropagationDelay	2 μs

Table 2

NS-2 PHY settings for IEEE 802.11e

Priority	AIFS	CW_min	CW_max
0	2	7	15
1	2	15	31
2	3	31	1023
3	7	31	1023

Table 3

NS-2 EDCA settings for IEEE 802.11e

4.1 Simple topology with different route length

4.1.1 Simulation setup

In this set of simulations we use a simple linear topology as shown in Fig 3. The distances between node A and B, node B and C, are 180 meters. The distances between nodes B, D, E, F are all 200 meters. There are three flows:

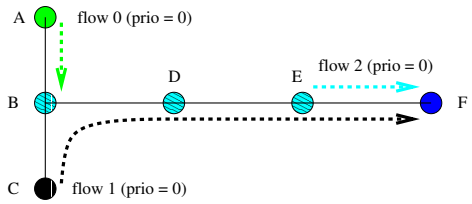

Fig. 3. A simple topology: same priority, different route length

flow 0 between node A and B, flow 1 between node C and F, and flow 2 between node E and F. All three flows are real-time audio traffic and thus have the same end-to-end delay requirement. Packet size is 150 bytes for all three flows, and we three tests using different source rates. In pure EDCA tests, we set priority level 0 for all three flows. In APHD tests, we set their end-to-end delay requirement to be 1000 *ms*.

As we can see, flows 0 and 2 only have one hop, while flow 1 need to travel four hops. This simulation setup is intendedly to see if our scheme can provide end-to-end delay assurance and end-to-end fairness.

In our simulations, flow 0 will starts at time 0.1 second, flow 1 starts at time 60.0 seconds, and flow 2 starts at time 100.0 seconds. By selecting different start times for different flows, we can see the impact to existing flows when adding a new flow to the network. Meanwhile, we mainly observe and compare the performance between APHD and EDCA after all flows start.

4.1.2 Simulation results

The end-to-end delay for pure EDCA scheme is shown in Fig. 4. When source data rate is 10 and 50 packets/second, the delay difference between flow 1 and flows 0/2 are not much, and all three flows satisfy end-to-end delay requirement. As the source rates are increased to 100 packets/second, the delay difference becomes innegligible. Flow 1 (which has longest route) experience much higher delay compared to flows 0/2. Moreover, all three flows fail to meet the end-to-end delay requirement.

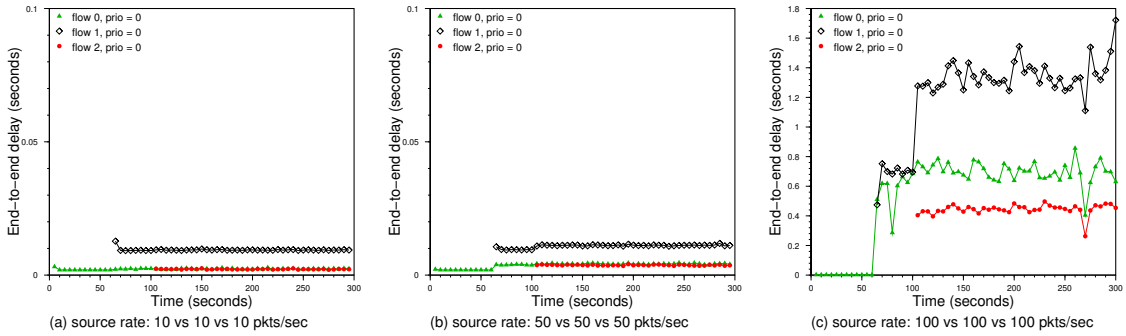

Fig. 4. EDCA: end-to-end delay (different route length)

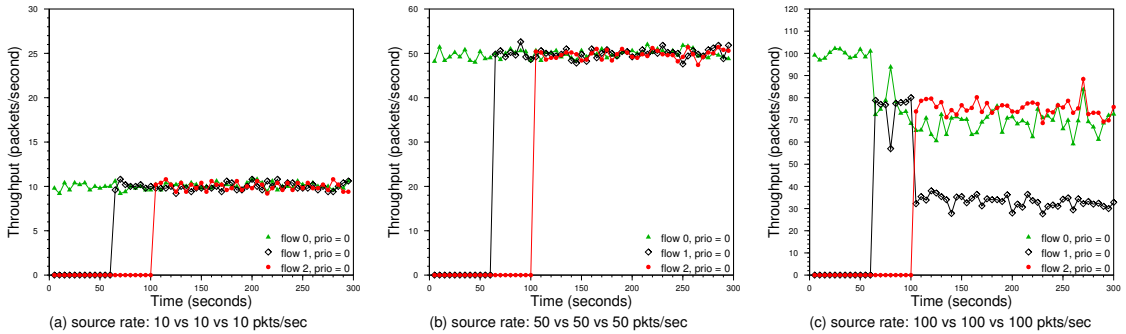

Fig. 5. EDCA: end-to-end throughput (different route length)

Now let's look at the throughput results. As shown in Fig. 5, at source rate 100 packets/second, EDCA scheme has great throughput drop in all three flows. Flow 1 (with longest route) achieves only half of the throughput of flows 0/2.

