Achieving Delay Guarantees in Ad Hoc Networks Using Distributed Contention Window Adaptation

Yaling Yang, Robin Kravets
University of Illinois at Urbana-Champaign
{yyang8, rhk}@cs.uiuc.edu

***Abstract*— In this paper, we propose a new protocol, named DDA (Distributed Delay Allocation), which provides average delay guarantees to real-time multimedia applications in wireless ad hoc networks. By adapting the contention window sizes of IEEE 802.11, DDA schedules packets of flows according to their individual delay requirements. The novelty of DDA is that it imposes no control message overhead on the network and does not depend on explicit knowledge of channel capacity. We rigorously prove the convergence property of DDA and show that it always converges to a contention window allocation that satisfies all competing realtime flows' delay requirements, if the requirements of all realtime flows do not exceed the capacity of the network.**

I. INTRODUCTION

The fast development of wireless ad hoc networks requires support for both best effort and realtime applications. Unlike best effort applications, realtime applications require quality of service (QoS) guarantees. Depending on the type of QoS requirements, realtime flows can be divided into two categories: throughput-sensitive realtime flows and delay-sensitive realtime flows. While throughput-sensitive realtime flows, such as on-demand multimedia retrieval or video/audio broadcasting, require only throughput guarantees, delay-sensitive realtime flows, such as video/audio teleconferencing, require both throughput and end-to-end delay guarantees.

Due to the wide availability and inexpensive price of IEEE 802.11 [1] technology, many studies have focused on providing delay and throughput guarantees in ad hoc networks based on IEEE 802.11. Effective throughput guarantees can be provided by performing admission control based on measurements of idle channel time [2], [3], [4], [5], which ensure that a newly admitted realtime flow gets its desired throughput and does not affect the throughputs of existing realtime flows. Such guarantees can be made as long as the bandwidth requirement of a new realtime flow does not exceed the amount of idle channel time in the network before the new flow starts. However, similar approaches that use admission control based on delay measurements to provide delay guarantees, such as VMAC [6] and SWAN [7], have been shown to be not as successful [8]. Because the delay of a flow is related to the packet scheduling between the competing flows at neighboring nodes, the arrival of a new flow at a node changes the packet scheduling and affects the delay of all of the nearby flows. Hence, the packet delay measured before the new flow starts is usually much smaller than the packet delay that the new flow experiences after it starts and does not reflect the impact of the new flow on the delay of existing flows at other nodes. Therefore, it is very difficult to design measurement-based admission control protocols to provide delay guarantees to new flows without admitting new flows that may degrade the delay of existing realtime flows in the neighboring area.

Since admission control based approaches cannot effectively support delay requirements, approaches based on scheduling protocols must be used to provide delay guarantees to realtime flows. Current scheduling approaches, such as [9], [10], try to provide per-flow delay guarantees by mimicking centralized scheduling algorithms from wired networks. This type of approach requires nodes to exchange packet deadline information with competing neighbors through IEEE 802.11's MAC layer handshakes. However, exchanging such information not only imposes high message overhead, but may not be possible in an ad hoc network since competing nodes may be located outside each other's transmission range and inside each other's carrier-sensing range. Alternative approaches explore the fact that the contention window size in IEEE 802.11 is related to the schedules of packets, which in turn affect the delay of a flow. Based on this observation, protocols such as IEEE 802.11e [11], DWTP [12], DFS [13] and [14] have been proposed to provide delay differentiation by allocating different contention window sizes to different classes of flows. The benefit of such an approach is that it does not require any message exchange between competing neighboring nodes. However, this type of approach only provides delay differentiation and does not guarantee the actual average delay of an individual flow.

Because of the limitations of existing scheduling approaches, the goal of our research is to design a new scheduling protocol for IEEE 802.11-type of networks, named DDA (**D**istributed **D**elay **A**llocation). Assuming that the throughputs of flows are already ensured by the use of existing admission control approaches that provide throughput-guarantees [2], [3], [4], [5], DDA is able to provide average delay guarantees so that delay-sensitive realtime applications, which require both delay and throughput guarantees, can be supported. The novelty of DDA is that it provides per-flow average delay guarantees, while requiring no message exchanges among neighboring nodes.

DDA achieves delay guarantees by adapting the contention window size of competing nodes that carry delay-sensitive flows according to the flows' individual delay requirements. The design of the adaptation algorithm is based on our novel analysis of the relationship between the delay of flows and the contention window sizes of nodes. Based on this analysis, we design DDA so that at its converged point, individual delay-sensitive flows' average delay can meet their requirements.

There are four major contributions of this paper. First, we identify and model, for the first time, the closed-form relationship between contention window size and the distribution of the delay of a flow in an unsaturated network. Existing delay models [15], [16] only provide closed-form delay expressions when every node in the network is saturated, which is an unlikely and undesirable situation in a network that supports realtime traffic. Current models that analyze unsaturated networks [17] do not provide closed form relationships between packet delay and contention window sizes. Second, even though the packet delay at a node depends on the contention window sizes of *all* competing nodes, the design of DDA does *not* require any information exchanges between neighbors. In DDA, a node adjusts its own contention window size based on locally available information. Third, even though adjusting the contention window size at one node may affect the delay at other nodes, we rigorously prove that under DDA, the system automatically converges to a contention window allocation that can satisfy the delay requirements of all competing flows if such an allocation exists. Finally, extensive simulation results confirm that DDA has very good performance for providing delay guarantees.

The remainder of this paper is organized as follows. Section II briefly reviews the IEEE 802.11 protocol. Section III decomposes end-to-end packet delay requirements and shows that they can be translated into the requirements for the MAC layer's contention delay at each hop of the flow. Section IV then analyzes the relationship between contention delay and contention window size. Based on this relationship, Section V presents our DDA algorithm that provides delay guarantees to competing flows by adapting contention window size. Section VI shows the convergence property of DDA. Section VII deals with mobility-caused conflicts between delay requirements of competing flows. Section VIII evaluates the performance of DDA via simulation. Section IX concludes our work.

II. Contention Resolution

Packet delay in any network with a shared medium is related to the contention resolution algorithm in its MAC layer. To understand how delay can be guaranteed in IEEE 802.11 [1], we need to understand how contention is resolved. In IEEE 802.11 DCF, before a transmission, a Node i must determine whether the medium is busy or idle. If the medium remains idle for DIFS time units, Node i can transmit. If the medium was initially busy or changed from idle to busy during the DIFS, Node i must defer its transmission until the medium turns from busy to idle and remains idle for DIFS time. Then, Node i starts a backoff timer which expires after a certain period of time called *backoff time*. The backoff time is determined as *Backoff Time = Random()* $\times \epsilon$, where $Random()$ is a pseudo-random integer uniformly distributed in $[0, W_i - 1)$, *backoff slot*, ϵ, is a very small time period and the contention window W_i is a positive integer larger than 1. For every idle ϵ period, the backoff timer is decremented by ϵ. The timer is stopped when the medium is busy and restarted after the medium is idle for a DIFS. When the timer expires, Node i can transmit. The transmission includes either a four-way RTS-CTS-DATA-ACK handshake or just a two-way DATA-ACK handshake. After each transmission, an additional backoff process must be performed whether Node i has additional packets for transmission or not.

Demonstrated by experiments, simulations and theoretical analysis [11], [15], W_i is related to the QoS experienced by Node i. Essentially, $2/W_i$ is the probability that Node i transmits in one of its backoff slots. Reducing W_i at Node i gives Node i higher transmission probability when it competes with neighboring nodes, which results in shorter packet delays for Node i's traffic. Existing protocols, such as IEEE 802.11e [11], DWTP [12], DFS [13] and [14], use this fact to provide service differentiation. The focus of our work is to exploit this property to provide average delay guarantees.

III. Delay Requirement

An application's delay requirement is typically presented in terms of end-to-end packet delay. Since end-to-end packet delay is the aggregation of the delays from each hop of the flow, for any algorithm to guarantee end-to-end delay, it must control the packet delay at each hop. Therefore, the end-to-end delay requirement of a flow must be divided into per-hop delay requirements. In this paper, we use a simple strategy where the end-to-end delay requirement is evenly divided into per-hop delay requirements based on the hop count of the flow. We are currently investigating other options, such as allocating per-hop delay requirements based on channel utilization at each hop.

Given a per-hop delay requirement, a relaying node of a realtime flow should control its delay to meet this requirement. However, the per-hop delay at a node is composed of multiple components. Since some of these components are correlated, by changing one component, a delay-aware algorithm may affect other components. Therefore, for any delay-aware algorithm to ensure per-hop delay requirements, it must understand the relationship between these components. Then, by identifying a single dominant controllable component that determines all other controllable components, the algorithm can translate per-hop delay requirements into requirements for this single dominant component and focus on meeting these requirements.

In the rest of this section, we identify this single dominant delay component as the mean of contention delay, which will be defined in Section III-B. To support this claim, we first decompose the expected per-hop packet delay into three components, *average queueing delay*, *average transmission delay* and *average contention delay*, and show that average transmission delay is fixed and average queueing delay is determined by both the mean and the variance of contention delay. Next, we categorize the contention delay into two different types: *busy delay* and *idle delay*. Finally, through analysis of the distribution of busy delay and idle delay, we demonstrate that the variance of the contention delay is determined by the mean of the contention delay. Therefore, the average per-hop delay is essentially a function of the average contention delay, which is the single dominant delay component that we are looking for. By translating per-hop delay requirements into requirements for average contention delay, we can then design our algorithm to focus on achieving average contention delay requirements.

A. Assumptions

To simplify the analysis, given a certain contention window allocation of a network, we make the following four assumptions. First, the packet inter-arrival times, A_i, for any Node i are independent and identically distributed variables and so are the MAC layer packet service times, X_i. Second, A_i is independent of both X_i and any other Node j's A_j. Our simplification does not consider the impact of the service time that a flow receives at one hop on the packet arrival processes at the following hops, which is very difficult to model given the complexity of wireless networks and is still an open problem. However, simulation results show that this simplification is valid since our DDA protocol, designed based on this simplification, has very good performance. Third, $E[A_i] \geq E[X_i]$. This relationship holds because the use of both flow control for best effort traffic, such as TCP, and admission control for realtime traffic, such as [2], [3], [4], [5], ensures that the average packet arrival rate at any Node i is the same as Node i's average packet departure rate. Finally, we assume that every node has a large enough queue size so that packet delay can be approximated using queueing models that assume infinite queue size.

Notation for the entire paper can be found in Appendix B.

