APHD: End-to-End Delay Assurance in 802.11e Based MANETs

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*—**

In this paper we present an adaptive per hop differentiation (APHD) scheme towards achieving end-to-end delay assurance in multihop wireless networks. Our APHD scheme extends the capability of IEEE 802.11e EDCA technique into multihop environments by taking end-to-end delay requirement into consideration. 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 a 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.

I. Introduction

Quality of services (QoS) support in mobile ad hoc networks (MANET) is very challenging due to their limited resources and the dynamic nature. Various techniques, from physical layer upto application layer, have been proposed to provide QoS support in MANET environments [1]. In this paper we propose an adaptive per hop differentiation (APHD) scheme towards achieving end-to-end delay assurance in multihop wireless networks.

APHD scheme is based on IEEE 802.11e EDCA technique which is proposed for service differentiation in sing hop environments (WLAN). EDCA 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. 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 organization of the rest of the paper is as follows. In Section II we discuss some background knowledge and the motivation of our work. Detailed design of APHD scheme is presented in Section III, followed by simulation based performanace evaluation in Section IV. Related work of interest to our work is discussed in Section V, and then the paper is conluded in Section VI.

II. Background and Motivation

A. IEEE 802.11

IEEE 802.11 standard [2] defines two access modes in wireless local area network (WLAN), namely, Distributed Coordinate Function (DCF), and Point Coordinate Function (PCF).

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: $backoff\_time = random() \times SlotTime$, 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 $CW\_min \leq CW \leq CW\_max$.

B. IEEE 802.11e

IEEE 802.11e [3] was proposed to supplement IEEE 802.11 MAC in order to provide service differentiation in WLAN. 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.

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 $CW\_min[AC] \leq CW[AC] \leq CW\_max[AC]$.

At 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. 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 and get more chance to utilize the channel.

C. Motivation for Our Design

In multihop wireless networks, time sensitive applications normally require end-to-end delay assurance. How can we utilize single hop service differentiation as defined in 802.11e EDCA to satisfy an end-to-end 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 piggyback this information to intermediate nodes along the route (for example, see [8]). 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. In APHD, 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 channel status (i.e., per class delay PCD[i]) 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.

III. APHD Design

In this section we discuss three major components of our APHD scheme and then put them together.

A. 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 I.

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 I
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 nodes 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 III-B.

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.

B. 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, it 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. 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:

$$pkt\_delay(i) = T_b - T_a. \quad (1)$$

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]:

$$PCD[i] = (1 - \alpha) * PCD[i] + \alpha * pkt\_delay(i). \quad (2)$$

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:

$$T'_b = T_g + \frac{packet\_size}{transmission\_rate}. \quad (3)$$

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.

We would like to point out that, in the above dissusion, the computation of PCD[i] information at a node is dependent on transmitting several packets belonging to that class. An alternative (or complementary) way would be to eavedrop packets passing-by in the neighborhood.

C. 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 a first hop node, we optimistically select a low service level (i.e. with larger priority number) which fits the packet's per hop budget. We would like to point out that, at first hop, packet priority adaptation 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.

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 Algorithm 1, written in a pseudo code similar to the C language.

Algorithm 1 Middle Hop Priority Adaptation

```
at intermediate node:
MAC receives an incoming packet p
e2e_budget_perhop = e2e_delay_req / e2e_hops;
budget_so_far = e2e_budget_perhop × hops_so_far;
slack = budget_so_far − delay_so_far;
if ( slack ≤ 0)   /* being late; try to speed up */
  for ( i = PRIO_min; i ≤ PRIO_max; i++ )
    if ( PCD[i] ≤ PCD_THRESHOLD[i] )
      break;   /* 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 = budget2go / hops2go;
  for ( i = PRIO_max; i ≥ PRIO_min; i−− )
    if ( PCD[i] ≥ PCD_THRESHOLD[i] )
      continue;   /* skip this priority level */
    endif
    if ( PCD[i] ≤ budget2go_perhop )
      break;   /* select priority level i */
    endif
  endfor
endelse
set p's priority: prio = i;
```

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

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.