From Fig. 4 and Fig. 5, we can see pure EDCA scheme cannot provide end-to-end assurance nor fairness in terms of end-to-end delay and throughput.

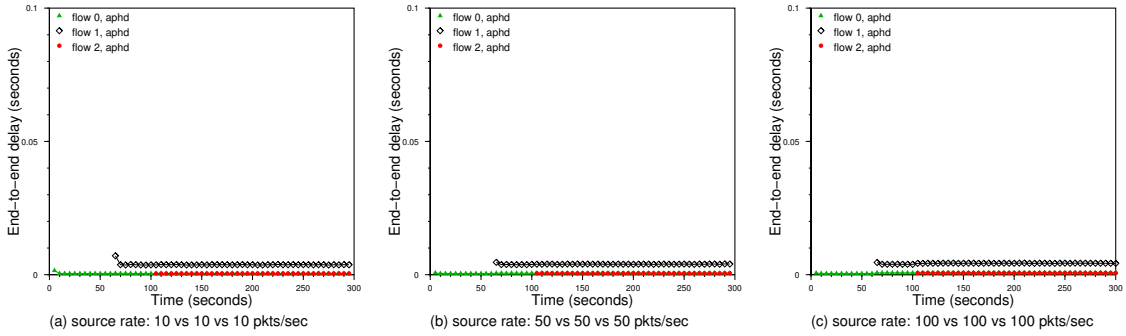

Fig. 6. APHD: end-to-end delay (different route length)

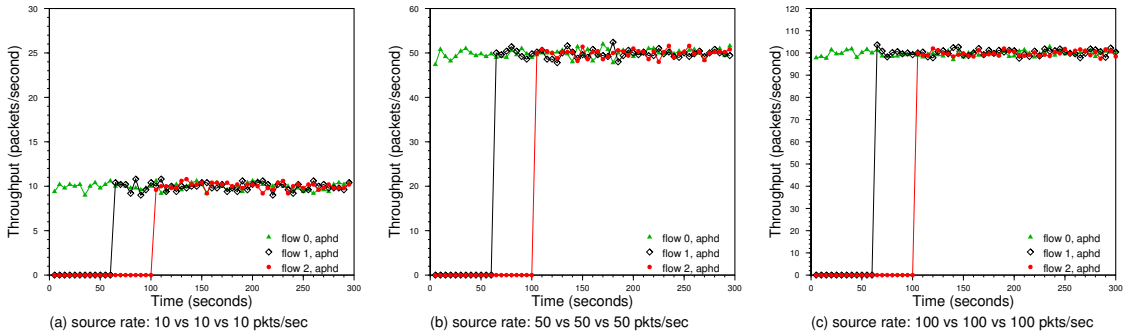

Fig. 7. APHD: end-to-end throughput (different route length)

Now let's move forward to see the results of APHD tests. The end-to-end delay for APHD scheme is shown in Fig. 6. Even under source rate 100 packets/second, the end-to-end delay for all three flows still keep at very low level and there is no noticeable difference between flow 1 and flows 0/2.

Next let's see throughput results for APHD as shown in Fig. 7. All three flows achieve the same throughput under all three source rates. We can say APHD achieve end-to-end fairness regardless of route length. Moreover, there is no packet drop even when source rate is 100 packets/second, where EDCA scheme has showed severe packet drop already. This shows APHD can improve network utilization by smartly adapting MAC priority level for individual packets.

4.2 Simple topology with different priority

4.2.1 Simulation setup

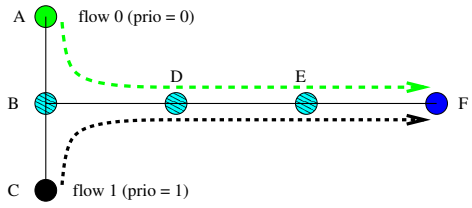

Fig. 8. A simple topology: same route length, different priority.

As shown in Figure 8, we use the same linear topology as previous section. Now there are two flows with the same route length. Flow 0 from node A to F

is real-time audio, while Flow 1 from node C to F is real-time video playback. Flow 0 has packet size as 150 bytes, while flow 1 as 800 bytes. The source rates we use in the simulation are 25 vs 25, 50 vs 25, and 100 vs 50 packets/second. In EDCA tests, we set flow 0 at priority level 0, while flow 1 at priority level 1. In APHD tests, we set the end-to-end delay requirement for flow 0 as 100 *ms*, while flow 1 as 1000 *ms*, assuming the receiver of flow 1 has enough playback buffer to smooth out the delay variance.