B. Decomposition of Per Hop Delay Requirement

In IEEE 802.11, the delay that a realtime packet experiences at Node i, d_i, is composed of three components: the queueing delay, the contention delay at the MAC layer and the transmission delay. The *queueing delay*, d_i^q, is the interval between the time that the packet arrives at Node i and the time that the packet becomes the head of line (HOL) packet in Node i's queue. The *contention delay*, d_i^c, is the interval between the time that the packet becomes the HOL packet and the time that the packet actually starts to be transmitted on the physical medium. This contention delay is unique for contention-based channel access schemes. It captures the fact that when a packet becomes the HOL packet at Node i, Node i may need to backoff before transmitting the packet on the physical medium. During Node i's backoff time, if a neighbor of Node i transmits, Node i must pause its backoff timer until this neighbor finishes its transmission. Therefore, the contention delay is related to the characteristics of the contention for the channel between neighboring nodes. The *transmission delay*, d_i^t, is the duration of a successful packet transmission at the physical medium, which equals the duration of a whole RTS-CTS-DATA-ACK or DATA-ACK handshake depending on the operating mode of IEEE 802.11. For simplicity of presentation, we assume that packet sizes are fixed for all nodes. Therefore, d_i^t is a constant. Other packet size distributions only require a straightforward modification of the analysis. Given the above decomposition, the expected packet delay at Node i can be expressed as:

$$E[d_i] = E[d_i^q] + E[d_i^c] + d_i^t. \tag{1}$$

To determine $E[d_i^q]$, according to queueing theory [18], d_i^q is determined by packet inter-arrival time, A_i, and MAC layer's packet service time, X_i. Since $X_i = d_i^c + d_i^t$,

$$E[X_i] = E[d_i^c] + d_i^t, \tag{2}$$
$$Var(X_i) = Var(d_i^c). \tag{3}$$

Therefore, based on G/G/1 queueing theory [18], $E[d_i^q]$ can be bounded as follows:

$$\frac{\rho_i(\rho_i - 2) + \lambda_i^2 Var(X_i)}{2\lambda_i(1-\rho_i)} \leq E[d_i^q] < \frac{Var(A_i) + Var(X_i)}{2(1-\rho_i)/\lambda_i}, \tag{4}$$

where $\rho_i = \lambda_i E[X_i] < 1$ and $\lambda_i = 1/E[A_i]$ is the average packet arrival rate at Node i. Equations (2), (3) and (4) imply that it is necessary to limit both $E[d_i^c]$ and $Var(d_i^c)$ to provide a bound on the queueing delay, d_i^q, which is required to bound the per-hop delay, d_i. While $E[d_i^c]$ is controllable by a node, $Var(d_i^c)$ is hard to control directly without understanding the distribution of d_i^c. Sections III-C and III-D examine the distribution of d_i^c and demonstrate that $Var(d_i^c)$ is bounded by a function of $E[d_i^c]$. Therefore, to bound the queueing delay, we only need to limit $E[d_i^c]$.

C. Types of the Contention Delay

To calculate $E[d_i^c]$ and $Var(d_i^c)$, it is necessary to understand the distribution of d_i^c. In this section, we show that d_i^c is related to the channel state at the arrival time of packets. Section III-D examines the distribution of d_i^c under different channel states.

Note that the channel state at Node i can be classified into three states: busy, idle and backoff. During the busy state, Node i's CSMA/CA mechanism indicates that the channel is busy since Node i or Node i's neighbors are actively communicating with each other. If a packet arrives at Node i during the busy state, the contention delay of the packet is the *busy delay*, d_i^b. During the idle state, none of the nodes in the neighborhood of Node i have backlogged packets so that the channel at Node i stays idle. If a packet arrives at Node i during the idle state, the contention delay of the packet is the *idle delay*, d_i^f. During the backoff state, nodes with backlogged packets are counting down their backoff timers and when the backoff timer of one of these nodes expires, a transmission happens on the channel and the backoff state turns into the busy state. The duration of a backoff state is usually less than 0.2 milliseconds [15], which is much smaller than the duration of a busy state or an idle state, which usually last several milliseconds. Therefore, the backoff state's effect on the distribution of d_i^c is negligible. Hence, the distribution of d_i^c is determined by the idle delay, d_i^f, and the busy delay, d_i^b.

To calculate d_i^f, note that when a packet k arrives at Node i's queue in the idle state, packet k must see an empty queue. Otherwise, if Node i's queue is not empty, packet k should see that either Node i is busy transmitting existing packets from the queue or Node i is waiting for other nodes to finish their transmissions. In both cases, the channel should be in the busy state, which violates the assumption that packet k sees an idle state when it enters Node i's queue. Hence, packet k will be transmitted by the MAC layer immediately after a DIFS defer time. Therefore,

$$d_i^f = DIFS. \tag{5}$$

To calculate d_i^b, note that when a packet k arrives during the busy state, Node i's queue may have packets or be empty. If packet k sees an empty queue when it arrives, packet k will be sent to the MAC immediately. Since the channel is busy at this time, packet k must wait a full backoff process before it can finally be transmitted. On the other hand, if the queue is not empty, packet k will only be sent to the MAC layer for transmission at the moment that all the packets ahead of packet k finish their transmissions. After every packet transmission in IEEE 802.11, a full backoff process must be performed. Therefore, in this case, packet k experiences a full backoff process too. Hence, d_i^b can be expressed as:

$$d_i^b = DIFS + w_i\epsilon + m_iT_d + m_i^cT_c, \tag{6}$$

where w_i is the number of backoff slots. T_d is the duration of a successful data transmission and T_c is the duration of a collision. m_i is the number of data packets transmitted by the neighboring nodes during Node i's backoff process and m_i^c is the number of collisions during the backoff process.

Combining Equations (5) and (6), d_i^c can be expressed as:

$$d_i^c = DIFS + \begin{cases} 0, & \text{in the idle state;} \\ w_i\epsilon + m_iT_d + m_i^cT_c, & \text{in the busy state.} \end{cases} \tag{7}$$

Equation (7) shows that the distribution of d_i^c is determined by the distributions of w_i, m_i, m_i^c and the probability that a packet arrives during the busy state. These distributions are examined in Section III-D to show that $Var(d_i^c)$ is determined by $E[d_i^c]$.

D. *Distribution of Contention Delay* d_i^c

By analyzing the distribution of d_i^c, we next show that $Var(d_i^c)$ is bounded by a function of $E[d_i^c]$, the single dominant component for per-hop delay, so that requirements for per-hop delay can be translated into requirements for $E[d_i^c]$. Our analysis includes two steps. First, we express $E[d_i^c]$ and $Var(d_i^c)$ with the moments of w_i and m_i. Then, by calculating these moments, we obtain the relationship between $E[d_i^c]$ and $Var(d_i^c)$.

1) Expressing $E[d_i^c]$ and $Var(d_i^c)$: Based on Equation (7), $E[d_i^c]$ and $Var(d_i^c)$ are related to the distribution of the three components: w_i, m_iT_d and $m_i^cT_c$. However, not all of the three components contribute equally to the value of $E[d_i^c]$ and $Var(d_i^c)$. In fact, we next show that the effects of $m_i^cT_c$ can be omitted in Equation (7), which greatly simplifies the calculation of $E[d_i^c]$ and $Var(d_i^c)$.

Given the probability that a collision happens in a backoff slot, P_c, and the probability that a successful transmission happens in a backoff slot, P_s, $E[m_i^cT_c] = w_iP_cT_c$, $E[m_iT_d] = w_iP_sT_d$, $Var[m_i^cT_c] = w_iP_c(1-P_c)T_c^2$ and $Var[m_iT_d] = w_iP_s(1-P_s)T_d^2$. Hence, to demonstrate that $m_i^cT_c$ can be omitted from Equation (7), we only need to show that $P_cT_c \ll P_sT_d$ and $P_c(1-P_c)T_c^2 \ll P_s(1-P_s)T_d^2$, which can be demonstrated by examining the relationship between channel load and the probability of collision as follows.

Consider a network with M nodes competing for the channel, where each node transmits its packet in a backoff slot with probability τ. The maximum channel capacity is achieved when the average duration between two successful transmissions, F, is minimized. Since the average number of backoff slots between two successful transmissions is $(1/P_s - 1)$,

$$F = \big[\frac{P_I}{1-P_s}\epsilon + \frac{P_c}{1-P_s}T_c\big]\big(\frac{1}{P_s} - 1\big), \tag{8}$$

where P_I is the probability that a backoff slot is idle. Since:

$$\begin{aligned} P_I &= (1-\tau)^M, \\ P_c &= 1-(1-\tau)^M - M\tau(1-\tau)^{(M-1)}, \\ P_s &= M\tau(1-\tau)^{(M-1)}, \end{aligned}$$

Equation (8) becomes:

$$F = \frac{(1-\tau)^M\epsilon + [1-(1-\tau)^M - M\tau(1-\tau)^{(M-1)}]T_c}{M\tau(1-\tau)^{(M-1)}}. \tag{9}$$

Letting $\frac{\partial F}{\partial \tau} = 0$ results in:

$$1 - \frac{\epsilon}{T_c} = \frac{1-M\tau}{(1-\tau)^M} \overset{M\to\infty}{=} (1-M\tau)e^{M\tau}. \tag{10}$$

From Equation (10), the optimal value of $M\tau$, $(M\tau)^*$, that results in a minimum F and achieves maximum channel capacity can be solved using MATLAB.

Note that in the targeted environment of DDA, existing admission control protocols for throughput-guarantee of realtime traffic [2], [3], [4], [5] and flow control protocols (e.g., TCP) for best effort traffic are used to ensure that the traffic load of flows are no larger than the maximum channel capacity. Hence, $M\tau < (M\tau)^*$. In this range $M\tau < (M\tau)^*$, $P_cT_c \ll P_sT_d$ and $P_c(1-P_c)T_c^2 \ll P_s(1-P_s)T_d^2$ as demonstrated Figure 1, which shows the relationship between P_cT_c, P_sT_d,$P_c(1-P_c)T_c^2$ and $P_s(1-P_s)T_d^2$ for various numbers of competing nodes as calculated in MATLAB. Therefore, $m_i^cT_c$ can be omitted from Equation (7).

Omitting $m_i^cT_c$ from Equation (7), $E[d_i^c]$ and $Var(d_i^c)$ are:

$$E[d_i^c] = DIFS + \{E[w_i]\epsilon + E[m_i]T_d\}p_i^b, \tag{11}$$

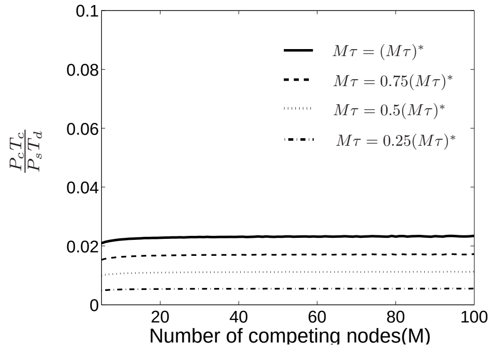
$$\begin{aligned} Var(d_i^c) = p_i^b\left\{E[(w_i)^2]\epsilon^2 + E[(m_i)^2]T_d^2 + 2E[m_iw_i]T_d\epsilon\right\} \\ - (p_i^b)^2\left\{E[w_i]\epsilon + E[m_i]T_d\right\}^2, \end{aligned} \tag{12}$$

where p_i^b is the probability that a packet sees a busy state upon arrival. We next calculate the moments of w_i and m_i, $E[m_iw_i]$ and p_i^b to show the relationship between $E[d_i^c]$ and $Var(d_i^c)$.

