Enabling Distributed Throughput Maximization in Wireless Mesh Networks - A Partitioning Approach

Andrew Brzezinski, Gil Zussman, and Eytan Modiano
Laboratory for Information and Decision Systems
Massachusetts Institute of Technology
Cambridge, MA 02139
{brzezin, gilz, modiano}@mit.edu

ABSTRACT

This paper considers the interaction between channel assignment and distributed scheduling in multi-channel multi-radio Wireless Mesh Networks (WMNs). Recently, a number of distributed scheduling algorithms for wireless networks have emerged. Due to their distributed operation, these algorithms can achieve *only a fraction* of the maximum possible throughput. As an alternative to increasing the throughput fraction by designing new algorithms, in this paper we present a novel approach that takes advantage of the inherent multi-radio capability of WMNs. We show that this capability can enable *partitioning* of the network into subnetworks in which *simple distributed scheduling algorithms can achieve 100% throughput*. The partitioning is based on the recently introduced notion of Local Pooling. Using this notion, we characterize topologies in which 100% throughput can be achieved distributedly. These topologies are used in order to develop a number of channel assignment algorithms that are based on a matroid intersection algorithm. These algorithms partition a network in a manner that not only expands the capacity regions of the subnetworks but also allows distributed algorithms to achieve these capacity regions. Finally, we evaluate the performance of the algorithms via simulation and show that they significantly increase the *distributedly achievable* capacity region.

Categories and Subject Descriptors: C.2.1 [Computer-Communication Networks]: Network Architecture and Design — *Wireless communication*; G.2.2 [Mathematics of Computing]: Graph Theory — *Graph algorithms*

General Terms: Algorithms, Performance, Design

Keywords: Stability, Channel assignment, Scheduling, Distributed algorithms, Wireless mesh networks, Local Pooling, Matroid intersection

1. INTRODUCTION

Wireless Mesh Networks (WMNs) have recently emerged as a solution for providing last-mile Internet access [1]. Several such networks are already in use, including testbeds and commercial deployments. A WMN consists of mesh routers, that form the network backbone, and mesh clients. Mesh routers are rarely mobile and usually do not have power constraints. The mesh routers are usually equipped with multiple wireless interfaces operating in orthogonal channels. Therefore, a major challenge in the design and operation of such networks is to allocate channels and schedule transmissions to efficiently share the common spectrum among the mesh routers. Several recent works focused on *multi-radio multi-channel* WMNs (e.g. [2, 3, 15, 22]). Specifically, [2, 22] study the issues of channel allocation, scheduling, and routing in WMNs, assuming that the traffic statistics are given. In this paper, we study the issues of channel allocation and scheduling but unlike most previous works, we *do not* assume that the traffic statistics are known. Alternatively, we assume a *stochastic arrival process* and present a novel approach that enables throughput maximization by distributed scheduling algorithms.

Joint scheduling and routing in a slotted multihop wireless network with a stochastic packet arrival process was considered in the seminal paper by Tassiulas and Ephremides [24]. In that paper they presented the first *centralized* policy that is guaranteed to stabilize the network (i.e. provide 100% throughput) whenever the arrival rates are within the stability region. The results of [24] have been extended to various settings of wireless networks and input-queued switches (e.g. [18, 20], and references therein). However, optimal algorithms based on [24] require repeatedly solving a *global optimization problem*, taking into account the queue backlog information for every link in the network. Obtaining a centralized solution to such a problem in a wireless network does not seem to be feasible, due to the communication overhead associated with continuously collecting the queue backlog information. On the other hand, distributed algorithms usually provide only approximate solutions, resulting in significantly reduced throughput.

Hence, the design of distributed scheduling algorithms has attracted a lot of attention recently. Lin and Shroff [17] studied the impact of imperfect scheduling on cross-layer rate control. Regarding primary interference constraints[1], they showed that using a distributed maximal matching algorithm along with a rate control algorithm may achieve as low as 50% throughput. Similar results for different settings were also obtained in [7, 8, 23, 25]. A novel distributed *ran-*

[1]Under primary interference constraints, each station can converse with at most a single neighbor at a time. Namely, the set of active links at any point of time is a matching.

MobiCom'06, September 23–26, 2006, Los Angeles, California, USA.

domized approach that can achieve 100% throughput has been recently presented in [19].

In this paper, we show that *the multi-radio and multi-channel capabilities of WMNs provide an opportunity for simple deterministic distributed algorithms to obtain 100% throughput.* Mesh routers are usually equipped with multiple radios (transceivers) and can transmit and receive on multiple channels simultaneously [2, 3, 15]. Hence, channels have to be allocated to the links and the transmissions on each link have to be scheduled to avoid collisions. By allocating different channels to different links, several non-interfering subnetworks can be constructed. We study which subnetwork topologies enable simple distributed scheduling algorithms to achieve 100% throughput. Based on these results, we develop network partitioning algorithms that decompose the network into such subnetworks.

Although in *arbitrary topologies* the worst case performance of simple distributed maximal scheduling algorithms can be very low, there are some topologies in which they *can achieve 100% throughput.* This observation is based on a recent theoretical work by Dimakis and Walrand [10] in which they study the performance of the Longest Queue First (LQF) scheduling algorithm in a graph of interfering queues[2]. The LQF algorithm is a greedy maximal weight scheduling algorithm that selects the set of served queues greedily according to the queue lengths. We note that unlike a *maximum* weight (i.e. optimal) solution a *maximal* weight solution can be easily obtained in a distributed manner. Dimakis and Walrand [10] present sufficient conditions for a maximal weight algorithm to provide 100% throughput. These conditions are referred to as *Local Pooling* (LoP) and are related to the properties of all maximal independent sets in the conflict graph.[3]

In this paper we conduct the first thorough study of the implications of the LoP conditions on the network performance. We start by presenting a motivating example demonstrating that channel allocation algorithms that take into account LoP can enable distributed throughput maximization while increasing the overall capacity. We then conduct an extensive numerical study of the satisfaction of LoP by conflict graphs of up to 7 nodes. We show that *out of 1,252 graphs, only 14 do not satisfy LoP*. It is an indication of the strength of maximal weight scheduling for achieving 100% throughput regardless of the network topology, aside from a few "bad" topologies. Due to computational limitations, exhaustively verifying the satisfaction of LoP in graphs with more than 7 nodes seems infeasible. In order to be able to utilize larger graphs, we study what general properties of conflict graphs assist or hinder the LoP conditions. For example, we show that cliques (complete graphs) that are connected to each other in different manners satisfy LoP.

These observations provide several building blocks for partitioning a graph into subgraphs satisfying LoP. In order to demonstrate this capability and for the ease of presentation, we focus on scheduling under primary interference constraints (studied in [4, 7, 8, 19, 23, 25, 26]). For example, we show that a tree network graph, when subject to the primary interference constraints, yields an interference graph which satisfies LoP. Hence, *in such a tree, maximal weight algorithms achieve 100% throughput.* We also study bipartite network graphs that provide insights regarding the number of required subgraphs. For instance, we show that in any $K_{2,n}$ bipartite graph (i.e. a $2 \times n$ input-queued switch) maximal weight matching algorithms achieve 100% throughput.

Building upon our observations, we design channel allocation algorithms. Similarly to [2] and to the static channel assignment in [15], we assume that a channel is assigned to a radio interface for an extended period of time. Under this assumption, using the minimum number of channels requires a partitioning of the network into the minimum number of subnetworks satisfying LoP. The general LoP conditions are extremely challenging to incorporate into a channel allocation algorithm. Fortunately, our study provides some useful building blocks. Since tree network graphs satisfy LoP, a possible approach (which we pursue) is to partition the network into non-overlapping forests, such that each edge will be part of a single forest and each forest will use a different channel. This problem is closely related to the *matroid intersection* and *matroid partitioning* problems.

Given k channels, the problem of partitioning the graph into k forests such that the number of edges included in the forests is maximized is referred to as the k-forest problem [11]. A simple approach is to obtain an *approximate* solution by a Breadth First Search (BFS) algorithm. Alternatively, since the k-forest problem is actually a specific case of a Matroid Cardinality Intersection problem, an *optimal* solution can be found by the Matroid Cardinality Intersection (MCI) algorithm of [16] (having polynomial complexity). We show that the MCI algorithm can be adapted to take into account the scenario in which different nodes have different numbers of radios. Using either the BFS algorithm or the MCI algorithm enables a simple distributed scheduling algorithm to achieve the capacity region (i.e. achieve 100% throughput). Yet, the capacity region itself may not be the best possible. This results from the *undesirable property* that the sizes (number of edges) of the forests are unbalanced. Therefore, and since the capacity of the largest forest may be significantly lower than the capacity of the smallest forest, the network capacity may be affected.

We present three algorithms that aim to expand the capacity region, while maintaining the LoP conditions in all the subnetworks. The main objective is to balance the number of edges across channels and to reduce the node degrees in each channel. Two of these novel capacity expansion algorithms make use of augmenting paths (in the spirit of the MCI algorithm of [16]) to balance the node degree across channels. Thus, they can be viewed as *balanced* Matroid Cardinality Intersection algorithms. We evaluate the performance of the algorithms via simulation. We show that the MCI algorithm significantly outperforms the BFS algorithms. We also compare the performance of the capacity expansion algorithms and the MCI algorithm and show that a large capacity improvement can be gained by using these algorithms. We conclude by exploring the tradeoffs between the capacities and the algorithms' complexities.