As in previous tests, flow 0 starts at time 0.1 second, and flow 1 starts at time 60.0 second.

4.2.2 Simulation results

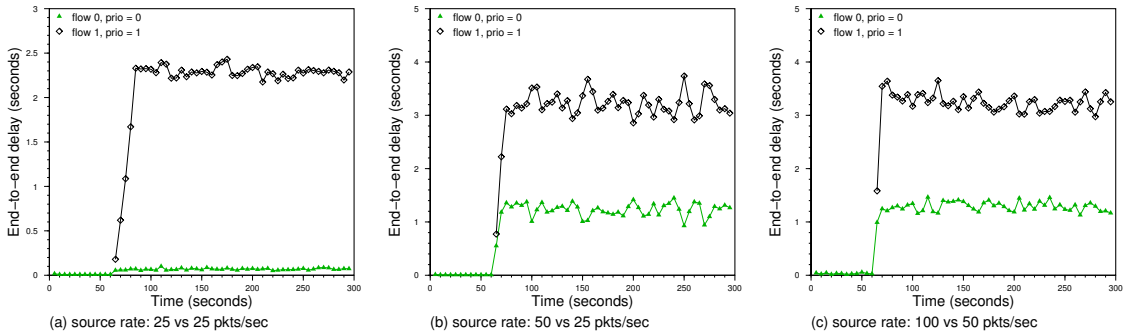

Fig. 9. EDCA: end-to-end delay (same route length, prio=0 vs prio=1)

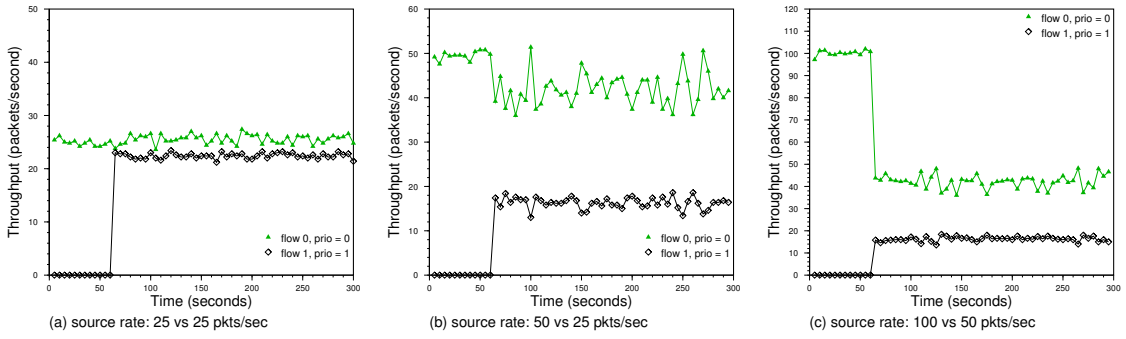

Fig. 10. EDCA: end-to-end throughput (same route length, prio=0 vs prio=1)

Simulation results for EDCA scheme are shown in Fig. 9 and Fig. 10. When source rate is 25 vs 25 packets/second, EDCA achieve fairly good throughput

for both flows, but flow 1 has experienced delay as high as 2 seconds. As we increase source rate for both flows, real-time audio flow 0 starts to experience end-to-end delay as high as one second, which is unacceptable in real application. Moreover, with the increase of source rates, both flows see great throughput degrade. From Fig. 9 and Fig. 10, we can see pure EDCA scheme cannot provide end-to-end delay assurance, although the traffic load is not very high (440 Kbps for source rate 100 vs 50 packets/second).

Next let's see APHD's results as shown in Fig. 11 and Fig. 12. With all three source rates, both flows achieves very low end-to-end delays, which are always within the end-to-end delay requirement. The throughput shows no packet drop even when source rate is at 100 vs 50 packets, where EDCA scheme's throughput is very poor already. We can see that APHD scheme have great improvement in network utilization and can provide excellent end-to-end delay assurance.

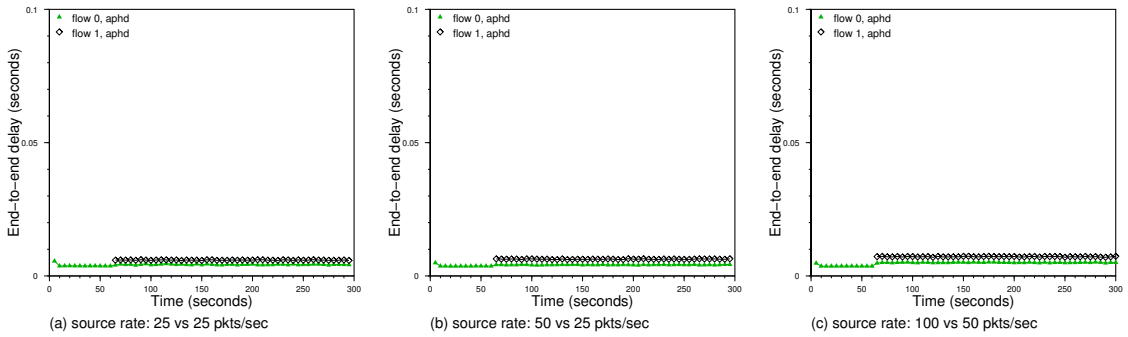

Fig. 11. APHD: end-to-end delay (same route length)