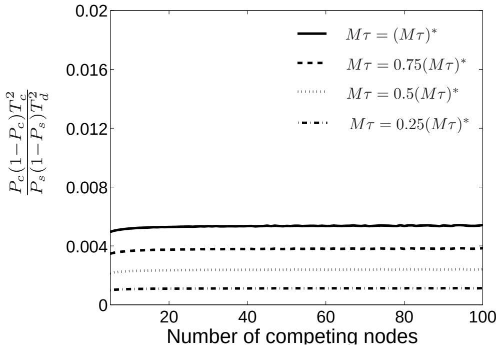
2) Moments of w_i, m_i and Mean of m_iw_i: Assuming that the probability that Node i successfully transmits a packet in a backoff slot is π_i, the first and second moments of w_i are:

$$E[w_i] = 1/\pi_i, \tag{13}$$

$$E[(w_i)^2] = \frac{2/\pi_i - 1}{\pi_i}. \tag{14}$$



(a) $\frac{P_c T_c}{P_s T_d}$



(b) $\frac{P_c(1-P_c)T_c^2}{P_s(1-P_s)T_d^2}$

Fig. 1. Relationship between $m_i^c T_c$ and $m_i T_d$. The channel transmission rate is 2Mbps and the data packet size is 512Byte.

If q_i is the probability that some neighbor transmits in one of Node i's backoff slots, then:

$$E[m_i] = E[E[m_i|w_i]] = E[q_i w_i] = q_i/\pi_i, \tag{15}$$

$$\begin{aligned} E[(m_i)^2] &= E[E[(m_i)^2|w_i]] = E[w_i q_i - w_i q_i^2 + q_i^2 w_i^2] \\ &= 2(1/\pi_i - 1)q_i^2/\pi_i + q_i/\pi_i, \end{aligned} \tag{16}$$

$$E[m_i w_i] = E[E[m_i w_i|w_i]] = q_i\left(2/\pi_i - 1\right)/\pi_i. \tag{17}$$

3) The Probability of the Busy State p_i^b: Note that p_i^b equals the probability that an arriving packet sees a busy channel, which happens when Node i or some of Node i's contending neighbors are transmitting. Therefore, assuming that the set of Node i's neighbors inside the carrier-sensing range is n_i,

$$p_i^b = \sum_{j\in n_i} \alpha_{j,i}\frac{\lambda_j}{\chi_j} + \frac{\lambda_i}{\chi_i}, \tag{18}$$

where χ_j is the physical channel transmission rate at Node j, λ_j is the average packet arrival rate at Node j and $\alpha_{j,i}$ is a positive discount factor in $[0,1]$. Essentially, the right side of the equal sign is the fraction of time that the channel at Node i is busy. We introduce the discount factor $\alpha_{j,i}$ because some of the neighbors of Node i may transmit concurrently if they are not in each other's carrier-sensing range. Therefore, even though $\frac{\lambda_j}{\chi_j}$ is the fraction of time that Node j transmits on the channel, the fraction of the busy period that Node i sees is not simply the summation of $\frac{\lambda_j}{\chi_j}$. For example, in Figure 2, Nodes A and C are not in each other's carrier-sensing range and can transmit concurrently. If Nodes A and C's packets always arrive simultaneously so that they always transmit concurrently, the fraction of the busy period that Node B experiences is $\max(\frac{\lambda_A}{\chi_A}, \frac{\lambda_C}{\chi_C})$. However, if Nodes A and C's packets always arrive sequentially, Nodes A and C may always transmit sequentially. In this case, the fraction of the busy period that Node B experiences is $\frac{\lambda_A}{\chi_A} + \frac{\lambda_C}{\chi_C}$. These two values bound the fraction of channel busy time that Node B sees. Hence, the actual amount of busy channel time at Node B is $\frac{\alpha_A\lambda_A}{\chi_A} + \frac{\alpha_C\lambda_C}{\chi_C}$, where the values of α_A and α_C are in the range of [0,1] and are determined by the level of concurrent transmissions from Nodes A and C. In general, if a Node i has multiple neighbors, Node i's busy channel time is captured in Equation (18).

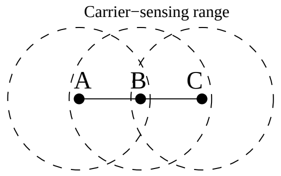


Fig. 2. Example: Nodes A and C can transmit concurrently.

It is important to note that the level of concurrent transmissions between Node i's neighbors is only related to their packet arrival distributions and is not related to their contention window sizes. Essentially, this means that p_i^b can be viewed as a fixed value when our DDA algorithm adapts contention windows at nodes. To understand why p_i^b is a fixed value, consider the example in Figure 2. During the period of time that Node B has no packets to transmit, it is obvious that Nodes A and C's concurrent transmission level is not related to contention window sizes. During the period of time that Node B is transmitting a packet, if both Nodes A and C have a packet arrival, Nodes A and C always transmit their packets concurrently. The contention window allocations of Nodes A, B and C only determine whether Node B will transmit its next packet before or after the concurrent transmissions of Nodes A and C. This is because after Node B finishes its current transmission, if Node B has another packet for transmission, both Nodes A and C compete with Node B. Depending on contention window size allocations, Node B may win the channel and both Nodes A and C need to wait again for Node B to finish its transmission. Finally, if either Node B finishes all of its transmissions or if either Node A or C wins the channel, both Nodes A and C will immediately transmit concurrently since Node B cannot transmit when either Node A or C is transmitting. Therefore, Nodes A and C are guaranteed to transmit concurrently if they both have packet arrivals during Node B's transmitting period. Given Node B's packet arrival rate, the fraction of time that Node B is transmitting on the channel is fixed. Therefore, the probability that Nodes A and C both have packet arrivals during Node B's transmitting period is not related to the allocation of contention window sizes. Hence, considering all of the cases discussed above, the concurrent transmission level of Nodes A and C is not related to the allocation of contention window sizes.

4) Relationship Between $E[d_i^c]$ and $Var(d_i^c)$: Combining Equations (11), (12), (13), (14), (15), (16) and (17), $E[d_i^c]$ and $Var(d_i^c)$ become:

$$E[d_i^c] = DIFS + \big(\frac{\epsilon}{\pi_i} + \frac{q_i}{\pi_i}T_d\big)p_i^b, \tag{19}$$

$$\begin{aligned} Var(d_i^c) &= p_i^b\big[\tfrac{2/\pi_i - 1}{\pi_i}\epsilon^2 + \tfrac{2(1/\pi_i - 1)q_i^2}{\pi_i}T_d^2 + \tfrac{q_i}{\pi_i}T_d^2 \\ &\quad + 2\tfrac{q_i(2/\pi_i - 1)}{\pi_i}T_d\epsilon\big] - (p_i^b)^2\big[\tfrac{\epsilon}{\pi_i} + \tfrac{q_i}{\pi_i}T_d\big]^2 \\ &< (E[d_i^c] - DIFS)^2(\tfrac{2}{p_i^b} - 1) + (E[d_i^c] - DIFS)T_d. \end{aligned} \tag{20}$$

Since p_i^b cannot be controlled by Node i, Equation (20) essentially shows that $Var(d_i^c)$ can be determined by $E[d_i^c]$. Therefore, $E[d_i^c]$ is the single dominant component in the average per-hop delay at Node i. Based on Equations (1), (2), (3), (4) and (20),

$$\begin{aligned} E[d_i] < \big[&Var(A_i)\lambda_i + (E[d_i^c] - DIFS)^2\big(\tfrac{2}{p_i^b} - 1\big) \\ &+ (E[d_i^c] - DIFS)T_d\big] / \big[2[1 - \lambda_i(E[d_i^c] + d_i^t)]/\lambda_i\big] \end{aligned} \tag{21}$$

Therefore, to ensure that a per-hop delay requirement, D, is satisfied, we only need to ensure that:

$$\begin{aligned} D = \big[&Var(A_i)\lambda_i + (E[d_i^c] - DIFS)^2\big(\tfrac{2}{p_i^b} - 1\big) \\ &+ (E[d_i^c] - DIFS)T_d\big] / \big[2[1 - \lambda_i(E[d_i^c] + d_i^t)]/\lambda_i\big] \end{aligned} \tag{22}$$

By solving $E[d_i^c]$ from Equation (22), the per-hop delay requirement D can be translated into a requirement on $E[d_i^c]$. Therefore, any QoS-aware protocol that provides end-to-end delay guarantees only needs to ensure that $E[d_i^c]$ at each hop is below the applications' requirements.

IV. Contention Delay vs. Contention Window Size

Intuitively, since contention window sizes of competing nodes statistically determine which node wins the channel during the competition for channel access, contention window size should be related to $E[d_i^c]$. Therefore, in this section, we analyze the relationship between contention window size and $E[d_i^c]$, so that we can design DDA to allocate different contention window sizes to different flows to ensure that their delay requirements for $E[d_i^c]$ are all satisfied.

To find the relationship between contention window size and $E[d_i^c]$, note that in Equation (19), the unknown component is q_i and π_i. The following analysis shows that both q_i and π_i are related to the packet arrival rates, λ_j, and the contention window sizes, W_j, at Node i's neighbors.

To calculate q_i, recall that multiple neighbors of Node i may transmit concurrently if they are not in each other's carrier-sensing range (See example in Figure 2). Therefore, if Node j transmits in a backoff slot of Node i with probability $q_{j,i}$,

$$q_i = 1 - \prod_{j \in n_i}(1 - \beta_{j,i}q_{j,i}), \tag{23}$$

where $\beta_{j,i}$ is the discount factor due to the concurrent transmissions between Node i's neighbors in a backoff slot. Similar to the $\alpha_{j,i}$ in Equation (18), $\beta_{j,i}$ is determined by the packet arrival processes of the neighboring nodes and is not related to contention window sizes. In addition, since a collision happens when Node i and some neighbors of Node i transmit in the same backoff slot, q_i is also the collision probability of Node i's transmission.