The main contributions of this paper are two-fold. First, we conduct a rigorous study of the properties of network graphs satisfying Local Pooling. The second contribution is the development of network partitioning (i.e. channel allocation) algorithms that generate subnetworks with large capacity regions, while enabling distributed throughput maximization in each of the subnetworks.

[2]A graph of interfering queues can be constructed from the network graph according to the interference constraints and is usually referred to as an interference or conflict graph [14].

[3]More technical details can be found in Section 3.

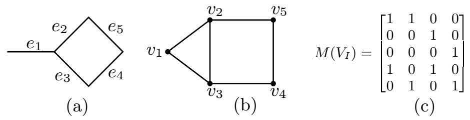


Figure 1: (a) A network graph G_N, (b) the corresponding interference graph G_I under the primary interference constraints, and (c) the matrix $M(V_I)$ of maximal independent sets in G_I.

To the best of our knowledge, this is the first attempt to study the algorithmic implications of Local Pooling. This work is not only different from previous works on distributed stability, due to the focus on partitioning mesh networks, but also different from previous works on optimizing mesh networks that mostly rely on traffic statistics.

This paper is organized as follows. In Section 2 we present the network model and formulate the problem. In Section 3 we present and clarify the LoP conditions and demonstrate their effect on the channel assignment problem. Section 4 studies the characteristics of conflict graphs satisfying LoP. In Section 5 we present network partitioning and capacity expansion algorithms and in Section 6 we evaluate their performance. We summarize the results and discuss future research directions in Section 7. Due to space constraints, some of the proofs are omitted and can be found in [6].

2. MODEL

We consider the backbone of a Wireless Mesh Network modeled by a *network graph* $G_N = (V_N, E_N)$, where $V_N = \{1, \ldots, n\}$ is the set of nodes (mesh routers) and $E_N = \{(i,j) : i,j \in V_N\}$ is the set of bi-directional links, with $m \triangleq |E_N|$. Depending on the context, we denote a link either by (i,j) or by e_k. We assume that the time is slotted, denoted by t, and that the packet length is normalized so as to be transmittable in a unit time slot. We denote by K_n a clique having n vertices and by $K_{i,j}$ a complete bipartite graph with i and j vertices.

Different wireless technologies pose different constraints on the set of transmissions that can take place simultaneously. For example, under *primary interference constraints*, the set of possible transmissions is the set of all possible matchings on G_N. More generally, in many cases an *interference graph* (also known as a conflict graph) $G_I = (V_I, E_I)$ can be defined based on the network graph G_N [14]. We assign $V_I \triangleq E_N$. Thus, each edge e_i in the network graph is represented by a vertex v_i of the interference graph and an edge (v_i, v_j) in the interference graph indicates a conflict between network graph links e_i and e_j (i.e. transmissions on e_i and e_j cannot take place simultaneously). In graph theoretic terminology, the interference graph resulting from primary interference constraints is called a *line graph* [12]. For example, Figure 1 illustrates a network graph and the corresponding interference graph under primary interference constraints (i.e. the line graph corresponding to the network graph). Here, We note that the model can be easily generalized to capture network graphs with directional links. In such a case, link (i,j) may interfere with different links than those link (j,i) interferes with. Accordingly, the interference graph will include a node for each directional link.

We consider the application of Local Pooling to multi-radio multi-channel WMNs. Following the model of [2], we assume that each node v is equipped with $R(v)$ interfaces (radios). There are k available orthogonal channels and it is assumed that each of the $R(v)$ interfaces operates on a different channel. Similarly to [2] and to the static model of [15], we consider a static channel allocation model in which a channel is allocated to each interface for an extended period of time. Such an approach enables the use of commodity 802.11 radios [2]. We note that the extension of the model for a dynamic channel allocation is a subject for further research. We assume that transmissions in different channels cannot collide. Therefore, once the different channels are allocated, k disjoint interference graphs are generated.

For the simplicity of presentation, we consider single-hop bi-directional traffic.[4] Let $A_{ij}(t)$ denote the number of packets arriving at node i or node j that need to be transmitted on link (i,j) by the end of time-slot t. $A_{ij}(t)$ can be viewed as the cumulative number of packets arriving at node (i,j) of the interference graph. We assume that arrivals are mutually independent and temporally i.i.d. processes with arrival rate λ_{ij}, that is $\mathbb{E}[A_{ij}(1)] = \lambda_{ij}$. Let the column vector $\Lambda = (\lambda_{ij}, (i,j) \in E_N)$ denote the arrival rate vector.

Let $Q_{ij}(t)$ denote the number of packets waiting to be transmitted on link (i,j) at the beginning of time-slot t and $Q(t)$ denote the queue-size vector. We will use $Q(t)$ as the system state at time t. Let $\Pi(G_N)$ denote the set of all feasible link activations in the network graph G_N. In particular, let $\pi = (\pi_{ij}, (i,j) \in E_N) \in \Pi(G_N)$ be a $(0,1)$ column vector representing a possible link activation. Under primary interference constraints, $\Pi(G_N)$ includes all possible matchings, while in general, it corresponds to all independent sets in the interference graph G_I. Following the notation of [10], we denote by $M(V_I)$ the matrix that includes all the *maximal* independent sets in G_I (i.e. all the maximal elements of $\Pi(G_N)$). For example, Figure 1(c) shows the matrix $M(V_I)$ for the interference graph G_I in Figure 1(b). We can now define the *stability region* (also known as the *capacity region*).

Definition 1 (Admissible Rate-Vector). *An arrival rate vector Λ is called admissible, if there exists a collection of link activations, $\pi_l, 1 \le l \le L$ such that*

$$\Lambda \le \sum_{l=1}^{L} \alpha_l \pi_l, \quad \alpha_l \ge 0, \quad \sum_{l=1}^{L} \alpha_l < 1.$$

Definition 2 (Stability Region). *The set of all admissible rate vectors Λ is called the stability region and is denoted by Λ^*.*

A scheduling algorithm has to select a schedule that satisfies the transmission constraints at each time slot. Let $S_{ij}(t) \in \{0,1\}$ be the indicator variable of whether link (i,j) is active at time t and $S(t)$ denote the scheduling decision vector. Then, $S(t) \in \Pi(G_N)$. Under a scheduling algorithm, the state of the system $(Q(t), t \ge 0)$ evolves according to a Markov Chain. A stable algorithm is defined as follows. We will also refer to it as an algorithm that achieves 100% throughput or a *throughput optimal algorithm.*

Definition 3 (Stable Algorithm). *A scheduling algorithm is stable, if for any admissible Λ the Markov Chain $(Q(t), t \ge 0)$ is positive recurrent.*

[4]Under this assumption, the joint routing and scheduling problem reduces to a scheduling problem.

Tassiulas and Ephremides [24] established the existence of a stable scheduling algorithm. In particular, the algorithm that schedules according to $S^*(t)$ where

$$S^*(t) = \arg \max_{\pi \in \Pi(G_N)} Q'(t)\pi \qquad (1)$$

is a stable algorithm (Q' denotes the transpose of vector Q). Given an interference graph G_I, the algorithm of [24] has to find the *maximum weight independent set* in G_I at each time slot. Namely, it has to solve an NP-Complete problem in every time slot. In the context of switch scheduling and primary interference constraints, this algorithm has to schedule the edges of the *Maximum Weight Matching* at each time slot, where the edge weights are the queue sizes. The maximum weight matching in any graph can be found in $O(n^3)$ computation time, using a centralized algorithm [16]. However in wireless networks, implementing a centralized algorithm is not feasible and distributed algorithms (e.g. [13]) can obtain only an approximate solution, resulting in a fractional throughput. Hence, even under very simple transmission constraints, it is difficult to obtain 100% throughput in a distributed manner. This motivates us to develop channel allocation methods that will enable simple distributed scheduling algorithms to obtain 100% throughput. Therefore, we provide a general definition of the *Channel Allocation Problem* below. In Section 5 we will develop algorithms for *specific* versions of this problem.

Definition 4 (Channel Allocation Problem). *Given a network graph G_N, k channels, and $R(v)$ radios at each node $v \in V_N$, assign channels to links (i,j) $\forall (i,j) \in E_N$ such that at most $R(v)$ channels are used by links adjacent to v and* simple *(e.g. greedy) distributed algorithms are stable in each subnetwork operating in a different channel.*

3. LOCAL POOLING CONDITIONS

3.1 Definitions

In this section we restate the definition and implications of Local Pooling (LoP) presented in [10]. We also present and demonstrate a somewhat simpler set of definitions. Recall that $M(V_I)$ is the collection of maximal independent vertex sets on G_I, organized as a matrix (an example appears in Figure 1). Denote by $\mathrm{Co}(M)$ the convex hull of the columns of matrix M. We now restate the definition of LoP.

