

# TCP flow-aware Channel Re-Assignment in Multi-Radio Multi-Channel Wireless Mesh Networks

Juan J. Galvez  
DIIC, Computer Science Faculty  
University of Murcia, Spain  
Email: jjgalvez@um.es

Pedro M. Ruiz  
DIIC, Computer Science Faculty  
University of Murcia, Spain  
Email: pedrom@um.es

Antonio F. Gomez Skarmeta  
DIIC, Computer Science Faculty  
University of Murcia, Spain  
Email: skarmeta@um.es

**Abstract**—We consider the traffic-aware channel assignment problem in a multi-radio Wireless Mesh Network that involves assigning channels to radio interfaces to optimize the performance of a set of TCP flows (flow throughput and fairness). The resulting optimization problem is NP-hard. At the TCP flow level, rapid traffic fluctuations are expected, imposing the need for frequent channel re-assignment to adapt to current traffic conditions. We develop a centralized greedy channel assignment algorithm suited for this task that can support frequent channel re-assignment by two properties: (i) low computational complexity, permitting it to quickly calculate a solution, and (ii) by taking into account the previous channel assignment to generate a new one with minimum variation. Focusing on the gateway access scenario, we observe an important benefit to optimizing the performance of TCP flows with the proposed algorithm.

## I. INTRODUCTION AND MOTIVATION

Wireless Mesh Networks (WMNs) have recently attracted much attention. They are comprised of mesh routers and mesh clients. Mesh routers are stationary nodes interconnected by wireless links. They serve as an infrastructure wireless backbone, providing connectivity to mesh clients. Typically, a subset of routers have direct connectivity to a fixed infrastructure (e.g. a wired network such as the Internet) and serve as gateways to the mesh nodes. WMNs provide a cost-effective way to deploy a wide-area network and offer broadband Internet access.

The reduction in hardware costs permits equipping routers with multiple radio interfaces. In these networks, the effective use of multiple non-overlapping channels (e.g. 3 in IEEE 802.11b/g and 12 in IEEE 802.11a) can significantly enhance the network capacity by allowing more concurrent transmissions. In this context, a key issue is the assignment of channels to radio interfaces to minimize interference.

Traffic-aware *channel assignment* (CA) has the important benefit of optimizing radio resources taking into account real traffic patterns. The majority of traffic-aware approaches either assume knowledge of a traffic profile specifying the expected load between nodes, or allocate rate so that the load on each link is known [1]–[5]. In practice, most traffic is constituted by elastic TCP flows. We consider that it is not accurate to assume knowledge of a traffic profile in WMNs for two main reasons: (a) because the capacity of a WMN is limited,

users will tend to use all available capacity. This means that routing and channel assignment affect resulting demands, and those demands don't necessarily reflect user requirements. As soon as routing or channel assignment changes, demands can change; (b) traffic is dynamic and can constantly change.

A difficulty of scheduling link-rates in a multihop wireless network is the requirement of an accurate throughput model to guarantee that the allocation is feasible. When considering the SINR (Signal to Interference and Noise Ratio) model of interference and a random access-based medium access model (e.g. DCF used by 802.11), analytically modeling throughput in wireless multihop networks is highly complex and so most works rely on simplifying assumptions, such as synchronized time-slotted mode [2], [3], binary interference models and throughput estimation [1], [4], [5].

To our knowledge, no previous work has attempted to calculate a channel assignment to optimize the performance of a specific set of TCP flows. One of the main challenges of this approach is that, at the TCP flow level, rapid traffic fluctuations are expected and this most likely imposes the need for frequent channel re-assignment. A change in channel allocation requires disseminating the new assignment to nodes in the network<sup>1</sup>. The time required for this, along with the interface switching delay, can temporarily disrupt network activity. In [6] we show that channel re-assignment with a period of a few seconds is realizable in gateway access scenarios<sup>2</sup>. To effectively support this, a CA algorithm must be able to quickly calculate a solution while minimizing the number of channel re-adjustments needed.

In this work we study the benefit of optimizing the performance of a set of TCP flows via CA, focusing on the gateway access scenario. We present a centralized CA algorithm suited for this task that can support frequent channel re-assignment. The proposed algorithm meets the following goals:

- Optimizes the performance of a set of TCP flows.
- Low computational complexity.

<sup>1</sup>Due to space constraints, a protocol for CA dissemination is outside the scope of this paper. See [6] for our current proposal.

<sup>2</sup>In gateway access scenarios there is a reduced set of *critical* edges that determine performance (those closest to the gateway). To adapt to traffic variations, it generally suffices to modify the CA of a subset of these edges.

- Given an existing CA, calculates a new CA with limited channel re-adjustment.
- Assumes the SINR model of interference.

When considering traffic-aware CA, there is a well-known interdependency between routing and channel assignment. The joint routing and CA problem is NP-hard and is frequently approached by decomposing the problem into separate routing and CA subproblems (e.g. [1], [2]). Due to space limitations, in this work we only consider the CA algorithm and assume that routes are already given.

The rest of the paper is organized as follows. In section II we review related work. In section III we describe the system model, and define and formulate the Channel Assignment problem. Section IV describes the channel assignment algorithm. Section V analyzes the performance of the algorithm using graph-theoretic metrics and by simulation with *ns-3*. Finally, in Section VI we summarize our conclusions.

## II. RELATED WORK

Early work on the use of multiple channels in multihop wireless networks assumed a single radio per node [7], [8]. These mechanisms require each node to dynamically switch between channels, coordinating with neighboring nodes to ensure communication over a common channel for some period. Such coordination is usually based on tight time synchronization among nodes, which is difficult to realize in a multihop wireless network, and/or very fast channel switching capability that is not yet available with commodity hardware.

In multi-radio networks, we can broadly classify CA in two categories: traffic-aware and traffic-independent strategies.

Traffic-independent approaches assume no knowledge of traffic patterns and focus on minimizing interference in the network given connectivity and potential interference characteristics. Examples include centralized [9]–[11] and distributed approaches [12], [13].

Traffic-aware approaches assume knowledge of expected or real traffic patterns in the network. Most centralized approaches either assume a priori knowledge of the traffic profile or allocate rate to optimize throughput [1]–[5]. These strategies consider the problem jointly with routing, where routing determines the actual link loads and a feasible channel assignment to support it is calculated. Distributed protocols have also been proposed, which do not assume a priori knowledge of traffic and instead nodes estimate the current load in their vicinity by measurement [14], [15]. An issue in distributed approaches is the time required to converge to a solution. This time can be higher than the actual time between variations in traffic conditions. For example, simulation results for a sample scenario in [14] show a convergence time of two minutes.

To our knowledge, the issue of channel re-assignment to minimize the number of channel re-adjustments was first studied in [16], with the objective of calculating a new CA without exceeding a specified number of re-adjustments.

Subramanian et al. proposed centralized and distributed algorithms for the CA problem [17]. The mathematical form of their traffic-aware objective function is equivalent to the

one used by our algorithm. We compare our algorithm with the centralized algorithm proposed in the above paper, and find that ours finds comparable solutions while being substantially faster.

Our proposal is also shown to perform better than a heuristic which, not taking into account real loads, establishes the priority of edges based on their distance to the gateway (this strategy is used e.g. in [11]).