To formulate the relationship between λ_j, W_j and $q_{j,i}$, consider the example shown in Figure 2. As shown in Section III-C, if a packet arrives during an idle state, Node B transmits immediately. However, since during the idle state, none of Node B's neighbors are in a backoff state, Node B's transmission does not happen in their backoff slots. Therefore, the probability that Node B transmits in any neighbors' backoff slots is 0. For every packet that arrives at Node B in a busy state, Node B transmits with probability $2/W_B$ in its backoff slots. Since the average packet arrival rate at Node B is λ_B, the average number of packets that arrive at Node B during a busy period is $\lambda_B T_d$. $\lambda_B T_d$ is smaller than 1 since $E[X_B] > T_d$ and both best effort's flow control and realtime traffic's admission control ensure that $\lambda_B E[X_B] \leq 1$ (See Section III-A). Therefore, considering the fact that $W_B > 1$, the probability that Node B transmits in a backoff slot of Node A is:

$$q_{B,A} = \lambda_B T_d \frac{2}{W_B} < 1. \tag{24}$$

It is also possible that even if Node B has backlogged packets, it does not compete with Node A. This is because when Node C is transmitting, while Node A sees an idle channel, Node B cannot transmit due to its busy channel. However, from Node A's perspective, this is the same as Node B not having any backlogged packets during Node C's transmission. Therefore, the probability that Node B transmits in Node A's backoff slots can still be expressed by Equation (24).

Based on the analysis of the above two examples, in general, $q_{j,i} = \lambda_j T_d \frac{2}{W_j}$. Hence, Equation (23) becomes:

$$q_i = 1 - \prod_{j \in n_i}(1 - \beta_{j,i}\lambda_j T_d \frac{2}{W_j}). \tag{25}$$

Since q_i is also the collision probability of Node i's transmission and Node i transmits in a slot with probability $2/W_i$, the probability that Node i have a successful transmission in a slot is $\pi_i = 2(1 - q_i)/W_i$. Therefore, Equation (19) becomes:

$$E[d_i^c] = DIFS + p_i^b\big(\frac{\epsilon}{1 - q_i} + \frac{q_i T_d}{1 - q_i}\big)\frac{W_i}{2}. \tag{26}$$

Similar to the analysis in Section III-D.1, we can calculate the upper bound on q_i, which equals 0.2. When $q_i \in [0, 0.2]$, $\frac{q_i}{1-q_i} \approx 1.1788q_i$ and $\frac{1}{1-q_i} \approx 1 + 1.1788q_i$. Combining these approximations with Equations (25) and (26),

$$E[d_i^c] = H_1 + H_2\Big[1 + \gamma - \prod_{j \in n_i}\big(1 - G_{j,i}\frac{2}{W_j}\big)\Big]\frac{W_i}{2}, \tag{27}$$

where $H_1 \equiv DIFS$, $H_2 \equiv 1.1788(T_d + \epsilon)p_i^b$, $\gamma \equiv \frac{\epsilon}{1.1788(T_d + \epsilon)}$ and $G_{j,i} \equiv \beta_{j,i}\lambda_j T_d$. Note that H_1, H_2, γ

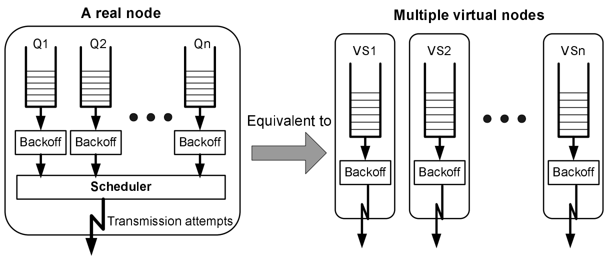


Fig. 3. Virtual nodes

and $G_{j,i}$ are all components of $E[d_i^c]$ that are not related to contention window size and cannot be controlled by relaying nodes.

Equation (27) shows that a node's contention delay is determined by both its own contention window size and the contention window sizes of all its competing neighbors. This implies three things. First, if there is enough network capacity for all of the realtime flows, by finding the right allocation of contention window sizes for nodes carry delay sensitive flows, the delay requirements of these flows can be satisfied. Second, since $G_{j,i}$, which depends on the packet arrival process at Node j, cannot be known to Node i, directly calculating the contention window allocation of Node i is very difficult. A more feasible method, which is used by DDA, is to use an iterative algorithm that adapts the contention window size at Node i based on local measurable information. Even though Node i does not know $G_{j,i}$, Node i can still gradually set its W_i to the right size. Third, adapting the contention window size at Node i may affect the delay of other neighboring delay-sensitive flows. Therefore, it is very important that any contention window adaptation algorithm does not cause system instability. In Section V, we discuss how DDA iteratively adapts the contention window sizes of nodes that carry delay-sensitive flows and Section VI proves that DDA does not cause system instability and that the converged point of DDA is the contention window allocation that satisfies every delay-sensitive flow's delay requirement.

V. Design of DDA (Distributed Delay Allocation)

For a delay-sensitive flow, there are both delay and throughput requirements. Since throughput is guaranteed through the use of existing methods, such as [2], [3], [4], [5], the goal of DDA is to ensure that the end-to-end packet delay is below the flow's end-to-end delay requirement.

DDA provides delay guarantees through the following process. First, the end-to-end delay requirement is evenly broken down into per-hop delay requirements and the first few packets of the flow piggyback the per-hop delay requirements to the relaying nodes. A relaying node, Node i, then translates its per-hop delay requirement into the requirement for $E[d_i^c]$ as discussed in Section III. Assuming that Node i only has one flow to relay and the requirement of the flow on $E[d_i^c]$ is Δ_i, Node i then controls its contention window size to ensure $E[d_i^c] \leq \Delta_i$. Since every relaying node along the route of the flow locally limits $E[d_i^c]$ below Δ_i, the aggregated end-to-end delay is maintained below the flow's requirement.

If Node i has multiple flows with different delay requirements on $E[d_i^c]$, Node i creates a new queue to hold the packets for each requirement. An existing intra-node scheduling method proposed in IEEE 802.11e [11] is used to ensure that each queue competes for the channel as a virtual IEEE 802.11 node as shown in Figure 3 (See [11] for detailed explaination). Briefly speaking, in this method, each queue has its own contention window size and backs off according to the channel state similar to an IEEE 802.11 node. The intra-node scheduler transmits the packets from the queue whose backoff timer expires first. DDA then can adapt the virtual node's contention window size to satisfy the delay requirements of flows belonging to this virtual node. Since the adaptation of contention window size of a virtual node is the same as the adaptation of contention window size of a real node carrying one flow, in the reminder of the section, we refer to both virtual and real node as "node".

The key issue in the design of DDA is the contention window adaptation algorithm, which must ensure $E[d_i^c] \leq \Delta_i$ without causing instability in the system. Therefore, in this section, we discuss the design of the contention window adaptation algorithm in DDA and Section VI examines its convergence.

To simplify the discussion of the algorithm design, we first introduce several simple notations. First, we define $v_i \equiv \frac{2}{W_i}$. Then, we denote the set of v_i for all active nodes in the network, which essentially is the allocation of contention window sizes in the network, as $\mathbf{v}$. Finally, to simplify the part in Equation (27) that captures the effect of other nodes' contention window sizes on the delay of Node i, we denote

$$I_i(\mathbf{v}) \equiv \left[1 + \gamma - \prod_{j \in n_i} (1 - G_{j,i} v_j)\right]. \tag{28}$$

Using the above new notations, Equation (27) becomes:

$$E[d_i^c] = H_1 + H_2 \frac{I_i(\mathbf{v})}{v_i}. \tag{29}$$

Denoting N as the set of active nodes that carry delay-sensitive flows, to ensure that $E[d_i^c] \leq \Delta_i$ for any Node $i \in N$, according to Equation (29), v_i must satisfy:

$$\Delta_i \geq H_1 + H_2 \frac{I_i(\mathbf{v})}{v_i}, \forall i \in N. \tag{30}$$

Solving for v_i from Inequality (30),

$$v_i \geq \frac{H_2 I_i(\mathbf{v})}{\Delta_i - H_1}, \forall i \in N. \tag{31}$$

For the rest of nodes, denoted as set $\overline{N}$, there is no requirement on their v_is. Any contention window allocation $\mathbf{v}$ that satisfies Inequality (31) is a feasible contention window allocation, meaning that this contention window allocation can satisfy the delay requirements of all delay-sensitive realtime flows. According to queueing theory, the packet delay at a node is only bounded if the packet departure rate matches the packet arrival rate at the node. Therefore, a contention window allocation that satisfies the delay requirements of realtime flows does not affect the throughput guarantees provided by existing methods.

However, among all the feasible contention window allocations, allocations with smaller contention window sizes introduce more contention collisions between competing flows, which wastes both energy and bandwidth. Therefore, the smallest feasible $\mathbf{v}$, in other words, the feasible contention window allocation that has the largest contention window size, is the most preferable contention window allocation. Hence, the desired convergent point of DDA, $\mathbf{v}^*$, should satisfy:

$$v_i^* = \frac{H_2 I_i(\mathbf{v})}{\Delta_i - H_1}, \forall i \in N. \tag{32}$$

From Equation (29), $I_i(\mathbf{v})$ at time t can be estimated as:

$$I_i(\mathbf{v}(\mathbf{t})) = v_i(t)\frac{\overline{d_i^c}(t) - H_1}{H_2}, \tag{33}$$

where $\overline{d_i^c}(t)$ is the average d_i^c measured at time t. Based on Equations (32) and (33), the contention window adaptation algorithm in DDA is designed as follows:

$$\boxed{\begin{aligned} v_i(n+1) &= \frac{H_2 I_i(\mathbf{v}(n))}{\Delta_i - H_1}, \\ &= v_i(n)\frac{\overline{d_i^c}(n) - H_1}{\Delta_i - H_1}, \forall i \in N. \end{aligned}} \tag{34}$$

Equation (34) essentially means that, at each iteration, the contention window size at Node i is set to be the largest contention window size that can ensure $E[d_i^c] \leq \Delta_i$, assuming that other nodes do not change their contention window sizes. This contention window adaptation algorithm is very easy to implement since $H_1 = DIFS$ is known and $\overline{d_i^c}(n)$ can be measured locally by Node i. Essentially, at each iteration of the algorithm, Node i only needs to compare its current average contention delay $\overline{d_i^c}$ with the contention delay requirement Δ_i based on Equation (34). If $\overline{d_i^c}$ is smaller than Δ_i, Node i increases its contention window size. If $\overline{d_i^c}$ is larger than Δ_i, Node i decreases its contention window size.[1].

However, when Node i adapts its v_i, not only does it change its own delay, it also affects the contention delays of its neighboring nodes that are also in N, which in turn affects their contention window adaptation. Therefore, an important question is whether DDA can converge if there is a valid allocation of contention window sizes that satisfies the per-hop delay requirements at all the nodes in the network. The answer to this question is presented in Section VI.

