Computer Networks 56 (2012) 2934–2948

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Backpressure scheduling in IEEE 802.11 wireless mesh networks: Gap between theory and practice

Jae-Yong Yoo [a], Cigdem Sengul [b], Ruben Merz [b], JongWon Kim [a,*]

[a] Gwangju Institute of Science and Technology, 1 Oryongdong, Bukgu, Gwangju, Republic of Korea
[b] Telekom Innovation Laboratories, Ernst-Reuter-Platz 7, D-10587 Berlin, Germany

ARTICLE INFO

Article history:
Received 7 September 2011
Received in revised form 15 March 2012
Accepted 17 May 2012
Available online 26 May 2012

Keywords:
Wireless mesh networks
IEEE 802.11
Backpressure scheduling
Wireless network testbeds
Experimental performance evaluation
Theoretical optimal throughput

ABSTRACT

Achieving efficient bandwidth utilization in wireless networks requires solving two important problems: (1) which packets to send (i.e., packet scheduling) and (2) which links to concurrently activate (i.e., link scheduling). To address these scheduling problems, many algorithms have been proposed and their throughput optimality and stability are proven in theory. One of the most well-known scheduling algorithms is backpressure scheduling which performs both link and packet scheduling assuming a TDMA (Time Division Multiple Access) MAC (Medium Access Control) layer. However, there has been limited work on realizing backpressure scheduling with a CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) MAC layer (e.g., IEEE 802.11). In IEEE 802.11 networks, it is expected that the throughput optimality will not be achieved. In this paper, we investigate the extent of this throughput gap between theoretical TDMA-based backpressure scheduling and an approximation of it for IEEE 802.11 WMNs (Wireless Mesh Networks). Through extensive testbed measurements, we verify that there is indeed a non-negligible throughput gap. We present two main reasons behind this gap: *Control inaccuracy* that results from approximation of link scheduling and *information inaccuracy* that results from late or incorrect information, for instance, about queue lengths or network topology. Our results show that losses by MAC-layer collisions and backoff, which mainly occur due to control inaccuracy plays a major role for the throughput gap. On the other hand, while losses by queue drops, typically due to information inaccuracy, do occur, their effect can be tolerated. Nevertheless, both types of inaccuracies need to be mitigated in order to improve throughput.

© 2012 Elsevier B.V. All rights reserved.

1. Introduction

WMNs (Wireless Mesh Networks) promise great potential to improve Internet coverage, especially in underprivileged regions of the world, due to their deployment flexibility and relatively low cost [1]. However, despite their benefits, WMNs require solving several networking challenges to operate at full capacity [2]. For one of the challenges, improving bandwidth utilization, there have been numerous scheduling proposals [3–12]. One of the most well-known algorithms is *backpressure scheduling* [3], which computes a backpressure value as the queue differential of a node and its next-hop node for every flow, and uses these values to determine the central scheduling sequence. Theoretically, it has been shown that backpressure scheduling can stabilize the network per-flow queues and hence, provides optimal throughput [4]. However, these results are obtained under the assumption of a TDMA (Time Division Multiple Access) MAC (Medium Access Control) protocol, which grants access to the wireless medium in a time-synchronized fashion. Furthermore, a typical assumption is the availability of complete per-flow queue and link information. Unfortunately, these assumptions do not hold in

* Corresponding author.
E-mail addresses: jyyoo@nm.gist.ac.kr (J.-Y. Yoo), cigdem.sengul@telekom.de (C. Sengul), ruben.merz@telekom.de (R. Merz), jongwon@gist.ac.kr (J. Kim).

1389-1286/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.comnet.2012.05.009

IEEE 802.11 WMNs (Wireless Mesh Networks), both in theory and in practice.

Realizing backpressure scheduling in practice requires, first, moving away from TDMA to CSMA (Carrier Sense Multiple Access), and second, collecting information (e.g., next-hop queue length, physical-layer link capacity, and network topology) to compute the backpressure values [9–11]. Due to these challenges, there has been limited work on backpressure scheduling in practice. One recent work [13] implements TDMA-based backpressure scheduling in IEEE 802.11 WMNs by heavily modifying IEEE 802.11. However, their implementation relies on Ethernet connections to wireless nodes for stable control channels, which is not a typical assumption for WMNs. Another example, Horizon [9], monitors the next-hop queue length by listening to the packets forwarded by the next-hop node, where the queue length is piggybacked to the packets. For the MAC layer, Horizon uses generic IEEE 802.11 without modification. Using backpressure values as a metric for multi-path routing, Horizon achieves better fairness and throughput for multi-hop flows compared to pure multi-path routing. Another example is DiffQ [10], which uses backpressure values for scheduling links using IEEE 802.11e [14] at the MAC layer. DiffQ shows better fairness among several transport-layer protocols (e.g. TCP, ATP [15]) for multi-hop flows. These examples show the potential of realizing backpressure scheduling in practice.

In addition, there has been work on analyzing the throughput optimality within a certain degree of realistic settings. In [16], it is theoretically shown that the imperfect backpressure scheduler, due to the lack of global knowledge, results in only a fraction of the maximal capacity region. As an extent to this theoretical analysis, there has been work on the experimental analysis of the variants of backpressure scheduling (e.g., optimal CSMA (oCSMA) [17–19]). In oCSMA, only the sender queue length is used to determine the scheduling sequence, thus it does not require message passing for information exchange. However, the throughput-optimality of oCSMA is only shown for single-hop flow scenarios in WMNs [18] (e.g., for the "flow-in-the-middle" scenario, where three parallel flows contend for the medium and hence, the flow in the middle has a disadvantage compared to the other two flows). Their results show that, in practice, oCSMA does not provide the theoretical optimal throughput due to the existence of hidden terminal nodes. In this paper, we go a step further in illustrating the differences between theory[1] and practice, and evaluate the throughput gap under *multi-hop flow* scenarios.

To analyze the throughput gap between theoretical and practical backpressure scheduling, we rely on IEEE 802.11-based implementation of backpressure scheduling, which carefully deals with several system-level issues. Our evaluation covers various traffic scenarios including multi-hop flows. All experiments are carried out in GIST WMN testbed, which consists of 20 nodes with off-the-shelf IEEE 802.11 wireless interfaces. In each scenario, we also vary several MAC-layer parameters to further understand the interaction between MAC layer and backpressure scheduling. Our experimental evaluation verifies the throughput gap, which we show to emerge from both control and information inaccuracies due to the underlying IEEE 802.11 MAC layer.

In summary, we make the following contributions in this paper:

- We perform extensive measurements (960 experiments in total) and quantify the throughput gap in our testbed for various traffic scenarios. In these scenarios, we look at the impact of the flow composition (number of flows, number of hops per flow, whether they intersect or interfere, etc.) and the MAC-layer parameters such as RTS/CTS on or off, using static or automatic rate control, and different MAC-layer contention parameters.
- We show that MAC-layer collisions and backoff are the main reason behind the throughput gap. For certain scenarios, however, losses due to queue drops also increase the throughput gap. We believe mainly two factors incur such losses: *information* and *control* inaccuracies. Information inaccuracy stems from the lack of complete network topology information for scheduling decisions and control inaccuracy is the infeasibility of scheduling packets with exact timing in IEEE 802.11 MAC layer.
- Finally, we show the impact of MAC-layer parameters on the throughput gap. The results show that MAC-layer contention parameters have the least impact on the throughput gap, whereas the use of automatic rate control and RTS/CTS has a significant impact. This indicates that backpressure scheduling and certain MAC-layer mechanisms should not operate independently of each other.

The remainder of the paper is organized as follows. Section 2 explains the throughput gap between theoretical TDMA-based and practical CSMA/CA-based backpressure scheduling. Section 3 explains how we analyze the throughput gap and Section 4 presents the throughput gap between the theoretical and practical backpressure scheduling and its reasons based on the discussion in Section 2. Finally, we conclude in Section 5.

2. Throughput gap between theoretical TDMA-based and practical CSMA/CA-based backpressure scheduling

In this section, we first explain the throughput gap between theoretical TDMA-based and practical CSMA/CA-based backpressure scheduling in WMNs. Next, we present the solutions to both problems using backpressure scheduling and explain how the throughput gap between them may arise.

Backpressure scheduling requires solving both packet and link scheduling problems. Generally, "packet scheduling" is the problem of determining the flow sequence among multiple flows that traverse the same link and "link scheduling" is the problem of determining the link activation sequence among multiple links. Packet scheduling is performed between the network and MAC layers, and the

[1] Note that, in this paper, we mention theoretical-optimal throughput in the sense that it is optimal under the assumption of TDMA-based MAC layer.

link scheduling is performed by the MAC layer. The scheduling problem can be solved differently according to the different MAC-layer assumptions such as TDMA or CSMA/CA (i.e., IEEE 802.11). In the CSMA/CA-based scheduling problem, each node determines which per-flow queue should dequeue a packet whenever the MAC layer is ready to transmit (i.e., packet scheduling). Then, the MAC layer decides when to transmit this packet taking into account the neighbor links (i.e., link scheduling). On the other hand, in the TDMA-based scheduling problem, the above scheduling is considered in larger timescales, where the decision of packet and link scheduling becomes equivalent to allocating link bandwidth to flows. In this case, backpressure scheduling performs an allocation that achieves queue length stability and it is proven that stabilizing the queue lengths leads to optimal throughput by preventing queue overflow and link under-utilization (due to queue underflow).