## III. CHANNEL ASSIGNMENT PROBLEM FORMULATION

### A. Network and traffic model

We consider WMNs which comprise of stationary wireless mesh *routers*, also called *nodes*. These nodes form a wireless multi-hop network. Mesh *clients*, also called *users*, connect to the mesh routers. A subset of nodes, referred to as *gateways*, are directly connected to a fixed infrastructure, which we will assume is the Internet for the rest of this paper. Each router has multiple radio interfaces with omni-directional antennas (e.g. 802.11a/b/g) for communication with other routers. Each interface can be tuned to one of several non-overlapping channels. Communication between nodes and users is done via a separate interface and channel (wired or wireless).

Although intra-WMN communication is possible, we assume that most of the traffic will be received from the Internet. Users are randomly located in the network. A user accesses the Internet through one or more links leading from its router to the gateway. We assume that all traffic entering the WMN is elastic, and that in any instant a user can initiate a connection to the Internet generating a download flow of any number of bytes. This is safe to assume, because the majority of Internet traffic today uses TCP.

### B. Interference model

In this work we assume the SINR model of interference. The average signal strength at the receiver depends on the sender's transmission power, the path loss (due to any number of factors such as distance and obstacles) and fading. The SINR at the receiver and the modulation used determines the expected Bit Error Rate (BER) [18], and consequently the expected Packet Error Rate (PER).

A transmission link is specified by a sender and a receiver. The information CA uses is the PER of links. Specifically, let  $\text{per}(u)$  be the PER of transmission link  $u$  in the presence of ambient noise and no external interference. Let  $\text{per}(u|v)$  be the PER of transmission link  $u$  in the presence of ambient noise and concurrent transmission of link  $v$ . In general  $\text{per}(u|v) \neq \text{per}(v|u)$ . If  $u$  and  $v$  are on different channels  $\text{per}(u|v) = \text{per}(u)$ . Although CA doesn't take cumulative interference directly into account, i.e. the PER of a link when multiple links transmit concurrently (e.g.  $\text{per}(u|v, w, x)$ ), we will show a simple mechanism to reduce its effect.

We assume the use of a technique which measures packet delivery ratios and interference. Measurement techniques have been proposed in [19]–[21].

### C. System model

Let  $F$  denote the set of flows in the network. We use the definition of flow in [22]. The network operator is responsible for choosing the exact properties that define a flow. Although the CA problem is independent of specific traffic patterns, practical reasons may dictate that the flows which are considered are those that traverse gateways, because they constitute the majority of load and can be easily classified and tracked by gateways using a system such as IPFIX [22], without needing to collect information from inside the WMN.

The topology is modeled by an undirected graph  $G = (V, E)$ . We use the notation  $e_{mn}$  to refer to an edge  $e$  with pair of vertices  $(m, n)$ . In the case of a directed edge, the order of the vertices in  $e_{mn}$  conveys direction. Let  $dir(u_{mn}) = \{v_{mn}, v_{nm}\}$  be the pair of directed edges associated with an undirected edge  $u_{mn}$ . Let  $D = \bigcup_{e \in E} dir(e)$  denote the set of all directed edges corresponding to edges in  $E$ . Directed edges model unidirectional transmission links (given by a transmitter and receiver pair). They are used to model interference and direction of traffic. Undirected edges model a bidirectional link, and are used for CA. All valid communication links are bidirectional, because every transmission requires acknowledgments. Let  $inv(e_{mn}) = e_{nm}$ . Let  $E(n) = \{e_{ab} \in E \mid n \in \{a, b\}\}$  be the set of edges incident on node  $n$ . A directed edge  $e$  exists between two nodes if and only if  $per(e) < X$  (in this paper  $X = 0.05$ ). Let  $\mathbf{IM}$  denote the interference matrix, where  $\mathbf{IM}_{u,v} = per(u|v) \forall u, v \in D$ .

For practical reasons, we assume only one undirected edge between two nodes. The use of multiple edges requires assigning them to different channels, and therefore to different interfaces. Having routes to the same neighbor through different interfaces adds complexity to the routing layer, and can lead to packets arriving in different order at the destination.

The set of channels is denoted by  $C$ . Let  $R(n)$  be the set of radio interfaces of node  $n$ . For ease of exposition we assume, without loss of generality, that  $|R(m)| = |R(n)| = R \forall m, n \in V$ . A channel assignment is a function  $\chi : E \mapsto C$ . We denote  $\chi(e) = c$  if channel  $c$  is assigned to edge  $e$ . Channels are assigned to undirected edges or, in other words, the same channel is assigned to a directed edge and its inverse edge (a node expects to receive acknowledgments on the same interface). A channel is assigned to every edge, and so channel allocation does not alter network connectivity.

Let  $t(e)$  be the number of flows traversing an edge. This is determined by the routing algorithm, which is outside the scope of this paper.

### D. Channel re-assignment problem

The goal is to find an edge-channel assignment  $\chi$ , to optimize the performance of flows (throughput and fairness). In addition, to support frequent channel re-assignments the calculated CA must vary as little as possible with respect to the previous CA. More specifically, given the link load  $t(v) \forall v \in D$  and the previous CA  $\chi_p$ , the problem is to assign a channel to each undirected edge, to minimize interference, while at the same time minimizing the number of edges which

must change their assignment with respect to  $\chi_p$ . In general, both objectives conflict and there is a trade-off involved in their optimization, i.e. it is not possible to simultaneously minimize both. Additionally, a CA must obey the interface constraint, which determines that the number of distinct channels assigned to edges incident on a node is at most  $R$ . We propose solving the following multi-objective problem:

$$\mathbf{P}_{CA} : \min_{\chi} [obj_1, obj_2] \quad (1)$$

$$obj_1 : \sum_{u \in D} \sum_{v \in D - \{u, inv(u)\}} t(u) t(v) per(u|v) \quad (2)$$

$$obj_2 : |\{e \in E \mid \chi_p(e) \neq \chi(e)\}| \quad (3)$$

subject to:

$$|\{\chi(e) \mid e \in E(n)\}| \leq R \quad \forall n \in V \quad (4)$$

The goal is to find an assignment  $\chi : E \mapsto C$  that optimizes the objectives. Recall that the per function is defined for directed edges, that the channel assigned to a directed edge is the same as that assigned to its corresponding undirected edge and that  $per(u|v) = per(u)$  if  $u$  and  $v$  are in different channels. The interface constraint is expressed in Eq. 4.

Accurately predicting throughput in a WMN requires complex models, due to a number of factors including asynchronous random access, interference and dynamic wireless medium. Integrating a TCP model adds more complexity. It is difficult to use such models as objective functions for optimization problems and at the same time develop an efficient solution. Our priority is to develop an efficient algorithm to support frequent channel re-assignment, and so we opt for an easily evaluable objective function which correlates with flow performance (as demonstrated in section V). Minimizing  $obj_1$  implies assigning interfering edges with traffic to different channels. A flow is expected to share capacity with flows in the same edge and interfering edges. For this reason, it is important to avoid interference in edges with more flows, to decrease unfairness and flow starvation. Minimizing  $obj_1$  is known to be NP-hard [17].

The second objective is to minimize the number of edges which have changed their assignment with respect to  $\chi_p$ .

