Wireless Pers Commun (2016) 90:301–323
DOI 10.1007/s11277-016-3347-0

Distributed and Energy Efficient Scheduling for IEEE802.11s Wireless EDCA Networks

Mojtaba MalekpourShahraki[1] · Hadi Barghi[1] · Seyed Vahid Azhari[1] · Saeed Asaiyan[1]

Published online: 18 July 2016

Abstract We consider scheduling over IEEE802.11s wireless mesh networks running the standard Hybrid Wireless Mesh Protocol (HWMP) which forms a tree rooted at the mesh gateway. We propose Time-Split, a very simple and distributed scheduling algorithm compatible with the conventional EDCA mode of access. Time-Split is equipped with a simple mechanism to remove off-path interference without knowledge of the interference graph. Furthermore, Time-Split limits time-line fragmentation by creating contiguous inactivity periods within a beacon interval that can be used for deep power saving. We compare the performance of Time-Split with a greedy scheduler which performs optimally over the tree formed by HWMP for traffic routed to/from the gateway. We show that Time-Split can achieve more than 90 % optimal throughput at a packet delivery ratio of almost 100 %. More importantly, we propose split-depth as a means to strike a balance between interference elimination and spectrum sharing. In addition, we introduce a multi-channel version of Time-Split improving connection acceptance ratio by about 15 % using a single radio interface. Furthermore, due to its contiguous deep sleep intervals, Time-Split can potentially provide up to 25 and 50 % energy savings compared to conventional EDCA with sleep capability and a greedy sleep scheduler, respectively.

Keywords CSMA scheduling · Sleep scheduling · Wireless EDCA networks · IEEE802.11s · Energy efficiency

✉ Mojtaba MalekpourShahraki
m_malekpour@comp.iust.ac.ir

Hadi Barghi
hadi_barghi@iust.ac.ir

Seyed Vahid Azhari
azharivs@iust.ac.ir

Saeed Asaiyan
s_asaiyan@comp.iust.ac.ir

[1] School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran

1 Introduction

Wireless mesh networks (WMN) are multi-hop networks which use wireless links for relaying traffic between nodes. They tend to be quasi-stationary and generally transport large traffic volume comparable to WiFi hotspots. WMNs are used for many applications such as neighborhood networks, tactical networks, urban monitoring and surveillance systems and search and rescue applications [1]. Such applications are either temporary in nature or have small profit margin, requiring cost efficient easy to setup infrastructure to minimize deployment CAPEX.

As a result, conventional off the shelf solutions based on IEEE 802.11 are preferred as opposed to more efficient specialized radio technologies based on TDMA. In fact, IEEE 802.11s, which is a multi-hop extension to the original IEEE 802.11 standard is now one of the major technologies for building WMNs [2].

However, it is well known that CSMA based protocols such as IEEE 802.11 have poor performance in multi-hop applications due to interference from many contending relay links, poor channel allocation and lack of a centralized channel access mechanism. TDMA access methods, on the other hand, provide the best throughput but are more complex and costly in terms of equipment.

Link scheduling over WMNs is a challenging problem in general. When a connection is to be established along a certain path, all contending links both lying on the connection path and those which are off-path should be assigned non-overlapping activity periods. Dealing with on-path interference is relatively simple as it can be resolved while a connection request is moving along the path without requiring additional signaling. Resolving off-path interference, on the other hand, is a more challenging problem, which involves additional signaling with other nodes in the network and requires some knowledge of network topology at least as far as two to three hops depending on the interference range. Moreover, any local scheduling decision can potentially trigger a chain of signaling that spreads through the network. This signaling is to inform relevant nodes of the medium availability times for communicating to a particular node.

Furthermore, energy efficient communications is required by applications lacking reliable power source for mesh nodes, such as tactical networks and in some cases surveillance and monitoring systems. Radio cards of such WMNs should be put into sleep mode during their idle time to save energy. It is generally preferred that sleep periods be contiguous and large enough allowing the node itself to go into deep sleep as well as the radio card.

In this paper we propose a scheduling algorithm that addresses both off-path interference mitigation and energy efficiency by providing contiguous deep slep intervals. Our link scheduling algorithm, *"Time-Split"*, is compatible with the widely used EDCA mode of access and operates over IEEE 802.11s WMNs employing the default tree based routing protocol, HWMP [2]. Time-Split eliminates both on/off-path interference using a very simple and efficient local heuristic. We compare the performance of our scheduling algorithm with a greedy scheduler and the default EDCA mode of access. Our main contributions are as follows:

- Proposing Time-Split, a very simple distributed and localized scheduling algorithm for IEEE 802.11s WMNs operating over the default HWMP routing tree. In Time-Split, scheduling decisions are entirely local where a node only considers its children. This reduces scheduling complexity to $o(d)$, where d is the degree of HWMP tree. Moreover, Time-Split requires no additional signaling to eliminate off-path interference.

- Providing substantial energy savings by limiting time-line fragmentation and allocating a contiguous sleep interval to mesh nodes. This allows nodes to go into deep sleep and also reduces the energy overhead of frequent transitions between sleep and active states.
- Decoupling the mechanisms for dealing with on-path and off-path interference. Moreover, a split-depth parameter is presented that can be used to provide a tradeoff between interference mitigation and resource sharing.
- Integrating an implicit call admission control (CAC) scheme with Time-Split which adapts to network topology and traffic demand.

We evaluate the performance of Time-Split in terms of connection acceptance ratio, throughput, packet delivery ratio and energy savings. We also develop a greedy scheduling algorithm which achieves optimal result for the tree topology of HWMP and compare its performance to that of Time-Split and default EDCA used by IEEE 802.11s. We have used NS3 simulator to obtain our results.

The rest of this paper is organized as follows. Related work is briefly summarized in Sect. 2. Section 3 describes notation and system model. Section 4 presents our proposed algorithms, i.e., Greedy and Time-Split schedulers. Detailed simulation results are presented in Sect. 5, and finally the paper is concluded in Sect. 6.

2 Related Work

A plethora of work exists on scheduling over wireless multi-hop networks, which can be categorized based on using CSMA or TDMA, and the particular traffic pattern being peer to peer or destined to a single gateway [1]. In particular, we are interested in those work which use CSMA based MAC and a tree-based routing approach to a single gateway similar to HWMP which is the default routing protocol proposed by the IEEE802.11s standard.

The majority of research that considers routing to a single gateway is conducted over wireless sensor networks (WSN). Some of these consider the problem of tree formation [3], while others address different problems such as data aggregation [4], which are out of the scope of our current work. Some other papers only consider broadcast traffic. For example, [5] constructs a joint power control, rate adaptation and scheduling algorithm which assigns a transmission power to each node. This will minimize the delay only for broadcast packets using an estimation of remaining hop delay. We, however, consider general applications generating unicast packets and do not use any form of data aggregation. In addision, we assume that tree formation is performed by an independent entity such as the HWMP protocol. Moreover, unlike sensor networks with sporadic traffic profile, we consider wireless mesh networks transporting broadband traffic with much larger volume.

Incel et al. [6] proposes TDMA scheduling for converge-cast traffic in WSNs with an attempt to minimize delay. They introduce single and multi-channel versions of their scheduler. Their single channel scheduler mitigates interference using the concept of a conflict graph requiring all off-path interfering links to be known.Their multi-channel approach, however, assumes that interference is mitigated using channel assignment and power control. Our Time-Split scheduler, on the other hand, is able to eliminate off-path interference without knowledge of a conflict graph using a simple heuristic. In addition, we also propose a multi-channel version for Time-Split that further improves achievable throughput.

A tree based scheduler for CSMA is proposed in [7] via a distributed local signaling protocol that is executed between a parent and its children. Power control is also leveraged to reduce interference among contending links in the tree. However, the scheduling algorithm picks slots randomly whereas in our Time-Split scheduler we make the schedule as compact as possible to allow for a large contiguous deep sleep period. Moreover, Time-Split takes advantage of the tree structure to implicitly eliminate off-path interference, whereas, [7] merely leverages the parent-child relationship as a coordination and synchronization means but still has to explicitly deal with both on-path and off-path interference in the tree.