The following lists the common assumptions that are made by both scheduling problems:

- *Per-flow queues:* Each node maintains per-flow[2] queues for separately storing the packets from different flows. This per-flow queue assumption could lead to higher complexity with the growing number of flows. But it is shown that when nodes fairly aggregate multiple flows into smaller sets of aggregated flows and schedulers are aware of the aggregated flows, quality of service can be guaranteed [20].
- *Static routing:* We use the shortest path routes that are determined apriori. We use this assumption to isolate the scheduling problem from possible side effects that can come from routing.

Based on these common assumptions, we present theoretical and practical scheduling in further detail in the following sections.

2.1. Backpressure scheduling with TDMA-based MAC layer

In this section, for TDMA-based MAC layer, we first formulate the scheduling problem and then present its theoretical solution (i.e., backpressure scheduling) that gives the optimal throughput. The original scheduling problem and the related source rate control and routing problems appear in [3,21,7].

We denote N as a set of nodes, E as a set of links and F as a set of flows. The link e: $(i,j) \in E$ exists between node i and node j if i and j are within the transmission range. The capacity of link e is denoted as c_e. Let x_i^f represent the rate packets are injected to the network at source node i of flow f. (If node i is not a source, $x_i^f = 0$.) For convenience, we denote x^f as the rate of flow f. The topology matrix T is a $|N| \times |E|$ matrix where, $T_{n,e}$ represents $1/c_e$, if link e is incident on node n, and 0 otherwise. Also, the allocated link capacity is represented as μ, which is a $|E| \times |F|$ matrix, where μ_e^f represents the capacity (bandwidth) of link e that is allocated to flow f. The network utility of flow f with rate x^f is denoted as $U^f(x^f)$. Finally, we denote ϵ as the constant MAC-layer overhead.

Based on this notation, the scheduling problem is to find x and μ that maximize the sum of network utilities $U^f(x^f)$ for all flows while satisfying capacity, flow preservation, and rate non-negativity constraints:

$$\max_{x,\mu} \quad \sum_{f \in F} U^f(x^f) \tag{1}$$

$$\text{subject to :} \quad \sum_{f \in F} T\mu^f \leqslant (1-\epsilon)\mathbf{1} \qquad \text{(capacity constraint)}, \tag{2}$$

$$x_i^f + \sum_{e:(j,i)\in E} \mu_e^f \leqslant \sum_{e:(i,j)\in E} \mu_e^f, i \in N, f \in F \quad \text{(flow preservation constraint)}, \tag{3}$$

$$x \geqslant 0 \qquad \text{(rate non-negativity constraint)}, \tag{4}$$

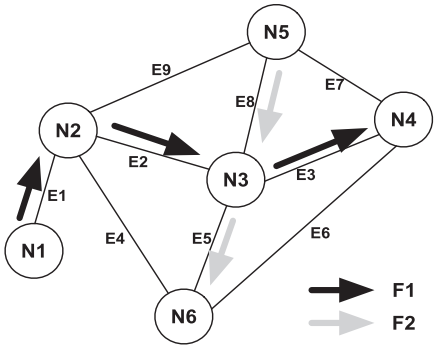
where $\mathbf{1}$ represents the $|N|$-dimension column vector with all 1s. Capacity constraint, given in Eq. (2), ensures that the sum of flow rates passing through a node should not exceed the capacity of its links. The flow preservation constraint in Eq. (3), guarantees that, at each node, the aggregate rate for incoming flows and the traffic of the node itself should not exceed the outgoing flow rates. Finally, Eq. (4) states that all rates x should always be non-negative.

Fig. 1 presents a simple example scenario. The scheduling problem is to find the rates of the source nodes $N1$ and $N5$, x^{F1} and x^{F2}, respectively, and the schedules for links $E1$, $E2$, $E3$, $E5$, and $E8$ that maximize the sum network utility while satisfying all constraints. For instance, the capacity constraint for the node $N3$ ensures that for $F1$ and $F2$, allocated link capacity for $E2$, $E3$, $E5$, and $E8$ does not exceed $1-\epsilon$. Also, for $F1$, the flow preservation constraint for $N3$ means that the incoming rate of $F1$ into $N3$ should be less than the outgoing rate.

Now, we present the theoretical backpressure scheduling that solves the problem above. The solution includes 3 steps which are taken at every TDMA time slot and is shown to converge to the throughput-optimal solution in [7].

1. Congestion price update: At each TDMA slot k, each node i updates its price for each flow as:
$$p_i^f(k+1) = \left[p_i^f(k) + \gamma \left(x_i^f + \sum_{e:(j,i)\in E} \mu_e^f - \sum_{e:(i,j)\in E} \mu_e^f \right) \right]^+, \tag{5}$$
where $[\cdot]^+$ gives only positive values (i.e., if negative value is given, it gives 0) and γ is a given positive constant. Note that $p_i^f(k)$ is interpreted as congestion price, i.e., the sum of extra traffic against the capacity allocated to flow f over the lifetime.
2. Backpressure value computation: Each node i collects congestion price information from its neighbor node j and calculates its differential as:
$$d_e^f = p_i^f - p_j^f, \tag{6}$$
where e is the link between node i and j.
3. Backpressure scheduling: We denote F_e as a set of flows that pass link e. Based on the collected and computed

[2] We identify a flow by 5-tuple information (source and destination IP addresses, source and destination ports, and protocol).



T	E1	E2	E3	E4	E5	E6	E7	E8	E9
N1	c_{E1}^{-1}	0	0	0	0	0	0	0	0
N2	c_{E1}^{-1}	c_{E2}^{-1}	0	c_{E4}^{-1}	0	0	0	0	c_{E9}^{-1}
N3	0	c_{E2}^{-1}	c_{E3}^{-1}	0	c_{E5}^{-1}	0	0	c_{E8}^{-1}	0
N4	0	0	c_{E3}^{-1}	0	0	c_{E6}^{-1}	c_{E7}^{-1}	0	0
N5	0	0	0	0	0	0	c_{E7}^{-1}	c_{E8}^{-1}	c_{E9}^{-1}
N6	0	0	0	c_{E4}^{-1}	c_{E5}^{-1}	c_{E6}^{-1}	0	0	0

Fig. 1. Example of a WMN topology with six nodes, nine links and two flows.

information, for each link e, the flow that has the highest backpressure value, $f(e)$, is computed as:

$$f(e) = \arg\max_{f \in F_e} d_e^f.$$

After that, the allocated link capacity vector μ^* is selected such that it satisfies:

$$\mu^* = \arg\max_{\mu} \sum_e \mu_e^{f(e)} d_e^{f(e)}$$

subject to

$$\mu \in \Pi(\epsilon, T), \tag{7}$$

where $\Pi(\epsilon, T)$ is the capacity region that contains all μ^f which satisfy the capacity constraint (Eq. (2)). That is, for each link e: (i,j), $\mu_e^{f(e)}$ is offered to flow $f(e)$.

This solution assumes full knowledge about the physical-layer link capacity and the network topology to construct the matrix T. Based on this information, the backpressure scheduler assigns how much bandwidth (μ) each flow should get, which is in turn scheduled by the TDMA-based MAC protocol.

2.2. Backpressure scheduling with practical CSMA/CA-based MAC layer

In this section, we present the approximation of backpressure scheduling for CSMA/CA. Due to the high complexity of modeling CSMA/CA behavior,[3] we do not present any mathematical models and focus mainly on how backpressure scheduling works with CSMA/CA.

IEEE 802.11 exploits DCF (Distributed Coordination Function) as a core function for medium access. In DCF, before a node starts a transmission, it senses whether the channel is idle for a DIFS (DCF Inter-Frame Space) duration, plus an additional backoff time that is randomly chosen between minimum (CWMin) and maximum (CWMax) contention window values. Only when the channel remains idle for a DIFS plus backoff time, the node can initiate transmission. To provide service differentiation among traffic classes in a distributed way, IEEE 802.11e uses EDCA (Enhanced Distributed Channel Access) which is built on DCF. EDCA provides up to four ACs (Access Categories) and its corresponding queues (we name them as priority queues) as shown in Fig. 2. By assigning smaller CWMin, CWMax, and AIFS (Arbitrary Inter-Frame Space) values to ACs with higher priorities, EDCA differentiates the transmission probability in favor of high-priority ACs in a statistical sense. Note that EDCA uses AIFS rather than DIFS to provide different initial waiting times for different traffic classes.

The medium access parameters and priority queues are typically implemented in hardware as the actions involving these parameters require microsecond granularity. Even the well-known open-source WiFi MAC driver, MadWiFi [23], does not provide much control over the medium access parameters and priority queues. In these circumstances, the only way to implement backpressure scheduling is to build per-flow queues on top of IEEE 802.11e standard implementation as shown in Fig. 2. Under IEEE 802.11e, practical backpressure scheduling is approximated with the following steps:

1. *Congestion price update:* The congestion price p_i^f in Eq. (5) can be interpreted as the queue length for flow f at node i, if γ is taken as the packet length in bits. Then,

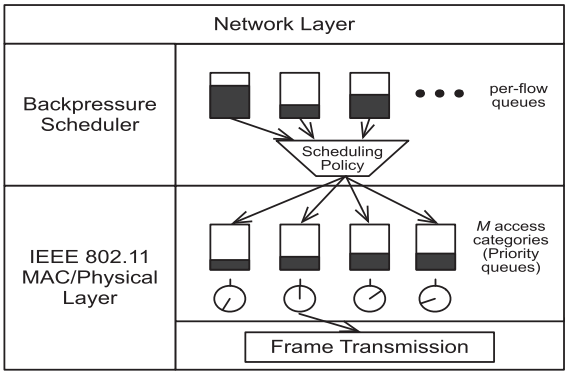


Fig. 2. Protocol stack of practical backpressure scheduling in IEEE 802.11e MAC layer.