Definition 5 (Local Pooling - LoP [10]). *The set of nodes (queues) $V \subseteq V_I$ satisfies local pooling, if there exists a nonzero vector $\alpha \in \mathbb{R}_+^{|V|}$ such that $\alpha'\phi$ is a positive constant for all $\phi \in \mathrm{Co}(M(V))$. Local pooling is satisfied, if every $V \subseteq V_I$ satisfies local pooling.*

In this paper, we separate the definition of Local Pooling to two different definitions and present a somewhat simpler definition for the satisfaction of LoP by a set of nodes. We show that although this definition does not take into account the convex hull of M, it is equivalent to the definition in [10]. We designate by e the vector having each entry equal to unity. We deliberately avoid specifying its size, because it will be obvious by the context of its use.

Definition 6 (Subgraph Local Pooling - SLoP). *An interference graph G_I satisfies Subgraph Local Pooling, if there exists $\alpha \in \mathbb{R}_+^{|V_I|}$ and $c > 0$ such that $\alpha' M(V_I) = ce'$.*

Lemma 1. *The definition of Subgraph Local Pooling and the satisfaction of Local Pooling by a set of nodes (Definition 5) are equivalent.*

Proof. Suppose the set of nodes $V \subseteq V_I$ satisfies local pooling as defined in Definition 5. Then, there exists $c > 0$ and $\alpha \in \mathbb{R}_+^{|V|}$ such that $\alpha'\phi = c$ for all $\phi \in \mathrm{Co}(M(V))$. Clearly each column of $M(V)$ belongs to $\mathrm{Co}(M(V))$, which gives $\alpha' M(V) = ce'$. Thus the subgraph of G_I over nodeset V satisfies SLoP. Conversely, suppose that the subgraph of G_I over nodeset V satisfies SLoP. Then there exist $c > 0$ and $\alpha \in \mathbb{R}_+^{|V|}$ such that $\alpha' M(V) = ce'$. Now consider $\phi \in \mathrm{Co}(M(V))$, which must equal by definition $M(V)\beta$ for $\beta \in \mathbb{R}_+^{|M(V)|}$ with $e'\beta = \sum_j \beta_j = 1$, $\beta_j \geq 0, \forall j$ and $|M(V)|$ equal to the number of columns in $M(V)$. Then, we have $\alpha\phi = \alpha M(V)\beta = ce'\beta = c$. Note that this value is constant regardless of the choice of ϕ. Thus, the set of nodes V satisfies local pooling as defined in Definition 5. □

We can now define the notion of Overall Local Pooling which requires that Subgraph Local Pooling (SLoP) will be satisfied in any subgraph of a given interference graph induced by selecting a *subset of the nodes*.

Definition 7 (Overall Local Pooling - OLoP). *Interference graph G_I satisfies Overall Local Pooling if each induced subgraph over the nodes $V \subseteq V_I$ satisfies SLoP.*

We continue with the example of the interference graph G_I and the corresponding matrix $M(V_I)$ depicted in Figure 1. We can see that G_I satisfies SLoP since for $\alpha = (1,1,1,1,1)$, $\alpha' M(V_I) = 2e'$. Similarly, the subgraph composed of the vertex set $\{2,3,4\}$ satisfies SLoP, since for $\alpha = (1,1,0)$, $\alpha' M(\{2,3,4\}) = e'$. In a similar manner, it can be shown that all subgraphs of G_I satisfy SLoP, and therefore, G_I satisfies OLoP.

We can now describe the stability of the system when the service in each time slot is scheduled according to the Longest Queue First (LQF) algorithm. This algorithm is an iterative greedy algorithm that selects the node of G_I with the longest queue, and removes it and its neighbors from the interference graph. This process is repeated successively until no nodes remain in the graph. When two queues have the same length a tie-breaking rule has to be applied. The set of selected nodes is a maximal independent set in the interference graph. Hence, since the nodes are selected according to their weights, we will refer to the LQF algorithm as the Maximal Weight Independent Set algorithm. Such a greedy algorithm can be easily implemented in a distributed manner. In [10] the following theorem is proved:

Theorem 1 (Dimakis and Walrand, 2006 [10]). *If interference graph G_I satisfies the OLoP conditions, a Maximal Weight Independent Set scheduling algorithm achieves 100% throughput.*

To conclude, the satisfaction of OLoP by an interference graph is a *sufficient* condition for distributed maximal weight algorithm to be throughput optimal (i.e. in that case, there is no need to obtain an optimal solution to (1) in each slot).

3.2 Channel Allocation Example

The following simple example demonstrates the application of the LoP conditions, presented above, to a channel allocation (network partitioning) problem. We consider the

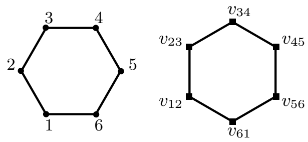


Figure 2: A 6-node ring network graph and the corresponding interference graph.

6-node ring network graph, depicted on the left in Figure 2. Under the primary interference constraints, this graph has a corresponding 6-node ring interference graph representation, which is illustrated on the right in Figure 2. Under primary interference constraints, the maximal weight independent set in the interference graph is equivalent to the maximal weight matching in the network graph. A maximal weight matching can be obtained in a distributed manner by the greedy algorithm of Hoepman [13].

If a single radio is located at each node of the 6-node ring illustrated in Figure 2(a), then no two adjacent edges can be simultaneously active. The *stability region* Λ^* is then characterized by the following inequalities:

$$\begin{aligned} &\lambda_{12}+\lambda_{23}\le b,\ \lambda_{23}+\lambda_{34}\le b,\ \lambda_{34}+\lambda_{45}\le b,\\ &\lambda_{45}+\lambda_{56}\le b,\ \lambda_{56}+\lambda_{61}\le b,\ \lambda_{61}+\lambda_{12}\le b, \end{aligned} \qquad (2)$$

where $b = 1$. This stability region can be achieved by a centralized algorithm that finds a maximum weight matching (i.e. obtains the optimal solution to (1)) in each time slot.

It was shown in [10] that in the 6-node ring, OLoP does not hold, and that in general a *maximal* weight matching algorithm does not achieve 100% throughput in the 6-node ring[5]. According to [17], a *maximal* weight matching algorithm can only guarantee stability for arrival rates that are 50% of the rates in the region above (Λ^*). Hence, the guaranteed distributedly achievable region is given by (2) with $b = 0.5$.

If we allow two channels to be used simultaneously, and provide two transceivers to each node, then in every time slot a node can transmit two packets on the selected link (similarly to a speedup of two, defined in [9]). Thus, the guaranteed achievable region (using maximal weight matching) is again given by (2) with $b = 1$.

Alternatively, links $(1,2),(2,3)$, and $(3,4)$ can use one channel, while the remaining links use the other channel. The interference graph on each channel is now a tree (e.g. the line connecting v_{12}, v_{23}, and v_{34}). Since [10] shows that the maximal weight independent set algorithm is throughput optimal in *tree* interference graphs, the *distributedly achievable* stability region is now given by

$$\begin{aligned} &\lambda_{12}+\lambda_{23}\le 1,\ \lambda_{23}+\lambda_{34}\le 1,\\ &\lambda_{45}+\lambda_{56}\le 1,\ \lambda_{56}+\lambda_{61}\le 1. \end{aligned} \qquad (3)$$

This provides a strict performance improvement over the region achievable by using two channels (speedup of two) in the interference graph represented in 2(b). Yet, it is clear that this channel allocation is not the best possible: the allocation in which links $(1,2),(3,4)$, and $(5,6)$ use one channel, while the remaining links use the other channel can provide each network link with a stable rate of one unit per time slot (i.e. $\lambda_{ij} \le 1\ \forall (i,j) \in E_N$).

For a general network operating under primary interference constraints with a speedup of two (similar to allocating two channels to each link), a greedy maximal weight algorithm (implementable in a distributed manner) can achieve the network stability region Λ^* [17]. Our example above shows for a particular network scenario that when two channels are allocated such that each component satisfies OLoP, the stability region (that can be *achieved* by a distributed algorithm) is *strictly larger* than the original stability region Λ^*. This strict performance improvement can be demonstrated in any network with primary interference constraints that can be partitioned into two non-trivial components satisfying OLoP (for more details see [6]).

This simple example demonstrates that careful channel allocation that takes into account topologies that satisfy OLoP can provide provable and significant improvements over arbitrary channel allocation. Moreover, it shows that partitioning into different OLoP-satisfying components can result in different capacity regions. Therefore, it provides the motivation to study the characteristics of network topologies satisfying OLoP and to design channel allocation algorithms that take advantage of these characteristics.

4. A STUDY OF LOCAL POOLING

4.1 Exhaustive Numerical Search

We performed a numerical study in which we searched over all interference graphs of up to 7 nodes. We employed Mathematica to identify all simple graphs, and Matlab to determine the maximal configurations (i.e. to obtain the matrices $M(V_I)$) and to verify the satisfaction of the OLoP conditions for each interference graph. The OLoP conditions are based on the SLoP conditions that were verified using the following linear program presented in [10].

$$\begin{aligned} c^* = {} & \max_{c,\mu,\nu} c \\ \text{s.t. } & M(V_I)\mu \ge M(V_I)\nu + ce \\ & e'\mu = 1,\ e'\nu = 1 \\ & \mu,\nu \in \mathbb{R}_+^{|V_I|},\ c \in \mathbb{R} \end{aligned}$$

It has been shown in [10, Proposition 1] that the graph G_I satisfies SLoP if and only if $c^* = 0$.