Note that the channel assigned to an edge without traffic does not affect the objective function, because such an edge does not produce interference. This holds true given the *current* set of flows. However, in a dynamic environment traffic conditions can change at any time and so some consideration must be given to edges which currently have no traffic. See discussion of this issue in the next section.

## IV. LOAD-AWARE CHANNEL ASSIGNMENT ALGORITHM

We propose a greedy algorithm to solve  $\mathbf{P}_{CA}$ . We will refer to it as the Load-aware Channel Assignment algorithm -LACA- for the rest of this paper. LACA is both fast and obtains good solutions. A low execution time is crucial to permit frequent channel re-assignment. The algorithm visits

TABLE I  
STRUCTURES USED BY CHANNEL ASSIGNMENT ALGORITHM.

$c.edges = \{e \in E \mid \chi(e) = c\}$	Set of edges assigned to channel $c$
$v.channels = \{\chi(e) \mid \forall e \in E(v)\}$	Set of channels assigned to edges of $v$
$v.edgesUsingChannel(c) = \{e \in E(v) \mid \chi(e) = c\}$	Set of edges of $v$ assigned to channel $c$

each edge and assigns it to a channel. In general, edges are visited only once, except when a merge operation is needed (see merge operation in later subsection). The quality of the solution depends on the order on which edges are visited. More critical edges are visited first, i.e. edges are visited in descending order of  $t(e)$ . In each iteration, the algorithm assigns the current edge  $e$  to the channel where the network interference (measured by Eq. 2) is minimized. The priority is thus minimizing  $obj_1$ , and an edge  $e$  will only be assigned to  $\chi_p(e)$  if it minimizes the current value of  $obj_1$ . In the studied scenarios, edge criticality is well-defined (the edges that carry more traffic are always those closest to the gateway). This leads to the fact that a greedy algorithm is sufficient to obtain good solutions, as we will show in section V.

Table I lists data structures used by LACA. Note that it is not necessary to build these structures every time they are needed, but rather they are built and modified as the algorithm progresses. For ease of exposition we obviate showing this.

The main algorithm is shown in Alg. 1. In each iteration, first it builds the list of valid channels  $vc$  which can be assigned to  $e$  without violating interface constraints (lines 5-14): when assigning edge  $e_{mn}$  to a channel, if both  $m$  and  $n$  have  $R$  channels assigned,  $e$  must be assigned to a channel shared by  $m$  and  $n$ . If they don't share channels, a merge procedure needs to be executed. If one of  $m$  or  $n$  has  $R$  channels assigned,  $e$  must be assigned to a channel in that endpoint. If no endpoint has  $R$  channels assigned, all channels can be used. In this case a random order is established in  $vc$ .

The selection of a random channel when an edge can be assigned to multiple equally good channels is a simple measure that carries two important benefits. One manifests in dynamic scenarios where sudden variations of traffic can occur in any instant, which can lead to situations of groups of edges on the same channel suddenly interfering. In addition, this measure also helps to reduce the effect of cumulative interference (see section III-B). Because the interference matrix does not account for this phenomenon, by uniformly distributing channels in the network when possible, its effect can be reduced.

The next step involves refining the list of valid channels to attempt to avoid future merge operations (explained later), and assigning the edge to the channel in  $vc$  where less interference is produced (lines 16-21). LACA attempts to maintain the same channel previously assigned to  $e$ , if possible (lines 18-19). The  $edgeInterference(u, V)$  function determines the increase in interference when the undirected edge  $u$  is in the same channel as the set of edges  $V$ . More specifically, the set  $\Delta$  contains the directed edges corresponding to the undirected

Algorithm 1 LACA: Load-aware channel assignment.

---

```

1:  $edges \leftarrow list(E)$ 
2: Sort  $edges$  in descending order of  $t(e)$ 
3: for  $e_{mn} \in edges$  do
4:    $merge \leftarrow false$ 
5:   if  $(|m.channels| = R) \wedge (|n.channels| = R)$  then
6:      $vc \leftarrow m.channels \cap n.channels$ 
7:     if  $|vc| = 0$  then
8:        $merge \leftarrow true$ 
9:     else if  $|m.channels| = R$  then
10:       $vc \leftarrow m.channels$ 
11:     else if  $|n.channels| = R$  then
12:       $vc \leftarrow n.channels$ 
13:     else
14:       $vc \leftarrow randomOrder(C)$ 
15:     if  $\neg merge$  then
16:       $vc \leftarrow avoidMerge(e, vc)$ 
17:       $bestC \leftarrow \arg \min_{c \in vc} edgeInterference(e, c.edges)$ 
18:      if  $\chi_p(e) \in bestC$  then
19:         $\chi(e) \leftarrow \chi_p(e)$ 
20:      else
21:         $\chi(e) \leftarrow \text{First element of } bestC$ 
22:      else
23:         $mergeop(e)$ 
24:
25: function  $edgeInterference(u, V)$  do
26:    $\Delta \leftarrow \bigcup_{v \in V} dir(v)$ 
27:   return  $\sum_{\vec{u} \in dir(u)} \sum_{\vec{v} \in \Delta} (t(\vec{u})t(\vec{v}) \text{per}(\vec{u}|\vec{v}) + t(\vec{u})t(\vec{v}) \text{per}(\vec{v}|\vec{u}))$ 

```

---

edges in  $V$  (line 26). The interference in directed edges of  $u$  by directed edges of  $\Delta$  and vice versa is calculated in line 27. Although not shown for ease of presentation, interference of  $u$  is only checked against its adjacent nodes in the conflict graph (i.e. against edges which can interfere with  $u$ ).

The basic structure of the greedy algorithm is similar to the one proposed by Raniwala et al. in [1]. However, we are optimizing a different objective function and network interference is measured differently. In addition, the merge operation explained in the following subsection has not been proposed before, and we introduce a novel procedure to avoid merge operations. Lastly, their algorithm is intended for static scenarios, where the traffic profile is expected to remain stable over long periods of time, and does not consider the issue of channel re-assignment.

#### A. Merge operation

The CA algorithm needs to perform a merge operation when assigning a channel to edge  $e_{mn}$  and both  $m$  and  $n$  have already been assigned  $R$  channels and don't have a channel in common, i.e.  $|m.channels| = R \wedge |n.channels| = R \wedge m.channels \cap n.channels = \emptyset$ . In this situation, a channel cannot be assigned to  $e$  without breaking interface constraints.

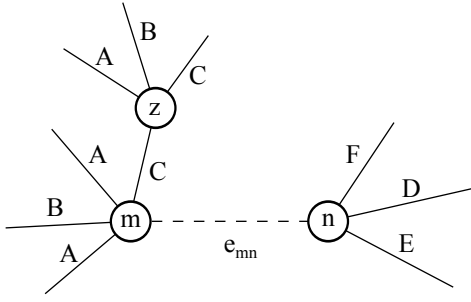


Fig. 1. Example of when a merge operation is needed. Nodes have 3 radio interfaces, and there are 6 channels {A,B,C,D,E,F}.

This condition can only occur in cases where both  $m$  and  $n$  have less radios than their node degree, the number of radios is less than the number of channels, and the algorithm happens to have previously assigned different channels to  $m$  and  $n$ . In other words, the likelihood of this occurring depends on the particular scenario: topology (node degree), number of radios in each node and number of channels.