[3] IEEE 802.11 employs CSMA/CA with binary exponential backoff and there is no accurate mathematical model of binary exponential backoff unless the model includes non-linear (such as exponential) terms in equations. Additionally, if we consider multiple collision domains (i.e., a typical condition in WMNs), the complexity of non-linear equations increases [22].

the scheduling sequence can be similarly decided. This is the reason why the recent implementations of backpressure scheduling use the queue length to control medium access [9,10,12].

2. *Backpressure value computation:* To compute backpressure value $\left(d_e^f\right)$, we need to know the per-flow queue length of the next-hop node. This queue length can be monitored by overhearing the packets transmitted by the next-hop node and extracting the per-flow queue length piggybacked in packets. This is also the method we use in our implementation.
3. *Backpressure scheduling:* In theory, the flow with higher backpressure value should be scheduled first. This can be easily implemented in a packet scheduler (backpressure scheduler in Fig. 2), since each node can compute the backpressure values for all flows. For link scheduling, since we need to operate based on IEEE 802.11e, the backpressure values should be mapped to the priority queues as shown in Fig. 2. Note that 1-to-1 mapping from backpressure values to priority queues would not be possible due to the finite number of priority queues in IEEE 802.11e. In this paper, similar to DiffQ [10], we use a linear mapping function. We assume all per-flow queues have the same maximum queue size and denote this size as q_{max}. Then, the backpressure value $\left(d_e^f\right)$ can be mapped to the priority queue index k, as:

$$k = \left\lfloor M \frac{q_{max} + d_e^f}{2q_{max}} \right\rfloor, \tag{8}$$

where M is the number of priority queues.

2.3. Possible reasons for the throughput gap

Now, we discuss the two possible reasons for throughput gap. The first one is *information inaccuracy* that results from the incomplete information about next-hop queue length and network topology, which are needed for scheduling decisions. This is due to the approximations of steps 2 and 3 described in Section 2.2. The second one is *control inaccuracy* that results from the approximation of link scheduling using IEEE 802.11e in step 3.

Information inaccuracy may result from not having exact information about:

- *Next-hop per-flow queue lengths* required to compute the backpressure values in Eq. (6). This information is the input to the backpressure scheduler for the scheduling decision.
- *Topology information T* required to compute the capacity constraint in Eq. (7). This information basically conveys how the medium is shared among links (e.g., how many links interfere with each other). Based on this, the TDMA-based scheduler decides how much bandwidth should be allocated to each flow to fully utilize the links.

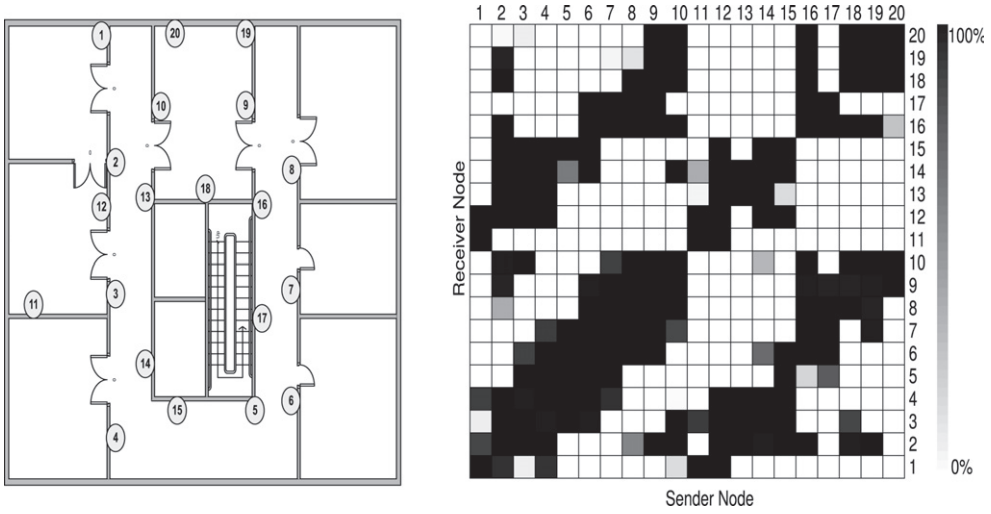
Unfortunately, both types of information cannot be accurately monitored due to following reasons. First, acquiring next-hop per-flow queue length requires message-passing between the next-hop node and the node itself (implemented as overhearing piggybacked information). But, this takes some time and this monitoring delay may trigger per-flow queue length fluctuation [24] and moreover, whenever there is link asymmetry, this information would not be correctly estimated. Second, topology information is essential to understand how link bandwidth could be shared among flows. However, trying to collect this information would require acquiring transmission and interference relations between every node pair and this results in a large amount of monitoring traffic [25].

Control inaccuracy results from IEEE 802.11e operation. In TDMA, a central controller coordinates the overall link scheduling (i.e., link activation sequence) over time-synchronized slots and transmits packets with an exact schedule. Note that the distributed version of TDMA-based backpressure scheduling [7] also assumes on-time transmissions. This is feasible because nodes are time-synchronized and complete knowledge of network topology including interference relations is assumed. However, in practice, synchronized operation cannot be assumed and moreover, we cannot acquire the accurate interference information [25]. On the other hand, the channel access in CSMA/CA is probabilistic, therefore, it is not possible to transmit packets on time based on a pre-determined sequence. Additionally, IEEE 802.11 has operational overheads such as backoff time and interframe spacings (e.g., AIFS in IEEE 802.11e), which are not incurred in TDMA.

3. Throughput gap analysis method

We *quantitatively show the throughput gap* by comparing the theoretically optimal throughput and the experimental throughput under various traffic scenarios ranging from a single flow to multiple flows with different parameter settings. In order to understand experimental throughput performance, we run our backpressure scheduling implementation in an IEEE 802.11 WMN testbed and perform throughput measurements. Also, we compute the theoretical throughput for these experiments using the model described in Section 2.1. To feed the model with the necessary input (e.g., network topology), we use the measured values from our testbed experiments. To distinguish the effects of the reasons behind the throughput gap, we *quantitatively show the wasted throughput* due to different reasons by inspecting all the delivered and dropped packets in all experiments.

Note that for some simple topologies and traffic scenarios we can immediately measure the throughput gap. We describe an illustrative example as follows. First, setup three nodes, A, B, and C. In the first experiment, set up two flows from B to A and from B to C and measure the throughput of both flows. In the second experiment, setup two flows from A to B and from C to B and measure the throughput of both flows. The difference of the resulting throughputs would be the throughput waste due to MAC-layer collisions and backoffs since, in the first experiment, Node B makes sure only one node is transmitting. However, once the topology or the traffic scenario becomes complicated, this method would be difficult to measure the throughput gap.



(a) Map of GIST WMN testbed deployed in GIST Korea.

(b) The delivery percentage matrix plot of GIST WMN testbed with the modulation rate of 36 Mbps.

Fig. 3. GIST WMN testbed and its node-connectivity plot.

3.1. Network testbed and traffic scenarios

For the network testbed, we use GIST WMN testbed whose deployment map is shown in Fig. 3a. GIST WMN testbed consists of 20 wireless nodes, equipped with a VIA Eden 1.0 GHz CPU and two Atheros-chipset-based IEEE 802.11 interfaces. As the operating system of wireless nodes, we use Linux Voyage distribution with kernel version 2.6.26. Also, to precisely measure the experimental throughput performance, we use PaPMo (Packet-accurate Protocol Monitoring) [26] that monitors the life of all packets (e.g., when and where it is generated, delivered, or dropped) and the internal protocol state (e.g., per-flow queue length, packet reception SINR, etc). Also, to repeat experiments in a controlled fashion, we use OMF (Orbit Management Framework) [27] that manages every testbed node in a centralized fashion and automates experiment setup and traffic generation. Note that OMF and PaPMo use a separate management/measurement network to operate the testbed and deliver monitoring traffic thus it does not affect the experimental throughput [26].

To show the overall connectivity of nodes in our testbed, we plot the node-connectivity matrix in Fig. 3b. We measure the node connectivity as the broadcast delivery ratio of IEEE 802.11 using the fixed modulation rate of 36 Mbps.[4] Each node takes turns and broadcasts 100 packets/s for three minutes. Black cells represent the case when all the broadcast packets are delivered and white cells represent the case when all packets are lost. Note that this connectivity information is used for computing the theoretically optimal throughput, presented in detail in Section 3.2.

[4] We choose 36 Mbps since it is the best modulation rate for overall links in GIST WMN testbed.

For our performance evaluations, we use a number of diverse traffic scenarios, which are listed in Table 1. These scenarios are categorized as follows:

- *Single flow:* This is the most basic scenario. However, note that a multi-hop single flow experiences intra-flow contention (i.e., medium access contention among the nodes in the same flow).
- *Parallel flows:* Two parallel flows with inter-flow contention (i.e., medium access contention between the nodes in neighboring flows).
- *Overlapping flows:* Two overlapping flows that run in opposite directions. The flows do not experience explicit inter-flow contentions, but intra-flow contention from two different flows.
- *Crossing flows:* Two flows with cross inter-flow contention. Here, the node where the two flows cross also needs to perform packet scheduling to decide the sequence of flows to serve.
- *Complex flows:* These scenarios include 3, 4, and 5 flows. In these scenarios, the parallel, overlapping and crossing flows exist in various combinations.

3.2. Theoretical throughput computation

Here, we describe how we compute the theoretical throughput of TDMA-based backpressure scheduling for a given traffic scenario in our testbed. Remember that we can obtain the theoretical throughput by solving the network utility maximization problem in Eq. (1) with the constraints of Eqs. (2)–(4) in Section 2.1. As a solver, we use the CVX (Convex) optimization toolkit in Matlab [28].