In order to simplify the presentation of the numerical results, we first show that the OLoP conditions are satisfied by the disjoint union of two graphs (not sharing any vertices in common) satisfying the OLoP conditions. This allowed us to restrict our search to connected simple graphs.

PROPOSITION 1. *A graph $G_I = G_I^1 \cup G_I^2$ (disjoint union) satisfies OLoP, if and only if G_I^1 and G_I^2 satisfy OLoP.*

PROOF. Suppose G_I satisfies OLoP. Consider all induced subgraphs restricted to the vertices of G_I^1. Then, any such induced subgraph satisfies the SLoP conditions by our assumption that G_I satisfies OLoP. Thus, G_I^1 satisfies OLoP. The same reasoning provides that G_I^2 satisfies OLoP.

Suppose that G_I^1 and G_I^2 satisfy OLoP. Then, any induced subgraph of G_I can be split into disjoint induced subgraphs on G_I^1 and G_I^2. For the induced graph on G_I^1, our assumption provides that there exists nonzero $\alpha_1 \ge 0$ that multiplies

[5] In [10], it was shown that under *restricted* arrival processes (subject to a variance constraint and a large deviation bound), a maximal weight matching algorithm is stable in the 6-node ring. In this work the arrival processes are not restricted in this way.

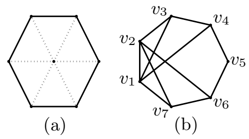


Figure 3: 7-node graphs that fail OLoP: (a) configurations where the induced graph over the outer 6 nodes is a 6-ring (the dotted lines indicate edges that can exist), (b) The only 7-node graph that has no induced 6-ring subgraph and fails SLoP.

any maximal independent vector on the induced subgraph to yield a constant c_1. Similarly, there exists α_2 and c_2 for the induced subgraph on G_I^2. Every maximal independent set of the induced subgraph of G_I must be the disjoint union of a maximal independent set of the induced subgraph on G_I^1 and a maximal independent set of the induced subgraph on G_I^2. Thus, the augmented vector (α_1, α_2) must yield a constant value of $c_1 + c_2$ for all maximal independent sets of the induced subgraph on G_I. $\square$

We note that in the following section we will present several additional theoretical results regarding LoP in general graphs. A specific case of one of the results that will be presented there (Lemma 2) is that graphs that have a node with degree 1 satisfy SLoP. This allowed us to restrict our search to graphs that do not have vertices of degree 1, thereby significantly reducing the computation time. We first considered all connected interference graphs having up to 5 vertices that do not have vertices of degree 1. There are 15 such graphs. We obtained the following numerical result.

Numerical Result 1. *All connected simple graphs of up to* 5 *nodes that do not have vertices of degree* 1 *satisfy SLoP.*

This immediately implies that all graphs having up to 5 vertices (there are 52 such graphs) satisfy OLoP. Next, we considered graphs of 6 vertices (there are 61 such connected graphs without degree 1) and obtained the following result.

Numerical Result 2. *All graphs of* 6 *vertices except the* 6*-node ring satisfy SLoP.*

Numerical Results 1 and 2 together imply that all graphs of up to 6 vertices except the 6-node ring satisfy OLoP.

Finally, we considered all graphs of 7 vertices. We first removed from consideration all such graphs having a 6-ring as an induced subgraph, since due to the failure of SLoP in a 6-ring, OLoP fails in these graphs by definition. There are 12 such graphs, and their general form is depicted in Figure 3(a). Among the remaining graphs of 7 vertices, we can then guarantee that there are no induced subgraphs, having 6 vertices or fewer, that fail the SLoP conditions.

Numerical Result 3. *There is one graph of* 7 *vertices which does not have an induced* 6*-ring on any subset of* 6 *nodes that fails the SLoP conditions. This graph is depicted in Figure 3(b).*

To conclude, almost all 1,252 graphs of up to 7 nodes satisfy OLoP (specifically, 14 fail OLoP). All attempts at numerical evaluations for graphs of greater than 7 vertices suffered computational difficulty. Therefore, in the following section we focus on generating large graphs satisfying OLoP from small components.

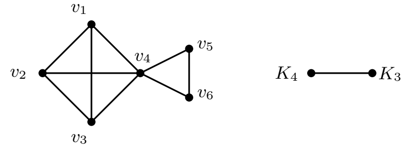


Figure 4: An interference graph composed of two cliques and the corresponding *tree of cliques* graph.

4.2 Constructive Approach

Our first observation is about connecting a graph and a clique (complete graph).

Lemma 2. *If G_I satisfies OLoP, then the graph G_I^*, which consists of G_I sharing a single vertex with clique $K_n, n \geq 2$, satisfies OLoP.*

Proof. Assume that G_I satisfies OLoP. Denote by v the vertex of G_I that is shared with clique K_n. We need only consider the induced subgraphs of G_I^* containing a vertex $v^* \neq v$ belonging to the clique K_n, since all other induced subgraphs are subgraphs of G_I and satisfy SLoP by our initial assumption. Clearly, the maximal independent sets of any such induced subgraph (whose vertex set is designated by V) either include vertex v or v^*, but never both vertices. Consequently, the vector α having all zero entries except at the indices corresponding to vertices of K_n, where the entries are set to 1, yields $\alpha' M(V) = e'$. Thus, such a subgraph satisfies SLoP. This holds for all induced subgraphs of G_I^* that include v^*, and we conclude that G_I^* satisfies OLoP. $\square$

From the proof of Lemma 2 it can be seen that a graph that has a node with degree 1 (such a graph can be viewed as a graph G_I sharing a node with K_2) satisfies SLoP. Recall that we have used this result in Section 4.1 to reduce the number of graphs in our numerical search. Moreover, the observation in [10] that any interference graph that is a tree (or forest) satisfies OLoP can be immediately obtained using Lemma 2. We note that in Section 4.3 we will show that even under the simple primary interference constraints, the only interference graph that can be a tree is a line. Therefore, we now study more complicated interference graphs.

Lemma 3. *Every complete graph satisfies OLoP.*

Proof. Consider the complete graph $G_I = K_n$. Then clearly any subset of the nodes of G_I, labeled V, also generates a complete induced subgraph. Each maximal independent set of a complete graph can only contain one vertex, from which we conclude that $M(V)$ is the identity matrix of size $|V|$. Thus, we can use $\alpha = e$, which yields $\alpha' M(V) = e'$ for any V, from which we conclude that every induced subgraph satisfies SLoP, and consequently that G_I satisfies OLoP. $\square$

We define a *tree of cliques* as follows (an example is provided in Figure 4) and derive the following Theorem.

Definition 8. *A* tree of cliques *is composed of cliques connected to each other in a tree structure. Its nodes can be equated to cliques and its edges imply a shared vertex between two adjacent cliques. No vertex can be shared by more than two adjacent cliques.*

Theorem 2. *A tree of cliques satisfies OLoP.*

PROOF. Consider any clique G_I^1 on the tree. By Lemma 3 this clique satisfies OLoP. Then, consider any clique adjacent to G_I^1 in the tree of cliques, and denote the graph of the two combined cliques G_I^2. Since G_I^1 and the adjacent clique share only a single vertex, we can apply Lemma 2 to conclude that G_I^2 satisfies OLoP. By iteratively adding successive cliques to the overall graph under consideration, we see that each resulting graph must satisfy OLoP by Lemma 2. Thus, the overall tree of cliques must satisfy OLoP. $\square$

The next theorem considers cliques connected by disjoint edges, where no two connecting edges share any vertices in common. Consequently, at most $\min\{m, n\}$ edges can connect K_m and K_n while maintaining an overall simple graph. The proof of Theorem 3 is omitted and can be found in [6]. It considers four possible subgraph configurations and demonstrates SLoP for each type. The main idea is that each clique usually contributes a single vertex to every maximal independent set of each subgraph.

THEOREM 3. *If two cliques are connected by any number of disjoint edges, the combined graph satisfies OLoP.*

We now consider a generalized structure of the one defined in Definition 8, which we term "tree-of-blocks". Here, we generalize the types of structures that can correspond to each vertex of a tree. We have already shown that a clique is one such structure. We next show that two cliques connected by any number of disjoint edges is another such structure. As before, we require that two "blocks" can only share at most one vertex in common. The proof of the following theorem is along similar lines as the proof of Theorem 3 and can also be found in [6].

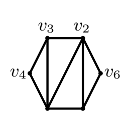
THEOREM 4. *A "tree-of-blocks", where each block is either a clique $K_n, n \geq 2$ or a pair of cliques $K_n, K_m,\ n, m \geq 1$, connected by any number of disjoint edges, satisfies OLoP.*

4.3 Primary Interference Constraints

As mentioned above, the primary interference constraints yield an interference graph G_I which is the line graph of the network graph G_N. In this section, we study the restrictions imposed on such interference graphs. We begin by considering the only 7-node graph, which does not have an induced 6-ring, that failed SLoP (depicted in Figure 3(b)).