The goal of the merge operation is for  $m$  and  $n$  to have a channel in common, and to assign  $e_{mn}$  to the common channel. To do this, a channel  $c_1$  from one of the endpoints is converted to a channel  $c_2$  from the other endpoint, and  $c_2$  is assigned to  $e$ . However, changing the channel  $c_1$  of edges of one endpoint may involve propagating the changes through the graph. To illustrate this, consider the example in Fig. 1. The algorithm wants to assign a channel to edge  $e$ . Nodes  $m$  and  $n$  have already been assigned 3 channels and don't have a channel in common. Suppose we want to reassign edges of  $m$  in channel C to a channel in  $n$ , say F. Note that this change affects node  $z$ , which must also reassign all its edges in C to F. And, for the same reason, this change in turn might involve successive propagation through the graph.

The end result is that a set of edges in  $c_1$  need to be reassigned to a different channel  $c_2$ , and the algorithm must choose the best pair  $(c_1, c_2)$  such that the overall interference (objective function) is minimized. The merge operation transforms the channel of one endpoint to a channel of the other endpoint. There are a total of  $2(R \times R)$  possible transformations, given by the Cartesian products of the set of channels in each endpoint, i.e.  $m.channels \times n.channels$  and  $n.channels \times m.channels$ , where each ordered pair  $(c_1, c_2)$  denotes that channel  $c_1$  is transformed into  $c_2$ . Each one of these transformations can require changing the channel of several edges in the graph. The merge operation tests every one of these pairs and chooses the one that produces least interference.

The algorithm is detailed in Alg. 2. For each one of these pairs (lines 2-3), the algorithm first calculates the set of edges  $E_x$  that have to be reassigned from  $c_1$  to  $c_2$  (lines 4-17). To this end, we first insert into a queue the endpoint of  $e$  which has  $c_1$  assigned to it (lines 7-10). Nodes in the queue will be processed to determine if their edges in  $c_1$  have to be changed to  $c_2$ . In the case of node  $m$  or  $n$ , all its edges in  $c_1$  have to be

---

### Algorithm 2 Merge operation initiated in edge $e_{mn}$ .

---

```

28:  $bestI \leftarrow \infty$ 
2:  $pairs \leftarrow (m.channels \times n.channels) \cup (n.channels \times m.channels)$ 
3: for  $(c_1, c_2) \in pairs$  do // test converting  $c_1$  to  $c_2$ 
4:   // propagation effect
5:    $E_x \leftarrow \emptyset$  // edges to change from  $c_1$  to  $c_2$ 
6:    $queue \leftarrow FifoQueue()$ 
7:   if  $c_1 \in m.channels$  then
8:      $queue.append(m)$ 
9:   else
10:     $queue.append(n)$ 
11:   while  $queue \neq \emptyset$  do
12:      $x \leftarrow queue.pop()$ 
13:     if  $(x \in \{m, n\}) \vee ((R - |x.channels| = 0) \wedge (c_2 \notin x.channels) \wedge (|x.edgesUsingChannel(c_1)| > 1))$  then
14:       for  $u_{xy} \in x.edgesUsingChannel(c_1)$  do
15:         if  $u \notin E_x$  then
16:            $E_x \leftarrow E_x \cup \{u\}$ 
17:           Insert  $y$  into  $queue$ 
18:     // measure effect of merge operation
19:      $I \leftarrow edgeInterference(e, E_x \cup c_2.edges)$ 
20:     for  $u \in E_x$  do
21:        $I \leftarrow I - edgeInterference(u, c_1.edges - E_x)$ 
22:        $I \leftarrow I + edgeInterference(u, c_2.edges)$ 
23:     if  $I < bestI$  then
24:        $bestI \leftarrow I$ 
25:        $bestE_x \leftarrow E_x \cup \{e\}$ 
26:        $bestC \leftarrow c_2$ 
27:     // perform merge operation
28:      $\chi(u) \leftarrow bestC \quad \forall u \in bestE_x$ 

```

---

reassigned. Next, the algorithm checks if these changes have to be propagated to other nodes. To this end, the endpoints of the edges marked for modification are inserted into the queue. When extracting a node from the queue, the following rule determines if the rest of its edges in  $c_1$  have to be reassigned to  $c_2$ : the node does not have free interfaces,  $c_2$  is not assigned to the node, and the node has more than one edge in  $c_1$  (line 13). Endpoints of edges marked for modification are inserted into the queue to continue propagating the changes and the process repeats until no more edges are added.

Once the set of edges  $E_x$  to change from  $c_1$  to  $c_2$  is obtained, the algorithm calculates the modification in network interference (lines 19-23). Edge  $e$  will interfere with all its neighbor edges in  $c_2$  (line 20). This includes the edges in  $E_x$ . And edges which were previously in channel  $c_1$  will no longer interfere with edges in this channel but will interfere with their neighbor edges in  $c_2$  (lines 21-23).

Finally, the algorithm chooses the transformation that produces least interference, and switches affected edges in  $c_1$  to

$c_2$  (lines 24-30).

As we will show, the simple rule we use to determine if a channel switch has to be propagated through the graph (line 13), when integrated in the merge procedure of the Tabu-based algorithm in [17] (which is a merge procedure different to ours), improves the solution of their algorithm.

### B. Avoiding merge operations

The merge operation can prove costly due to having to test, for each possible transformation  $(c_1, c_2)$ , the interference of every propagation of changes through the graph. The step we now explain is optional and is performed in order to reduce the likelihood of having to execute merge operations. As such, it can reduce the execution time of LACA, while maintaining a similar solution. We evaluate this empirically in section V.

First, let's call *critical* neighbors of a node the set of neighbors such that the edge that connects the node to those neighbors has not yet been assigned to a channel, the neighbors have already been assigned  $R$  channels and currently the node does not have a channel in common with them. The basic idea is that when first assigning a channel to an edge  $e_{mn}$ , the heuristic gives priority to assigning one that will permit  $m$  and  $n$  to have a channel in common with their *critical* neighbors.

The heuristic receives as input the current edge  $e$ , and the set of valid channels  $vc$  which can be assigned to  $e$ , determined by the interface constraint and calculated previously (see lines 5-14 of Alg. 1). The purpose of this heuristic is to further reduce  $vc$  to favor choosing channels which permit  $m$  and  $n$  to communicate with their critical neighbors, when necessary.

A channel is said to cover a node  $n$  if the channel is in  $n.channels$ . The following function returns the number of nodes in a set covered by a channel:

$$coveredNodes(c, N) = |\{n \in N \mid c \in n.channels\}| \quad (5)$$

The first step is to estimate the minimum number of channels the endpoints of  $e$  need to communicate with their critical neighbors. For each endpoint  $x$ , the algorithm initializes its list of critical neighbors. It then proceeds by selecting channels greedily (in each step selecting the channel which covers most critical neighbors), until all critical neighbors are covered. The size of the resulting set determines the minimum number of channels needed. Note that this heuristic does not guarantee optimality, and therefore only represents an estimate of the minimum number of channels needed.