In [8, 9] the problem of perfect periodic scheduling of wireless links over a tree is considered. The authors provide a polynomial algorithm for the special case of a binary tree. Their approach, however, considers a perfect TDMA system and uses a centralized algorithm to compute the schedule. Shrivastava and Pokle [10] proposes a node grouping scheme to manage interference and performs scheduling within each group using consecutive time-slots to reduce energy consumption and allow contiguous sleep and activity periods. However, they adopt a TDMA MAC and their approach is provided for WSNs having low traffic volume. Sengaliappan and Marimuthu [11] also proposes a TDMA scheduling algorithm for converge-cast traffic in WSNs. In addition, power control is used to further reduce the effect of interference. Power control, however, is not a viable option in 802.11 based networks as they lack fine grain control over link transmission power. DRAND is a randomized TDMA scheduling approach based on a state machine with four different states proposed in [12]. The state machine helps every node to choose its own slot without any strict synchronization. Choosing a slot, however, is performed via multiple packet forwarding resulting in numerous packet transmissions.

Several other methods and algorithms have also been proposed to resolve contention in wireless multi-hop networks. Most methods impose considerable amount of change to the standards. The work in [13], proposes a method based on compressive sensing to eliminate interference in a sparse wireless sensor network, which is only suitable for congested networks. PACE proposed in [14] is a method based on WiFix which is a tree based routing approach on top of IEEE802.11 and controls transmissions using an upper layer module. Synchronization and fairness are considered in PACE but spatial reuse is not addressed resulting in loss of capacity. A novel odd-even tagging approach is proposed in [15]. The tagging mechanism separates simultaneous transmissions while routing is used to further eliminate interference. This approach, however, is based on a TDMA MAC layer which also uses subchannelization but furthermore requires knowledge of interfering links unlike our proposed solution.

3 System Model

We consider a wireless mesh network running Hybrid Wireless EDCA Protocol (HWMP) consisting of a mesh gateway as root of the HWMP tree which operates in proactive registered mode. To provide a loose form of synchronization as required by 802.11 nodes, a beacon and management frames (BM) interval is put aside for all participating mesh nodes during which beacons and any management frames are sent and received. Normal 802.11s beacon offsetting mechanism is used to adjust beacon times when beacon collisions occur.

We perform scheduling by restricting each node's access mechanism only to be active during specific intervals based on their role in the tree formed by HWMP. This is achieved by forcing a node into sleep during other times. Hence, we define the following type of intervals (see also Table 1):

- *S* Sleep interval, which can be of type S_f during which a node is forced to sleep for preventing interference. Otherwise, the node is allowed to use its sleep period for communication according to our scheduling rules.
- *C* Child interval, during which a node will only communicate with its children in the tree. All other interfering nodes will be put into S_f mode.
- *P* Parent interval, during which a node will only communicate with its parent in the tree. All other interfering nodes will be put into S_f mode.

Figure 1 illustrates a simple interval configuration for link AB realized by overlapping C and P intervals of nodes A and B, respectively. Note that A's parent and B's children will have to be put into forced sleep S_f to eliminate interference. A scheduling algorithm is responsible for allocating different intervals from the set $\{S,S_f,C,P\}$ to each node's timeline. By eliminating interference and therefore collisions, the length of C and P intervals needed to serve a certain connection can be calculated. For instance, a connection k with packet arrival rate λ_k and packet transmission time $T_{pkt,k}$ requires $\delta_k = \lambda_k T_{pkt,k} BI$ as P and C intervals for each beacon interval of duration BI seconds.

In order to provide acceptable delay performance, we employ a cyclic scheduling approach where access intervals are repeated periodically according to the connection delay requirement [16]. Elastic best effort traffic needs to be scheduled only once during each beacon interval, however. It should be noted that our scheduling algorithms can be implemented without breaking the IEEE802.11 standard. In fact, the access restrictions are applied using the NAV mechanism along with APSD power saving features that mask and unmask channel access for a set of nodes [17]. Furthermore, we adopt an implicit connection admission control (CAC) scheme that rejects a connection request at setup and scheduling time if a feasible schedule cannot be found.

4 Proposed Algorithms

4.1 Greedy Algorithm

Connections in the tree formed by HWMP are from some source or destination node w to the root, along a pre-selected path Γ_w, represented as an ordered list of bidirectional links. For a connection between root R and some node w over a tree $T(V, E)$, the greedy scheduler is described by Algorithm 1.

Table 1 Different types of intervals in the proposed algorithms

	Specification	Purpose
P	Communicate with parent	Mask own Children
C	Communicate with child	Mask parent
S	Sleep	Usable for scheduling/sleeping
S_f	Forced sleep	Eliminate interference

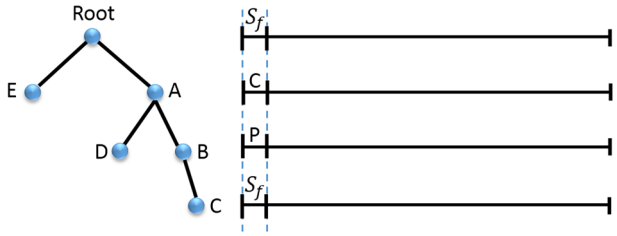

Fig. 1 Interval configuration for realizing link AB

Algorithm 1: Centralized Greedy Scheduling Algorithm

Data: $T(V, E)$, δ_k, w
Result: A feasible schedule or reject connection
Γ_w = Path from root R to source/destination w
$I(u, v)$ = Set of nodes interfering with bidirectional link (u, v)
for $L \in \Gamma_w$ **do**
  u= L.parent
  v= L.child
  $S(u, v) = u.S \cap v.S$ #Find intersection of sleep periods
  τ_L = first interval in $S(u, v)$ larger than δ_k
  if τ_L *exists* **then**
    $u.\tau_L$=C
    $v.\tau_L$=P
    $\forall\ x \in I(u, v)$: $x.\tau_L = S_f$ # Force sleep interferers
  else
    Reject Connection

The greedy scheduling algorithm is centrally executed at the root. It requires knowledge of the tree topology, but more importantly a complete view of the interference graph, which is represented by the set of nodes *I*(*u*, *v*) that should not be active at the same time with a given bidirectional link (u,v). This information is obtained by having nodes enumerate the ID of other nodes within their range in their beacon and in route maintenance messages that are sent to the root. The greedy scheduler also takes as input the amount of time required to service a new connection *k* during each service interval, (i.e., δ_k) and the source or destination node *w* that terminates the connection.

The scheduler starts by traversing links from source to destination allocating C and P intervals to the child-parent pairs that make up connection links along the path. These C and P intervals are carved out of mutual available sleep periods (denoted by S) of child-parent pairs, large enough to accommodate the connection traffic demand δ_k in a given service period. Moreover, to minimize delay only the first occurrence of such an overlap is taken. In addition, forced sleep (S_f) intervals are assigned to all interfering nodes eliminating both on-path and off-path interference. If there is no space for fitting C or P intervals then the connection is rejected.

The time complexity of the greedy scheduling algorithm for every new connection is $O(D|V|^2)$, where D is the maximum tree depth and $|V|$ is the number of nodes. Note that $O(|V|^2)$ is the time complexity of finding the sleep intersection for a child-parent pair and selecting the first one with appropriate size. Here we are assuming that all connections terminating or originating at a certain node *w* will be fit into a contiguous C or P interval so that there will be at most $|V|$ non-contiguous intervals to select from. This is, however, a

huge overestimate for a typical tree. We assume rescheduling is performed periodically, for example, by integrating it into the regular proactive route registration messages of HWMP. Moreover, performing periodic rescheduling prevents time-line fragmentation and improves system capacity. It should be noted however that, our Time-Split scheduling algorithm provides even more contiguous sleep intervals without requiring rescheduling.