D. Putting It Together

The overall architecture of APHD is shown in Fig. 1. 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, 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 1. 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. Now, both locally generated packets and incoming packets are handed to the Routing Agent (DSR). The output from DSR will be divided into two parts: local drop traffic and outgoing traffic. Local drop 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 levels. 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, and update packet header $delay\_so\_far$ field as well as the node state (PCD[i]) accordingly.

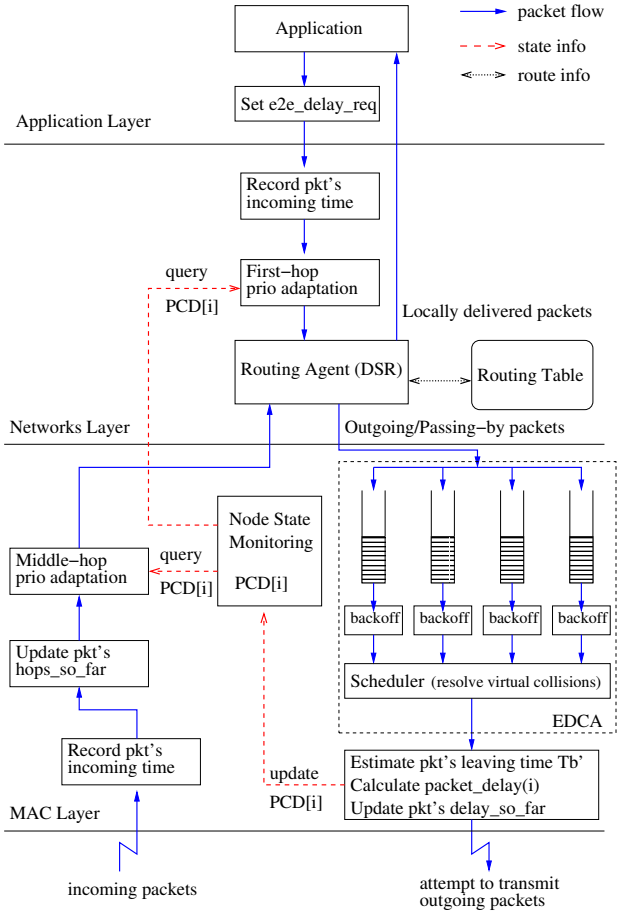


Fig. 1. APHD's overall architecture

IV. Performance Evaluation

We have done extensive simulations using NS-2 (version 2.26) [11] 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 [10]. 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 II and III, 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 *us*
CCATime	15 *us*
RxTxTurnaroundTime	5 *us*
SIFSTime	10 *us*
PreambleLength	144 bits
PLCPHeaderLength	48 bits
PLCPDataRate	1 *Mbits/s*
MaxPropagationDelay	2 *us*

TABLE II
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 III
NS-2 EDCA SETTINGS FOR IEEE 802.11E

A. Simple topology

1) Simulation setup: In this set of simulations we use a simple linear topology as shown in Fig 2. 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: 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.

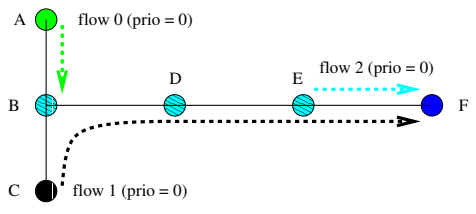


Fig. 2. A simple topology: same priority, different route length

2) Simulation results: The end-to-end delay for pure EDCA scheme is shown in Fig. 3. 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 inneglible. 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.

Now let's look at the throughput results. As shown in Fig. 4, 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. 3 and Fig. 4, we can see pure EDCA scheme cannot provide end-to-end assurance nor fairness in terms of end-to-end delay and throughput.