VI. Convergence of DDA

This section analyzes the convergence property of DDA. The proof consists of three steps. In the first step, we translate DDA into its vector form and prove that this vector form has three properties. These properties are used in the second and third steps to analyze the convergence property of DDA. In the second step, we examine the ideal situation, called the *synchronous case*, where all nodes in N update their contention window size synchronously and their estimation of $I(\mathbf{v})$ is always accurate and up to date. In this ideal case, we prove that if there exists a feasible allocation of contention window sizes that satisfies the delay requirements of every node, DDA always converges to a feasible solution. In the third step, we study the *asynchronous case*, where nodes may update their contention window size asynchronously and their estimation of $I(\mathbf{v})$ may be outdated. We show that DDA still converges to a feasible solution, if a feasible solution exists.

[1]Since the contention window size is under the control of DDA, there is no expontential increase of contention window size after a collision

A. The Vector Form of DDA

To examine whether the network is stable under the control of DDA, we need to consider how the contention window allocation in the whole network changes at each iteration of DDA. Therefore, we need to map Equation (34), which only shows how an individual node adapts its own contention window size, to a vector form, which describes the changes in contention window allocation for all nodes.

To map Equation (34) to a vector form, note that only nodes in set N adapt their contention window sizes. Denoting $\mathbf{u} = \{v_i : \forall i \in N\}$ and $\bar{\mathbf{u}} = \{v_i : \forall i \in \overline{N}\}$, only $\mathbf{u}$ is affected by DDA and $\bar{\mathbf{u}}$ is fixed. To capture the mutual effects between the contention window adaptations of the nodes in N, we define an *interference function*, $\mathbf{L}(\mathbf{u})$, as follows:

$$\mathbf{L}(\mathbf{u}) = \{L_i(\mathbf{u}) | i \in N\}, \tag{35}$$

$$\text{where } L_i(\mathbf{u}) = \frac{H_2 I_i(\mathbf{v})}{\Delta - H_1} = \frac{H_2 I_i(\mathbf{u} \cup \bar{\mathbf{u}})}{\Delta - H_1}. \tag{36}$$

Using $\mathbf{L}(\mathbf{u})$, the vector form of DDA can be expressed as:

$$\mathbf{u(n+1)} = \mathbf{L(u(n))}. \tag{37}$$

From Inequality (31), a $\mathbf{u}$ is a feasible solution if and only if:

$$\mathbf{u} \geq \mathbf{L}(\mathbf{u}). \tag{38}$$

We next present the three properties of $\mathbf{L}(\mathbf{u})$ that we will use to prove the convergence of DDA for both the synchronous case and the asynchronous case.

Theorem 1: For $\forall \mathbf{u} > \mathbf{0}$, $\mathbf{L}(\mathbf{u})$ has the following properties:

1) Positivity: $\mathbf{L}(\mathbf{u})$ is positive.
2) Monotonicity: If two contention window allocations $\mathbf{u}$ and $\hat{\mathbf{u}}$ satisfy $\mathbf{u} \geq \hat{\mathbf{u}}$, then $\mathbf{L}(\mathbf{u}) \geq \mathbf{L}(\hat{\mathbf{u}})$.
3) Scalability: For all $\phi > 1$, $\phi\mathbf{L}(\mathbf{u}) > \mathbf{L}(\phi\mathbf{u})$.

Proof:

1) Since the smallest possible contention delay for a node is DIFS, it is reasonable to assume that Δ is larger than DIFS. Hence, according to Equation (36), $\mathbf{L}(\mathbf{u})$ is positive.
2) If $\mathbf{u} \geq \hat{\mathbf{u}}$, $1 - G_{j,i}v_j \leq 1 - G_{j,i}\hat{v}_j, \forall j \in N$ and $1 - G_{j,i}v_j = 1 - G_{j,i}\hat{v}_j, \forall j \in \overline{N}$. Combining this with Equation (28), $I_i(\mathbf{u} \cup \bar{\mathbf{u}}) \geq I_i(\hat{\mathbf{u}} \cup \hat{\bar{\mathbf{u}}}), \forall i$. Therefore, $\mathbf{L}(\mathbf{u}) \geq \mathbf{L}(\hat{\mathbf{u}})$.
3) Note that for $\phi > 1$, $\phi\mathbf{L}(\mathbf{u}) > \mathbf{L}(\phi\mathbf{u})$ holds if and only if $\phi I_i(\mathbf{u} \cup \bar{\mathbf{u}}) - I_i(\phi\mathbf{u} \cup \bar{\mathbf{u}}) > 0, \forall i$. Based on Equation (28),

$$\begin{aligned} &\phi I_i(\mathbf{u} \cup \bar{\mathbf{u}}) - I_i(\phi\mathbf{u} \cup \bar{\mathbf{u}}) \\ &= (\phi - 1)(1+\gamma) - \textstyle\prod_{j \in n_i \cap \overline{N}}(1 - G_{j,i}v_j) \\ &\times \Big[\phi \textstyle\prod_{j \in n_i \cap N}(1 - G_{j,i}v_j) - \prod_{j \in n_i \cap N}(1 - G_{j,i}\phi v_j)\Big]. \end{aligned} \tag{39}$$

To demonstrate that $\phi I_i(\mathbf{u} \cup \bar{\mathbf{u}}) - I_i(\phi\mathbf{u} \cup \bar{\mathbf{u}}) > 0$ from Equation (39), note that Lemma 3 in Appendix A shows:

$$\phi \prod_{j=1}^{m}(1 - G_{j,i}v_j) - \prod_{j=1}^{m}(1 - G_{j,i}\phi v_j) \leq \phi - 1, \forall m > 0. \tag{40}$$

Combining Equation (40) with Equation (39) and based on the fact that $(1 - G_{j,i}v_j) \leq 1$,

$$\phi I_i(\mathbf{u} \cup \bar{\mathbf{u}}) - I_i(\phi\mathbf{u} \cup \bar{\mathbf{u}}) \geq (\phi - 1)(1 + \gamma) - (\phi - 1) > 0. \tag{41}$$

Therefore, $\forall \phi > 1$, $\phi\mathbf{L}(\mathbf{u}) > \mathbf{L}(\phi\mathbf{u})$. ■

These three properties of $\mathbf{L}(\mathbf{u})$ in Theorem 1 are used in Sections VI-B and VI-C to demonstrate DDA's convergence property.

B. Convergence in the Synchronous Case

In this section, we analyze the convergence property of DDA in the synchronous case based on the properties of $\mathbf{L}(\mathbf{u})$ from Theorem 1. In Section VI-C, the asynchronous case is analyzed. [2] To prove that DDA converges in the synchronous case, Theorem 2 shows that the fixed point of DDA is unique. Then, Theorem 3 shows that DDA converges to this unique fixed point starting from any initial contention window allocation, if there exists at least one feasible contention window allocation.

Theorem 2: If DDA in Equation (34) has a fixed point, that fixed point is unique.

Proof: If $\mathbf{u}^*$ and $\hat{\mathbf{u}}^*$ are distinct fixed points, $v_j^* = L_j(\mathbf{u}^*)$ and $\hat{v}_j^* = L_j(\hat{\mathbf{u}}^*)$ for $\forall v_j^* \in \mathbf{u}^*$ and $\hat{v}_j^* \in \hat{\mathbf{u}}^*$. Without loss of generality, assume that there exists j such that $v_j^* < \hat{v}_j^*$. Hence, there exists $\phi > 1$ such that $\phi\mathbf{u}^* \geq \hat{\mathbf{u}}^*$ and for some j, $\phi v_j^* = \hat{v}_j^*$. From the monotonicity and scalability properties of $\mathbf{L}(\mathbf{u})$,

$$\hat{v}_j^* = L_j(\hat{\mathbf{u}}^*) \leq L_j(\phi\mathbf{u}^*) < \phi L_j(\mathbf{u}^*) = \phi v_j^*. \tag{42}$$

Since $\hat{v}_j^* = \phi v_j^*$, we have found a contradiction, implying that the fixed point must be unique. ■

Theorem 2 shows that DDA has at most one fixed point. However, we still need to show that this unique fixed point $\mathbf{u}^*$ exists and that DDA converges to this fixed point asymptotically, which are demonstrated in Theorem 3. The proof of Theorem 3 works as follows. Starting from an initial contention window allocation $\mathbf{u}$, n iterations of DDA produce a sequence of contention window allocations $\{\mathbf{L}^1(\mathbf{u}), \mathbf{L}^2(\mathbf{u}) \cdots \mathbf{L}^n(\mathbf{u})\}$. In Lemma 1 and Lemma 2, we identify two special sequences of contention window allocations that eventually converge to $\mathbf{u}^*$. By showing that any sequence of contention window allocations generated by DDA can be bounded by these two special sequences, we prove the convergence of DDA starting from any initial allocation $\mathbf{u}$.

Lemma 1: If $\mathbf{u}$ is a feasible contention window allocation, then $\mathbf{L}^n(\mathbf{u})$ is a monotonically non-increasing sequence of feasible contention window allocations that converges to a unique fixed point $\mathbf{u}^*$.

Proof: This Lemma can be proved by induction. First, let $\mathbf{u}(0) = \mathbf{u}$ and $\mathbf{u}(n) = \mathbf{L}^n(\mathbf{u})$. Since $\mathbf{u}$ is a feasible contention window allocation, according to Inequality (38), $\mathbf{u}(0) \geq \mathbf{L}(\mathbf{u}(0)) = \mathbf{u}(1)$. Second, suppose $\mathbf{u}(n-1) \geq \mathbf{u}(n)$. Then, based on the Monotonicity of $\mathbf{L}(\mathbf{u})$, $\mathbf{L}(\mathbf{u}(n-1)) \geq \mathbf{L}(\mathbf{u}(n))$. Therefore, based on Equation (37), $\mathbf{u}(n) \geq \mathbf{u}(n+1)$. Hence, $\mathbf{u}(n)$ is a non-increasing sequence of contention window allocations. Since the sequence $\mathbf{u}(n)$ is bounded below by zero, $\mathbf{u}(n)$ must converge to a fixed point $\mathbf{u}^*$. Based on Theorem 2, $\mathbf{u}^*$ is unique. Therefore, $\mathbf{u}(n)$ must converge to a unique fixed point $\mathbf{u}^*$ ■

Lemma 2: If there exists a feasible contention window allocation, then starting from $\mathbf{z}$, the all zero vector, DDA produces a monotonically non-decreasing sequence of contention window allocations $\mathbf{L}^n(\mathbf{z})$ that converges to the unique fixed point $\mathbf{u}^*$.