Figure 2 is a simple example of how greedy scheduling works. Here, a connection is to be established from D to R (root), passing through $\Gamma_D = \{D, B, A, R\}$. Scheduling is started by assigning a P interval to D and a corresponding C interval to its parent B. The greedy scheduler allocates this from the beginning of the S interval for both B and D, forming the bidirectional link BD. Next, all interfering nodes with D and B, namely I(D,B)={R,A,F,G} receive a forced sleep at exactly the same place as C and P. The scheduler then continues the same process for links BA and AR. Note that in this schedule, node E remains free to communicate during its first S interval.

In a tree topology, the root is the anchor point for all traffic in the network. However, the bottleneck nodes are those in the first level which require both C and P intervals. Any scheduling algorithm must efficiently allocate C/P intervals to the first level to maximize throughput. This observation justifies the optimization problem formulated in Eqs. (1–8) for assigning time intervals to a set of connections in a tree such that the remaining available sleep time of the bottleneck node is maximized.

It is assumed that time is split into N time-slots large enough to hold the smallest traffic demand on the network. Traffic generated by some source node w is denoted by its equivalent amount of time slot λ_w. Each time-slot k of node v is marked as S/P/C by setting the appropriate function S / P / $C(v, k)$ to one, otherwise a forced sleep time-slot is implied (Eq. 5). Child-parent time-slots are aligned through constraint (2), while constraint (3) enforces flow feasibility of the solution. Also, interference constraints are captured by (4). Note that this is a non-linear constraint but which can be linearized using proper selection of variables. However, we do not intend to provide a linear programming formulation as this is only used to compare our proposed algorithms with the optimal solution.

$$Maxz \tag{1}$$

$$\begin{aligned} &st. \\ &\quad P(u,k) \geq C(v,k), \forall uv \in T(V) \end{aligned} \tag{2}$$

$$\sum_{k=1}^{N} C(v,k) \geq \sum_{w \in SubTree(v)} \lambda_w, \forall v \in V \tag{3}$$

$$\begin{aligned} &P(w,k) + S(w,k) + C(w,k) + P(u,k)C(v,k) \\ &\quad \leq 1, \forall uv \in T(V), w \in I(uv) \end{aligned} \tag{4}$$

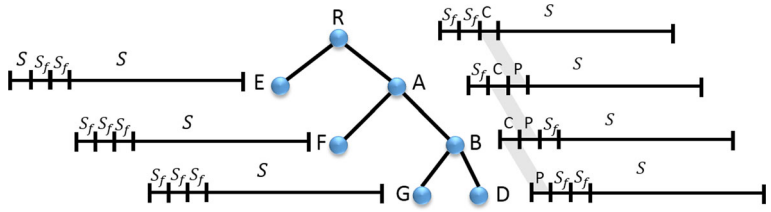

Fig. 2 Sample greedy interval allocation

$$S(v,k) + C(v,k) + P(v,k) \leq 1, \forall v \in V, \quad k = 1..N \tag{5}$$

$$S_v = \sum_{k=1}^{N} S(v,k), \forall v \in V \tag{6}$$

$$z \leq S_v, \forall v \in V \tag{7}$$

$$S(v,k), C(v,k), P(v,k) \in \{0,1\}, \forall v \in V, \quad k = 1..N \tag{8}$$

4.2 Time-Split Scheduling Algorithm

The greedy scheduling algorithm has a major shortcoming; it is centralized and requires knowledge of time-line schedule as well as interference graph for all tree nodes to be available at the root. We would like to have a distributed algorithm that limits both knowledge of schedule and interference graph to be local to each node.

Recall that a scheduler has to consider both on/off-path interference. While on-path interference is easily prevented during the connection setup process, off-path interference is much harder to handle as it involves nodes which are not participating in the connection setup process. Our previous greedy algorithm handles these off-path interferers using complete knowledge of the interference graph, which is passed up to it in management and routing control frames.

Let us consider two interfering nodes u and v in the tree formed by HWMP. These nodes, either share some ancestor w or one is in fact the ancestor of the other. In the latter case, any interference between u and v is prevented using appropriate on-path interference prevention methods enforced as part of the distributed connection establishment procedure along the tree. However, when interfering nodes u and v share some ancestor w then their interference must be handled only through an off-path mechanism.

Observing the structure of a tree, we can prevent off-path interference by simply allowing different sub-trees to access the channel during mutually exclusive intervals. Let us assume that w has two children q and r. We split the available S (sleep) time-line of w into two non-overlapping sub-intervals, namely, τ_1 and τ_2 and force each of the children to be active exclusively during one sub-interval. This is easily done by setting $\tau_1^r = S_f, \tau_2^r = S$ for r and $\tau_1^q = S, \tau_2^q = S_f$ for q.

Figure 3 illustrates a more general case of the proposed *"Time-Splitting"* technique for a node A having N children $B_1 ... B_N$. Here, the S time-line of the parent A is subdivided into N sub-intervals $\tau_1 ... \tau_N$, originally of type S. All communication between A and its ith child B_i will be confined to τ_i by setting,

$$\forall j \neq i, \tau_i^{B_j} = S_{f,off}. \tag{9}$$

For better illustration, we have distinguished between forced sleep due to on-path interference prevention and that due to off-path denoting them by $S_{f,on}$ and $S_{f,off}$, respectively. It follows that any communication happening in each sub-tree T_{B_i} rooted at B_i will take place during mutually exclusive intervals τ_i, hence preventing off-path interference without requiring any knowledge of the interference graph. Therefore, the Time-

Split algorithm can make scheduling decisions only using local information in a distributed way. A pseudo code for this scheduler is provided in Algorithm 2.

Algorithm 2: Decentralized Time-Split Scheduling Algorithm - **@ Receiving New C-REQ Message**

```
   Result: Schedule local C/P/S intervals or reject connection
 1 C-REQ=(v, C/P, C/P, ...) # List of all connection requests from parent/child v
 2 v = C-REQ sender
 3 u = current node
 4 Update θ according to specific splitting schema (see Sections 4.2.1,4.2.2)
 5 Send θ in next Beacon
 6 MyC-REQ ← (u, ) #Create empty connection request
 7 for Interval ∈ C-REQ do
 8     if Interval ⊈ u.S then
 9         Reject connection request
10     if C-REQ received from parent then
11         if Interval = C then
12             u.τ ← (u.S ∩ Interval) #Mark overlapping P interval on current node
13             u.τ = P
14             Append u.τ to MyC-REQ
15             if current node not source or destination then
16                 q ← child of u which is next hop for C-REQ
17                 τ' = first large enough interval in q.S
18                 if τ' exists then
19                     u.τ' = C # Mark new interval for child link
20                     Append u.τ' to MyC-REQ
21                 else
22                     Reject Connection
23         else if Interval = P then
24             u.τ ← (u.S ∩ Interval) #Eliminate on-path interference on current node
25             u.τ = S_f
26     else if C-REQ received from child then
27         if Interval = C then
28             u.τ ← (u.S ∩ Interval) #Eliminate on-path interference on current node
29             u.τ = S_f
30         else if Interval = P then
31             u.τ ← (u.S ∩ Interval) #Mark overlapping C interval on current node
32             u.τ = C
33             Append u.τ to MyC-REQ
34             if current node not source or destination then
35                 q ← parent of u which is next hop for C-REQ
36                 τ' = first large enough interval in q.S
37                 if τ' exists then
38                     u.τ' = P # Mark new interval for parent link
39                     Append u.τ' to MyC-REQ
40                 else
41                     Reject Connection
42 Send MyC-REQ
```

A few snapshots of the operation of the Time-Split algorithm are illustrated in Figure 4, where a connection is to be establish from R to D. The process starts at the source node (i.e., R), by picking a suitable C-interval from the interval split that belongs to A