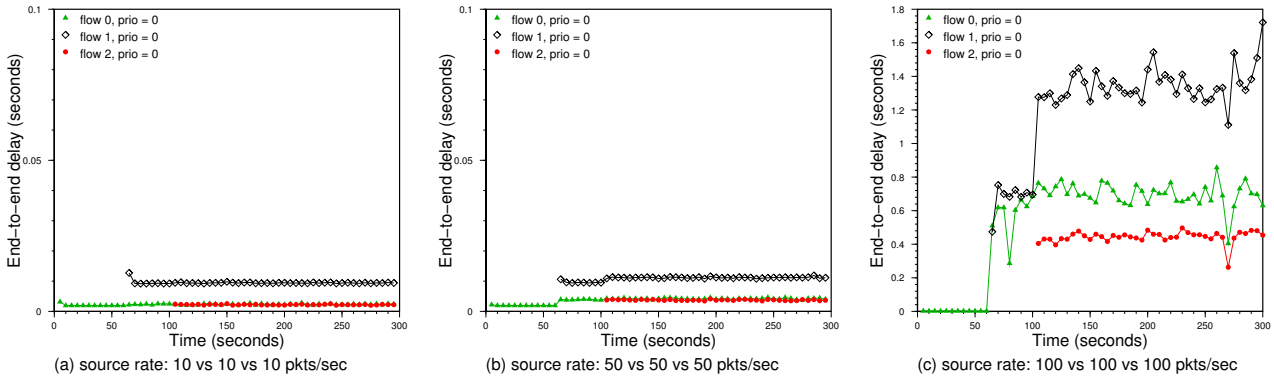
Now let's move forward to see the results of APHD tests. The end-to-end delay for APHD scheme is shown in Fig. 5. Even under source rate 100 packets/second, the end-to-end delay for all three flows still keep at very low level and there is no noticeable difference between flow 1 and flows 0/2.

Next let's see throughput results for APHD as shown in Fig. 6. 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.

B. Large Networks

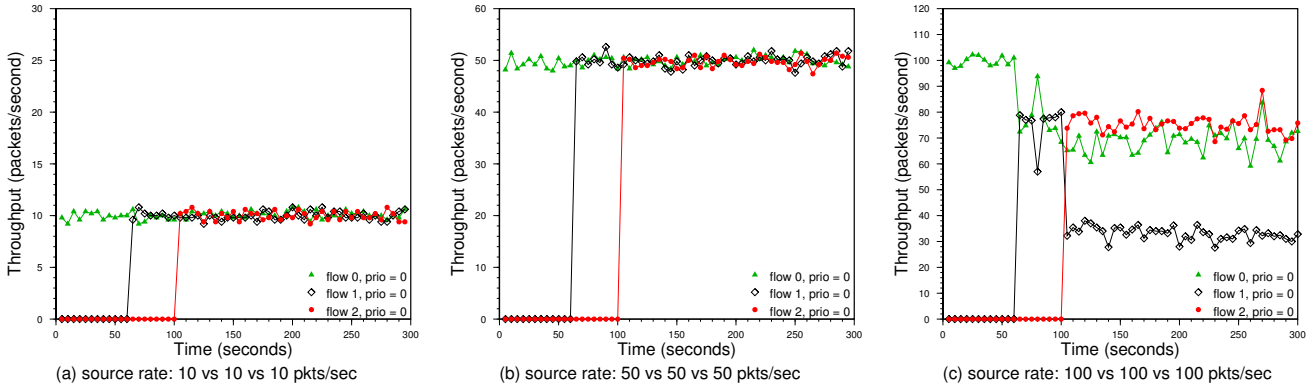
1) Simulation setup: To see how APHD performs in large networks, we use a 10x10 Manhattan networks. Adjacent nodes on each row (and each column) are separated 200 meters in distance. There are 12 flows in total among random pairs of source and destination. Flows 0-5 are real-time audio traffic with packet 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. In EDCA tests, 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.

2) Simulation results: First let see EDCA's performance in Fig. 7 and Fig. 8. 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.