Proof: This lemma can be proved by induction. Let $\mathbf{z}(n) = \mathbf{L}^n(\mathbf{z})$ and $\mathbf{z}(0) = \mathbf{z}$. Based on Lemma 1, the existence of a feasible solution implies the existence of a unique fixed point $\mathbf{u}^*$. Obviously, $\mathbf{z}(0) < \mathbf{u}^*$. Suppose for some $n \geq 0$, $\mathbf{z}(n) \leq \mathbf{u}^*$. According to the monotonicity of $\mathbf{L}(\mathbf{u})$ in Theorem 1,

$$\mathbf{z}(n+1) = \mathbf{L}(\mathbf{z}(n)) \leq \mathbf{L}(\mathbf{u}^*) = \mathbf{u}^*. \tag{43}$$

Therefore, the sequence $\mathbf{z}(n)$ is bounded by $\mathbf{u}^*$ from above. In addition, note that $\mathbf{z}(1) = \mathbf{L}(\mathbf{z}) \geq \mathbf{z}$. Suppose $\mathbf{z} \leq \mathbf{z}(1) \leq \cdots \leq \mathbf{z}(n)$. The monotonicity of $\mathbf{L}(\mathbf{u})$ implies:

$$\mathbf{z}(n+1) = \mathbf{L}(\mathbf{z}(n)) \geq \mathbf{L}(\mathbf{z}(n-1)) = \mathbf{z}(n). \tag{44}$$

Hence, $\mathbf{z}(n)$ is a non-decreasing sequence and bounded by $\mathbf{u}^*$ from above. Combined with the uniqueness of DDA's fixed point, $\mathbf{z}(n)$ must converge to the unique fixed point $\mathbf{u}^*$. ■

Theorem 3: If there exists a feasible contention window allocation, then for any initial contention window allocation $\mathbf{u}$, DDA converges to a unique fixed point $\mathbf{u}^*$.

Proof: According to Lemma 1, the existence of a feasible contention window allocation implies the existence of a unique fixed point $\mathbf{u}^*$. Since $v_j^* > 0$ for $\forall v_j^* \in \mathbf{u}^*$, for any initial $\mathbf{u}$, we can find $\phi \geq 1$ such that $\phi\mathbf{u}^* \geq \mathbf{u}$. By the scalability property, $\phi\mathbf{u}^* = \phi\mathbf{L}(\mathbf{u}^*) \geq \mathbf{L}(\phi\mathbf{u}^*)$. Therefore, $\phi\mathbf{u}^*$ must be feasible. Since $\mathbf{z} \leq \mathbf{u} \leq \phi\mathbf{u}^*$, the monotonicity property implies:

$$\mathbf{L}^n(\mathbf{z}) \leq \mathbf{L}^n(\mathbf{u}) \leq \mathbf{L}^n(\phi\mathbf{u}^*). \tag{45}$$

Since Lemma 1 and Lemma 2 imply that $\lim_{n\to\infty}\mathbf{L}^n(\phi\mathbf{u}^*) = \lim_{n\to\infty}\mathbf{L}^n(\mathbf{z}) = \mathbf{u}^*$, combined with Equation (45), we get $\lim_{n\to\infty}\mathbf{L}^n(\mathbf{u}) = \mathbf{u}^*$. Therefore, DDA converges to $\mathbf{u}^*$. ■

We have shown that for any initial contention window allocation $\mathbf{u}$, DDA converges to a unique fixed point $\mathbf{u}^*$ whenever a feasible contention window allocation exists. It is worth noting that Lemma 1 and Theorem 3 confirm our initial intuition about DDA. Recall that in DDA, each node in N chooses the largest feasible contention window size at each iteration with the assumption that the other nodes do not change their contention window sizes. We expect that by doing so, DDA's convergent point will be the feasible contention

[2]Similar proofs for both the synchronous and asynchronous cases can be found in prior works that study power control schemes in CDMA systems [19].

window allocation that has the largest contention window size. This is essentially confirmed by Lemma 1 and Theorem 3, since Lemma 1 implies that for any feasible contention window allocation $\mathbf{u}$, $\mathbf{u} \geq \mathbf{u}^*$ and Theorem 3 shows that DDA always converges to $\mathbf{u}^*$.

C. Convergence in the Asynchronous Case

Although we have proved that DDA converges to a unique fixed point whenever a feasible solution exists, the proof assumes that the iterations of DDA are run synchronously at each node in N, which is hard to achieve in ad hoc networks. Therefore, it is important to investigate whether DDA converges under asynchronous conditions. By asynchronous, we mean that some nodes may perform DDA iterations faster and execute more iterations than others and some nodes may perform the iterations using outdated information. In this section, we show that even under the asynchronous case, DDA still converges to the unique fixed point $\mathbf{u}^*$ whenever a feasible solution exists. The proof for DDA's convergence in the asynchronous case is composed of two steps. First, we translate DDA in Equation (34) to its asynchronous counterpart. Then we show that this asynchronous version of DDA still converges.

To translate DDA to its asynchronous version, let $v_i(t)$ be the value of v_i at time t, so that $\mathbf{u}(t) = \{v_j(t), \forall j \in N\}$. Note that Node i's estimation of the interference function may be outdated since Node i may have some delay in estimating $\overline{d_i^c}$. Additionally, due to the randomness of channel access, Node i's neighbors' adjustments of contention window sizes may not affect Node i's delay immediately. Therefore, we assume that when Node i adjusts its v_i at time t, its adjustment is performed based on the effect of an outdated contention window allocation:

$$\mathbf{u}(\tau^i(t)) = \{v_j(\tau_j^i(t)), \forall j \in N\}, \tag{46}$$

where $0 \leq \tau_j^i(t) \leq t$ and $\tau^i(t) = \{\tau_j^i(t), \forall j \in N\}$. Since nodes in the network may update their contention window sizes at different times, we denote the set of times at which Node i updates its contention window size as T^i. Given the sets $\{T^i, i \in N\}$, the asynchronous version of DDA can be expressed as:

$$v_i(t+1) = \begin{cases} L_i(\mathbf{u}(\tau^i(t))) & t \in T^i \\ v_i(t) & otherwise. \end{cases} \quad \forall i \in N \tag{47}$$

Next, we show that the asynchronous version of DDA in Equation (47) still converges using the Asynchronous Convergence Theorem in [20], which is repeated in Theorem 4.

Theorem 4: Given an asynchronous iterative algorithm

$$x_i(t+1) = \begin{cases} f_i(x_1(\tau_1^i(t)), \cdots, x_n(\tau_n^i(t))) & \forall t \in T^i, \\ x_i(t) & otherwise, \end{cases} \tag{48}$$

if there is a sequence of nonempty sets $\{\mathbf{X}(k)\}$ with

$$\cdots \subset \mathbf{X}(k+1) \subset \mathbf{X}(k) \subset \cdots \subset \mathbf{X}(0) \tag{49}$$

satisfying the following two conditions and the initial solution estimate $x(0)$ belongs to the set $\mathbf{X}(0)$, then every limit point of $x(t) = \{x_1(t), x_2(t) \cdots x_n(t)\}$ is a fixed point for $f = \{f_1, f_2, \cdots, f_n\}$. The two conditions are:

1) (*Synchronous Convergence Condition*)

$$f(x) \in \mathbf{X}(k+1), \forall k \text{ and } x \in \mathbf{X}(k). \tag{50}$$

Furthermore, if $\{y(k)\}$ is a sequence such that $y(k) \in \mathbf{X}(k)$ for every k, then every limit point of $\{y(k)\}$ is a fixed point of f.

2) (*Box Condition*) For every k, there exist sets $\mathbf{X}_i(k) \subset \mathbf{X}_i(0)$ such that $\mathbf{X}(k) = \mathbf{X}_1(k) \times \mathbf{X}_2(k) \times \cdots \times \mathbf{X}_n(k)$.

Using Theorem 4, we can prove the convergence of DDA as shown below in Theorem 5.

Theorem 5: If there is a feasible solution, then from any initial contention window allocation $\mathbf{u}(0)$, DDA converges to a unique fixed point $\mathbf{u}^*$.

Proof: Note that Theorems 2 and 3 show that if there is a feasible solution, DDA has a unique fixed point $\mathbf{u}^*$. Denoting the all zero vector as $\mathbf{z}$ and choosing a large enough $\phi > 1$, we can have $\phi\mathbf{u}^* \geq \mathbf{u}(0) \geq \mathbf{z}$. Given

$$\mathbf{X}(k) = \{\mathbf{u} | \mathbf{L}^k(\mathbf{z}) \leq \mathbf{u} \leq \mathbf{L}^k(\phi\mathbf{u}^*)\}, \tag{51}$$

it is easy to see that for all $k \geq 0$, $\mathbf{X}(k) = \mathbf{X}_1(k) \times \mathbf{X}_2(k) \times \cdots \times \mathbf{X}_n(k)$, where $\mathbf{X}_i(k) = \{v_i | L_i^k(\mathbf{z}) \leq v_i \leq L_i^k(\phi\mathbf{u}^*)\}$. Therefore, $\mathbf{X}(k)$ satisfies the box condition.

Lemma 1 and Lemma 2 imply that $\mathbf{X}(k+1) \subset \mathbf{X}(k), \forall k \geq 0$ and, for $\forall k \geq 0$ and $\forall \mathbf{u} \in \mathbf{X}(k)$, $\mathbf{L}(\mathbf{u}) \in \mathbf{X}(k+1)$. In addition, since $\lim_{k\to\infty} \mathbf{L}^k(\phi\mathbf{u}^*) = \lim_{k\to\infty} \mathbf{L}^k(\mathbf{z}) = \mathbf{u}^*$, any sequence $\{\hat{\mathbf{u}}(k)\}$ such that $\hat{\mathbf{u}}(k) \in \mathbf{X}(k)$ for all k must converge to $\mathbf{u}^*$. Note that $\mathbf{u}$, $\hat{\mathbf{u}}$ and $\mathbf{L}(\mathbf{u})$ correspond to x, y and $f(x)$ in the statement of Theorem 4 respectively. Therefore, $\mathbf{X}(n)$ satisfies the synchronous convergence condition.

Since the initial contention window allocation $\mathbf{u}(0)$ satisfies $\mathbf{u}(0) \in \mathbf{X}(0)$, Theorem 4 implies that DDA converges to $\mathbf{u}^*$. ■

In summary, we have shown that if there exists at least one feasible contention window allocation, DDA converges to a unique fixed point starting from any initial contention window allocation, regardless whether it is run synchronously or asynchronously. This fixed point is the feasible contention window allocation that has the largest contention window sizes.

VII. Divergent Case and Its Solution

In the proof of convergence of DDA, it is assumed that there exists at least one feasible contention window allocation. However, it is possible that there is no feasible contention window allocation. This may happen frequently in ad hoc networks, since active delay-sensitive flows may move into each other's carrier-sensing range, causing the total requirements of the realtime flows to exceed the capacity of the network. In such situations, DDA may diverge, which is natural, since when the total requirements of the realtime flows exceed the network capacity, no scheduling algorithm can ever provide QoS guarantees to all of the flows.