After the minimum number of channels  $mc$  needed by each endpoint is calculated, the algorithm does one of the following:

- If  $mc$  of both endpoints equals their number of free interfaces, it forces the choice of a channel belonging to their set of critical neighbors. Channels which cover more neighbors are given priority.
- If  $mc$  of one endpoint equals its number of free interfaces, it forces the choice of a channel belonging to its set of critical neighbors. Channels which cover more neighbors are given priority.
- If the endpoints have more free interfaces than  $mc$ , no restriction is enforced on the candidate channels.

TABLE II  
PHYSICAL WIRELESS PARAMETERS.

Wifi standard	802.11b @ 11mbps constant rate
Path loss model	Log-distance, exponent = 2.7
RTS/CTS (in <i>ns-3</i> )	Disabled
Transmit power	$\min\{p \mid per(e) \leq 5\%\} \quad \forall e \in D$
Energy detection threshold (EDth) (W)	RxPower(max neighbor distance)
Carrier sense threshold (CSth) (W)	$EDth \times 10^{-9/10}$

We recommend implementing this procedure in scenarios where merge operations are likely (as explained earlier, this depends on node degree, number of radios in each node and number of channels).

## V. PERFORMANCE EVALUATIONS

We evaluate the performance of LACA in gateway access scenarios from both a graph-theoretic perspective and through simulation in *ns-3* [23].

In all cases, the WMN topologies used are the same. They consist of static networks in a square 1000 m  $\times$  1000 m area. The gateway is located in the center, surrounded by 8 so-called “ring-nodes”. Communication between the gateway and each of these nodes is assumed to not interfere with other communications (this can be achieved with wired or directional wireless links). The purpose of this architecture is to alleviate the bottleneck around the gateway [6]. The distance from the gateway to ring-nodes is 120 m. There are 70 mesh nodes placed randomly outside the gateway-ringnode zone. All topologies are connected, with a minimum distance of 100 m between mesh nodes. Maximum transmit range between mesh nodes is 150 m, i.e. an undirected edge exists between mesh nodes when the distance is less than or equal to 150 m (transmit power is adjusted to achieve this).

In all tests, we use the physical model of *ns-3* (Yans<sup>1</sup>) [23], [24]. The parameters used to configure wireless interfaces and propagation are shown in Table II. The transmit power chosen is the minimum power that guarantees a Packet Error Rate (PER) of at most 5% for all links in the network (with no concurrent transmissions). The energy detection threshold is the received power at the maximum link distance. Carrier sense threshold is derived based on a ratio of  $CSth/EDth$  of  $-9$  dB. We choose this ratio because it produced maximum aggregate throughput in numerous simulation tests.

Given a network topology (node positions), the Yans physical model, and the parameters listed in Table II, we calculate the Interference Matrix **IM** for packet sizes of 2048 bytes (this is a conservative measure because the larger the packet size the larger the PER).

Additionally, we assume the use of shortest path routing (with hop-count metric).

<sup>1</sup>In the Yans physical model of *ns-3* successful reception depends on the BER. The BER depends on the SINR and modulation used. The SINR is calculated based on signal strength, noise and cumulative interference.

### A. Graph theoretic performance metric

In this section we analyze LACA based on the value of the objective function  $obj_1$  (Eq. 2). Given a network graph  $G$ , its interference matrix  $\mathbf{IM}_{u,v} = \text{per}(u|v) \forall u, v \in D$ , and a set of flows, we apply and compare different CA algorithms.

Traffic is considered to be based on TCP flows. We generate random download flows (source is the gateway, destination is a mesh node). The number of concurrent flows in a scenario can be one of  $\{10,20,30,40,50\}$ . For each number of flows, we generate 100 random scenarios, totaling 500 unique scenarios per topology. In each scenario, the destination of a flow is chosen randomly (a node can be selected multiple times).

Tests have been performed with  $R = 3$  and  $R = 6$ . The maximum number of usable interfaces per node in these scenarios is six because the maximum node degree is six. Note that when  $R = 6$  no merge operations are needed (no node has less radios than its node degree).

We compare LACA with the Tabu-based algorithm of Subramanian et al. [17] (we refer to it as TABU)<sup>1</sup>. We test two variants of LACA: one with the avoid merge procedure (LACA\_AM) and one without it (LACA\_NOAM). We test two variants of TABU: the original one (TABU\_ORIG) and a modification where we adapt the propagation rule of our merge procedure (line 13 of Alg. 2) to the merge procedure of TABU (we refer to this variant as TABU\_MODIF). Parameters chosen for TABU are  $i_{max} = |E|$  and  $r = 20$ . All implementations are in the Python language.

Performance results are shown in Fig. 2. Each point is the average result of the scenarios that fall in that class. Figs. 2 (a) and (c) show the value of the objective of the CA problem (Eq. 2) while Figs. 2 (b) and (d) show the time to calculate a solution. A number of conclusions can be extracted. First, the solution found by LACA is very similar to the one found by TABU, and in most cases better than the original TABU when merge operations are needed ( $R = 3$ ). This is due to inefficiencies in the merge operation of TABU. Note that our modification of TABU improves this. The main advantage of LACA over TABU is that it is substantially faster. As we stated earlier, this is crucial to support frequent channel re-assignment. The execution time of TABU is several orders of magnitude higher to that of LACA (see table III for details). Also, the execution time of TABU suffers considerably with lower number of channels. We have found that in these cases, due to the nature of the TABU algorithm, the interference of edges is repeatedly tested against large groups of edges, because the probability of edges residing in the same channel is much higher. In contrast, we have found the execution time of LACA fairly constant independently of traffic patterns, number of radios and channels.

We can observe a logarithmic decrease in the objective value as the number of channels increases. This translates into increased network performance (due to assigning interfering edges with traffic to different channels). When the objective

<sup>1</sup>The mathematical form of the traffic-aware objective function optimized by TABU is equivalent to the one optimized by our algorithm.

TABLE III  
LACA AND TABU EXECUTION TIME.

	Average time (ms)	
	$R = 3$	$R = 6$
LACA_NOAM	466	356
LACA_AM	312	366
TABU_ORIG	12298	10344

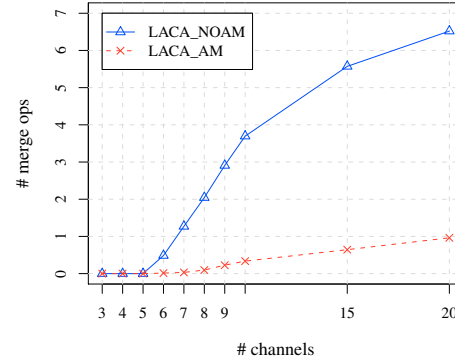


Fig. 3. Number of merge operations of LACA when  $R = 3$ .

reaches zero, it correlates with interference-free performance. This can be achieved with much less channels than number of undirected edges, and therefore it is not necessary to assign each edge to a unique channel to achieve interference-free performance (in the scenarios tested the average number of undirected edges is 146). This behavior will be observed in the simulation results. Also note that, without sufficient interfaces, the objective may never reach zero regardless of how many channels are available (see  $R = 3$  where the objective almost reaches zero).