PROPOSITION 2. *Under primary interference constraints, the interference graph presented in Figure 3(b) cannot correspond to any valid network graph.*

PROOF. According to [12] a graph is a line graph, if and only if it does not contain any one of 9 specific induced subgraphs. In particular, the following graph is one of the 9 subgraphs, with vertices of Figure 3(b) labeled appropriately to show the correspondence.



$\square$

We conclude that only the 6-ring leads to failure of the OLoP conditions in any network graph having 7 edges or fewer. By similar arguments, we can show that other interference graphs cannot exist under primary interference constraints. For example, we can show that there is no network

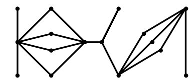


Figure 5: An example of a network graph whose interference graph satisfies OLoP.

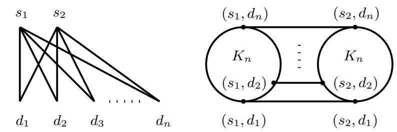


Figure 6: A network graph for a $K_{2,n}$ bipartite graph ($2 \times n$ input-queued switch) and the corresponding interference graph.

graph whose interference graph (line graph) is a tree having a node degree greater or equal to 3. Any such tree has as an induced subgraph the complete bipartite graph $K_{1,3}$ (also known as the "claw"). According to [12], the existence of such an induced subgraph precludes the possibility that this interference graph is the line graph of any network graph.

Although there is no interference graph that is a tree, a network graph that is a tree can of course exist. It can be shown that the interference graph of such a network graph is always a tree of cliques, defined in Definition 8. The following corollary is an immediate result of Theorem 2. According to this corollary, *maximal weight matching algorithms are stable (provide 100% throughput) in trees.* To the best of our knowledge, this corollary provides the first non-trivial network structure in which simple distributed algorithms are stable. The channel allocation algorithms that will be presented in Section 5 are based on this observation.

COROLLARY 1. *Under primary interference constraints, the interference graph of a tree network graph satisfies OLoP.*

Based on the results presented in Section 4.2, we can construct other non-trivial networks in which maximal weight matching algorithms are stable. For example, Theorem 4 implies that the network described in Figure 5 satisfies OLoP, and thus is stable under distributed scheduling. Developing network partitioning algorithms that efficiently take advantage of such topologies is a subject for further research.

We have obtained additional results that concern bipartite graphs. Although mesh networks are usually not bipartite, bipartite graphs provide insight regarding the performance of our partitioning algorithms. Since input-queued switches are bipartite graphs with primary interference constraints, an additional byproduct is insight regarding switches. The following corollary generalizes a recent result presented in [5] regarding a 2×2 input-queued switch.

COROLLARY 2. *A maximal weight matching algorithm achieves* 100% *throughput in a $K_{2,n}$ bipartite graph (i.e. in a $2 \times n$ input-queued switch).*

PROOF. A $K_{2,n}$ bipartite network graph is depicted on the left in Figure 6. Its interference graph can then easily be shown to be two cliques of size n (K_n), connected by n disjoint edges, as depicted on the right in Figure 6. The result is then directly derived from Theorem 3. $\square$

It follows that a $K_{4,n}$ bipartite graph can be partitioned into two subgraphs, each of whose interference graphs satisfies OLoP. In Section 5.2, we will use this observation to evaluate the performance of our channel allocation algorithms.

5. CHANNEL ALLOCATION

The Channel Allocation Problem, introduced in Definition 2, seeks to assign a channel to every link such that each partition (operating in a different channel) can achieve 100% throughput by a distributed maximal weight scheduling algorithm. In this section our objective is to develop channel allocation algorithms that: (i) provide a large stability region and (ii) allow simple distributed algorithms to achieve 100% throughput in this region. As in Section 4.3, in order to demonstrate the presented concept, we assume that primary interference constraints hold.

In terms of LoP conditions, we seek to partition the network edges into channels such that the interference graph in each channel satisfies OLoP. The OLoP requirement is extremely challenging to incorporate into an optimization algorithm that generates a channel allocation, because it seeks the SLoP property for every subgraph on each channel. However, Corollary 1 shows that network graphs that are trees satisfy OLoP. Thus, it is sufficient to partition the edges of the network graph into channels such that each channel's network graph is a forest. This is the basis for our channel allocation algorithms.

Our channel allocation problem is equivalent to a coloring problem on the network graph. Namely, we seek to color the network edges such that edges of a single color do not compose a cycle (i.e. each color composes a forest). The minimum number of colors is known as the graph arboricity and can be found by an $O(m^2)$ algorithm [11].

Initially, we assume that all nodes have the same number of radios and that this number is equal to the number of channels (i.e. $R(v) = k \; \forall v \in V_N$).[6] When the number of available colors (channels) k is fixed, the k-forest problem [11, 16] seeks to find the maximum number of edges of the graph that can be colored using only k colors without closing a single color cycle. This problem can be formulated as a *matroid*[7] *partitioning* or a *matroid intersection* problem. In order to enable the development of capacity expansion algorithms, we focus on the matroid intersection formulation. Under this formulation, the k-forest problem makes use of two matroids: the *graphic matroid* and the *partition matroid*. In our setting, we define these matroids by considering the graph $G_N^k = (V_N^k, \mathcal{E})$, equal to k disjoint *copies* of the network graph G_N. The graphic matroid $\mathcal{M}_1 = (\mathcal{E}, \mathcal{I}_1)$ assigns to $\mathcal{I}_1$ all possible forests in G_N^k. The partition matroid $\mathcal{M}_2 = (\mathcal{E}, \mathcal{I}_2)$ partitions $\mathcal{E}$ into $m \triangleq |E_N|$ sets, where the i-th set, $\mathcal{E}_i$, contains all k copies of edge i. The collection $\mathcal{I}_2$ contains all sets of edges that have no more than a single element in any set of the partitions: $I \in \mathcal{I}_2$ implies $|I \cap \mathcal{E}_i| \leq 1$ for $i = 1, \ldots, m$. By associating with each copy of G_N in G_N^k a *unique color*, it can be seen that the sets belonging to $\mathcal{I}_1 \cap \mathcal{I}_2$ can be equated to colorings, where each subgraph of a particular color is a forest. This directly corresponds to a valid channel allocation, where each channel's network graph is a forest. The k-forest problem is to find for a given k the largest set of edges belonging to the matroid intersection of the graphic and partition matroids.

5.1 Partitioning Algorithms

Our first algorithm for the k-forest problem is the suboptimal Breadth-First Search (BFS) algorithm. Such an algorithm was used in [21] as a heuristic solution to this problem. Its major advantage is its low complexity of $O(k(m + n))$. Yet, in Section 6 we will show that there is a large gap between the BFS solution and the optimal solution.

Therefore, we selected an optimal algorithm as a basis for developing our capacity expansion algorithms. The optimal solution to the k-forest problem can be found in polynomial time [11, 16] by several algorithms. One of these algorithms is the *Matroid Cardinality Intersection* (MCI) algorithm of Lawler [16]. Given a valid coloring $I \in \mathcal{I}_1 \cap \mathcal{I}_2$, the MCI algorithm searches for an *augmenting path*, consisting of an alternating sequence of edges not in I and edges in I, such that when the edges of the path belonging to I are removed from I and those not belonging to I are added, the resulting coloring (channel allocation) belongs to $\mathcal{I}_1 \cap \mathcal{I}_2$ and its cardinality has increased by 1 (for more details see [16]). The complexity of the MCI algorithm is $O(km^2n' + k^2mn(n')^2)$, where $n' = \min\{n, m/k\}$. In the description of the following algorithms, we refer to two copies of the same edge on different colors in G_N^k as *parallel edges*.

Our channel allocation framework admits the practical situation where each node v is equipped with $R(v)$ radios (interfaces). Namely, different nodes have a different number of radios. In the formulation of the matroid intersection problem, we define the graph G_N^k as the disjoint union of k *identical* copies of the network G_N. This corresponds to the case, where each node is equipped with exactly k radios. Essentially, rather than generating k copies of each network graph edge, each network link should only have an edge represented in the i-th copy of the network graph G_N when there is a radio for that link available for use of the i-th channel.[8] Without loss of generality we refer to any graph defined in this manner as $G_N^k = (V_N^k, \mathcal{E})$. The matroid intersection properties, the MCI algorithm, and the algorithms described in Section 5.2 can then be applied to G_N^k.

Once the channel allocation is performed, at each time slot, one can use the distributed approximation algorithm of [13] that finds the maximal weight (greedy) solution, thereby providing 100% throughput. The (local) computational complexity of this algorithm is $O(1)$. This complexity is extremely low relative to the $O(n^3)$ [16] complexity of a centralized optimal algorithm required to solve (1). In addition, the centralized algorithm has to collect queue backlog information from all the nodes at each time slot (for an extended comparison see [19]).