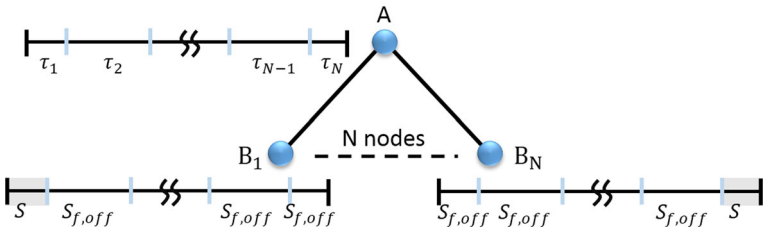

Fig. 3 Time-split algorithm interval allocation concept

(Figure 4a). Node R will then form a connection request (C-REQ) indicating the type of requested interval (C) and send it to its next hop child A (Figure 4b). When A receives a C-REQ it executes Algorithm 2. This algorithm iteratively handles all interval requests within a C-REQ through a loop (line 7), which in this case is a single request for a C-interval sent by A's parent R. Node A will first secure the requested C-interval and if not possible the C-REQ will be rejected and the connection will fail (lines 8–9). Since the C-REQ is received from A's parent (line 10), A will first pair the C-interval with an overlapping P-interval (lines 11–14). If node A is not the final destination (line 15) a new C interval is needed to communicate with A's next hop child, B. Lines 16–22 are responsible for allocating this C-interval on the current node, followed by requesting it from the next hop child node B, by issuing a new C-REQ message denoted by MyC-REQ which is then sent by A to B.

Figure 4c shows the next step where A forwards MyC-REQ to its next hop child B. Node B will match the C-interval request within the C-REQ with both a P-interval (lines 10–14) and a new C-interval (lines 15–22). In addition, node B will use the P-interval in this message to label a matching forced sleep interval on its own time-line for preventing on-path interference (lines 23–25). Note that unlike the greedy algorithm, Time-Split requires no treatment of off-path interference as that is automatically handled by the hierarchy of intervals among nodes. This process continues until the C-REQ reaches the destination node D (Figure 4d) or the connection is rejected at some point along the path. Note that lines 26–41 are executed for the case where the connection source is not the root.

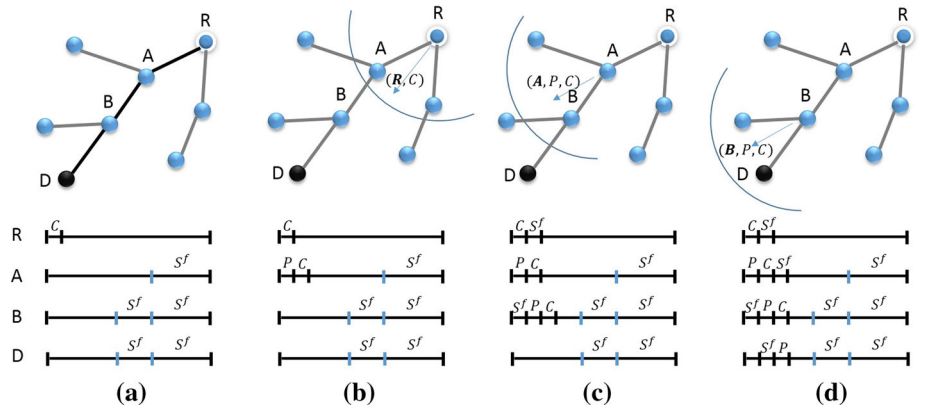

Fig. 4 Snapshots of time-split algorithm

It should be noted that the split assigned to different children τ_i, determines system throughput and fairness of the CAC scheme across the network. We next investigate two alternative splitting approaches, namely, static and dynamic which are based on tree topology and traffic demand, respectively.

4.2.1 Static Sub-tree Splitting

This method allocates equal Time-Splits (S-intervals) to all siblings of a certain parent. Static Time-Splitting is very simple and requires no rescheduling. However, it suffers from unfairness toward connections deeper down the tree and those which pass through congested parts of the tree. Figure 5 illustrates an example of static Time-Split for a binary tree, for which the available S time-line of a parent is split into half and assigned to its two children. Clearly, node B is receiving more than a fair share of bandwidth as it has no other nodes to serve while nodes A,C and D have to share the same amount of bandwidth.

4.2.2 Dynamic Sub-tree Splitting

Our dynamic splitting schema will adjust the Time-Split allocated to each child according to its traffic. This is to better accommodate connections deeper down the tree and traffic hotspots. In particular, the splitting fraction for a child u of node v will be,

$$\theta_u = \frac{\delta_u}{\sum_{\forall j \in Child(v)} \delta_j}, \tag{10}$$

where δ_u is the amount of communication time required to service traffic to/from node u. It follows that the amount of S-interval allocated to each child u will be $\tau_u = \theta_u S^v$, where S^v is the available S-interval of node v.

4.3 Channel Assignment Algorithm

Most applications of wireless mesh networks benefit from limited number of hops [18]. However, as the size of the network increases the reduced capacity resulting from single channel operation will eventually render the network useless. This is true for any type of scheduling approach and routing protocol. In particular, Figure 6 shows how the Time-Split scheduling scheme may fail to accept connections to/from nodes deep down the tree. In this example, a binary tree is formed where every node divides time-line into two intervals. Eventually at some depth (e.g., four) the available sleep interval assigned to some node G will be too small to accommodate a new connection.

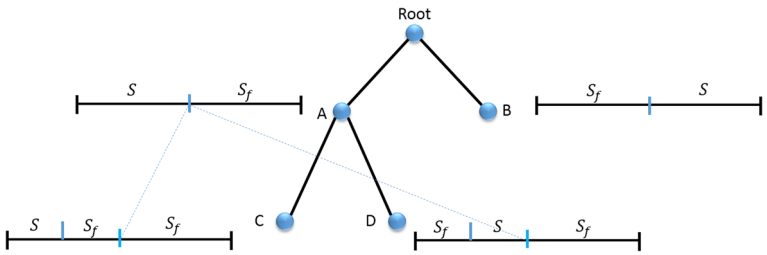

Fig. 5 Sample static sub-tree splitting

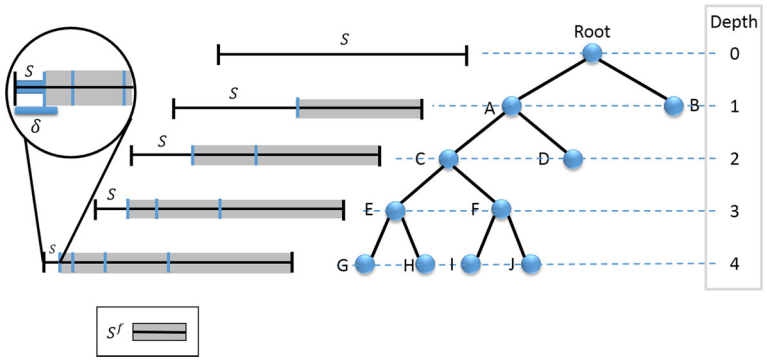

Fig. 6 Example of deep tree with too small time-splits

Although proper tree formation can, to some extent, prevent the aforementioned problem, it eventually happens if the network becomes large enough. For such cases, we propose a multi-channel version of the Time-Split scheduling approach, which assigns non overlapping channels to certain sub-trees in the network. It should be noted that, our multi-channel scheduling approach continues to operate on a single radio card. As a result, it can only improve capacity by interference mitigation as opposed to providing parallel links.