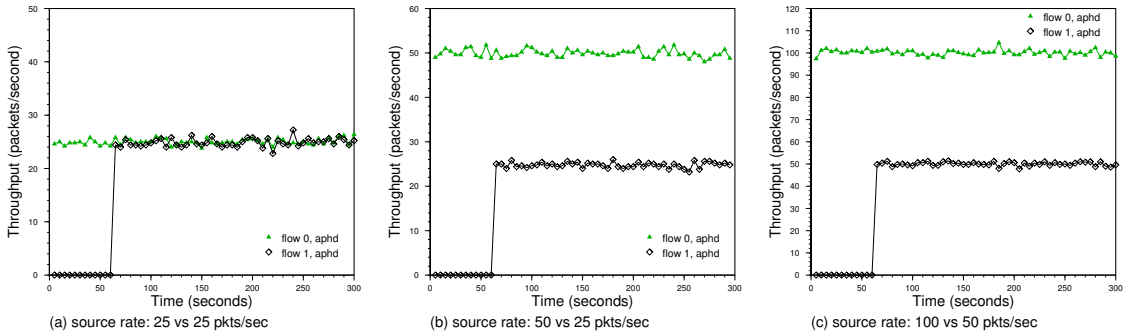

Fig. 12. APHD: end-to-end throughput (same route length)

4.2.3 What if EDCA raises flow 1's priority?

As we discuss earlier, the alternative approach is end-to-end priority adaption. Specifically, when the receiver of flow 1 detects the current priority (level 1) cannot provide the desired end-to-end delay, it will notify the sender to raise the priority level (to level 0) in hope that it will reduce the end-to-end delay. Our simulation results show that this does not help.

As shown in Fig. 13 and Fig. 14, even when source rate is 25 vs 25 packets/seconds, flow 0 (real-time audio) has unacceptable end-to-end delay upto one second. This degrade is due to the contention from flow 1 raising priority to the same level as flow 0. As the source rate increases, both end-to-end delay and throughput experience much more degrade. The network utilization is even worse compared to the results when flow 1 uses priority level 1, as shown in Fig. 9 and Fig. 10. From this, we can say that the end-to-end priority selection approach may be harmful in term of delay assurance and network utilization.

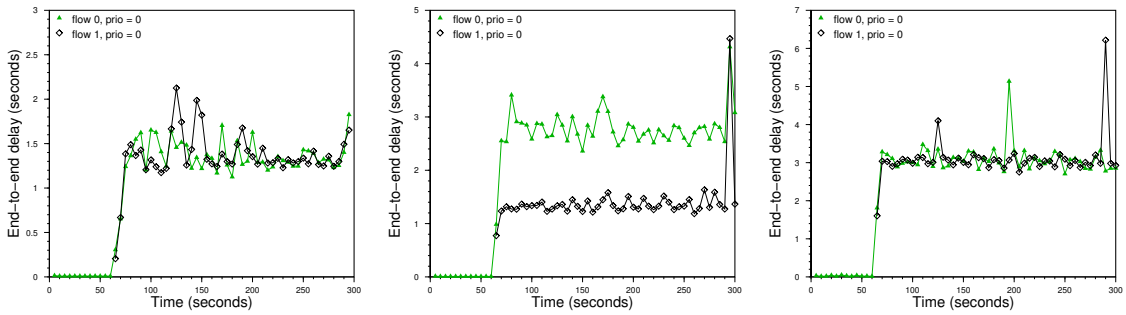

Fig. 13. EDCA: end-to-end delay (same route length, flow 1 raised priority)