Finally, Fig. 3 shows the number of merge operations performed by LACA when  $R = 3$ . Each point shows the average number of times a merge operation (Alg. 2) is performed during a execution of LACA. As we can see, the avoid merge process effectively reduces the number of merge operations, and thus the execution time of the algorithm (see table III). And this does not affect the quality of the solution.

### B. Channel re-adjustments

We study the number of affected edges by channel re-assignment when the traffic pattern of a network randomly varies in time. Given a network graph  $G$ , we generate a list  $L = \{t_1, t_2, \dots, t_{100}\}$  of 100 random traffic patterns. Each pattern consists of 50 random download flows. For each pattern  $t_i$  a CA is calculated, using as previous CA the one calculated for  $t_{i-1}$ . We evaluate the objective function  $obj_2$  (Eq. 3).

Table IV shows the average percentage of edges that need to re-adjust their channel in each re-assignment. It is calculated as  $100 \times \frac{\overline{obj_2}}{|E|}$  where  $\overline{obj_2}$  is the average value of  $obj_2$  over all runs. Note that only a small percentage of edges are affected.

### C. Simulation results

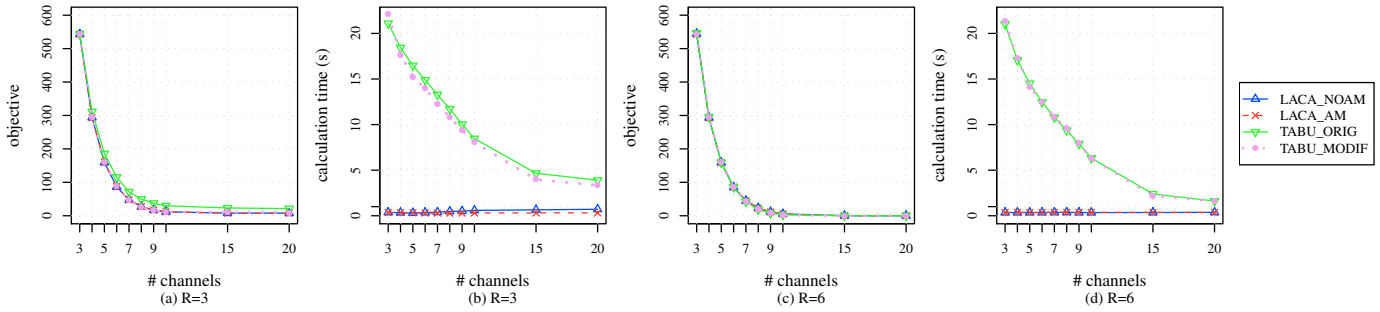


Fig. 2. Performance comparison of LACA and TABU with varying channels and  $R = 3$  and  $R = 6$ .

TABLE IV  
NUMBER OF CHANNEL RE-ADJUSTMENTS.

		Channels								
		2	3	4	5	6	7	8	9	10
% edges	$R = 3$	13	17	19	18	16	14	14	12	11
	$R = 6$	12	16	18	15	14	12	10	8	5

1) *Simulation environment*: We have evaluated the performance of LACA by simulation in *ns-3*. Tests are implemented in the following manner:

There is a server on the Internet connected to the gateway by a point-to-point 1000 Mbps link. Similarly, the connection from the gateway to each ring-node is implemented as a point-to-point 100 Mbps link. Users connect to the Internet server, which sends data to them. The gateway runs LACA. It knows the complete topology, active download flows and Interference Matrix *IM*. The *IM* is calculated once at the start of simulation using the underlying *ns-3* physical model.

Generated traffic consists of “long-lived” TCP download flows, which start at a random instant in  $30 \pm 2.5$  seconds, and have a size of 512 KB. A flow implies that a user associated with a given mesh node connects to the server to request a download of 512 KB (i.e the request originates at a mesh node, and reaches the server which generates a download flow toward the destination). The number of flows in a scenario can be one of  $\{10, 20, 30, 40, 50\}$ . For each number of flows, we generate 100 random scenarios, totaling 500 unique scenarios. Note that although flows have the same size, not all flows will be active simultaneously due to different start times, and different achieved rate.

For every scenario, we generate an initial CA which is not traffic-aware but however attempts to give more bandwidth to edges which are more likely to carry traffic. To this end, we give more priority to edges closer to the gateway. A similar heuristic is proposed in [11]. More specifically, if  $dist(n)$  denotes the distance in hops from node  $n$  to the gateway, the priority of an edge is inversely proportional to the average distance of its endpoints to the gateway, i.e.  $t(e_{mn}) = \frac{1}{(dist(m) + dist(n))/2}$ . We solve LACA assigning this weight to edges.

In all cases, simulations last until all flows finish. Points

shown in graphs represent the average result of scenarios that fall in that class. Confidence intervals are shown at the 95% level.

2) *Performance metrics*: We use the following metrics to study protocol performance. Flows refer to TCP download flows: (i) *Average network throughput* - total data bytes delivered by the network divided by the time elapsed since the first packet was sent and the last packet was received; (ii) *Peak network throughput* - the maximum throughput of the network observed in a one second interval; (iii) *Flow duration* - time elapsed since the first packet of a flow was sent and the last packet was received; (iv) *Flow throughput* - the number of bytes delivered divided by the duration of the flow; (v) *Flow throughput fairness* - rates the fairness of the throughput achieved by flows in one scenario using Jain’s fairness index.

Regarding flow throughput, in each scenario we measure the minimum and median throughput of flows. For flow duration, we measure the median. The median is chosen due to the frequent presence of outliers and skewed distribution of the flow metrics. For example, there can be scenarios where one flow benefits from a much higher throughput than other flows.

3) *Results and analysis*: To study the benefit of TCP flow aware channel assignment, we compare performance with and without LACA. When not using LACA, channel assignment is static based on the initial one. When using LACA, channel allocation is calculated and disseminated every second.

The results are shown in Figs. 4 and 5. Fig. 4 shows results with varying number of channels and  $R = 3$  interfaces in each node. Fig. 4 shows results with  $R = 5$ . The black line represents the average performance in a interference-free case (i.e. every edge assigned to a unique channel). This represents the best performance that can be achieved with the routing strategy used.

We can observe an important benefit of optimizing CA for the given set of TCP flows, with improvement across all performance metrics. When  $R = 3$ , there is an increase in minimum flow rate of up to 67% and median flow rate of up to 50% (with  $|C| = 6$ ). When  $R = 5$ , minimum flow rate increases up to 72% (when  $|C| = 6$ ), and median flow rate up to 59% (when  $|C| = 8$ ). A consequence of improving minimum rate is that fairness improves.