Algorithm 3: Distributed Channel Assignment Algorithm For Time-Split

Data: l: depth of truncated BFS, L_{ch}: channel list
Result: A channel is assigned to the current node
1 execute distributed truncated BFS at current node
2 **if** *current node determined as a pseudo-leaf* **then**
3 pick new channel from L_{ch}
4 set current node as root on new sub-tree
5 initiate distributed truncated BFS on new sub-tree
6 **else**
7 set channel to that advertised for the current sub-tree

Algorithm 3 provides a high level description of our proposed distributed channel assignment algorithm. It is simply based on a generic distributed breadth-first tree traversal algorithm that is truncated to a maximum depth of l. The channel assignment algorithm can be triggered upon various conditions, such as tree updates, connection rejects, or any other suitable triggering criteria. Each node executing the algorithm may be labeled as a pseudo-leaf. Pseudo-leaves are nodes which lie at the bottom of the sub-tree formed by the truncated BFS of depth l. These nodes will then select a new channel from the list of available channels and initiate a new truncated BFS over their decendents advertising the newly selected channel (lines 2–5). All other nodes will simply adopt the advertised channel by their sub-tree root (lines 6–7).

The result of Algorithm 3 is a number of sub-trees with a depth of at most l which are mutually interference free. More importantly, each sub-tree will be using its own time-line. Figure 7 illustrates channel assignment using a depth of $l = 2$ for a binary tree topology. The sub-trees formed by nodes $\{Root, A, B, C, D\}$ and $\{C, E, F, G, H, I, J\}$, are assigned to

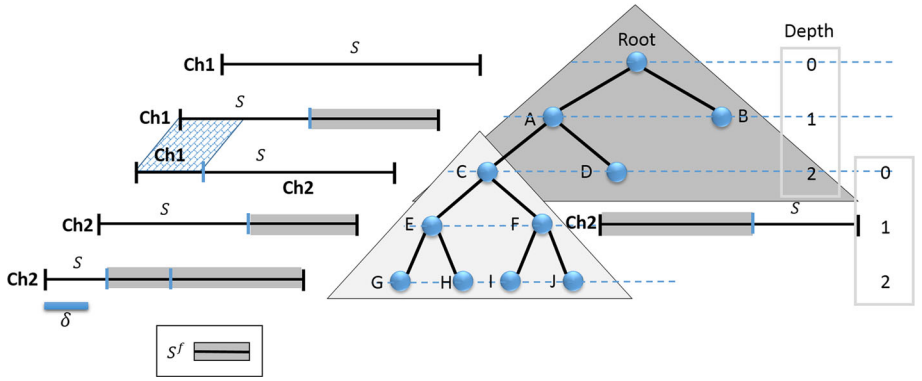

Fig. 7 Sample channel assignment for time-split scheduler

channels one and two, respectively. Node C which is a pseudo-leaf on the first sub-tree and the root of the second sub-tree must operate on both channels. It will use Ch1 for communicating with its parent during its S-interval and switches to Ch2 to serve traffic to/from its sub-tree during all other force sleep periods S_f, which would have been unavailable if a single channel were used. As a result, node G which is deep down the tree will now have larger time-line available for scheduling its connections. We will show the throughput gain of the multi-channel version of Time-Split scheduling in Sect. 5.3.

5 Performance Evaluation

We use the NS3 simulator to evaluate our scheduling algorithms. We have chosen two different topologies, a 4×4 grid and 80 instances of a 16-node random topology with simulation parameters selected as in Table 2. Nodes are uniformly distributed over a square of 300 m $\times$ 300 m for the random topology. The gateway node is randomly picked for random topologies, while for the grid case it is always at the top left corner of the grid, resulting in a worst case deep tree for our scheduling algorithms. In addition, we have included 95 % confidence intervals over all our simulation results as error bars. It should also be noted that in all simulation scenarios of Time-Split, the entire network was re-scheduled in at most 11.468 msec.

Table 2 Simulation parameters

Simulation time (s)	600
Node communication range (m)	102
Packet generation rate (pkt/s)	50
Packet size (bits)	1280
Beacon interval (ms)	500 and 100
Power consumption during transmit/receive	5.6 W
Power consumption during sleep	3.5 W
Power consumption during deep sleep	0.7 W
Deep sleep duration threshold (τ_0)	40 ms

5.1 Throughput, Delay and Admission Control Performance

In this section we evaluate the performance of single-channel Time-Split scheduling algorithm when call admission control (CAC) is present. Figure 8a, b compare throughput and packet delivery ratio (PDR) of Time-Split for the 4x4 grid to that of simple 802.11s EDCA access mechanism labeled as "EDCA". The curves labeled with "Split-Static" and "Split-Dynamic" correspond to the Time-Split scheduler when static topology based and dynamic traffic based splitting is employed, respectively (see Sects. 4.2.1,4.2.2). Figure 8e, f show the same set of results for the random graphs generated over 16 nodes. The results for random graphs are averaged among different graphs but the narrow 95 % confidence intervals suggest small variation. To make our comparisons fair we have applied the exact same set of connections to all different schedulers.

These results show the performance improvement of increasing the split depth for Time-Split scheduler. A splitting depth of l means that time-line of nodes beyond level l of the tree will not be further divided and they will share whatever time-line is allocated to them with their siblings. As a result, off-path interference is not completely eliminated but it is possible that more connections are admitted to the network due to time-line sharing. A depth of zero corresponds to no split at all, leaving off-path interference untreated. This is obviously the cause of worst case performance even lower than that of conventional EDCA access scheme as shown in Figure 8a, b. Interestingly, by allowing only a single level of split, the throughput almost doubles to 800Kbps, slightly better than conventional EDCA access. Applying Time-Splitting deeper down the tree improves throughput and PDR at a declining rate, however. In particular, throughput and PDR start to saturate at a splitting depth of two and reach their maximum value of 1Mbps when the depth becomes three. Surprisingly, the saturation happens at a much smaller depth of only one for the random topology. This is because our grid topology results in a tree of largest depth compared to those created over the random topologies.

Moreover, as suggested by Figure 8a, b Time-Splitting with a depth of only three is able to achieve the same throughput and PDR as our greedy scheduler for the grid topology. This further reduces to a split depth of one for random topologies, as suggested by Figure 8e, f. It is important to note that for all our simulations, the greedy scheduler achieved exactly the same performance as the optimal solution to the optimization problem in (1–8). This is attributed to the fact that nodes on the first level of the tree are the bottleneck as all network traffic passes through them and they both send and receive traffic. As a result, any scheduling algorithm that manages to optimally assign time intervals to these first level nodes will be able to solve the problem optimally. It turns out that the greedy choice of time intervals is an optimal choice in this case.

Surprisingly, static Time-Splitting performs slightly better than dynamic Time-Splitting in terms of throughput as observed in Figure 8a. This is because static Time-Splitting favors connections closer to the root which consume less total bandwidth from the network, whereas, dynamic Time-Splitting adjusts S and S_f intervals based on traffic demand so that connections deep down the tree are given the same chance as connections closer to the root. We believe dynamic Time-Splitting is a more suitable option since it equally considers connections which are distributed along the tree.

Figure 8d compares the number of connections admitted for different split depths and different topologies. As expected, the number of accepted connections for a split depth of zero (no split) is the largest due to maximum interval sharing among sub-trees. This, however, comes at the price of more collisions and less throughput and PDR per