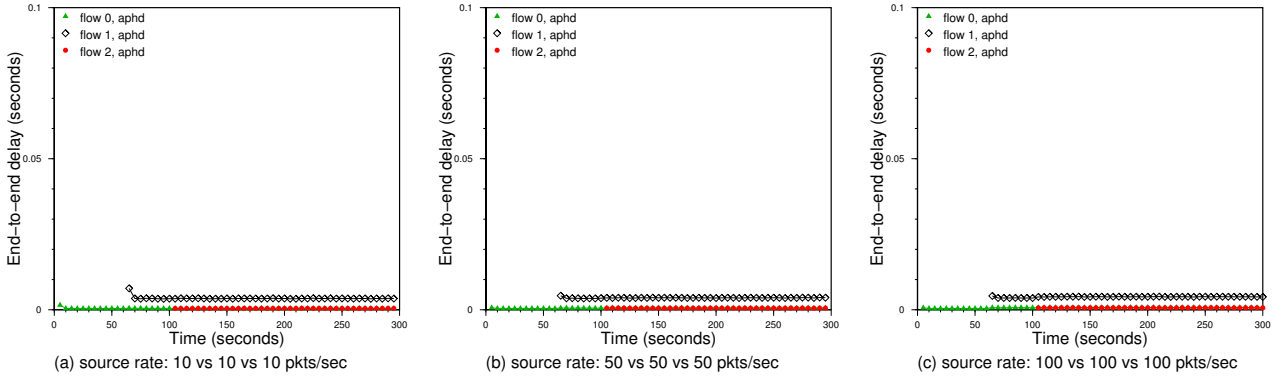
(a) source rate: 10 vs 10 vs 10 pkts/sec (b) source rate: 50 vs 50 vs 50 pkts/sec (c) source rate: 100 vs 100 vs 100 pkts/sec

Fig. 3. EDCA: end-to-end delay (different route length)



(a) source rate: 10 vs 10 vs 10 pkts/sec (b) source rate: 50 vs 50 vs 50 pkts/sec (c) source rate: 100 vs 100 vs 100 pkts/sec

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



(a) source rate: 10 vs 10 vs 10 pkts/sec (b) source rate: 50 vs 50 vs 50 pkts/sec (c) source rate: 100 vs 100 vs 100 pkts/sec

Fig. 5. APHD: end-to-end delay (different route length)

Now let's see APHD's results as shown in Fig. 9 and Fig. 10. Under all three source rates, both audio and video traffic can achieve very low end-to-end delays. The throughput shows no packet loss even when the source rate is 16x6 vs 16x6 packets/second. These results show that APHD performs efficiently in large networks as well.

V. Related Work

Dynamic Packet State (DPS) [4] 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 [5] and DiffServ [6], that have been proposed for Internet QoS provisioning. In DPS technique, end-to-end delay requirement is carried in packet header, intermediate hop routers process incoming packets based on their packet

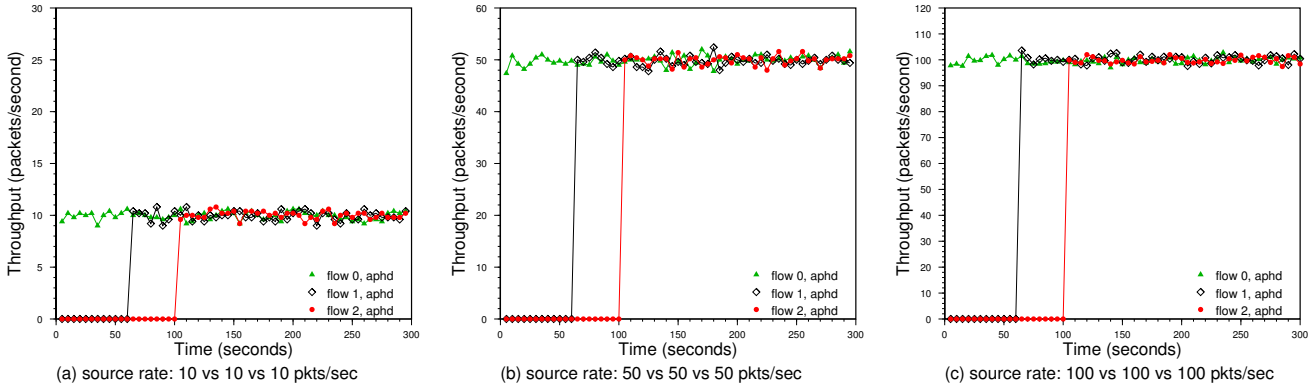


Fig. 6. APHD: end-to-end throughput (different route length)