To solve the problem, we need to choose the utility function and the variables that represent the network connectivity and the flows. For the utility function U^f of flow f,

we use the log function which is widely used as it provides proportional fairness [29]. We ignore the MAC-layer overhead by setting $\epsilon = 0$. For the topology matrix, we use the node-connectivity matrix shown in Fig. 3b. For each link, we set its capacity as the product of modulation rate and delivery ratio, where the modulation rate is 36 Mbps.

It must be noted that we do not attempt to compute the true optimal throughput. Indeed, our computation provides an over-estimation. First, in our theoretical computation, we do not consider the concurrent transmissions in a collision domain. Second, we use 0 for ϵ which is not typical in the real-world MAC layer. However, by comparing this estimation to the practical throughput, we still obtain a valuable insight on the extent of throughput gap and different causes (i.e., information and control inaccuracies) for the observed gap.

3.3. Practical throughput computation

To understand the achievable throughput in practice, we carefully implement backpressure scheduling in our testbed, paying attention to reducing the execution time overhead as much as possible. Our implementation uses Click 1.8 [30] and MadWiFi 0.9.4 [23]. Click is a well-known tool that provides packet-level flexible programmability between the transport and the MAC layer. MadWiFi provides an implementation of IEEE 802.11e MAC layer and has the multiple priority queues as defined in the standard [14]. We extend the number of priority queues from 4 to 8 ($M = 8$ in Fig. 2) to provide more fine-granularity medium access control.[5] We then implement a flow classifier, a per-flow table for maintaining flow-specific information, per-flow queues and backpressure scheduler as Click elements and place them between the original network and MAC layers. During the validation of our implementation, we notice that additional delays are introduced between packet scheduling (at per-flow queues) and link scheduling (at priority queues) due to the existence of additional queues in Linux networking kernel. We make sure to bypass these queues to avoid any delay, and potential queue fluctuations (see Appendix A for more details).

For packet processing, the flow classifier receives packets from the network layer and classifies them according to a unique flow identifier. Based on this identifier, packets are stored in their corresponding FIFO (First-In First-Out) queue. The per-flow table also maintains the per-flow queue length of the next-hop nodes and uses it to compute a backpressure value d_e^f for a given link using Eq. (6). For backpressure scheduling, the packet scheduler runs as a thread and checks if the MAC-layer priority queue has a space to store a packet. Then, the packet scheduler performs scheduling according to the backpressure values $\left(d_e^f\right)$ and pushes the packet to the corresponding priority queue of the MAC layer. To indicate which priority queue to insert the packet, we put the index of the corresponding priority queue to the TOS (Type of Service) field in the IP header. For the index computation, we use Eq. (8). Note that, in our implementation, packets in the per-flow queues are not scheduled when the priority queues are full to prevent priority-queue overflows.

For the next-hop per-flow queue length monitoring, each node monitors the upstream node queue length similar to DiffQ [10]. The IPv4 header contains 13 bits FO (Fragmentation Offset) field.[6] When a node forwards a packet to the next-hop node, the node piggybacks the per-flow queue length of the corresponding flow in the FO field. The previous-hop node promiscuously listens to the medium and records the FO field of packets based on their flow IDs. Again, during tests, we observe the order of forwarding packets could be mixed at the priority queues of the MAC

Table 1
Various traffic scenarios (from 1 to 5 flows) with the given routing paths. Note that the right-most node is the source node and the left-most node is the destination node.

A single flow (Scenario 1)	N20 → N10 → N9 → N16 → N7 → N6
A single flow (Scenario 2)	N11 → N12 → N3 → N2 → N18 → N9 → N8
Crossing flows (Scenario 3)	N20 → N10 → N9 → N16 → N7 → N6 N11 → N12 → N3 → N2 → N18 → N9 → N8
Parallel flows (Scenario 4)	N6 → N7 → N16 → N9 N4 → N3 → N2 → N10
Parallel flows (Scenario 5)	N4 → N3 → N2 → N10 N8 → N7 → N6 → N15
Overlapping flows (Scenario 6)	N11 → N12 → N3 → N2 → N18 → N9 → N8 N9 → N18 → N2 → N3 → N12
Complex three flows (Scenario 7)	N1 → N12 → N3 → N14 N6 → N7 → N16 → N9 N19 → N10 → N2
Complex three flows (Scenario 8)	N20 → N10 → N9 → N16 → N7 → N6 N12 → N3 → N4 → N15 N19 → N9 → N16 → N7
Complex four flows (Scenario 9)	N1 → N12 → N3 → N14 N11 → N12 → N3 → N2 N4 → N3 → N2 → N10 N19 → N10 → N2
Complex four flows (Scenario 10)	N1 → N12 → N3 → N14 N11 → N12 → N3 → N2 N19 → N9 → N16 → N7 N19 → N10 → N2
Complex five flows (Scenario 11)	N1 → N12 → N3 → N14 N10 → N18 → N9 → N16 → N7 N11 → N12 → N3 → N2 N2 → N10 → N18 → N9 → N8 N19 → N10 → N2
Complex five flows (Scenario 12)	N10 → N18 → N9 → N16 → N7 N19 → N9 → N16 → N7 N4 → N3 → N2 → N10 N8 → N7 → N6 → N15 N19 → N10 → N2

[5] We use 8 priority queues since Atheros 5 k series support 10 hardware queues and 2 of them are designated as beacon and UAPSD (Unscheduled Automatic Power Save Delivery) queues.

[6] We assume that all nodes use IPv4 and there is no fragmentation of packets at the network-layer and the FO field is always zero.

layer. This results in errors in the monitoring order and leads to the computation of wrong backpressure values. To filter out-of-order packets, we introduce sequence numbers and used the ID field in the IPv4 header as the sequence number. (For more details, see Appendix A.) Note that this monitoring also incurs delayed information due to monitoring delay, but we leave this as information inaccuracy since the monitoring delay inevitably happens.

For source rate control, we implement threshold-based source rate control similar to [10] as follows. Whenever the per-flow queue at source node becomes smaller than some pre-defined threshold, T_q, the source node sends more packets to fill up the per-flow queue to the threshold. Although our source rate control is implemented as a heuristic manner, we have validated that, in selected scenarios (see Table 1), the performance of our source rate control marginally differs to the one in appeared at [7], whose stability property has been proved in theory. Note that the source rate control at [7] deals with log utility function which is the limited case of general source rate controls. This source rate control can prevent the per-flow queue overflows at source node but the intermediate nodes are still vulnerable to queue overflows.

3.4. Quantifying throughput gap

To quantify the throughput gap, we denote the total number of bits carried over the network by all flows using the theoretical TDMA-based backpressure scheduling as T_{theory} (bits). Also, we denote the throughput for practical CSMA/CA-based backpressure scheduling as $T_{practical}$ (bits). We expect the throughput gap to be a result of wasted bits because of queue losses (Δ_q), collisions (Δ_c), MAC-layer overhead (Δ_o) (e.g., IFS times, and ACKs that are not used in TDMA), and MAC-layer backoff (Δ_b). We assume that the following relationship holds:

$$T_{theory} = T_{practical} + \Delta_q + \Delta_c + \Delta_o + \Delta_b. \quad (9)$$

We believe the primary cause behind Δ_c and Δ_b is control inaccuracy (approximated link scheduling) and Δ_q is information inaccuracy (especially in the case of topology information). Essentially, if a node does not strictly control the time of packet transmissions, more than two nodes could concurrently transmit in a collision domain, which leads to a collision, which also leads to unnecessary backoff. Similarly, Δ_q quantifies the information inaccuracy, since a node might overflow the next-hop queue, if it does not have exact knowledge of the neighbor topology or the queue length of the next-hop node, and sends faster than it should.

Finally, from Eq. (9), we derive the throughput gap, $\delta(0 \leqslant \delta \leqslant 1)$, as:

$$\delta = \frac{T_{theory} - T_{practical}}{T_{theory}} = \frac{\Delta_q + \Delta_c + \Delta_o + \Delta_b}{T_{theory}}. \quad (10)$$

We additionally define:

$$W_q = \frac{\Delta_q}{T_{theory}},\ W_c = \frac{\Delta_c}{T_{theory}},\ W_o = \frac{\Delta_o}{T_{theory}},\ \text{and } W_b = \frac{\Delta_b}{T_{theory}}. \quad (11)$$

We denote δ^s to represent the δ for a scenario s. Similarly, W_q^s, W_c^s, W_o^s, and W_b^s represent the corresponding values for scenario s.

We next explain how we compute Δ_q, Δ_c, Δ_o and Δ_b.

- *Computing* Δ_q: We compute Δ_q by counting all the dropped packets and multiplying the hop counts that the dropped packets traveled. More specifically, we denote the number of dropped packets due to per-flow queue overflow at node i for flow f as ql_i^f. We denote $H(i,f)$ as the hop count of node i to the source node of flow f and $R(f)$ as the set of nodes that construct the route of flow f. PS denotes packet size. Hence, Δ_q is:

$$\Delta_q = \sum_{f \in F} \sum_{i \in R(f)} PS \cdot ql_i^f \cdot H(i,f)$$

- *Computing* Δ_c: To compute Δ_c, we count all the dropped packets due to collisions and compute the lost bits by multiplying with packet size, PS. Additionally, we divide the lost bits by the number of nodes affected by each collision. The reason is that only one packet among colliding packets can be received successfully by the receiver, regardless of how many packets are involved in the collision. We define cl_i^f as the number of dropped packets due to MAC-layer collisions and $S_C(p)$ as the number of interfering nodes for each packet p. Then, we can derive Δ_c as:

$$\Delta_c = \sum_{f \in F} \sum_{i \in R(f)} PS \cdot \sum_{p=1}^{cl_i^f} \frac{1}{S_C(p)}.$$

Since it is very hard to measure $S_C(p)$ for each packet, we approximate this as the number of interfering nodes of a node i.