In the realistic situation where the number of channels k is fixed and *insufficient* to partition all the network edges into k forests, we apply the MCI algorithm (or BFS) to generate an initial allocation that is a k-forest, and assign the unallocated network edges to the k-th channel. Thus, the first $k-1$ channels are guaranteed to satisfy OLoP, while the k-th channel operates at a worst-case 50% throughput.

A (theoretical) optimal solution will partition the graph

[6] We will show below that this assumption can be relaxed.

[7] A matroid is a combinatorial structure $\mathcal{M} = (\mathcal{E}, \mathcal{I})$ in which $\mathcal{E}$ is a finite set of elements, and $\mathcal{I}$ is a collection of subsets of $\mathcal{E}$ satisfying (i) $\emptyset \in \mathcal{I}$, and if $I \in \mathcal{I}$, then all proper subsets of I belong to $\mathcal{I}$, and (ii) if $I_1, I_2 \in \mathcal{I}$ with $|I_2| = |I_1| + 1$, then there exists $e \in I_2$ such that $I_1 \cup \{e\} \in \mathcal{I}$.

[8] When different nodes have a different number of radios, the specific allocation of the links to the different copies may affect the capacity region. An efficient allocation algorithm is a subject for further research.

into the minimum number of OLoP satisfying components, whereas our algorithms partition into forests. In order to evaluate the performance of our algorithms, we consider complete bipartite graphs. It can be shown that two channels are necessary and sufficient to guarantee the satisfaction of OLoP in $K_{3,3}$. Applying MCI, we find that the arboricity of $K_{3,3}$ is 2 and conclude that MCI achieves the minimum number of channels to guarantee OLoP. This and similar results point to the strong performance of the MCI algorithm in partitioning the network into a small number of channels satisfying OLoP. Yet, the following lemma provides a lower bound on the performance in general. Before presenting the lemma, we define $\kappa^*(G_N)$ as the minimum number of channels necessary to partition the edges of a network graph G_N such that the interference graph of each partitioned subgraph satisfies OLoP.

LEMMA 4. *For $\varepsilon > 0$ there is no approximation algorithm that partitions a network graph G_N into $\kappa(G_N)$ forests, where*

$$\kappa(G_N) \leq (1.5 - \varepsilon)\kappa^*(G_N), \forall G_N.$$

PROOF. Consider a $K_{4,4}$ bipartite network graph. It can be partitioned into two $K_{2,4}$ network graphs. According to Corollary 2, under primary interference constraints, an interference graph of $K_{2,4}$ satisfies OLoP. Therefore, 2 channels are sufficient to guarantee the satisfaction of OLoP in $K_{4,4}$. Namely, $\kappa^*(K_{4,4}) = 2$. Since $K_{4,4}$ has 8 nodes, any forest in such a graph can have at most 7 edges. Since $K_{4,4}$ has 16 edges, its arboricity must be at least 3 (i.e. $\kappa(K_{4,4}) = 3$). Hence, there exists a graph G_N for which $\kappa(G_N) = 1.5\kappa^*(G_N)$. □

5.2 Capacity Expansion Algorithms

An important undesirable feature of the MCI and BFS algorithms is that each successive channel has a *maximal* number of network edges assigned to it, given the assignment to the previous channels. We wish to balance the trees in order to expand the capacity, thereby expanding the achievable throughput.

We present three algorithms for improving the network capacity properties. Since the admissible region restricts the summed throughput of all edges incident on the same vertex in the network graph to 1, it is desirable to minimize the maximum vertex degree over the network graphs on each channel. The first algorithm is called R-GREEDY, and it operates by greedily selecting edges incident on vertices of maximum degree and seeking any channel that they can be reallocated to, such that the new allocation belongs to $\mathcal{I}_1 \cap \mathcal{I}_2$ and the allocation has an improved maximum degree. We note that $e = (v_i, v_j)$ implies that $v_i \in e$ and $v_j \in e$. The algorithm makes use of the function $\mathrm{TF}_1(I)$, which returns a negative value when the maximum degree or number of vertices at maximum degree under allocation I improves upon that of a reference allocation, I_0.

$$\mathrm{TF}_1(I) = \Delta_I^* - \Delta_{I_0}^* + 1_{\{\Delta_I^* = \Delta_{I_0}^*\}} \left(\textstyle\sum_v 1_{\{\Delta_I(v) = \Delta_I^*\}} - \sum_v 1_{\{\Delta_{I_0}(v) = \Delta_{I_0}^*\}} \right).$$

Above, $\Delta_I(v)$ denotes the degree of vertex v in graph (V_N^k, I), Δ_I^* indicates the maximum vertex degree in graph (V_N^k, I), and $1_{\{.\}}$ is the indicator function. The complexity of the R-GREEDY algorithm is $O(dnmkn')$, where d is the maximum vertex degree in G_N.

Algorithm Greedy Reallocation (R-GREEDY)

begin with any edge set $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ (this could be the output of BFS or MCI)
repeat
  $I_0 \leftarrow I$
  if $\exists e_1 \in I, e_2 \notin I$ such that $\exists v \in e_1, \Delta_I(v) = \Delta_I^*$, $\mathrm{TF}_1((I \setminus \{e_1\}) \cup \{e_2\}) < 0$ **then**
    $I \leftarrow (I \setminus \{e_1\}) \cup \{e_2\}$
until I equals I_0

Our second and third capacity expansion algorithms search for capacity improvements by directly attempting to balance the vertex degrees over all channels. They make use of augmenting paths in the spirit of the MCI algorithm to find new locations for edges that are incident on heavily-loaded vertices. The *maximum degree reallocation* algorithm (R-MAXD) seeks to minimize the maximum degree over vertices in all channels. It proceeds by disabling edges incident on maximum degree vertices and searching for augmenting paths that do not use such edges. The algorithm uses the function TF_1 for evaluating channel allocations, and the function $\mathrm{ESF}_1^0(I)$ for selecting candidate edges to disable. $\mathrm{ESF}_1^0(I)$ returns all edges incident on vertices having maximum degree in graph (V_N^k, I),

$$\mathrm{ESF}_1^0(I) = \{e \in I : v \in e,\, \Delta_I(v) = \Delta_I^*\}.$$

The *average degree reallocation* algorithm (R-AVGD) seeks to reduce *any* vertex degree in the graph so long as the reduction does not lead to higher vertex degrees or more vertices of maximum degree elsewhere in the graph. R-AVGD employs the performance evaluation function TF_2,

$$\mathrm{TF}_2(I) = \textstyle\sum_{i=1}^{\Delta_I^*} 2^i \mathrm{sign}\left(\sum_v 1_{\{\Delta_I(v)=i\}} - 1_{\{\Delta_{I_0}(v)=i\}} \right).$$

Above, the function $\mathrm{sign}(x) = -1$ if $x < 0$, $\mathrm{sign}(x) = 1$ if $x > 0$, and $\mathrm{sign}(0) = 0$. The function $\mathrm{TF}_2(I)$ returns a negative value when the first entry at which the degree sequence[9] of (V_N^k, I) differs from that of (V_N^k, I_0) is lower in the sequence of (V_N^k, I) than that in (V_N^k, I_0). This function encourages trading higher degree vertices for more vertices of lower degree. R-AVGD also makes use of the function $\mathrm{ESF}_2^v(I)$, which returns all edges incident on vertex v in I, $\mathrm{ESF}_2^v(I) = \{e \in I : v \in e\}$. We simultaneously present both algorithms as Algorithms 1/2, making use of the parameter PARAM_i, with $\mathrm{PARAM}_1 = \{0\}$, and $\mathrm{PARAM}_2 = V_N^k$.

Algorithm 1/2 Maximum Degree/Average Degree Reallocation algorithms (R-MAXD $[i = 1]$/R-AVGD $[i = 2]$)

begin with any edge set $I \in \mathcal{I}_1 \cap \mathcal{I}_2$
repeat
  $I_0 \leftarrow I$
  for $v \in \mathrm{PARAM}_i$ **do**
    $I \leftarrow \arg\min_{\tilde{I}} \{\mathrm{TF}_i(\tilde{I}) :$
    $\tilde{I} = \text{CE-MCI}(I,\{e\},\mathrm{ESF}_i^v,\mathrm{TF}_i,1),\ e \in \mathrm{ESF}_i^v(I)\}$
until I equals I_0

R-MAXD and R-AVGD employ the recursive procedure CE-MCI that successively disables edges until an improved augmenting path is found, or all possible configurations are

[9]The degree sequence of a graph G is a *nondecreasing* sequence of the vertex degrees of G.