The divergence of DDA provides a valuable indication that signals the conflicts of service requirements between competing flows. By monitoring the divergence of DDA, we

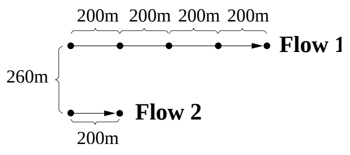


Fig. 4. Simulation topology

can use a reactive admission control scheme similar to the scheme in [21] to resolve the conflicts as follows. If the total requirements of the realtime flows exceed the network capacity, DDA diverges and the contention window sizes at competing nodes reduce repeatedly without improving the delay of the nodes. Therefore, in DDA, each node maintains a lower bound on its contention window size. When one node's contention window size is reduced to below the lower bound, this node assumes that there is not enough resources for its flow so that it stops sending its flow and informs the source of its flow to either end the session or reroute the flow. DDA avoids rejecting existing flows by decreasing their lower bounds on contention window size as they age. In this way, newly arrived flows always have the highest lower bounds and hence are always rejected first if there is not enough resources for all of the flows. When enough flows in the network are rejected so that there is enough resources for the remaining flows, DDA again will converge to a contention window allocation that ensures the delay of the remaining flows.

VIII. Evaluation

The goal of our evaluation is to demonstrate the performance of DDA in terms of its ability to provide delay guarantees to delay-sensitive flows while at the same time achieving high network utilization. Our evaluation is conducted in both simple topologies with two flows (Sections VIII-A and VIII-B) and randomly generated large topologies with 20 flows (Section VIII-C). We compare the performance of DDA with SWAN [7] and IEEE 802.11 using the NS2 simulator [22]. We choose SWAN since it also aims to provide delay guarantees with low overhead by using admission control based on delay measurements and adapting the sending rate of best effort traffic. We also compare DDA with IEEE 802.11 as a baseline for comparison. The NS2 implementation of SWAN is the latest distribution by the SWAN project. The routing protocol used in the simulations is DSR [23]. The channel bandwidth is 11Mbps. The transmission range is 250m and the carrier-sensing range is 550m.

A. End-to-End Delay Guarantees

To illustrate DDA's ability to provide end-to-end delay guarantees, we simulate two flows competing with each other in a simple topology as shown in Figure 4. Flow 1 and Flow 2 are both delay sensitive realtime flows with an end-to-end delay requirement of 30ms. Flow 1 sends 10 packets/second and Flow 2 sends 190 packets/second. Although the throughputs of both flows match their throughput requirements (due to the space limitations, the throughput of flows are not presented), the end-to-end delay of the two flows are quite different under IEEE 802.11, SWAN and DDA as shown in Figure 5, where the solid line represents the end-to-end delay requirement. Since Flow 1 has a larger hop count, Flow 1's end-to-end delay is much larger than the end-to-end delay requirement under IEEE 802.11 and SWAN due to their lack of per-hop delay management. DDA, however, can translate the end-to-end delay requirement into per-hop delay requirements and manage the delay at each hop. Therefore, the average end-to-end delay of both flows can be managed below their end-to-end delay requirement.

B. Delay Guarantees in the presence of Best Effort Traffic

To illustrate DDA's ability to provide guarantees of end-to-end delay in the presence of best effort traffic, we run a similar simulation as in Section VIII-A except that Flow 2 is set to be a best effort TCP flow. Since Flow 2, as a best effort flow, always competes for channel bandwidth and IEEE 802.11 has no QoS-aware management, Flow 1's end-to-end delay can be as high as several seconds as shown in Figure 6(a). Both SWAN and DDA, however, maintain the end-to-end delay of Flow 1 below its requirement. SWAN achieves this by adapting the transmission rate of Flow 2, while DDA achieves this through adapting the contention window size of Flow 1. However, compared to IEEE 802.11, SWAN greatly increases the delay of Flow 2 and reduces its throughput significantly as shown in Figure 6. (Note the scales for end-to-end delay are different for Figures 6(a), (b) and (c).) DDA, however, only increases the delay and reduces the throughput of Flow 2 slightly, demonstrating DDA's excellent ability to maintain high network utilization.

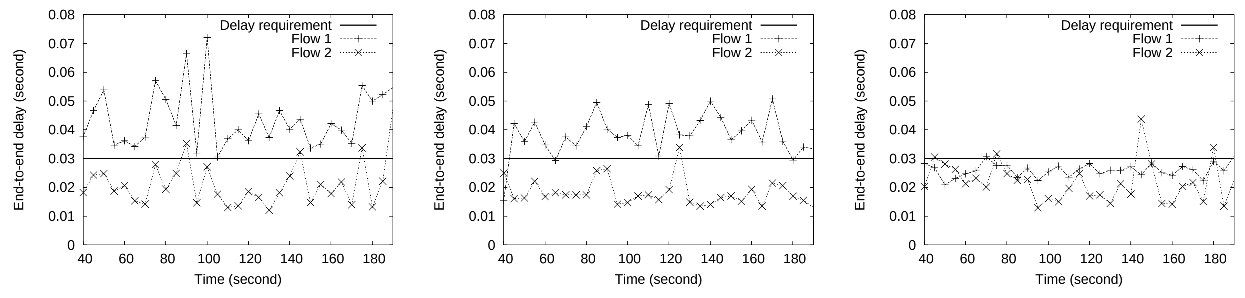
C. Delay Guarantees in Large Networks

In this section, we evaluate DDA by varying the rate of the delay-sensitive flows and the type of its competing flows in large randomly generated topologies. To measure the performance of DDA, we use two metrics. The first metric is delay violation, which is 0 if the average end-to-end delay of a flow is below its delay requirement and equals

$$\frac{\text{(average end-to-end delay) - (delay requirement)}}{\text{(delay requirement)}},$$

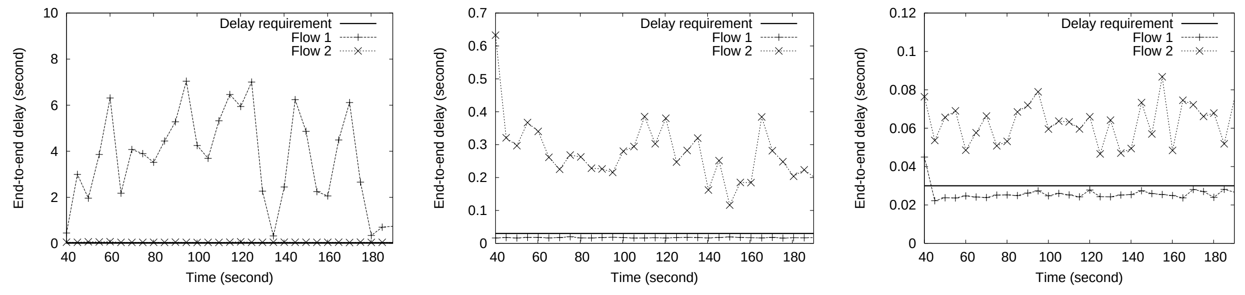
if the average end-to-end delay of a flow is larger than its delay requirement. The second metric is the total network throughput, which essentially examines whether a protocol affects the capacity of the network.

The simulations are run in 84 randomly generated 1000m$\times$1000m networks with 80 nodes. In the first set of simulations, 10 delay-sensitive flows with 50ms delay requirements are competing with 10 best effort TCP flows. Figures 7 and 8 show the average delay violations and total network throughput as the rates of the delay-sensitive flows range from 10 to 100 packets/second. Both SWAN and IEEE 802.11 show significant amounts of delay violations, while in DDA, delay violations are always 0, demonstrating DDA's excellent ability to keep delay guarantees in the presence of best effort flows. In addition, DDA has a higher total network throughput than

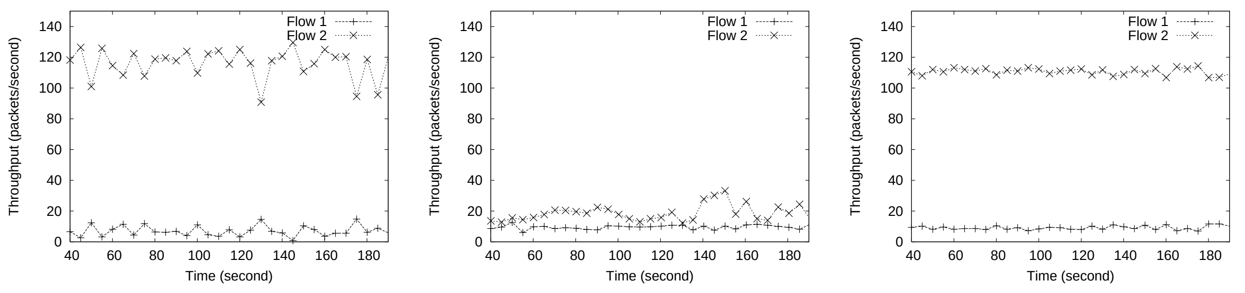


(a) IEEE 802.11 (b) SWAN (c) DDA

Fig. 5. End-to-end delay when both flows are delay-sensitive flows



(a) Delay in IEEE 802.11 (b) Delay in SWAN (c) Delay in DDA



(d) Throughput in IEEE 802.11 (e) Throughput in SWAN (f) Throughput in DDA

Fig. 6. Throughput and end-to-end delay when Flow 2 is a TCP flow

SWAN since SWAN unnecessarily reduces the throughput of best effort traffic and hurts the capacity of the network.

To ensure that DDA can provide delay guarantees in the presence of throughput-sensitive flows, in the second set of simulations, we change the competing flows from TCP flows to CBR flows and set the rates of the CBR flows to be the same as the delay-sensitive realtime flows. As shown in Figure 9, the delay violations of DDA are 0 while IEEE 802.11 and SWAN show significant delay violations as the load on the network increases. As shown in Figure 10, the total network throughput of DDA in this case is also comparable to both IEEE 802.11 and SWAN, showing that DDA does not hurt the capacity of network when competing with throughput-sensitive flows.