Table 2
Common parameters used in all experiments.

Parameter	Value
Maximum queue length, q_{max}	250 packets
Source rate control threshold, T_q	100 packets
Maximum priority queue length	10 packets
Packet size, PS	1500 bytes
Frequency	5.26 GHz
Transmit power	18 dBm

Table 3
IEEE 802.11e extended link access priority parameters.

Priority	7	6	5	4	3	2	1	0
AIFS (linear increasing set)	2	3	3	3	6	6	6	7
AIFS (expon. increasing set)	2	3	4	5	6	7	8	9
CWMin (linear increasing set)	1	2	3	4	5	5	5	5
CWMin (expon. increasing set)	2	3	4	5	6	7	8	9
CWMax (linear increasing set)	3	3	4	5	7	8	9	10
CWMax (expon. increasing set)	4	5	6	7	8	9	10	11

Table 4
Parameters that we combinate.

Parameter	Value
Modulation rate	Static 36 Mbps/automatic (SampleRate)
RTS/CTS	On/off
Medium access parameters (AIFS, CWMin, and CWMax)	Linear/exponential increasing sets (Table 3)

Table 5
δ over all scenarios by varying modulation rate, RTS/CTS on/off, and medium access parameters (linear/exponential increasing sets are shown as L and E, respectively). All values (except W_b) are given as (average, the standard deviation).

RTS/CTS	MAC	Rate	δ	W_q	W_c	W_o	W_b
Off	E	Auto	0.64, 0.06	0.03, 0.01	0.43, 0.08	0.03, 0.01	0.15
Off	L	Auto	0.64, 0.06	0.03, 0.01	0.43, 0.08	0.03, 0.01	0.15
On	E	Auto	0.68, 0.04	0.00, 0.00	0.08, 0.02	0.10, 0.01	0.50
On	L	Auto	0.68, 0.04	0.00, 0.00	0.08, 0.01	0.10, 0.01	0.50
Off	E	36	0.47, 0.08	0.01, 0.01	0.19, 0.04	0.07, 0.01	0.20
Off	L	36	0.48, 0.08	0.01, 0.01	0.19, 0.04	0.07, 0.01	0.21
On	E	36	0.49, 0.06	0.00, 0.00	0.08, 0.02	0.12, 0.02	0.29
On	L	36	0.49, 0.07	0.00, 0.00	0.08, 0.02	0.13, 0.02	0.28

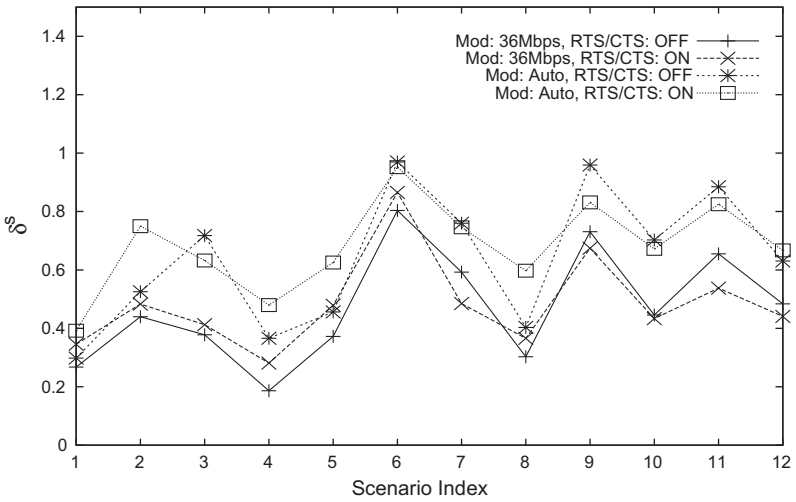


Fig. 4. δ^s using medium access parameters with exponential increasing set.

- *Computing* Δ_o: To compute Δ_o, we add up the SIFS, DIFS, MAC-layer ACK frame transmission, and physical-layer preamble times of all dropped and transmitted packets.
- *Computing* Δ_b: As we need multiple 5 GHz-supporting spectrum analyzers to measure backoff time accurately with a micro-second granularity, we do not measure Δ_b directly but estimate it based on the others. Therefore, to compute Δ_b, we subtract $T_{practical}$, Δ_q, Δ_c, and Δ_o from T_{theory} (see Eq. (9)). It is important to note that since every term has a measurement noise, this computation aggregates all the noise values in Δ_b. We leave more accurate calculation of Δ_b as future work.

4. Throughput gap analysis results

4.1. Experiment setup

In every experiment, each node uses static routing to minimize effects from routing dynamics. Table 2 denotes the common parameters of all our experiments. We setup the maximum per-flow queue length (q_{max}) to 250, which is large enough to avoid per-flow queue overflow drops due to minor queue length fluctuations [24]. We set the source rate control threshold, T_q, to 100. We set the maximum priority queue length (i.e., MadWiFi driver queue) to 10, the smallest value that does not hurt the throughput performance due to system overhead. It is preferable to set this value as small as possible since additional queuing delays could result in fluctuations of per-flow queues [24].

Note that there are 8 priority queues in the MadWiFi driver. Each priority queue exploits different medium access parameters (AIFS, CWMin, and CWMax) and these medium access parameters may significantly affect the throughput performance. For a complete experimental analysis, we would need to experiment all possible combinations of these parameters. To avoid this but still observe the impact of these parameters, we use the approaches of exponentially and linearly increasing these parameters. Exponential (or linear) increase means that as the priority of the queues increases, we exponentially (or linearly) increase the medium access parameters.[7] The corresponding medium access parameters are shown in Table 3. Note that the values in the table are the exponent of 2. We use the combination of all parameters shown in Table 4. Thus, for each scenario and parameter settings, we perform eight experiments. We repeat each experiment 10 times.

[7] Note that [10] uses the linear increase approach.

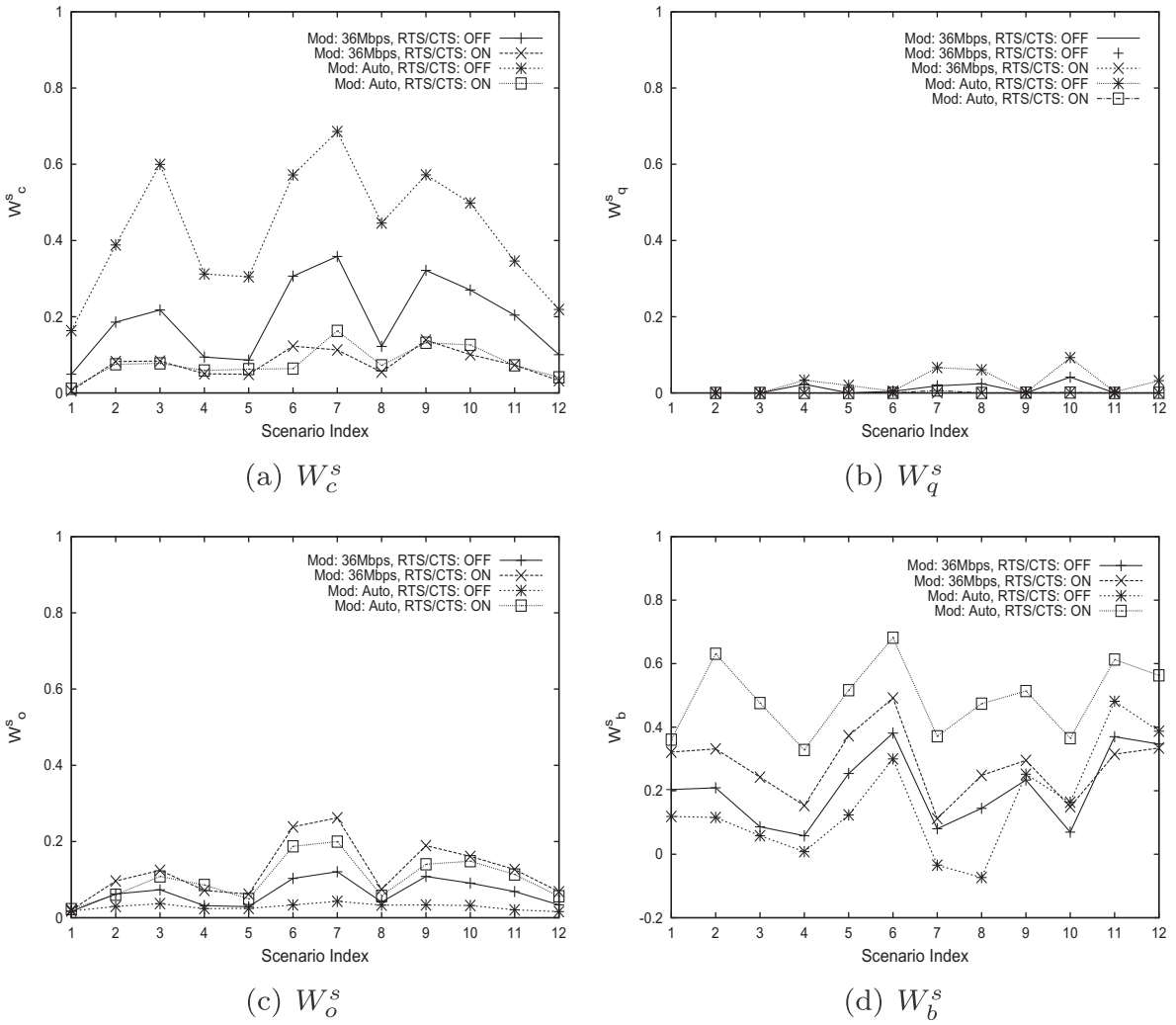


Fig. 5. Throughput wastes with exponential increasing medium access parameters.