The performance gained by LACA with increasing number of channels follows a similar behavior to that observed from

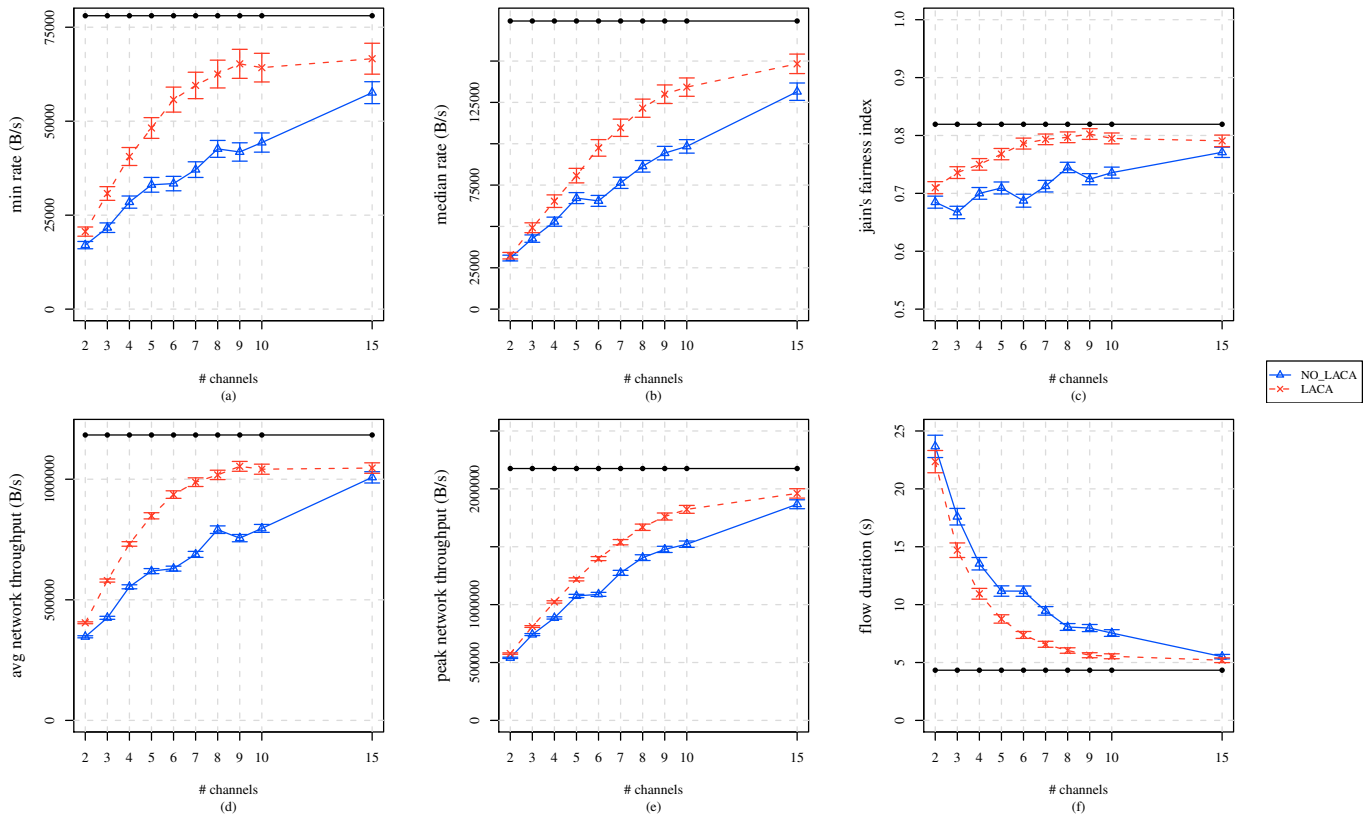


Fig. 4. Performance results of LACA with  $R = 3$ .

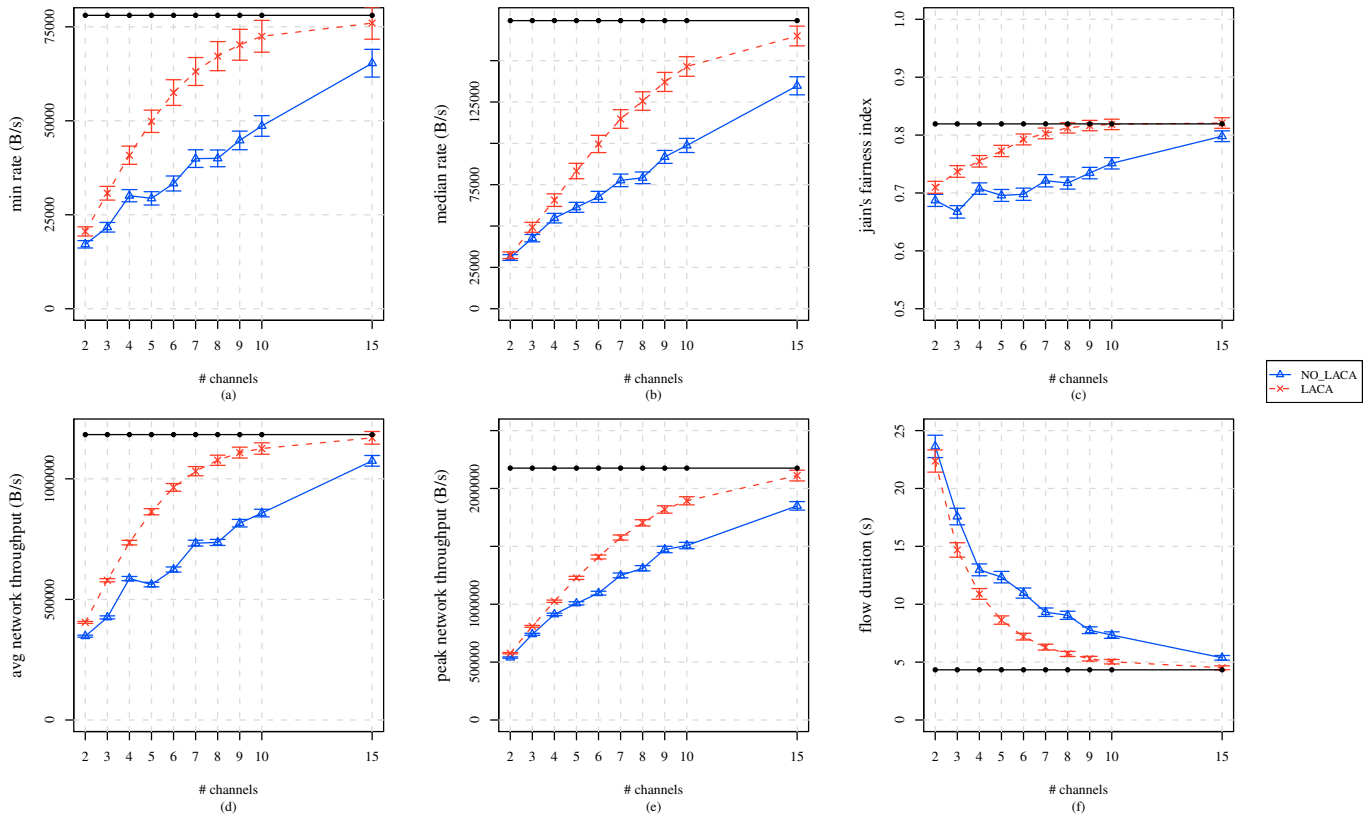


Fig. 5. Performance results of LACA with  $R = 5$ .

the graph-theoretic perspective. When  $R = 3$  the performance of LACA gets close but never reaches interference-free performance. This correlates with the behavior observed previously. With sufficient number of interfaces and channels, the performance can match that of a interference-free network. In this case, with  $R = 5$  and ten or more channels. Note that in practice it is not necessary to equip all routers with the same number of interfaces. First, the maximum number of usable interfaces per node is equal to its node degree, and second, only the routers which carry more traffic (those closest to the gateway) will effectively take advantage of more interfaces.