Algorithm CE-MCI(I_0,E_0,ESF,TF,Depth)

$\mathcal{I} = \{I_0 \setminus E_0\}$
while $\exists I \in \mathcal{I}$ with $|I| < m$ **do**
  $\mathcal{I} \leftarrow \mathcal{I} \setminus \{I\}$
  remove labels from all edges; assign $I_+ = I_- \leftarrow \emptyset$
  label '+' on every edge e such that $I \cup \{e\} \in \mathcal{I}_1$ and $e \cap E_0 = \emptyset$
  while $e =$ [edge with oldest unscanned label] $\neq \emptyset$ **do**
    if e is labeled '+' *and* $I \cup \{e\} \in \mathcal{I}_2$ **then**
      trace the alternating path of '+' and '-' labels that lead to the '+' label at e by assigning edges labeled '+' to I_+ and those labeled '-' to I_-
      $\mathcal{I} \leftarrow \mathcal{I} \cup \{(I \setminus I_-) \cup I_+\}$
    else if e is labeled '+' **then**
      label '-' on the edge in I that is parallel to e (if the edge is unlabeled)
    else
      label '+' on each unlabeled edge in the unique cycle in $(V_N^k, I \cup \{e\})$
$\mathcal{I} \leftarrow \mathcal{I} \cup \{I_0\}$; $I_{\text{rmci}} \leftarrow \arg\min_{I \in \mathcal{I}} \text{TF}(I)$
if $\text{TF}(I_{\text{rmci}}) = \text{TF}(I_0)$ **then**
  (failed to generate an improved augmenting path)
  if Depth $<$ D_MAX **then**
    $I_{\text{rmci}} \leftarrow \arg\min_I \{\text{TF}(I) :$
    $I = \text{CE-MCI}(I_0, E_0 \cup \{e\}, \text{ESF}, \text{TF}, \text{Depth}+1),$
    $e \in \text{ESF}(I_0 \setminus E_0)\}$
  else
    $I_{\text{rmci}} \leftarrow I_0$
return I_{rmci}

exhausted. CE-MCI takes as input the initial channel allocation I, the set of edges E_0 to exclude when it attempts to search for augmenting paths, the functions ESF and TF, and an integer to track the depth of the recursion. The maximum depth of the recursion can be set using the constant D_MAX. While the MCI algorithm modifies the channel allocation at each iteration upon the discovery of its first augmenting path, CE-MCI labels over the entire graph and *selects the best augmenting path available* between all such paths found, in terms of the function TF.

The complexity of the algorithms is a function of the complexity of the MCI algorithm, which we denote by $c(\text{MCI})$. The complexity of R-MaxD is $O(dnm^{\text{D\_MAX}}c(\text{MCI}))$ and of R-AvgD is $O(d^{\text{D\_MAX}}nmc(\text{MCI}))$. As long as the search depth D_MAX is low, the complexity is reasonable. In the following section, we will see that significant capacity improvement is achieved for D_MAX = 2.

6. PERFORMANCE EVALUATION

The partitioning and capacity expansion algorithms presented in Section 5 were implemented in Matlab and tested on numerous randomly generated networks. In this section we briefly describe the numerical results obtained for a number of representative cases. All presented results have been obtained for randomly generated instances in which the nodes are uniformly distributed in a plane of size $1000m \times 1000m$, with a link existing between two nodes if the distance between them is at most $250m$. We intentionally present results regarding relatively dense networks, since in very sparse networks the partitioning solution is often trivial and does not shed light on the tradeoffs involved in capacity expansion. As in the previous sections, we assumed that primary interference constraints hold. The presented results were obtained under the assumption that the number of radios is equal to the number of channels and is the same for all the nodes (i.e. $R(v) = k\ \forall v$). As described in Section 5.1, this assumption can be easily relaxed.

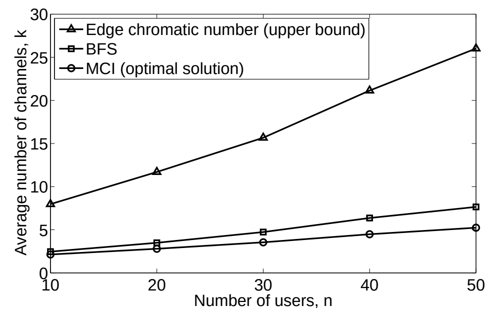


Figure 7: Average number of channels in the optimal solution, the number required by the BFS algorithm, and the upper bound.

6.1 Partitioning Algorithms

Figure 7 compares the average number of channels (k) required by the BFS and the MCI algorithms. The results are presented as a function of the number of nodes in the network (n), where for each value of n, the average was obtained over 100 different random instances. Over all cases tested, the BFS algorithm required on average 32% more channels than the optimal MCI algorithm. Such a performance gap was observed throughout our numerical studies. Consequently, it seems that despite the higher computational complexity, using a matroid intersection algorithm is beneficial. This is one of the reasons the MCI algorithm was chosen as the basis for our capacity expansion algorithms.

Figure 7 also presents an *upper bound* on the edge chromatic number, which is the minimum number of colors (channels) such that an edge coloring exists having no two equally colored edges incident on the same vertex. The large gap between the optimal solution and the edge chromatic number upper bound results from the fact that under edge coloring, all edges can be active simultaneously, while MCI creates trees on which transmissions still have to be scheduled. Hence, by using edge coloring, the capacity region is enlarged to $\lambda_{ij} \leq 1\, \forall (i,j) \in E_N$. In many network instances, such a large capacity expansion requires numerous channels.

6.2 Capacity Expansion Algorithms

We now demonstrate the operation of the different capacity expansion algorithms on a specific randomly generated network with 20 nodes. Figure 8 illustrates an example of the channel allocations performed by the different algorithms in a network in which the required number of channels is 4. The figure presents the network and then, for each algorithm, the 4 forests. Figure 8(a) presents the solution obtained by the MCI algorithm. It can be seen the leftmost forest is relatively very dense, while the rightmost tree is very sparse (it includes only a single edge). It is clear that the capacity is not efficiently allocated in this solution. Namely, most of the nodes do not use the fourth channel, while the first channel has to be shared by many links.

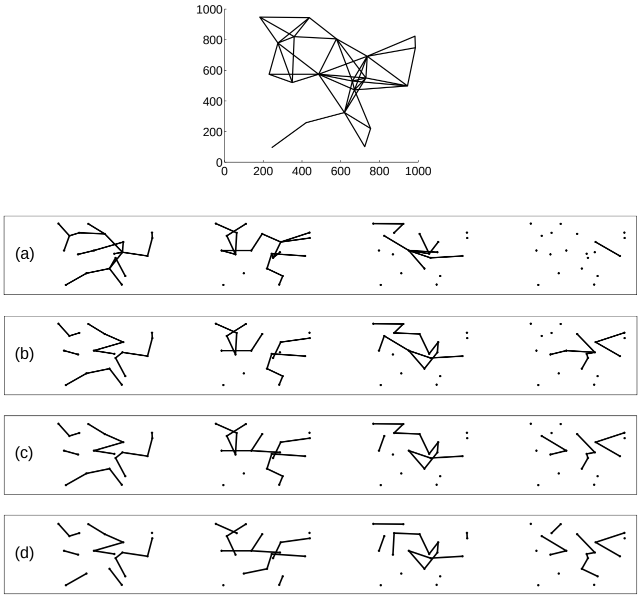


Figure 8: Channel assignments by (a) MCI (b) R-Greedy (c) R-MaxD, and (d) R-AvgD.

Figure 8(b) presents the allocation performed by algorithm R-Greedy, using the MCI solution as input. It can be seen that several edges have now been migrated to the fourth (rightmost) channel. Figure 8(c) presents the allocation performed by algorithm R-MaxD, using the R-Greedy solution as input. The R-Greedy solution had two vertices of degree three, and R-MaxD manages to manipulate the allocation such that only a single vertex has degree three. Finally, the solution from R-MaxD is used as input in R-AvgD to obtain the channel allocation of Figure 8(d). Though the maximum vertex degree remains at three, lower degree vertices have had their degrees improved, with many more edges in this allocation entirely disconnected.

The example above demonstrates the operation of the capacity expansion algorithms. We now quantitatively evaluate their performance. Given a specific channel allocation it is not straightforward to represent the capacity region. This results from the fact that it is a polytope in $\mathbb{R}_+^m$. Yet, in order to obtain some insight, we make the following simplifying assumption regarding the capacity allocation that takes place once the channels are assigned to the links. We assume that some degree of fairness exists, and therefore, if possible, all edges connected to a node receive an equal share of the node capacity. This is sometimes impossible, due to a capacity limit resulting from the other node connected to an edge. Consequently, under this assumption the throughput on an edge (i, j) operating in channel k will be at least $(\max(\Delta_{i,k}, \Delta_{j,k}))^{-1}$, where $\Delta_{i,k}$ is the number of edges adjacent to node i that are using channel k.

Accordingly, the first performance measure is *Average Capacity*, which is the average over all edges $(i, j) \in E_N$ of the above value. The second performance measure is the *Worst-Case Capacity*, which is the lowest capacity allocated to a link in the network. This is inversely proportional to the maximum node degree over all nodes and all channels. Using the above notation, it is equal to $(\max_{i,k} \Delta_{i,k})^{-1}$.

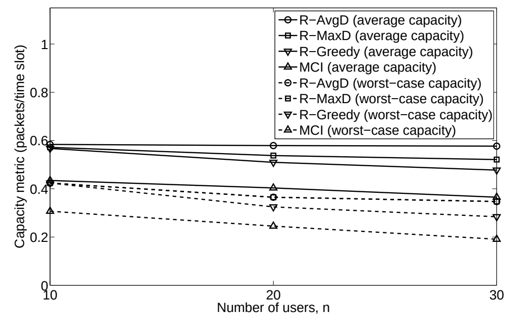


Figure 9: Average and worst-case capacities.