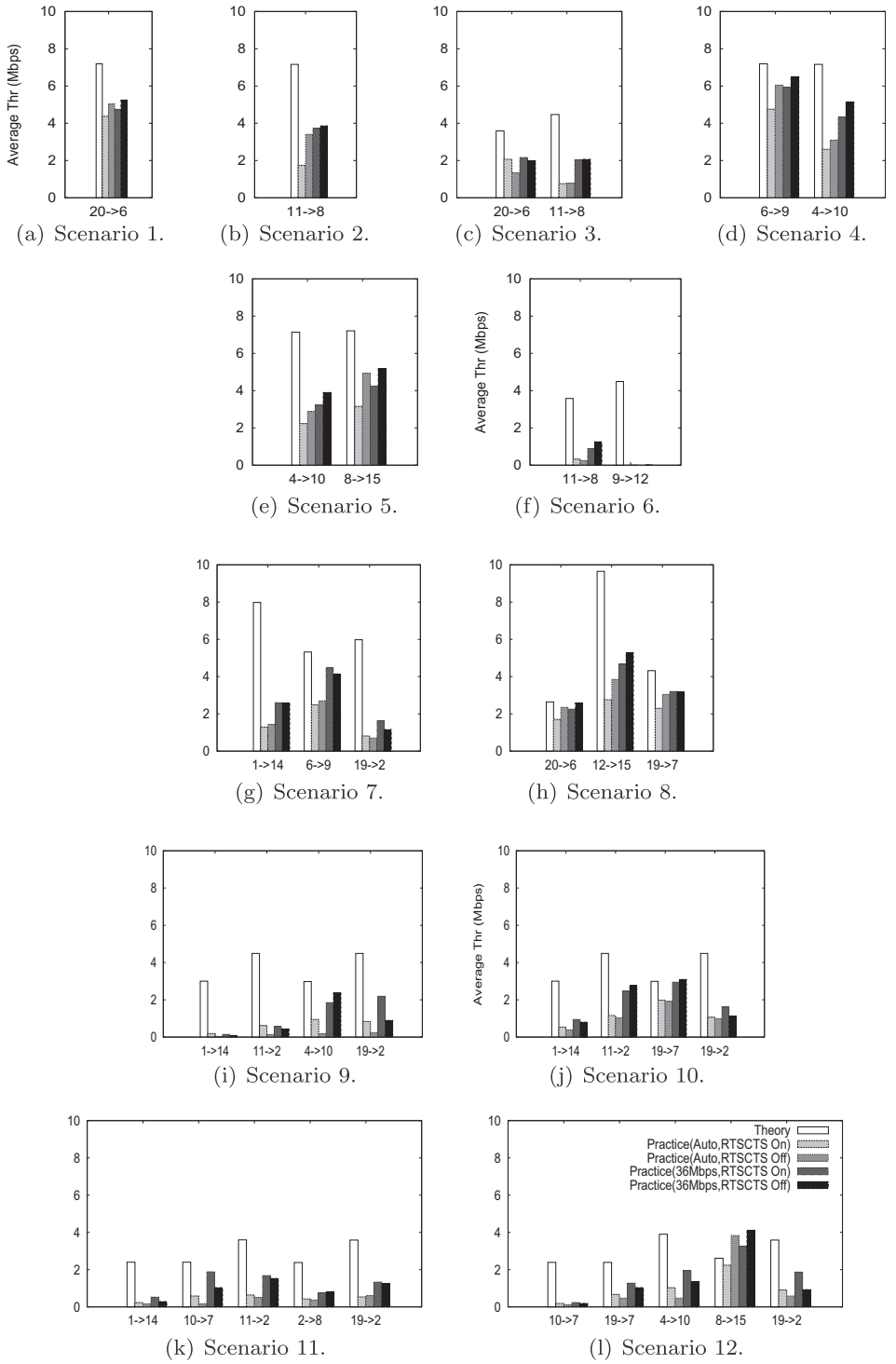
4.2. Average throughput gap result over all scenarios

We perform overall 960 experiments and our experiment traces are public [31]. The average δ values of the overall scenarios are shown in Table 5. We also take a more detailed look at the throughput gap based on W_q, W_c, W_o, and W_b results. For this, we inspect all the dropped packets due to MAC-layer collisions and per-flow queue overflows monitored by PaPMo [26]. Note that, for W_b, we do not present standard deviation since it is computed by using averages of others as mentioned in Section 3.4.

In overall, we observe that δ is higher than 47% for all possible combinations of parameters. We also observe that δ increases when automatic rate control is used. This is intuitive because automatic rate control tries to find the best rates and adapts modulation rates based on packet drops. However, the rate control algorithm does not have information about the causes of packet drops and might reduce the rate unnecessarily. For example, in case of MAC-layer collisions, the modulation rate should not be decreased. Therefore, in our experiments, automatic rate control achieves worse throughput than the static 36 Mbps setting, which is the best rate based on our measurements. Also, the packet transmission time with different modulation rates changes the overlapping probability of packets and hence, the collision probability among senders. This is easily seen by increasing W_c with automatic rate control and no RTS/CTS in Table 5.

On the other hand, RTS/CTS does not seem to affect δ. Note that even though turning on RTS/CTS saves some bandwidth by mitigating the hidden terminal problem [32] and reduces collisions, the overhead of RTS/CTS cancels the saved bandwidth. This again can be observed by increasing W_o and W_b when RTS/CTS is on, while W_c significantly decreases in this case (see Table 5). Finally, the medium access parameters show negligible throughput performance difference.

In summary, the results show that W_c is significantly higher than W_q. Furthermore, W_b also shows non-negligible throughput waste, even though its computation is not accurate. Nevertheless, based on these results, we conjecture that control inaccuracy has more impact on the throughput gap then the information inaccuracy. However, so far, we have only considered the average results over all flows and scenarios. When we look into individual scenario, we see some cases where the throughput waste due to per-flow queue overflows also becomes non-negligible. Next, we show the results of throughput gap δ and throughput waste, W_q and W_c, for each scenario.



(a) Scenario 1. (b) Scenario 2. (c) Scenario 3. (d) Scenario 4.

(e) Scenario 5. (f) Scenario 6.

(g) Scenario 7. (h) Scenario 8.

(i) Scenario 9. (j) Scenario 10.

(k) Scenario 11. (l) Scenario 12.

Fig. 6. Throughput comparison of backpressure scheduling between theoretical TDMA and practical CSMA/CA-based MAC layer.

4.3. Throughput gap results of individual scenario

When we consider scenarios individually, we observe diverse throughput gap as shown in Fig. 4. Interestingly, δ^s is not correlated to the number of flows (the scenarios with higher index have more flows). This is surprising since lower throughput is expected as more nodes compete to send traffic. However, in our experiments, for some scenarios with one or two flows, we observe relatively high δ (such as Scenarios 2, 3 and 6). Similarly, for some scenarios with 3, 4, and 5 flows (Scenarios 8, 10, and 12), we can observe relatively low δ^s. Looking deeper, we observe that a particular set of nodes (Node 2, 3, 11, 12, and 18) are most commonly involved in the scenarios with high throughput gap (Scenario 2, 3, 6, 9, and 11). From this, we conjecture that this set of nodes forms multiple hidden terminal conditions (e.g., Node 2 is hidden to Node 11 and Node 3 is hidden to Node 18 as also seen from 0% delivery ratio between these nodes in Fig. 3b). This affects the throughput significantly by incurring many collisions. This is also clearly seen in Fig. 5a, which plots W_c^s values for each scenario. The scenarios with high throughput gap also have high W_c^s values when the RTS/CTS is off. On the other hand, using automatic rate control, which also uses slower rates, a higher number of collisions is observed due to the potential increase in overlapping times between two senders.

Similarly, Fig. 5b depicts W_q^s values for each scenario. Looking at individual scenarios, we see that the throughput waste due to the per-flow queue overflow is not negligible for some scenarios. For instance, in Scenario 9 with RTS/CTS off and automatic rate control, W_q^s is around 10%. Furthermore, when we use RTS/CTS, we do not observe many packet drops due to per-flow queue overflow. We suspect this behavior as RTS/CTS changes the transmission opportunity regarding the neighbor topologies (by using RTS/CTS, the sender transmits when the receiver is also idle).

In Fig. 5c, we depict W_o^s values for each scenario. As shown in Fig. 5c, RTS/CTS incurs huge overhead due to the additional transmission times of RTS and CTS frames. Also, we plot the throughput waste conjectured by backoff time in Fig. 5d. Although we cannot guarantee the accuracy of W_b^s values, we can roughly observe that W_b^s values are not negligible.

Next, we present the throughput of individual flows in Fig. 6. We can see that, for some scenarios, a flow takes all the network bandwidth that should be shared with the other flows (see Fig. 6f). Also, for Flow 8 → 15 in Scenario 12 (Fig. 6l), even though the theoretically optimal throughput is around 3 Mbps, the experimental throughput result is larger. It means that although in theory, the backpressure scheduler with log as utility function provides proportional fairness [29], in practice it may not be possible to achieve this due to the information and control inaccuracies.[8]

To sum up, a significant throughput gap between theoretical TDMA and practical CSMA/CA exists. Major reasons behind this throughput gap are packet drops due to MAC-layer collisions, backoffs and per-flow queue overflows which result from the two inaccuracy problems.