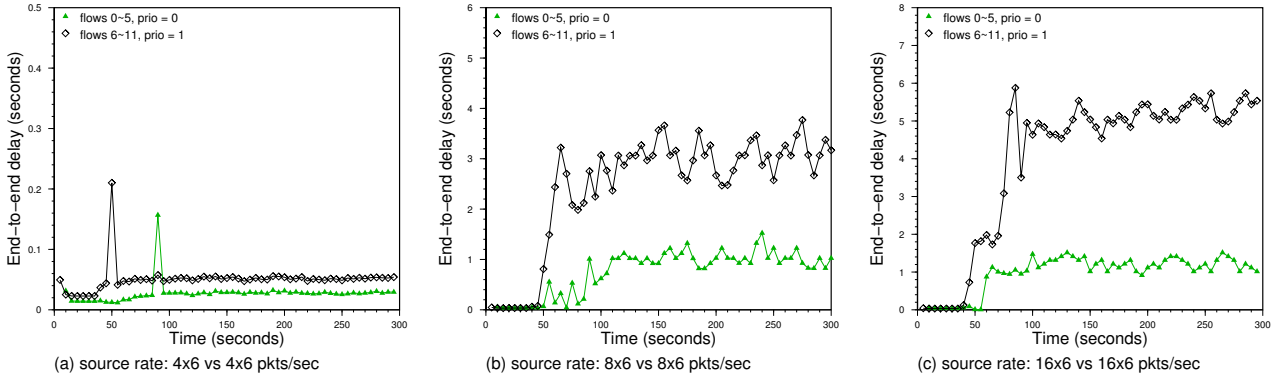


Fig. 7. EDCA: end-to-end delay (large network)

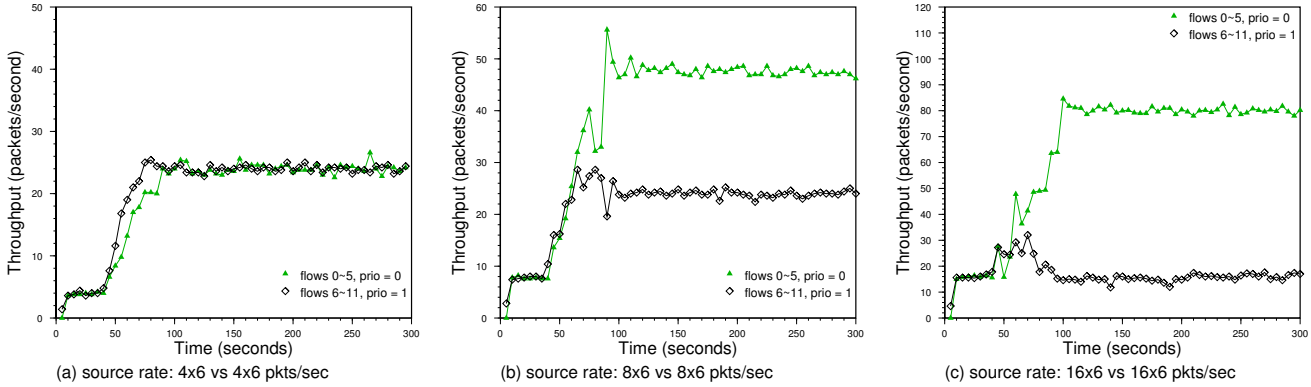


Fig. 8. EDCA: end-to-end throughput (large network)

header information and the current hop node state. In their implementation, they managed to find a way to encode all extended information in a standard IP packet header. Our APHD scheme follows this stateless approach, which does not require flow state maintenance at intermediate hop nodes.

INSIGNIA [7] 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 MANET environments 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. In contrast, APHD scheme utilizes localized per hop adaptation to satisfy an end-to-end requirement.