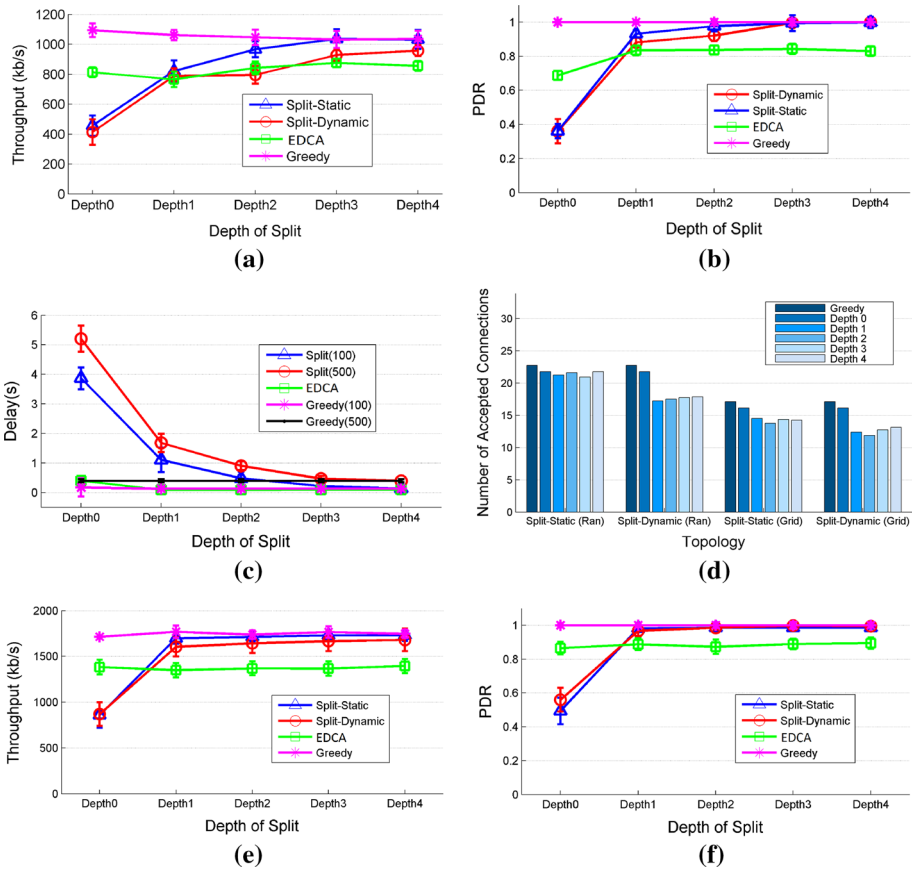

Fig. 8 Impact of split depth on throughput, PDR and average delay for the 4×4 grid (**a**–**c**) and random topology (**e**, **f**). Graph (**d**) shows the number of accepted connections by The CAC mechanism for both topologies

connection. In particular, the throughput achieved under full interval sharing (depth of zero) is close to 450Kbps which is significantly less than the 800Kbps value achieved by conventional EDCA access.

Simply increasing the split depth to one, improves throughput and PDR making it almost identical to EDCA while only reducing accepted connections by about 10 %. In fact, Figure 8a, b, d show the trade-off between eliminating off-path interference and connection acceptance ratio. One important feature of our proposed Time-Splitting scheduler is that this trade-off can be tuned as desired by adaptively changing the split depth based on network conditions and the level of QoS requirement.

Recall that the performance of our greedy scheduler matches that of the optimal scheduler over the tree. As a result Figure 8d also shows how Time-Splitting CAC performs compared to the maximum number of admissible connections. In particular, for the grid topology and for a depth of four it admits 76 % of maximum possible connections while it achieves on average 95 % of maximum admissible connections for random networks under static splitting regime. This gap with the optimal capacity is the price paid for

Time-Split's simple off-path interference avoidance approach. By adopting the more fair dynamic splitting approach, on average 73 % of the maximum admissible connections are accepted.

In order to test the accuracy of our admission control mechanism we disable it and provide the network with connections beyond its admissible range. We expect PDR to drop sharply when extra connections are admitted into the network. Figure 9 compares the PDR for Time-Split with a depth of four, our greedy scheduler and conventional EDCA access for the 4×4 grid topology as the number of simultaneous connections increases. As expected, both Time-Splitting and greedy schedulers maintain a PDR of almost 100 % up to their maximum number of admissible connections, which is 15 and 17, respectively. Note that at these points, the PDR for EDCA is 61 and 65 %, respectively. However, as we exceed beyond the admissible number of connections, both Time-Split and greedy schedulers experience substantial packet loss. For instance, by accepting only two more connections above their capacity, the PDR of both Time-Split and greedy will drop below 90 %.

The relatively sharp decline in PDR suggests that our CAC mechanism along with its respective scheduler tightly pushes the network to its performance bound. Otherwise, one could have argued that conventional EDCA is able to provide roughly the same PDR performance if it were introduced the same connection load. Clearly, Figure 9 opposes such an argument. In fact, the PDR performance of Time-Split becomes similar to that of EDCA at a value of 50 %, which is far from acceptable. It follows that, for all reasonable values of PDR, Time-Split performs better than conventional EDCA, not by merely limiting the number of connections but more importantly by eliminating off-path interference and reducing the likelihood of collisions.

Lastly, Figure 8c shows path delay for Time-Split with a service cycle of 100msec and 500msec and compares it to EDCA. Conventional EDCA access has the lowest delay as nodes send whenever packets become available. However, increasing split depth significantly improves delay for Time-Split due to removing collisions and the resulting back-off and re-transmissions. In fact, at a split depth of three and with a service cycle of 100msec the delay performance of both Time-Split and EDCA become almost identically equal to 0.096s.

We have also evaluated jitter with service cycles of 100 and 500msec in Figure 10. It can be seen that Time-Split causes larger jitter. Nevertheless, after a split depth of two the

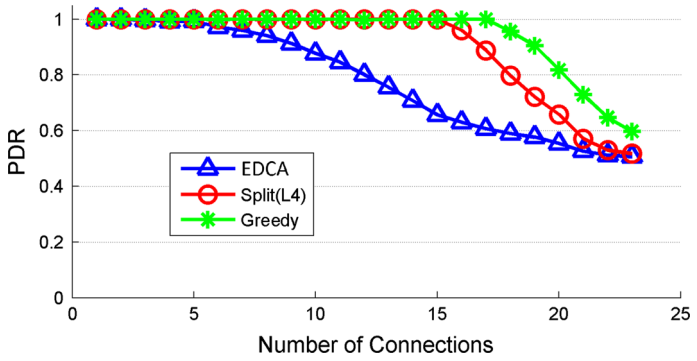

Fig. 9 PDR versus number of connections in 4×4 grid topology With no CAC

jitter becomes lower than 40msec, which is acceptable for a voice call according to values reported in [19].

5.2 Energy Performance Evaluation

In this section, we evaluate the potential energy savings of Time-Split and greedy schedulers when nodes put their radio into sleep during sleep intervals shorter than τ_0, and go into deep sleep during sleep intervals larger than τ_0. In deep sleep, an entire node transitions into low-power mode allowing more energy saving. Table 2 lists sample power consumption figures under each mode of operation, as well as the value of τ_0 chosen for our simulations. Our energy analysis of the network is focused on the node with minimum energy content across the wireless mesh network as it determines the lifetime of the network.

Figure 11a shows power consumption of the energy-bottleneck node over a 4×4 grid topology with various schedulers and compares it to EDCA. For the sake of fair comparison, we have applied the exact same set of connections to all schedulers including EDCA. According to this figure, Time-Split with any non-zero split depth reduces power consumption of the bottleneck node by a factor of four compared to conventional EDCA access.

Having a highly fragmented time-line, the greedy scheduler cannot take much advantage of deep sleep, however. The Time-Split scheduler, on the other hand, is designed such that contiguous intervals of sleep are provided to nodes. Hence, in practice more power saving can result from the use of Time-Split compared to greedy scheduling especially at larger splitting depths. In particular, Time-Split consumes half the power of greedy scheduler on the bottleneck node as suggested by Figure 11a.

In addition, lifetime results shown in Figure 11b closely follow that of bottleneck power consumption. We consider network lifetime as the time it takes for the first mesh node to deplete its energy storage and go into outage. We believe this definition of lifetime to be suitable for wireless mesh networks where each node also serves local traffic and its failure causes service outage for those clients connected to it. Figure 11b suggests that network lifetime improves by 280 % using Time-Split with a split depth of only one, compared to EDCA. In other words, under conventional EDCA access, the network will experience

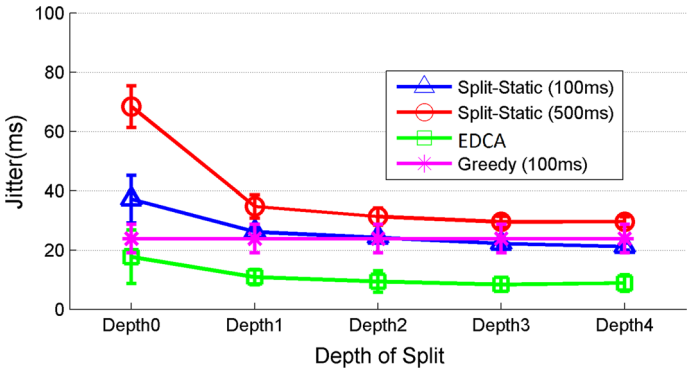

Fig. 10 Comparison of jitter for different split depths

outage nearly a third through the same simulation run as Time-Split. These results clearly show that our proposed Time-Split scheduler can properly take advantage of deep sleep techniques to provide extensive power savings for energy-constrained wireless mesh networks.