[8] This fairness problem also comes from the poor approximation of source rate control.

5. Conclusion

In this paper, we quantitatively show the throughput gap when a backpressure scheduler uses theoretical TDMA or practical CSMA/CA. Our thorough experimental analysis results show a significant throughput gap due to control and information inaccuracies. To operate IEEE 802.11 WMNs more efficiently, we need to reduce this gap. Recently, there has been various work on throughput-optimal CSMA/CA scheduling algorithms [34,35]. We consider that the throughput gap between this theoretical CSMA/CA scheduling algorithms and practical scheduling algorithms could be smaller, which we consider as future work.

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2012R1A2A2A01014687) and by Telekom Innovation Laboratories, in the framework of the BOWL project.

Appendix A. Implementation issues of backpressure scheduling

The two main issues we have faced during our implementation of backpressure scheduling are: (1) dealing with the *hidden queues in Linux networking kernel* and (2) *next-hop queue length monitoring*. The hidden queues in Linux networking kernel causes additional queuing delay between the per-flow queue in Click and the priority queue at the MAC layer. This additional queuing delay results in time difference between the operations of packet and link scheduling, and leads to a significant fluctuation in the per-flow queues [24]. To resolve this issue, we by-pass the hidden queues by modifying Linux networking kernel and MadWiFi. Fig. A.7 shows the function-call map of Linux networking kernel. At top left, user-space function *send* is the entry point to the networking kernel. It passes through all the IP operations and reaches the device driver (depicted with solid line). As observed, there are two hidden queues shown as circles: (1) the IP buffer for optimal fragmentation [33] and (2) the Qdisc which is formally known as the interface queue. These hidden queues have an effect on the performance due to additional queuing delay. To avoid this, we by-pass the hidden queues as depicted in dotted line of Fig. A.7. This ensures that there are only two queues: per-flow queues at the network layer and priority queues at the MAC layer.

The monitoring of next-hop queue length also experienced some problems since we use multiple priority queues at the MAC layer and the packet order might change due to using multiple priority queues. For instance, consider the case when packets get queued at different priority queues at the MAC layer. Here, there is a possibility that a packet which falls into a high priority queue can be dequeued faster than earlier packets in a lower priority

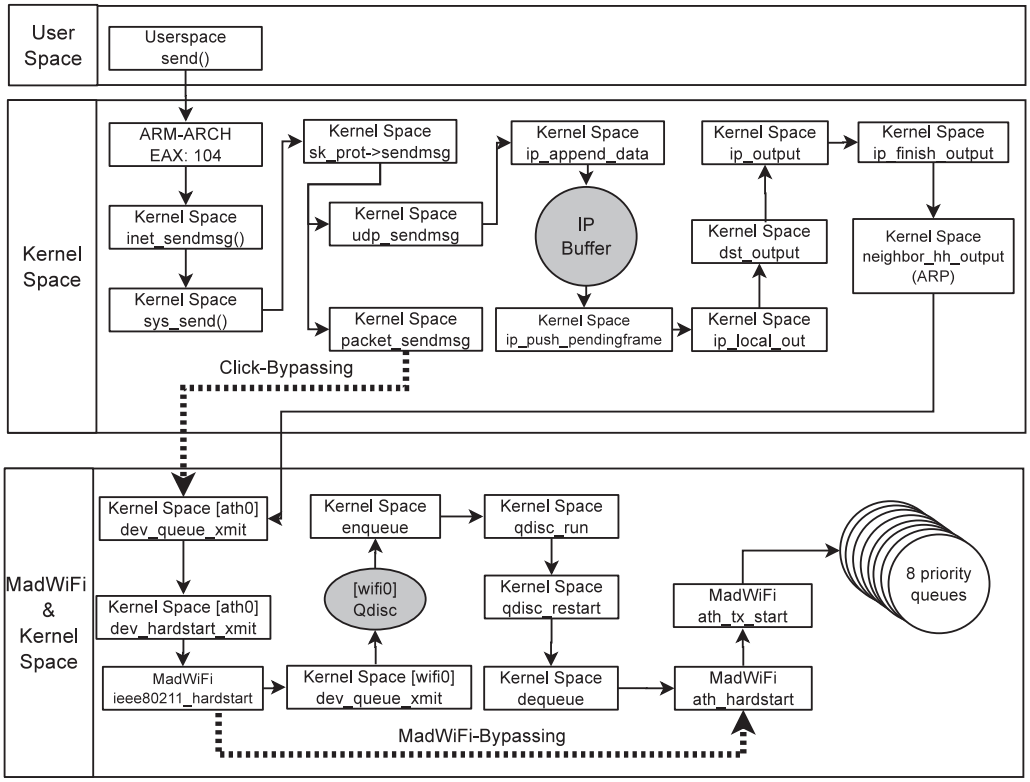


Fig. A.7. Linux networking kernel and MadWiFi function call map where hidden queues are depicted as gray colored circles.

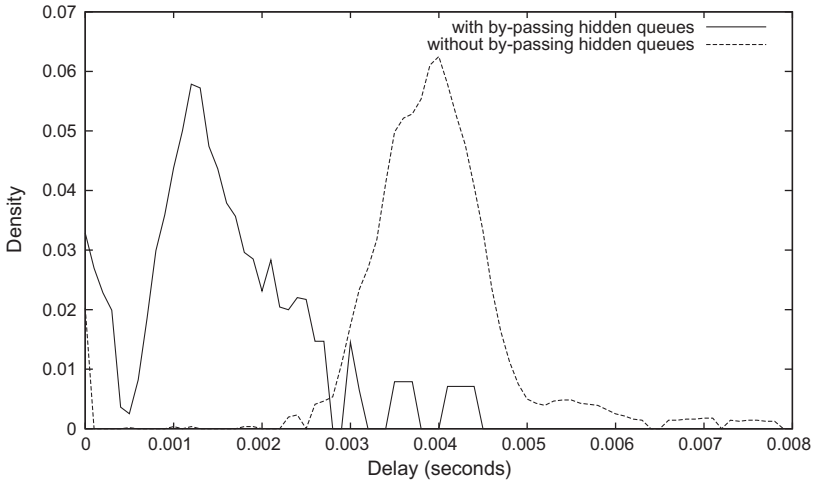
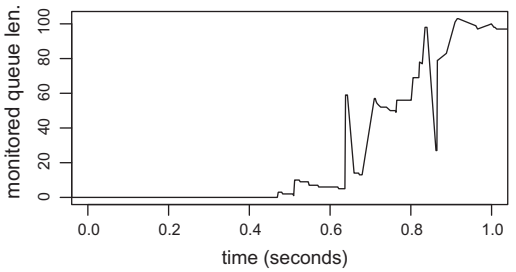


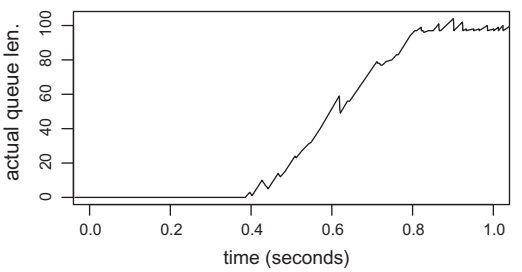
Fig. A.8. The distribution of delay between send function in userspace and ath_hardstart in MadWiFi with and without using by-passing the hidden queues.

queue. This results in out-of-order packets. Since the queue length information is piggybacked on data packets, nodes may not monitor the queue lengths in the right order, which leads to computing backpressure values for the wrong period, and hence, results in untimely scheduling decisions. To filter out-of-order packets, we use sequence numbers. The sequence number is increased each time a packet is transmitted. At the monitoring node, if the sequence number is larger than the previously monitored sequence number, then the sequence number is recorded and the queue length is updated. If the sequence number is less than the previously recorded sequence number, the monitoring packet is discarded. In our implementation, the ID field in the IPv4 header is used as the sequence number.

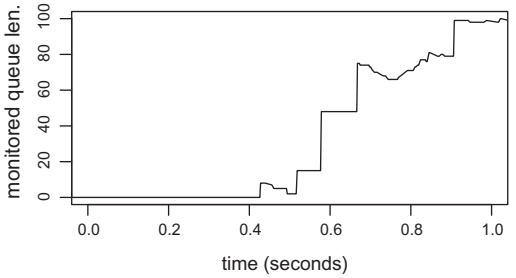
To validate our implementation, we ran an experiment where we generated 2-hop UDP traffic from Node 1 to Node 3 and observe the delay between send function in



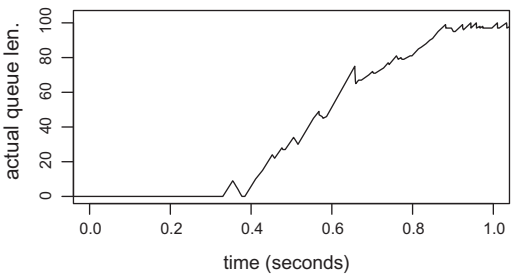
(a) The monitored queue length without sequence numbers (Node 1 monitors Node 2's queue).



(b) The actual queue length of Node 2 without sequence numbers.



(c) The monitored queue length with sequence numbers (Node 1 monitors Node 2's queue).



(d) The actual queue length of Node 2 with sequence numbers.

Fig. A.9. The test scenario shows that using sequence numbers solves the queue length monitoring inversion.