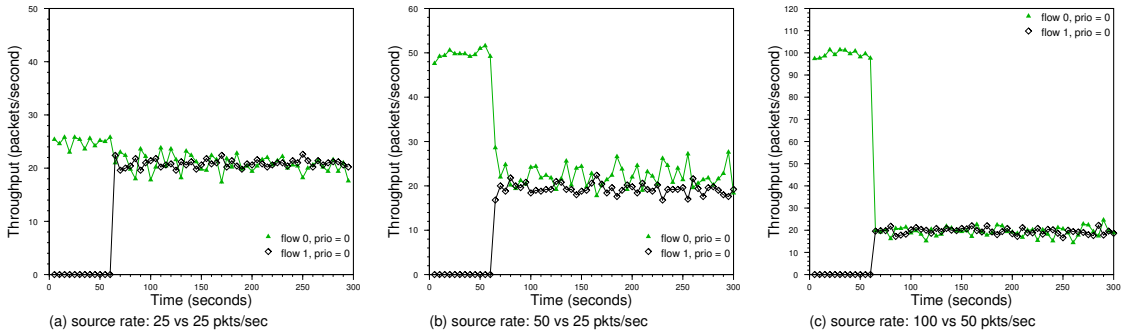

Fig. 14. EDCA: end-to-end throughput (same route length, flow 1 raised priority)

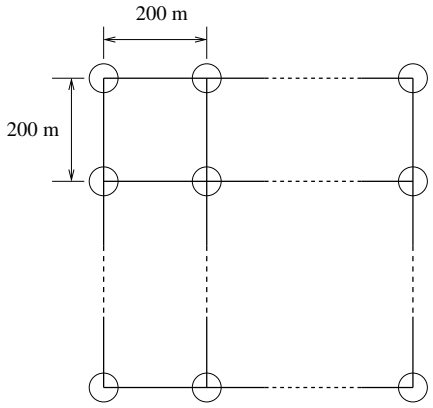

Fig. 15. A 10x10 regular topology.

4.3 Large Networks

4.3.1 Simulation setup

To see how APHD performs in large networks, we use a 10x10 Manhattan networks shown in 15. Adjacent nodes on each row (and each column) are separated 200 meters in distance. Note that in the worst case the hop count is 18 hops between diagonal corners, so we increase DSR maximum route length to 18 hops (default maximum is 16 hops). There are 12 flows in total among random pairs of source and destination. Flows 0-5 are real-time audio traffic with packet size 150 bytes, while flows 6-11 are video playback with packet size 800 bytes. All the flows start randomly at different times between [0, 100] seconds.

As previous simulations, in EDCA, we set audio traffic at priority level 0 while video playback at level 1. In APHD tests, we set end-to-end delay requirement as 100 *ms* for audio traffic and 1000 *ms* for video playback.

4.3.2 Simulation results

First let see EDCA's performance in Fig. 16 and Fig. 17. When source rate is at 4x6 vs 4x6 packets/seconds, both audio and video flows can maintain satisfied performance. As the source rate increases, both type of traffic meet unacceptably high delay and very poor throughput.

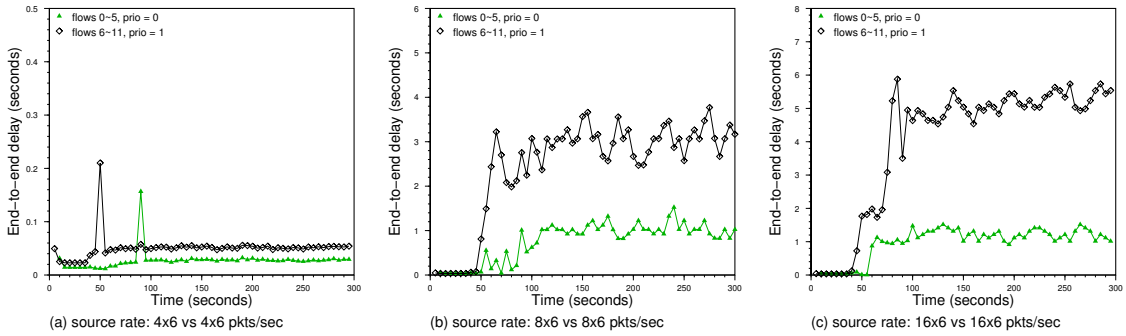

Fig. 16. EDCA: end-to-end delay (large networks)

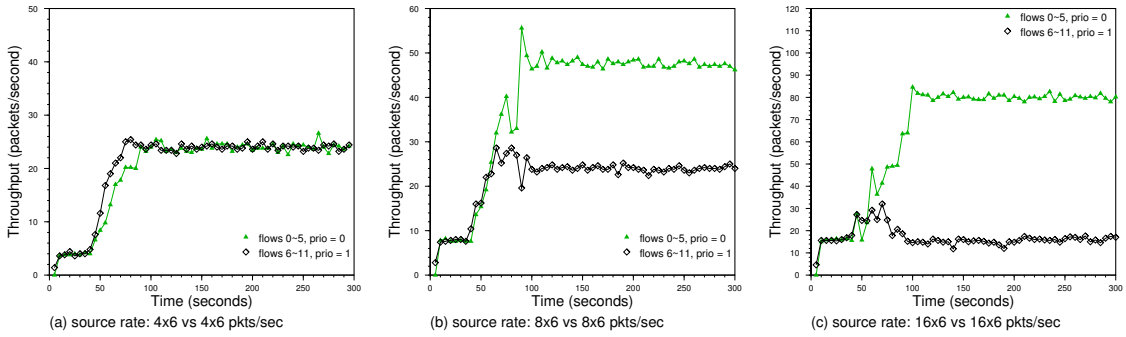

Fig. 17. EDCA: end-to-end throughput (large networks)

Now let's see APHD's results as shown in Fig. 18 and Fig. 19. Under all three source rates, both audio and video traffic can achieve very low end-to-end delay. The throughput shows no packet even when the source rate is 16x6 vs 16x6 packets/second. These results show that APHD performs efficiently in

large networks as well.