IX. Conclusion

In response to the limitations of current protocols for supporting delay-sensitive realtime flows, in this paper, we introduce a new protocol DDA, which provides delay guarantees to delay-sensitive realtime flows. Based on intensive analysis of the distribution of packet delay, we show that by allocating contention window sizes, delay guarantees can be provided. Based on this observation, we design DDA, which iteratively adapts the contention window sizes of nodes that carry delay-sensitive traffic and converges to a contention window allocation that ensures every delay-sensitive flow's delay requirement. DDA is simple, lightweight and does not impose any communication overhead. Every node running DDA only needs local measurable information. Simulations compare the

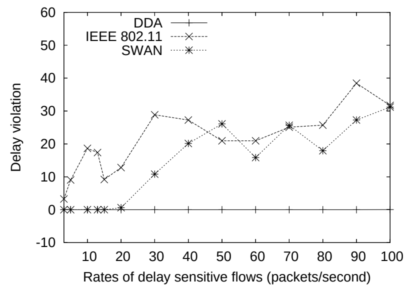


Fig. 7. Delay violation for delay-sensitive flows while they compete with TCP flows

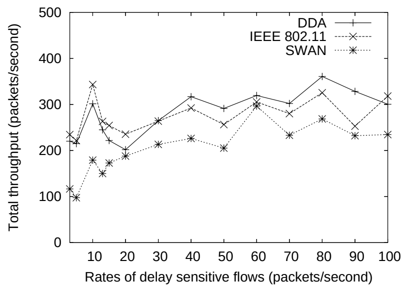


Fig. 8. Total network throughput when delay-sensitive flows compete with TCP flows

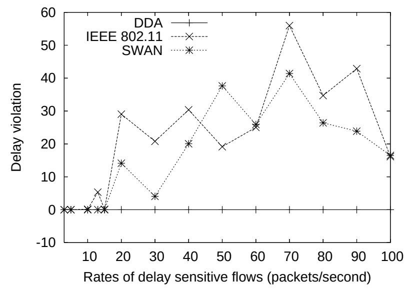


Fig. 9. Delay violation for delay-sensitive flows when they compete with CBR flows

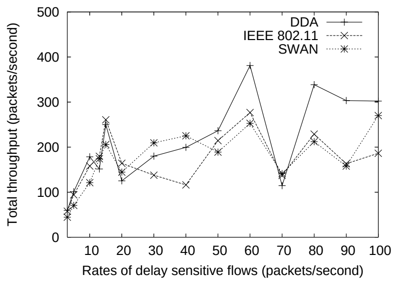


Fig. 10. Total network throughput when delay-sensitive flows compete with CBR flows

performance of DDA with SWAN and IEEE 802.11 and demonstrate DDA'a ability to provide delay guarantees.

In the future, we plan to extend the contention window adaptation algorithm of DDA so that instead of relying on existing protocols for throughput guarantees, DDA can provide both throughput and delay guarantees. We will also examine other methods for breaking end-to-end delay requirements to per-hop delay requirements.

References

[1] IEEE Computer Society, "802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," .

[2] T.-W. Chen, J. T. Tsai, and M. Gerla, "QoS Routing Performance in Multihop Multimedia Wireless Networks," in *IEEE International Conference on Universal Personal Communications (ICUPC)*, 1997.

[3] C.R. Lin and Chung-Ching Liu, "An On-demand QoS Routing Protocol for Mobile Ad Hoc Networks," in *IEEE GLOBECOM*, 2000.

[4] D. Maltz, "Resource Management in Multi-hop Ad Hoc Networks," in *Technical Report CMU CS 00-150, School of Computer Science, Carnegie Mellon University*, July 2000.

[5] Yaling Yang and Robin Kravets, "Contention-Aware Admission Control for Ad Hoc Networks," *IEEE Transactions on Mobile Computing*, vol. 4, no. 4, pp. 363–377, 2005.

[6] Michael G. Barry, Andrew T. Campbell, and Andras Veres, "Distributed Control Algorithms for Service Differentiation in Wireless Packet Networks," in *IEEE INFOCOM*, 2001.

[7] Gahng-Seop Ahn, Andrew Campbell, Andras Veres, and Li-Hsiang Sun, "SWAN: Service Differentiation in Stateless Wireless Ad Hoc Networks," in *IEEE INFOCOM*, 2002.

[8] Yaling Yang and Robin Kravets, "Throughput Guarantees for Multi-priority Traffic in Ad Hoc Networks," in *International Conference on Mobile Ad-hoc and Sensor Systems (MASS)*, October 2004.

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

[10] Haiyun Luo, Songwu Lu, Vaduvur Bharghavan, Jerry Cheng, and Gary Zhong, "A Packet Scheduling Approach to QoS support in Multihop Wireless Networks," *ACM Journal of Mobile Networks and Applications(MONET), Special Issue on QoS in Heterogeneous Wireless Networks*, 2002.

[11] Stefan Mangold, Sunghyun Choi, Peter May, Ole Klein, Guido Hiertz, and Lothar Stibor, "IEEE 802.11e Wireless LAN for Quality of Service," in *Proceedings of European Wireless*, 2002.

[12] Yuan Xue, Kai Chen, and Klara Nahrstedt, "Proportional Delay Differentiation in Wireless LAN," in *IEEE International Conference of Communications (ICC)*, 2004.

[13] Nitin Vaidya, Paramvir Bahl, and Seema Gupta, "Distributed Fair Scheduling in a Wireless LAN," in *ACM Mobicom*, 2000.

[14] Imad Aad and Claude Castelluccia, "Differentiation Mechanisms for IEEE 802.11," in *IEEE INFOCOM*, 2001.

[15] Giuseppe Bianchi, "Performance analysis of the IEEE 802.11 distributed coordination function," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 3, 2000.

[16] Bo Li and Roberto Battiti, "Performance Analysis of An Enhanced IEEE 802.11 Distributed Coordination Function Supporting Service Differentiation," in *International Workshop on Quality of Future Internet Service*, 2003.

[17] Gion Reto Cantieni, Qiang Ni, Chadi Barakat, and Thierry Turletti, "Performance Analysis under Finite Load and Improvements for Multirate 802.11b," *Elsevier Computer Communications Journal*, April, 2005.

[18] Leonard Kleinrock, *Queueing Systems*, Wiley-Interscience, 1975.

[19] Roy D. Yates, "A Framework for Uplink Power Control in Cellular Radio Systems," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 7, pp. 1341–1348, Sept 1995.

[20] Dimitri P. Bertsekas and John N. Tsitsiklis, *Parallel and Distributed Computation*, Athena Scientific, 1997.

[21] Yaling Yang and Robin Kravets, "Distributed QoS Guarantees for Realtime Traffic in Ad Hoc Networks," in *IEEE International Conference on Sensor and Ad Hoc Communications and Networks (SECON)*, 2004.

[22] Kevin Fall and Kannan Varadhan, "NS notes and documentation," in *The VINT Project, UC Berkely, LBL, USC/ISI, and Xerox PARC*, 1997.

[23] David B Johnson and David A Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," in *Mobile Computing*. 1996, vol. 353, Kluwer Academic Publishers.

Appendix

A. Lemma 3

Lemma 3: For any $m \geq 1$, $\phi \prod_{j=1}^{m}(1-G_{i,j}v_j) - \prod_{j=1}^{m}(1-G_{i,j}\phi v_j) \leq \phi - 1$.

Proof: This lemma can be proved using induction. Note that for $m = 1$, $\phi(1 - G_{i,1}v_1) - (1 - G_{i,1}\phi v_1) = \phi - 1$. Assume that for $m = k, k \geq 1$,

$$\phi \prod_{j=1}^{k}(1 - G_{i,j}v_j) - \prod_{j=1}^{k}(1 - G_{i,j}\phi v_j) \leq \phi - 1. \quad (52)$$

Then for $m = k + 1$,

$$\begin{aligned}
&\phi \prod_{j=1}^{k+1}(1 - G_{i,j}v_j) - \prod_{j=1}^{k+1}(1 - G_{i,j}\phi v_j) \\
&\quad = \phi \prod_{j=1}^{k}(1 - G_{i,j}v_j)(1 - G_{i,k+1}v_{k+1}) \\
&\qquad\quad - \prod_{j=1}^{k}(1 - G_{i,j}\phi v_j)(1 - \phi G_{i,k+1}v_{k+1}) \\
&\quad = [\phi \prod_{j=1}^{k}(1 - G_{i,j}v_j) - \prod_{j=1}^{k}(1 - G_{i,j}\phi v_j)] \\
&\qquad\quad + \phi G_{i,k+1}v_{k+1}[\prod_{j=1}^{k}(1 - G_{i,j}\phi v_j) - \prod_{j=1}^{k}(1 - G_{i,j}v_j)].
\end{aligned}$$

Since $\phi > 1$, $\prod_{j=1}^{k}(1 - G_{i,j}\phi v_j) - \prod_{j=1}^{k}(1 - G_{i,j}v_j) < 0$. Therefore, Based on Equation (52),

$$\begin{aligned}
&\phi \prod_{j=1}^{k+1}(1 - G_{i,j}v_j) - \prod_{j=1}^{k+1}(1 - G_{i,j}\phi v_j) \\
&\qquad < [\phi \prod_{j=1}^{k}(1 - G_{i,j}v_j) - \prod_{j=1}^{k}(1 - G_{i,j}\phi v_j)] \leq \phi - 1.
\end{aligned}$$

Hence, Lemma 3 is true. ■

B. Notation

1) ϵ: duration of a backoff slot
2) A_i: packet inter-arrival time at Node i
3) X_i: MAC layer's packet service time at Node i
4) d_i: per hop packet delay at Node i
5) d_i^q: queueing delay at Node i
6) d_i^t: transmission delay at Node i
7) d_i^c: contention delay at Node i
8) d_i^f: idle delay at Node i
9) d_i^b: busy delay at Node i
10) λ_i: packet arrival rate at Node i
11) P_I: probability of no transmission attempt in a backoff slot
12) P_c: a collision happens in a backoff slot
13) P_s: a successful transmission happens in a backoff slot
14) W_i: contention window size of Node i. $W_i > 1$.
15) w_i: number of backoff slots
16) m_i: number of data packets transmitted during Node i's backoff process
17) m_i^c:number of collision periods during Node i's backoff process
18) T_d: average duration of a data transmission period at the channel. It includes the whole duration of a RTS-CTS-DATA-ACK or DATA-ACK handshake.
19) T_c: average duration of a collision period.
20) τ_i: probability of Node i transmits in a backoff slot. $\tau_i = 2/W_i$
21) π_i: probability of Node i successfully transmit in a backoff slot
22) p_i^b: probability that an arrival packet at Node i sees a busy channel
23) n_i: set of Node i's carrier-sensing range neighbors
24) q_i: probability that some node transmits in a backoff slot of Node i. It is also the collision probability of Node i's transmission attempts.
25) $q_{j,i}$: probability that Node j transmits in a backoff slot of Node i.
26) $\alpha_{j,i}$, $\beta_{j,i}$: discount factors due to concurrent transmissions among neighboring nodes
27) Δ_i : requirement on the average contention delay at Node i
28) H_1: $DIFS$
29) H_2: $1.1788(T_d + \epsilon)p_i^b$
30) γ: $\frac{\epsilon}{1.1788(T_d+\epsilon)}$
31) $G_{j,i}$: $\beta_{j,i}\lambda_j T_d$
32) v_i: $2/W_i$
33) $\mathbf{v}$: $\{v_j |$ Node j is an active node in the network$\}$.
34) N: the set of nodes carrying delay-sensitive flows
35) $\overline{N}$: the set of nodes not in N.
36) $\mathbf{u}$: $\{v_j : j \in N\}$.
37) $\bar{\mathbf{u}}$: $\{v_i : i \in \overline{N}\}$