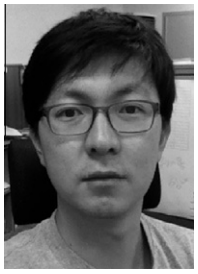
userspace send and ath_hardstart function in MadWiFi (see Fig. A.7).

Fig. A.8 plots the distribution of this delay. We can observe that the significant portion of the packets are less delayed (around 3 ms faster) if we by-pass the hidden queues, which can affect the proper operation of backpressure scheduling [24]. We also plot the queue length of the intermediate node (Node 2) and the monitored queue length by the source node (Node 1) over time in Fig. A.9. When sequence numbers are not used, shown in Fig. A.9a, there exists sharp drops in monitored queue length which is not representative of Node 2's actual queue length (see Fig. A.9b). These sharp drops lead to the wrong backpressure value calculations, and possibly cause sharp fluctuations in the queue evolution. Note that using sequence numbers eliminates such sharp drops (see Fig. A.9c). However, these solutions *do not solve* the root cause of the information inaccuracy in the queue length as also seen from comparing Fig. A.9c and d. It just *mitigates* the issue of out-of-order per-flow queue length monitoring.

References

[1] AirJaldi-Empowering Communities through Wireless Networks, <http://drupal.airjaldi.com/> (accessed 15.03.12).
[2] I.F. Akyildiz, X. Wang, W. Wang, Wireless mesh networks: a survey, Elsevier Computer Networks 47 (4) (2005) 445–487.
[3] L. Tassiulas, A. Ephremides, Stability properties of constrained queuing systems and scheduling for maximum throughput in multihop radio networks, IEEE Transactions on Automatic Control 37 (12) (1992) 1936–1948.
[4] L. Tassiulas, Adaptive back-pressure congestion control based on local information, IEEE Transactions on Automatic Control 40 (2) (1995) 236–250.
[5] N. McKeown, Scheduling algorithms for input-queued cell switches, Ph.D. thesis, University of California, Berkeley, 1995.
[6] A. Dimakis, J. Walrand, Sufficient conditions for stability of longest queue first scheduling: second order properties using fluid limits, Advanced Applied Probability 38 (2) (2006) 505–521.
[7] L. Chen, S.H. Low, M. Chiang, J.C. Doyle, Cross-layer congestion control, routing and scheduling in ad hoc wireless networks, in: Proc. of INFOCOM, 2006.
[8] G. Zussman, A. Brzezinski, E. Modiano, Multihop local pooling for distributed throughput maximization in wireless networks, in: Proc. of IEEE INFOCOM, 2008.
[9] B. Radunovic, C. Gkantsidis, D. Gunawardena, P. Key, Horizon: balancing TCP over multiple paths in wireless mesh network, in: Proc. of ACM MobiCom, 2008.
[10] A. Warrier, S. Janakiraman, S. Ha, I. Rhee, DiffQ: practical differential backlog congestion control for wireless networks, in: Proc. of IEEE INFOCOM, 2009.
[11] A. Aziz, D. Starobinski, P. Thiran, Understanding and tackling the root causes of instability in wireless mesh networks, IEEE/ACM Transactions on Networking 19 (4) (2011) 1178–1193.
[12] J. Ryu, V. Bhargava, N. Paine, S. Shakkottai, Back-pressure routing and rate control for ICNs, in: Proc. of ACM MobiCom, 2010.
[13] R. Laufer, T. Salonidis, H. Lundgren, P.L. Guyadec, XPRESS: a cross-layer backpressure architecture for wireless multi-hop networks, in: Proc. of ACM MobiCom, 2011.
[14] Specific requirements part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications amendment 8: Medium access control (MAC) quality of service enhancements, 2005.

[15] K. Sundaresan, V. Anantharaman, H.-Y. Hsieh, R. Sivakumar, ATP: a reliable transport protocol for ad hoc networks, IEEE Transactions on Mobile Computing 4 (6) (2005) 588–603.
[16] X. Lin, N.B. Shroff, The impact of imperfect scheduling on cross-layer congestion control in wireless networks, IEEE/ACM Transactions on Networking 14 (2) (2006) 302–315.
[17] B. Nardelli, J. Lee, K. Lee, Y. Yi, S. Chong, E.W. Knightly, M. Chiang, Experimental evaluation of optimal CSMA, in: Proc. of INFOCOM, 2011.
[18] J. Liu, Y. Yi, A. Proutiere, M. Chiang, H.V. Poor, Towards utility-optimal random access without message passing, Wiley Journal of Wireless Communications and Mobile Computing 10 (1) (2010) 115–128.
[19] A. Proutiere, Y. Yi, M. Chiang, Throughput of random access without message passing, in: Proc. of IEEE CISS, 2008.
[20] J.A. Cobb, Preserving quality of service guarantees in spite of flow aggregation, IEEE/ACM Transactions on Networking 10 (1) (2002) 43–53.
[21] L. Georgiadis, M.J. Neely, L. Tassiulas, Resource Allocation and Cross-Layer Control in Wireless Networks, Now Publishers Inc., 2006.
[22] M. Garetto, T. Salonidis, E.W. Knightly, Modeling per-flow throughput and capturing starvation in CSMA multi-hop wireless networks, in: Proc. of INFOCOM, 2006.
[23] Multiband Atheros driver for WiFi, <http://madwifi-project.org/> (accessed 15.03.12).
[24] J.Y. Yoo, C. Sengul, R. Merz, J. Kim, Experimental analysis of backpressure scheduling in IEEE 802.11 wireless mesh networks, in: Proc. of ICC, 2011.
[25] J. Lee, S.J. Lee, W. Kim, D. Jo, T. Kwon, RSS-based carrier sensing and interference estimation in 802.11 wireless networks, in: Proc. of IEEE SECON, 2007.
[26] J.Y. Yoo, T. Huehn, J. Kim, Active capture of wireless traces: overcome the lack in protocol analysis, in: Proc. of WinTECH Workshop at MobiCom, 2008.
[27] OMF: Orbit Management Framework, <http://omf.mytestbed.net/> (accessed 15.03.12).
[28] CVX: Matlab software for disciplined convex programming, <http://cvxr.com/cvx/> (accessed 15.03.12).
[29] J. Mo, J. Walrand, Fair end-to-end window-based congestion control, IEEE/ACM Transactions on Networking 8 (5) (2000) 556–567.
[30] Click modular router, <http://read.cs.ucla.edu/click/> (accessed 15.03.12).
[31] GIST wireless mesh network testbed, <http://flownet.nm.gist.ac.kr> (accessed 15.03.12).
[32] K. Xu, M. Gerla, S. Bae, How effective is the IEEE 802.11 RTS/CTS handshake in ad hoc networks? in: Proc. of IEEE GlobeCom, 2002.
[33] C. Benvenuti, Understanding Linux Network Internals, O'Reilly, 2006.
[34] L. Jiang, J. Walrand, Approaching throughput-optimality in distributed CSMA scheduling algorithms with collisions, IEEE/ACM Transactions on Networking 3 (19) (2011) 816–829.
[35] L. Jiang, D. Shah, J. Shin, J. Walrand, Distributed random access algorithm: scheduling and congestion control, IEEE Transactions on Information Theory 12 (56) (2010) 6182–6207.



Jae-Yong Yoo received the B.S. degree in computer science and engineering from Chung-Ang University, Seoul, Korea and the M.S. degree from the Department of Information and Communications at Gwangju Institute of Science and Technology (GIST), Gwangju, Korea in 2004 and 2006, respectively. He is pursuing a Ph.D. degree in the School of Information and Communications at GIST. His research interests include wireless networking and heterogeneous flow coordination.

Cigdem Sengul is a Senior Research Scientist in Deutsche-Telekom Labs at TU-Berlin. She got her PhD and MS degrees in Computer Science in University of Illinois at Urbana-Champaign. Her PhD and MS research were funded by Vodafone, Fulbright, UIUC Computer Science Department fellowships. She is currently involved in the design and deployment of a large-scale outdoor heterogeneous wireless network in TU-Berlin. Her research interests are configurable wireless testbeds, wireless network measurements, MAC protocols for multi-hop wireless, and energy-efficient wireless.

Ruben Merz is a Senior Research Scientist at Deutsche Telekom Laboratories in Berlin, Germany. He earned his MSc and PhD degrees in communication systems from EPFL, School of Computer and Communication Systems, in 2003 and 2008, respectively. His research interests focus on the design, modeling, and performance evaluation of wireless communication systems and networks. He is a recipient of a best student paper Award at the 2005 IEEE International Conference on Ultra-Wideband (ICU). He is a member of the IEEE and ACM.

JongWon Kim received the B.S., M.S., and Ph.D. degrees from Seoul National University, Seoul, Korea, in 1987, 1989 and 1994, respectively, all in control and instrumentation engineering. During 1994–2001, he was an Assistant Professor of Electronics Engineering Department at the KongJu National University, KongJu, Korea, and then a Research Assistant Professor of Electrical Engineering – Systems Department at the University of Southern California, Los Angeles, USA. From September 2001, he has joined the Department of Information & Communications (DIC), Gwangju Institute of Science and Technology (GIST), Gwangju, Korea, where he is currently working as a full Professor. Under the slogan of "Dynamic composition of immersive media-centric services over the wired/wireless IP convergence networks," he is focusing on networked media systems and services. He is the senior member of IEEE, and the members of ACM, SPIE, KICS, IEEK, KIISE (former KISS), KIPS, and HCI-Korea. He has been serving as the editorial board member of Elsevier JVIS (Journal of Visual Communication and Image Representation), several KIISE, and KIPS.