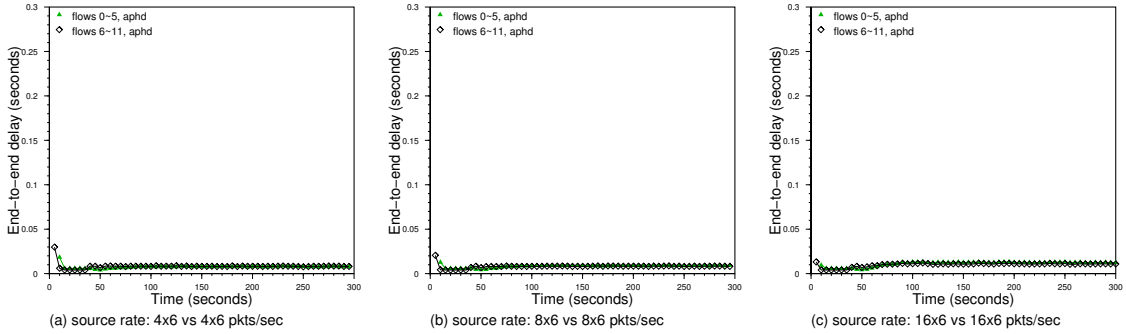

Fig. 18. APHD: end-to-end delay (large networks)

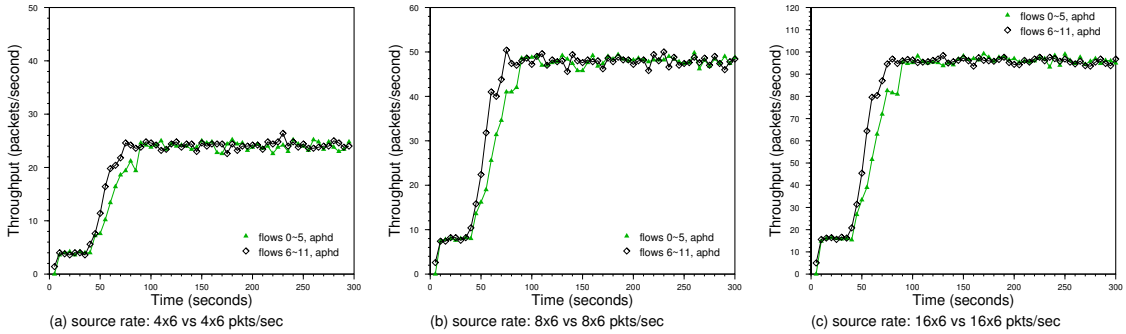

Fig. 19. APHD: end-to-end throughput (large networks)

5 Related work

In this section we discuss some related works on end-to-end delay assurance that are of highly interest to our APHD scheme.

Dynamic Packet State (DPS) [11] technique was proposed to achieve end-to-end QoS guarantee without per flow state maintenance. This technique attempts to take advantage of the two QoS models, namely, IntServ [12] and DiffServ [13], that have been proposed for Internet QoS provisioning. In IntServ model, per flow state is maintained on intermediate routers along the path. The scheme is complicated but it can achieve guaranteed end-to-end QoS on per flow granularity. In DiffServ model, a simplified per hop behavior (PHB)

model is defined, and flow states are aggregated based on a small set of service classes. DiffServ scheme is more scalable, but it only provide service differentiation without hard end-to-end QoS guarantee. In DPS technique, end-to-end delay requirement is carried in packet header, intermediate hop routers process incoming packets based on their packet header information and the current hop node state. Our APHD scheme follows this stateless approach, which does not require flow state maintenance at intermediate hop nodes.

INSIGNIA [5] is a signaling protocol which is based on crosslayer interaction to achieve end-to-end QoS reporting and adaptation. However, we argue that end-to-end feedback is not always desirable for wireless ad hoc networks due to their highly dynamic nature. First, it incurs long feedback delay. In face of fast changing network topology, it may lead to more communication overhead.