5.3 Performance of Multi-Channel Time-Split

We compare the performance of our multi-channel Time-Split scheduler using three orthogonal channels with its single-channel version for different splitting depths. Figure 12a, b illustrate total network throughput and PDR, respectively. Multi-channel Time-Split improves throughput by at least 30 %, while maintaining a PDR of above 90 % even when splitting at the first level. Note that decreasing the split depth provides more time sharing among nodes that could potentially increase throughput if interference is dealt with appropriately.

The delay performance of multi-channel Time-Split is also illustrated by Table 3. A certain delay objective can be simply met by appropriately setting the scheduling cycle. For instance, Table 3 clearly shows that a maximum delay of 500msec is satisfied by setting the scheduling cycle to 500 msec. Furthermore, these results suggest that multi-channel operation does not significantly change the delay performance of Time-Split.

One may expect that using three channels instead of one should almost triple total throughput. In fact, this is a reasonable expectation if the number of radios used by each node matches the number of channels to allow simultaneous use of channels. However, we have confined all our nodes to a single radio so that each node may only use a single

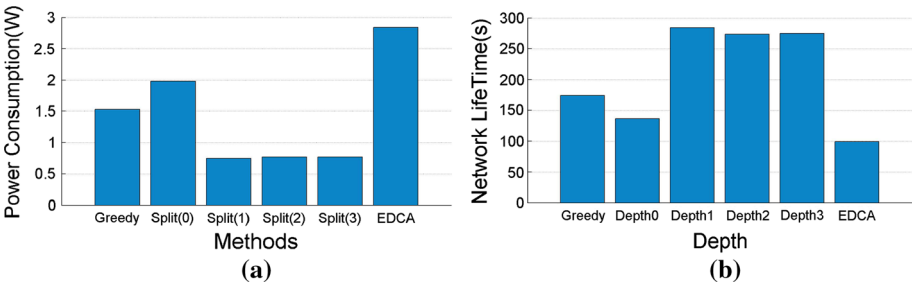

Fig. 11 **a** Average power consumption at bottleneck node, **b** network lifetime

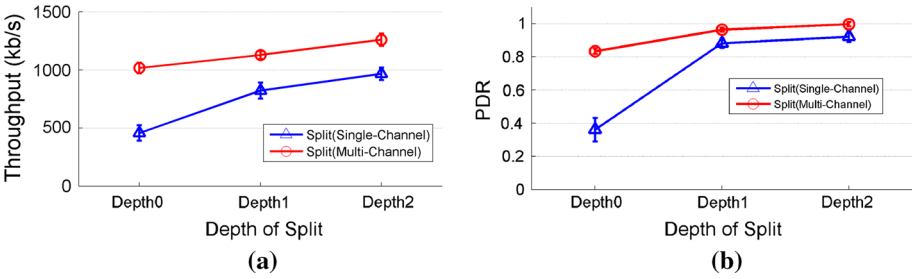

Fig. 12 Throughput (**a**) and PDR (**b**) comparison of single-channel and multi-channel time-split

Table 3 Path delay evaluation of scheduling algorithms

	EDCA	Greedy	Split (single-channel)	Split (multi-channel)
Delay(s)	0.096	0.38	0.394626	0.401534

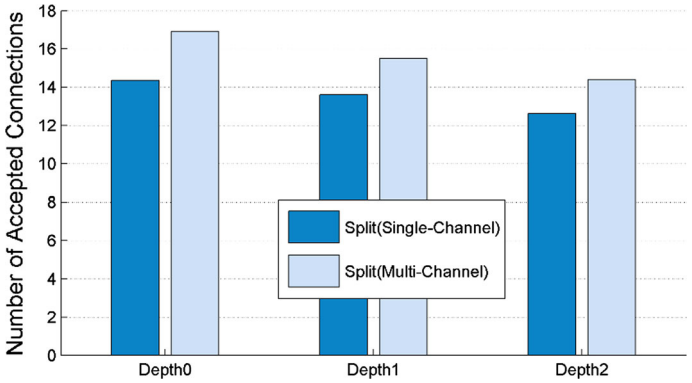

Fig. 13 Improvement in accepted number of connections using multiple channels

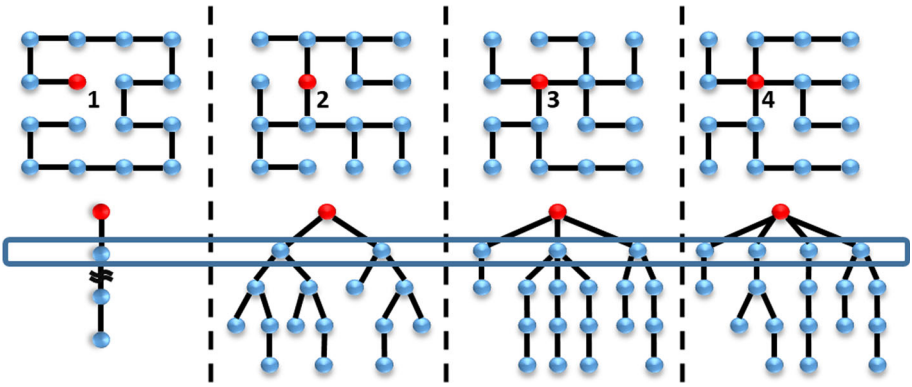

Fig. 14 Different tree formations considered for a 4×4 grid

channel to send or receive at a time. As a result, the capacity that can be achieved by any given node will never exceed that of a single channel at full utilization. Essentially, the role of multi-channel operation is to allow nodes to communicate during otherwise forced-sleep intervals that were installed to prevent interference. Multi-channel operation avoids this interference by simply switching the node to a different channel rather than putting it into forced sleep. Nevertheless, as suggested by Figure 13 the number of accepted connections using the multi-channel approach improves between 14 and 17 % over a 5×5 grid topology.

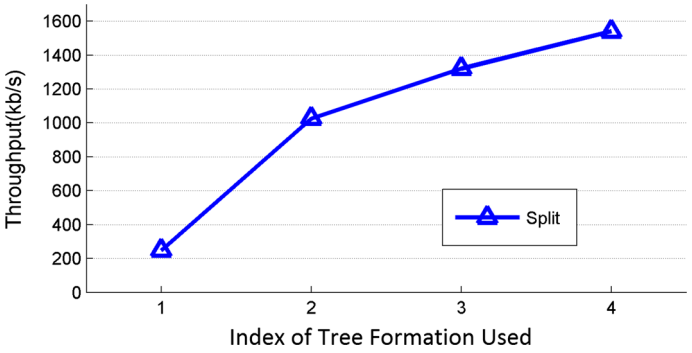

Fig. 15 Impact of tree formation on throughput of time-split

5.4 The Effect of Tree Topology

It is fair to assume that the performance of our Time-Split scheduler depends on the actual tree that is formed over the wireless mesh network. For example, a very tall tree can result in excessive splitting of the time-line toward leaf nodes leaving a small chance for connections to be initiated to/from the leaves and reducing network throughput.