## VI. CONCLUSIONS

In this paper we have proposed a novel channel assignment (CA) algorithm to optimize the performance of a set of TCP flows (flow throughput and fairness). To the best of our knowledge, this is the first CA algorithm to optimize the performance of a specific set of TCP flows. At the TCP flow level, rapid traffic fluctuations are expected, imposing the need for frequent channel re-assignment to adapt to current traffic conditions. The proposed algorithm can support frequent channel re-assignment by two properties: (i) low computational complexity, permitting it to quickly calculate a solution, and (ii) by taking into account the previous CA to generate a new one with minimum variation.

Extensive evaluations show that the algorithm can efficiently calculate good solutions in gateway access scenarios. Furthermore, tests have shown that channel re-assignment only affects less than 20% of edges, which means that adapting to varying traffic conditions only requires re-adjusting the channel of a few edges. We have compared the performance between our flow-aware CA and a static CA that establishes the priority of edges based on their distance to the gateway. We have seen an improvement of up to 72% in minimum rate of TCP flows (resulting in increased fairness) and 59% in median rate.

## ACKNOWLEDGMENT

This work has been partially funded by the Spanish CARM under Grant PIIIC09-0101-9476.

## REFERENCES

- [1] A. Raniwala, K. Gopalan, and T.-c. Chiueh, "Centralized channel assignment and routing algorithms for multi-channel wireless mesh networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 8, no. 2, pp. 50–65, 2004.
- [2] M. Alicherry, R. Bhatia, and L. E. Li, "Joint channel assignment and routing for throughput optimization in multi-radio wireless mesh networks," in *MobiCom '05: Proceedings of the 11th annual international conference on Mobile computing and networking*. New York, NY, USA: ACM, 2005, pp. 58–72.
- [3] M. Kodialam and T. Nandagopal, "Characterizing the capacity region in multi-radio multi-channel wireless mesh networks," in *Proceedings of the 11th annual international conference on Mobile computing and networking*, ser. *MobiCom '05*, 2005, pp. 73–87.
- [4] J. Tang, G. Xue, and W. Zhang, "Maximum Throughput and Fair Bandwidth Allocation in Multi-Channel Wireless Mesh Networks," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications*. IEEE Computer Society, 2006.
- [5] A. Mohsenian-Rad and V. Wong, "Joint logical topology design, interface assignment, channel allocation, and routing for multi-channel wireless mesh networks," *IEEE Transactions on Wireless Communications*, vol. 6, pp. 4432–4440, December 2007.
- [6] J. J. Galvez, P. M. Ruiz, and A. F. Gomez-Skarmeta, "Responsive On-line Load-Balancing Routing and Load-aware Channel Re-Assignment in Multi-Radio Multi-Channel Wireless Mesh Networks," University of Murcia, Spain, Tech. Rep., 2011, TR-DIIC 1/2011. [Online]. Available: <http://ants.dif.um.es/staff/pedrom/papers/lbca-TR.pdf>
- [7] J. So and N. H. Vaidya, "Multi-channel MAC for Ad hoc networks: handling multi-channel hidden terminals using a single transceiver," in *Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, ser. *MobiHoc '04*, 2004, pp. 222–233.
- [8] P. Bahl, R. Chandra, and J. Dunagan, "SSCH: slotted seeded channel hopping for capacity improvement in IEEE 802.11 ad-hoc wireless networks," in *Proceedings of the 10th annual international conference on Mobile computing and networking*, ser. *MobiCom '04*, 2004, pp. 216–230. [Online]. Available: <http://doi.acm.org/10.1145/1023720.1023742>
- [9] J. Tang, G. Xue, and W. Zhang, "Interference-aware topology control and QoS routing in multi-channel wireless mesh networks," in *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, ser. *MobiHoc '05*, 2005, pp. 68–77. [Online]. Available: <http://doi.acm.org/10.1145/1062689.1062700>
- [10] M. K. Marina, S. R. Das, and A. P. Subramanian, "A topology control approach for utilizing multiple channels in multi-radio wireless mesh networks," *Comput. Netw.*, vol. 54, pp. 241–256, February 2010. [Online]. Available: <http://dx.doi.org/10.1016/j.comnet.2009.05.015>
- [11] K. N. Ramachandran, E. M. Belding, K. C. Almeroth, and M. M. Buddhikot, "Interference-Aware Channel Assignment in Multi-Radio Wireless Mesh Networks," in *Proceedings of the 25th IEEE International Conference on Computer Communications*, ser. *INFOCOM '06*, 2006.
- [12] P. Kyasanur and N. H. Vaidya, "Routing and link-layer protocols for multi-channel multi-interface ad hoc wireless networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 10, January 2006.
- [13] A. Dhananjay, H. Zhang, J. Li, and L. Subramanian, "Practical, distributed channel assignment and routing in dual-radio mesh networks," in *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, ser. *SIGCOMM '09*, 2009.
- [14] A. Raniwala and T.-C. Chiueh, "Architecture and algorithms for an IEEE 802.11-based multi-channel wireless mesh network," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, 2005, pp. 2223–2234.
- [15] A. H. M. Rad and V. W. Wong, "Congestion-aware channel assignment for multi-channel wireless mesh networks," *Computer Networks*, vol. 53, no. 14, pp. 2502 – 2516, 2009.
- [16] S. Avallone, F. D'Elia, and G. Ventre, "A traffic-aware channel re-assignment algorithm for wireless mesh networks," in *Proceedings of European Wireless*. IEEE, April 2010, pp. 683–688.
- [17] A. P. Subramanian, H. Gupta, S. R. Das, and J. Cao, "Minimum interference channel assignment in multiradio wireless mesh networks," *IEEE Transactions on Mobile Computing*, vol. 7, pp. 1459–1473, December 2008. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1477071.1477512>
- [18] G. Holland, N. Vaidya, and P. Bahl, "A rate-adaptive MAC protocol for multi-hop wireless networks," in *Proceedings of the 7th annual international conference on Mobile computing and networking*, ser. *MobiCom '01*. New York, NY, USA: ACM, 2001, pp. 236–251. [Online]. Available: <http://doi.acm.org/10.1145/381677.381700>
- [19] J. Padhye, S. Agarwal, V. N. Padmanabhan, L. Qiu, A. Rao, and B. Zill, "Estimation of link interference in static multi-hop wireless networks," in *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, ser. *IMC '05*, 2005, pp. 28–28.
- [20] C. Reis, R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan, "Measurement-based models of delivery and interference in static wireless networks," in *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. *SIGCOMM '06*, 2006, pp. 51–62.
- [21] D. Niculescu, "Interference map for 802.11 networks," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, ser. *IMC '07*, 2007, pp. 339–350.
- [22] B. Claise, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information," Jan. 2008, RFC 5101. [Online]. Available: <http://www.ietf.org/rfc/rfc5101.txt>
- [23] "The ns-3 network simulator." [Online]. Available: <http://www.nsnam.org/>
- [24] M. Lacage and T. R. Henderson, "Yet another network simulator," in *Proceeding from the 2006 workshop on ns-2: the IP network simulator*, ser. *WNS2 '06*, 2006.