Wang et al [17] proposed a Neighborhood Proportional Delay Differentiation (NPDD) model which provides globally consistent proportional delays at all nodes in multihop WLAN environment. While attempting to achieve proportional service differentiation, NPDD utilizes the dynamic end-to-end class selection mechanism [18] to select the lowest (presumably cheapest) satisfactory class which can meet specific end-to-end delay bound. Our APHD shares the same spirit by choosing a lowest satisfactory priority level to achieve a given end-to-end delay requirement. But our APHD scheme is different from NPDD-DCS's end-to-end class selection approach. Instead, APHD smartly adjust a packet's priority level at each intermediate hop based on a packet's per hop budget and the current node state.

Barry et al [14] proposed VMAC, a passive monitoring technique which can estimate the level of service a node would get without actually loading the channel. While sharing the spirit of passive monitoring, our APHD approach mainly keeps track of the delay of individual packets flowing through a node, and update this information in packets' header such that the next-hop node can utilize this information to adjust a packet's priority level.

Kanodia et al [20] proposed a framework of distributed scheduling and MAC access to assurance end-to-end delay. In their proposal, intermediate node can adjust a packet's priority based on delay budget requirement. For MAC access, RTS/CTS packets will carry a node's head-of-line packet's priority so that a winner can be determined based on its highest priority. Our work shares a fundamentally common point in adjusting individual packet's access priority on the way. Our approach is distinguished from [20] in that our work is based on 802.11e where there are multiple queuing entities for individual priority classes, each with an independent set of MAC access parameters. Our priority adaption is to map a delay budget to a specific class index. This is different from [20] where MAC access is adjusted by changing the contention window size directly. Intuitively, being able to change backoff parameters directly could result in an arbitrary number of priority levels, which makes it more chanlleging to monitor or predict the access delay for a particular parameter setting. For such reasons, using a fixed (and small) number of different access parameter settings as in 802.11e could help improve the predictability of per class delay PCD[i].

Yang and Kravets [19] proposed a QoS protocol for ad hoc realtime traffic (QPART) to provide end-to-end QoS guarantees for ad hoc networks. QPART breaks down end-to-end delay requirement evenly and propagates the per hop requirement to intermediate hop nodes using piggyback packets. Intermediate nodes dynamically adjust the contention window size based on individual flow's per hop requirement. Our APHD scheme is distinguished from QPART mainly in three ways: how to infer the per hop delay requirement (budget), how to populate the per hop delay budget to individual hops, and how to carry out each hop's adaptation procedure. Specifically, APHD does not depend on precomputed per hop requirement. Instead, APHD dynamically infers per hop budget at individual middle hop. APHD packets carry delay requirement information in packet header, and this approach is expected to provide more resilience in presence of route change, at a cost of increased overhead in

packet size. APHD provides MAC priority adaptation on per packet granularity, which is more flexible and efficient. Finally, our scheme depends on a small set of priority queues as defined in 802.11e, which have fixed contention and backoff parameters. This makes it more predictable for intermediate hop nodes to conduct node state monitoring.

Carlson et al [21] proposed a distributed end-to-end reservation protocol for 802.11 based wireless mesh networks. Their protocol is based on reservation setup and packet time slot scheduling to achieve non-varying delay and constant throughput for each reserved flow. Our work is based on 802.11e and does not require any kind of reservation of resources in the network, which we believe is a virtue considering the dynamic nature of wireless multihop networks.

Lorenz and Orda [22,23] investigated the problem of optimal resource allocation for end-to-end QoS requirements on unicast paths and multicast trees. The proposed a framework where end-to-end QoS requirement into local requirement based on each network link's cost function, such that the overall cost is minimized. A more recent work [24] investigated the intra-path load balancing problem on how to partition the end-to-end QoS requirement of a network flow along the links of a given path such that the deviation in the loads on these links is as small as possible. Our APHD scheme adopts a different approach to break down end-to-end requirement. As stated previously, APHD calculates individual packets' per hop requirement on-the-fly while data packets propagate in the network. This distributed and localized attribute makes our scheme more suitable for dynamic wireless ad hoc networks.

6 Conclusion

We present an adaptive per hop differentiation (APHD) scheme towards providing end-to-end delay assurance in 802.11e based multihop wireless networks. APHD is based on a cross-layer design approach and it can utilize information from data packet header as well as from intermediate nodes' state monitoring module so as to smartly adjust individual data packets' priority level and thus achieve an end-to-end QoS requirement. Simulation results show that APHD can provide excellent end-to-end delay assurance while achieving much higher network utilization, compared to pure EDCA scheme.

Note: This paper is based on a preliminary version of our work [7]. we have made significant extensions to our previous work in several aspects. We add more details to the architecture design of our scheme. We obtain more complete simulation results from different scenarios to validate our proposal. Other extensions include more elaboration in background, motivation, as well as related work sections.

7 Acknowledgement

We would like to thank the anonymous reviewers for their insightful comments, which have helped improve the quality and presentation of our paper.