To investigate how tree topology affects our scheduler, we have pre-selected a certain number of trees that are all formed over a 4×4 grid as illustrated in Figure 14. The first tree is a very tall one in which all nodes have at most one child. This represents an extreme case in our study. The second tree is one in which all nodes have at most two children resulting in a more balanced tree. The third and fourth trees have three and four children at the top most level, respectively. It is expected that as the depth of a tree increases the performance of Time-Split should degrade.

Figure 15 shows how total network throughput behaves for each tree formation when splitting is performed down to the lowest level of the tree. The first tree formation having the most depth results in the worst throughput at only 250 Kbps which is almost six times smaller than the best value achieved for case number four at 1.5 Mbps. The amount of throughput variation can still be significant even for well balanced trees such as 2,3 and 4. As Figure 15 suggests this variation is close to 50 % in performance.

Clearly, those trees that provide more capacity to bottleneck nodes are able to achieve larger throughput. Recall that bottleneck nodes are children of the root which transport most of the traffic in both send and receive directions. Therefore, a tree which maximizes the number of nodes at the first level and at the same time has a small depth will be the one with larger throughput. Interestingly, the tree formation algorithm adopted by HWMP in 802.11s produces this type of tree which is labeled as “4” in Figure 14. In fact this is what we have used in all our simulations.

6 Conclusion

We have proposed a novel scheduling algorithm for IEEE802.11s wireless mesh networks using the standard HWMP routing protocol. Our scheduling algorithm, Time-Split, is distributed and fairly simple as it requires no knowledge of the interference graph of the WMN. Scheduling decisions are made at each node along the connection path as

connection establishment process proceeds. An implicit connection admission control (CAC) mechanism is also employed to block connections when interference cannot be appropriately avoided. Using extensive simulations we have shown that Time-Split performs close to optimal over a tree topology and also provides significant energy savings due to provisioning contiguous sleep intervals that can be used for deep sleep. In particular, Time-Split consumes almost half the energy required by a greedy sleeping mechanism. A multi-channel version of Time-Split is also proposed which considerably improves connection acceptance ratio with a single radio card. Furthermore, we show that desirable trade-off between bandwidth sharing and interference prevention can be achieved by proper tuning of the split-depth used with Time-Split. We also evaluate end-to-end delay and jitter and show that it can meet a certain bound by suitably limiting the scheduling interval.

References

1. Gabale, V., Raman, B., Dutta, P., & Kalyanraman, S. (2013). A classification framework for scheduling algorithms in wireless mesh networks. *IEEE on Communications Surveys & Tutorials*, *15*(1), 199–222.
2. IEEE Standards Association et al. (2012). 802.11-2012-ieee standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications.
3. Yigit, M., Incel, O.D., & Gungor, V.C. (2014). On the interdependency between multi-channel scheduling and tree-based routing for WSNs in smart grid environments. *Computer Networks*, *65*(0), 1–20.
4. Bagaa, M., Challal, Y., Ksentini, A., Derhab, A., & Badache, N. (2014). Data aggregation scheduling algorithms in wireless sensor networks: Solutions and challenges. *IEEE on Communications Surveys Tutorials*, *16*(3), 1339–1368.
5. Chang, Y., Liu, Q., Jia, X., Tang, X., & Zhou, K. (2012). Joint power control and scheduling for minimizing broadcast delay in wireless mesh networks. In *2012 IEEE Global Communications Conference (GLOBECOM)*, (pp. 5519–5524).
6. Incel, O. D., Ghosh, A., Krishnamachari, B., & Chintalapudi, K. (2012). Fast data collection in tree-based wireless sensor networks. *IEEE Transactions on Mobile Computing*, *11*(1), 86–99.
7. Hohlt, B., Doherty, L., & Brewer, E. (2004). Flexible power scheduling for sensor networks. In *Proceedings of the 3rd international symposium on information processing in sensor networks* IPSN '04, (pp. 205–214). New York, NY: ACM.
8. Kim, E.S., & Glass C.A. (2014). Perfect periodic scheduling for three basic cycles. *Journal of Scheduling, 17*(1), 47–65.
9. Kim, E.-S., & Glass, C.A. (2015). Perfect periodic scheduling for binary tree routing in wireless networks. *European Journal of Operational Research, 247*(2), 389–400.
10. Shrivastava, P., & Pokle, S. B. (Jan 2014). Energy efficient scheduling strategy for data collection in wireless sensor networks. In *2014 International conference on electronic systems, signal processing and computing technologies (ICESC)* (pp. 170–173).
11. Sengaliappan, M., & Marimuthu, A. (March 2014). Enhanced tree routing algorithms in wireless sensor network. In *2014 international conference on green computing communication and electrical engineering (ICGCCEE)* (pp. 1–11).
12. Rhee, I., Warrier, A., Min, J., & Xu, L. (2006). Drand: Distributed randomized tdma scheduling for wireless ad-hoc networks. In *Proceedings of the 7th ACM international symposium on mobile ad hoc networking and computing* (pp. 190–201).
13. Takita, D. (2013). Centralized scheduling for wireless mesh networks with contention-reduced media access. In *2013 International symposium on intelligent signal processing and communications systems (ISPACS)* (pp. 493–496).
14. Ribeiro, F., Campos, R., Rua, D., Pinho, C., & Ruela, J. (2013). Pace: Simple multi-hop scheduling for single-radio 802.11-based stub wireless mesh networks. In *2013 IEEE 9th international conference on wireless and mobile computing, networking and communications (WiMob)* (pp. 103–110).

15. Narlikar, G., Wilfong, G., & Zhang, L. (2010). Designing multihop wireless backhaul networks with delay guarantees. *Wireless Networks*, *16*(1), 237–254.
16. Vijayalayan, K. S., Harwood, A., & Karunasekera, S. (2013). Distributed scheduling schemes for wireless mesh networks: A survey. *ACM Computing Surveys (CSUR)*, *46*(1), 14.
17. Pérez-Costa, X., & Camps-Mur, D. (2010). IEEE 802.11 e qos and power saving features overview and analysis of combined performance [accepted from open call]. *IEEE on Wireless Communications*, *17*(4), 88–96.
18. Panigrahi, D., & Raman, B. (2009). Tdma scheduling in long-distance wifi networks. In *IEEE on INFOCOM 2009* (pp. 2931–2935).
19. Adegbenro, O., John, S. N., & Akinade, B. A. (2014). Modeling the contributory effect of impairment factors on voice transmitted over the internet. *International Journal of Computer Applications*, *89*(3), 42–47.

Mojtaba MalekpourShahraki received his B.Sc. degree in Information Technology Engineering from the University of Kurdistan, Sanandaj, Iran in 2011, and the M.Sc. degree in Information Technology-Computer Networks from Iran University of Science and Technology, Tehran, Iran in 2015. His research interests are Wireless Ad Hoc Networks, Wireless Mesh Networks, Software Defined Networks, Virtualization and Cloud Computing.

Hadi Barghi received his B.Sc. degree in Computer Hardware Engineering from Isfahan University, Isfahan, Iran in 1999, and the M.Sc. degree in computer architecture from Sharif University of Technology, Tehran, Iran in 2001. Now, he is a Ph.D. student at Iran University of Science and Technology, Tehran, Iran. His research interests are Wireless Networks, Computer Arithmetic.

Seyed Vahid Azhari Received his Ph.D. in Electrical and Computer Engineering from McMaster University, Canada in 2007. He is currently an Assistant Professor at Iran University of Science & Technology. His research interest includes energy efficient wireless networking. His work is mainly focused on developing energy efficient schemes for operation of IEEE802.11-based wireless networks. In addition, he works on SIP signaling networks and the IP Multimedia Subsystem. He is particularly interested in developing load aware and scalable network architecture and protocols for these systems.

Saeed Asaiyan received his B.Sc. degree in Computer Engineering from University of Guilan, Rasht, Iran in 2011, and the M.Sc. degree in Information technology-computer networks from Iran University of Science and Technology, Tehran, Iran in 2015. His research interests are Wireless Mesh Networks and CDN networks.