Yang and Kravets [8] proposed a QoS protocol for ad hoc realtime traffic (QPART) to provide end-to-end QoS guarantees for ad hoc networks. QPART breaks down

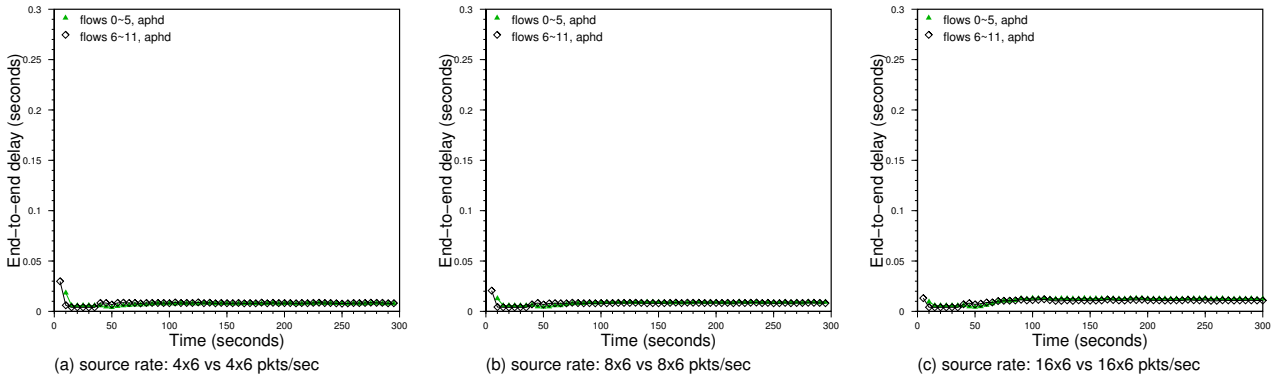


Fig. 9. APHD: end-to-end delay (large network)

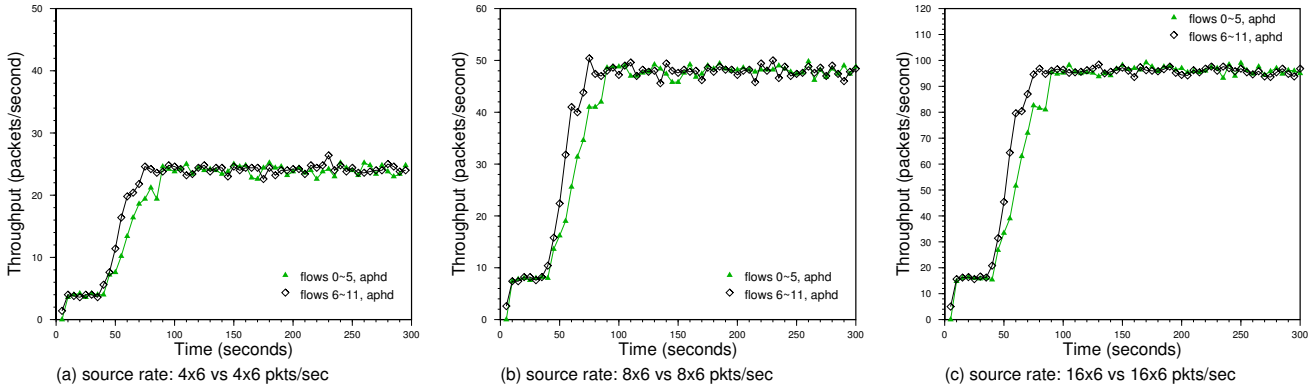


Fig. 10. APHD: end-to-end throughput (large network)

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 in the way on how to infer the per hop requirement as well as the granularity of adaptation procedure. APHD provides MAC priority adaptation on per packet granularity by using a small set of priority queues, which is more flexible and efficient.

Lorenz and Orda [9] investigated the problem of optimal resource allocation for end-to-end QoS requirements on unicast paths and multicast trees. APHD uses a different approach and calculates individual packets' per hop requirement on-the-fly while data packets propagate in the network.

VI. Conclusion

We present an adaptive per hop differentiation (APHD) scheme towards providing end-to-end delay assurance in 802.11e based multihop wireless networks. Simulation results show that APHD can provide excellent end-to-end delay assurance while achieving much higher network utilization, compared to a pure EDCA scheme.

References

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

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

[3] 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.

[4] I. Stoica and H. Zhang. Providing guaranteed services without per flow management. In Proc. ACM SIGCOMM 1999.

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

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

[7] 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.

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

[9] 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.

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

[11] NS-2, http://www.isis.edu/nsnam/ns/.