References

[1] D. Wu. QoS Provisioning in Wireless Networks. Wiley Wireless Communications and Mobile Computing (WCMC) journal, 2005, vol. 5, pp. 957–969.

[2] A. Duda, and C.J. Sreenan. Challenges for Quality of Service in Next Generation Mobile Networks. Proc. of Information Technology & Telecommunications Conference (IT&T), Oct. 2003.

[3] P. Mohapatra, J. Li, and C. Gui. QoS in Mobile Ad hoc Networks. IEEE Wireless Communications Magazine, June 2003.

[4] Q. Zhang, F. Yang, and W. Zhu. Cross-Layer QoS Support for Multimedia Delivery over Wireless Internet. EURASIP Journal on Applied Signal Processing 2005:2, 207-219.

[5] S.B. Lee et al. INSIGNIA: An IP-Based Quality of Service Framework for Mobile Ad Hoc Networks. J. Parallel and Dist. Comp., vol. 60 no. 4, Apr. 2000, pp. 374–406.

[6] S. H. Shah, K. Chen, and K. Nahrstedt. Cross Layer Design for Data Accessibility in Mobile Ad Hoc Networks. Proc. of SCI 2001/ISAS 2001, Orlando, Florida, July 2001.

[7] J. Li, Z. Li, and P. Mohapatra. APHD: End-to-End Delay Assurance in 802.11e Based MANETs. Proc. of Mobiquitous 2006.

[8] IEEE 802.11 WG. IEEE 802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

[9] IEEE 802.11 WG. IEEE 802.11e/D8.0, Draft amendment to 802.11: Medium Access Control (MAC) Quality of Service (QoS) Enhancement, Feb. 2004.

[10] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource ReSerVation Protocol. IEEE Network, September 1993.

[11] I. Stoica and H. Zhang. Providing guaranteed services without per flow management. In Proc. ACM SIGCOMM, Boston, MA, September 1999.

[12] S. Shenker, R. Braden, and D. Clark. Integrated services in the Internet architecture: an overview. Internet RFC 1633, June 1994.

[13] S. Blake et al. An architecture for Differentiated Services. Internet RFC 2475, Dec. 1998.

[14] M.G. Barry, A.T. Campbell, and A. Veres. Distributed Control Algorithms for Service Differentiation in Wireless Packet Networks. Proc. of INFOCOM 2001.

[15] Y. Yang and R. Kravets. Contention-Aware Admission Control for Ad Hoc Networks. IEEE Transactions on Mobile Computing, Vol. 4/4, pp.363-377, 2005.

[16] I. D. Chakeres and E. M. Belding-Royer. PAC: Perceptive Admission Control for Mobile Wireless Networks. Proc. of QShine 2004, Dallas, TX, October 2004.

[17] K.C. Wang and P. Ramanathan. End-to-end delay assurance in multihop wireless local area networks. IEEE Globecom, Dec. 2003

[18] C. Dovrolis, and P. Ramanathan. Dynamic class selection and class provisioning in proportional differentiated services. Computer Communications Journal, 26(3):204-221, February 2003.

[19] Y. Yang, and R. Kravets. Distributed QoS guarantees for realtime traffic in ad hoc networks. IEEE SECON 2004.

[20] V. Kanodia, C. Li, A. Sabharwal, B. Sadeghi, E. Knightly. Distributed Multi-Hop Scheduling and Medium Access with Delay and Throughput Constraints. MobiCom 2001.

[21] E. Carlson, C. Prehofer, C. Bettstetter, H. Karl, and A. Wolisz. A Distributed End-to-End Reservation Protocol for IEEE 802.11-Based Wireless Mesh Networks. IEEE JSAC, Volume 24, Issue 11, Nov. 2006 Page(s):2018 - 2027.

[22] D.H. Lorenz, and A. Orda. Optimal partition of QoS requirement on unicast paths and multicast trees. IEEE Infocom, 1999.

[23] D.H. Lorenz, and A. Orda. Optimal partition of QoS requirement on unicast paths and multicast trees. IEEE/ACM Transaction on Networking, 10(1):102–114, February 2002.

[24] K.Gopalan, T.C. Chiueh, and Y.J. Lin. Delay Budget Partitioning to Maximize Network Resource Usage Efficiency. IEEE INFOCOM 2004.

[25] Network Simulator (NS-2). Source code and documentation online http://www.isis.edu/nsnam/ns/.

[26] S. Wiethõlter, and C. Hoene. IEEE 802.11e EDCF and Simulation Model for NS-2 (online), http://www.tkn.tu-berlin.de/research/802.11e_ns2/.

[27] S. Wiethõlter, and C. Hoene. Design and Verification of an IEEE 802.11e EDCF Simulation Model in NS-2.26. Technical Report TKN-03-019, Telecommunication Networks Group, Technische Universität Berlin, November 2003.