Figure 9 illustrates these performance metrics for random networks with different numbers of nodes (n). For each value of n, the results were averaged over 50 different random network instances. It can be seen that both for the worst case and the average case, R-Greedy provides significant throughput improvement over the MCI algorithm (average improvement of 29% and 40% in the average and worst-case capacity, respectively). This is notable, since the complexity of the greedy capacity expansion algorithm is small relative to that of MCI. When using the R-MaxD and R-AvgD, we employed a maximum search depth of D_MAX = 2. This implies that the complexities of R-MaxD and R-AvgD are respectively $O(dnm^2)$ and $O(d^2nm)$ times the complexity of MCI. Despite the higher complexities, the value of these algorithms is evident from their ability to significantly improve the performance metrics. Relative to the MCI solution, R-MaxD achieves average improvements of 36% and 56% in the average and worst-case capacities, respectively, while R-AvgD achieves 45% and 56%, respectively.[10] There is an evident tradeoff between complexity and performance. Since the channel allocation problem is solved in a different time scale from the scheduling problem, it seems beneficial to use R-MaxD or R-AvgD.

In realistic situations the number of channels and radios is bounded. Figure 10 depicts the average capacity metric versus the number of available channels (k) for a network with 20 nodes. For each value of k, the results were averaged over 50 different random network instances. Given a fixed k, the MCI, R-Greedy, R-MaxD, and R-AvgD algorithms were enlisted to obtain and expand the capacity of k-forests. In instances where there were edges that could not be included in a valid k-forest, these edges were added to the last generated forest (at channel k). As explained in Section 5.1, the first $k-1$ channels are guaranteed to satisfy OLoP, while the k-th channel operates at a worst-case 50% throughput. If there was a cycle in the k-th channel, we assumed that the edges in the k-th channel achieve only 50% throughput when calculating the average capacity. Algorithms R-Greedy, R-MaxD and R-AvgD provide significant improvement over the MCI algorithm alone.

7. CONCLUSIONS

In this paper we have applied techniques stemming from stability theory and matroid theory to obtain novel results

[10]Note that the plots of the worst-case capacity for R-AvgD and R-MaxD overlap.

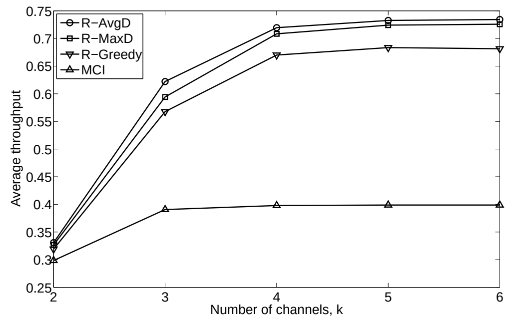


Figure 10: Average capacities given a fixed number of channels k.

regarding the design of Wireless Mesh Networks. The application of these theories allows us to develop algorithms for partitioning a mesh network into a number of high capacity subnetworks such that in each of the subnetworks simple distributed algorithms can obtain 100% throughput.

We have performed a thorough study of the implications of Local Pooling on network design and shown that although the notion of Local Pooling is rather abstract, its implications are quite powerful. Based on some of our observations, we developed matroid intersection algorithms for efficient network partitioning. In Section 6 we have shown that these algorithms perform very well in terms of capacity. We note that the scope of this work spans more than multi-radio multi-channel WMNs. It seems to be relevant to any wireless network with stochastic arrivals in which transmissions can be differentiated in the time domain (i.e. scheduling) as well as in other domains (frequency, code, etc.).

This paper primarily provides a *theoretical contribution* that lays the foundation for developing *practical algorithms*. Hence, there are still many problems to deal with. For example, a future research direction is to allow dynamic channel allocation. This will require to tailor the channel allocation algorithms for online and perhaps distributed operation. In addition, Lemma 4 indicates that partitioning into trees may be suboptimal. Therefore, we would like to develop matroid intersection algorithms that will partition into other components similar to the ones identified in Section 4. In general, we would like to develop algorithms that partition the network to the minimum number of OLoP-satisfying components. It seems that this may be done by utilizing connections between the maximal independent sets in the interference graph and the characteristics of the graphic and partition matroids.

Acknowledgments

We thank the reviewers for their helpful comments.

This work was supported by NSF ITR grant CCR-0325401, by ONR grant number N000140610064, by DARPA/ AFOSR through the University of Illinois grant no. F49620-02-1-0325, and by a Marie Curie International Fellowship within the 6th European Community Framework Programme.

8. REFERENCES

[1] I. F. Akyildiz, X. Wang, and W. Wang. Wireless mesh networks: a survey. *Computer Networks*, 47(4):445-487, March 2005.

[2] M. Alicherry, R. Bhatia, and L. E. Li. Joint channel assignment and routing for throughput optimization in multi-radio wireless mesh networks. In *Proc. ACM MOBICOM'05*, September 2005.

[3] A. Adya, P. Bahl, J. Padhye, A. Wolman, and L. Zhou. A multi-radio unification protocol for IEEE 802.11 wireless networks. In *Proc. Broadnets'04*, October 2004.

[4] R. Bhatia, A. Segall, and G. Zussman. Analysis of bandwidth allocation algorithms for wireless personal area networks. To appear in *ACM/Springer Wireless Networks*, 12, 2006.

[5] A. Brzezinski and E. Modiano. Greedy weigthed matching for scheduling the input-queued switch. In *Proc. CISS'06*, March 2006.

[6] A. Brzezinski, G. Zussman, and E. Modiano. Enabling distributed throughput maximization in wireless mesh networks via local pooling. *MIT/LIDS Technical Report #2696*, March 2006.

[7] P. Chaporkar, K. Kar, and S. Sarkar. Throughput guarantees through maximal scheduling in wireless networks. In *Proc. Allerton Conf. on Commun., Control, and Comp.*, September 2005.

[8] L. Chen, S. H. Low, M. Chiang, and J. C. Doyle. Optimal cross-layer congestion control, routing and scheduling design in ad hoc wireless networks. In *Proc. IEEE INFOCOM'06*, April 2006.

[9] J. Dai and B. Prabhakar. The throughput of data switches with and without speedup. In *Proc. IEEE INFOCOM'00*, March 2000.

[10] A. Dimakis and J. Walrand. Sufficient conditions for stability of longest queue first scheduling: second order properties using fluid limits. *Advances of Applied Probability*, 38(2):505–521, June 2006.

[11] H. N. Gabow and H. H. Westermann. Forests, frames, and games: algorithms for matroid sums and applications. *Algorithmica* 7(5&6):465–497, 1992.

[12] F. Harary. *Graph Thoery*, Addison-Wesley, Reading, MA, 1969.

[13] J.-H. Hoepman. Simple distributed weighted matchings. *eprint cs.DC/0410047*, October 2004.

[14] K. Jain, J. Padhye, V. N. Padmanabhan, and L. Qiu. Impact of interference on multi-hop wireless network performance. *ACM/Springer Wireless Networks*, 11(4):471–487, July 2005.

[15] M. Kodialam and T. Nandagopal. Characterizing the capacity region in multi-radio multi-channel wireless mesh networks. In *Proc. ACM MOBICOM'05*, September 2005.

[16] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, New York, 1976.

[17] X. Lin and N.B. Shroff. The impact of imperfect scheduling on cross-layer rate control in wireless networks. In *Proc. IEEE INFOCOM'05*, March 2005.

[18] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. *IEEE Trans. Commun.*, 47(8):1260-1267, August 1999.

[19] E. Modiano, D. Shah, and G. Zussman. Maximizing throughput in wireless networks via gossiping. In *Proc. ACM SIGMETRICS'06*, June 2006.

[20] M. Neely, E. Modiano, and C. Rohrs. Dynamic power allocation and routing for time-varying wireless networks. *IEEE J. Sel. Areas Commun.*, 23(1):89-103, January 2005.

[21] S. Ramanathan and E. L. Lloyd. Scheduling algorithms for multihop radio networks", *IEEE/ACM Trans. Net.*, 1(2):166–177, April 1993.

[22] A. Raniwala and T.-C. Chiueh. Architecture and algorithms for an ieee 802.11-based multi-channel wireless mesh network. In *Proc. IEEE INFOCOM'05*, March 2005.

[23] S. Sarkar, P. Chaporkar, and K. Kar. Fairness and throughput guarantees with maximal scheduling in multihop wireless networks. In *Proc. WiOpt'06*, April 2006.

[24] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Trans. Autom. Control*, 37(12):1936–1948, December 1992.

[25] X. Wu and R. Srikant. Regulated maximal matching: a distributed scheduling algorithm for multi-hop wireless networks with node-exclusive spectrum sharing. In *Proc. IEEE CDC-ECC'05*, December 2005.

[26] G. Zussman and A. Segall. Capacity assignment in bluetooth scatternets - optimal and heuristic algorithms. *ACM/Kluwer Mobile Networks and Applications*, 9(1):49–61, February 2004.