Distributed dynamic scheduling for end-to-end rate guarantees in wireless ad hoc networks

Theodoros Salonidis[†] and Leandros Tassiulas[†,‡]

[†] Department of Electrical and Computer Engineering and Institute for Systems Research
A.V. Williams Bldg. University of Maryland at College Park, MD 20742, USA
[‡] Department of Computer and Communication Engineering, University of Thessaly
Argonafton and Filellinon 38221 Volos, Greece

thsalon@glue.umd.edu, leandros@isr.umd.edu

***Abstract*—**

We present a novel framework for the provision of deterministic end-to-end bandwidth guarantees in wireless ad hoc networks. Guided by a set of local feasibility conditions, multi-hop sessions are dynamically offered allocations, further translated to link demands. Using a distributed TDMA protocol, nodes adapt to the demand changes on their adjacent links by local, conflict-free slot reassignments. As soon as the changes stabilize, the nodes must incrementally converge to a TDMA schedule that realizes the global link (and session) demand allocation.

We first identify an inherent trade-off between the degree of topology control and fraction of feasible allocations that can be captured by the local conditions. We show that tree topologies can be maximally utilized in this respect and that a converging distributed link scheduling algorithm exists in this case.

Decoupling end-to-end bandwidth allocation from link scheduling allows support of various end-to-end QoS objectives. Focusing on Available Bit Rate (ABR) service, we design an asynchronous distributed algorithm for sharing bandwidth to the sessions in a maxmin fair (MMF) manner.

Finally, we present the implementation of this framework over Bluetooth, an existing wireless technology that enables the formation of ad hoc networks. This implementation is free of the usual restrictive assumptions of previous TDMA approaches: it does not require any a-priori knowledge on the number of nodes in the network nor even network-wide slot synchronization.

I. Introduction

Ad hoc networks can be established on the fly and form an all-wireless infrastructure without the need of any centralized administration. Due to the multi-access nature of the wireless medium, the perceived QoS in ad hoc networks heavily depends on the underlying medium access (MAC) protocol. Such a protocol must use local information and coordinate transmissions so that bandwidth is shared among users in a controlled fashion. Satisfying both requirements is a well-known problem with no satisfactory solutions to date. Random access methods, such as the one used in the 802.11 standard, use local information at the expense of unpredictable transmission conflicts and lack of strict allocation guarantees. On the other hand, scheduled access methods such as Time Division Multiple Access (TDMA), achieve deterministic allocations via perfect coordination of transmissions but typically need global network knowledge to reach their goal.

According to TDMA, bandwidth can be allocated to the network links using a schedule of period T_{system} slots. During every slot, several links are activated for transmission such that no conflicts occur at the intended receivers. The number of conflict-free slots each link receives within a system period determines its allocated bandwidth.

TDMA has been used for QoS routing in mobile ad hoc networks [1] [2][3][4]. Chen and Nahrstedt [1] and Gerla and Tsai [2] focus on mobility issues but assume that conflict-free slots have already been pre-allocated to the links in an arbitrary manner. Admission control for multi-hop sessions is performed based on these static allocations. Better utilization of network resources can be achieved if the higher layer needs drive the link layer to allocate bandwidth accordingly. Zhu and Corson [4] and Lin [3], reserve slots for incoming sessions on links on an as-needed basis. Finding the maximum available bandwidth (number of conflict-free slots) on a path subject to the reserved slot positions of the existing sessions is an NP-complete problem. Distributed heuristic methods are proposed for admission control and slot allocation. The result is network underutilization in a different form–several blocked sessions would have been accepted had the arrangement of slots in the TDMA schedule been different.

Network utilization can be increased by allowing dynamic recomputation of the TDMA schedule upon session arrivals. An incoming session is admitted if the additional load it places on the links of its path is such that the induced demand allocation on the network links is realizable by a TDMA schedule. Existing results for the static version of the link scheduling problem are not encouraging even if global network topology information is available. According to the seminal works in [5], [6], determining feasibility of a set of link rates in an ad hoc network of arbitrary topology is an NP-complete problem. Several centralized [7][8], semi-centralized [9], or distributed [10][11] [12] heuristics for TDMA link scheduling have been proposed, but they either do not possess well-defined performance guarantees or cannot be applied easily to dynamic operational settings.

In this paper, we introduce a framework and implementation

for transparent integration of bandwidth allocation to multi-hop sessions with distributed dynamic TDMA link scheduling. The core idea is that we can achieve guaranteed performance by controlling the network topology or the set of supported allocations using a set of local conditions specific to the wireless setting. Guided by the local conditions, the end-to-end mechanism allocates feasible rates to multi-hop sessions sharing the network. These rates are translated to link demands to be realized by a TDMA schedule. The nodes adjust the rates on their adjacent links by local slot reassignments until the desired allocation is reached.

Using this framework, we present a distributed dynamic scheduling algorithm that realizes all feasible link demand allocations for tree topologies. This algorithm can start from any initial TDMA schedule and incrementally converge in a finite number of steps to a new TDMA schedule realizing a desired demand allocation. Trees manifest in various ad hoc networking applications. Existing topology construction algorithms for Bluetooth ad hoc networks [13] [14][15] generate tree topologies. According to the sensor network communication paradigm, sensors report data back to a single source over a tree structure [16], [17]. Tree topologies are also used for energy-efficient broadcasting [18]. Several non-tree ad hoc networks use a certain subset of nodes as a tree backbone for facilitating administrative purposes such as routing [19]. These structures can be leveraged and combined with our algorithm to provide bandwidth guarantees over the backbone and best effort to the edges of the ad hoc network.

The link scheduling algorithm focuses on converging to a TDMA schedule satisfying the link demands and is agnostic of the specifics of the higher layer process that allocates bandwidth to the end-to-end sessions. This allows definition and realization of more generalized service models than support of session rates known in advance. To this end, we consider provision of end-to-end Available Bit Rate (ABR) service and introduce an asynchronous distributed algorithm for sharing bandwidth to sessions in a maxmin fair manner. This algorithm borrows from rate-based flow control approaches currently used in wireline ATM networks. Combined with the identified local feasibility conditions, it can be applied to any topology form.

Another important issue is that TDMA for ad hoc networks is hard to implement in practice. Most approaches rely on global slot synchronization and require knowledge of the number of nodes in the network to split the periodic TDMA frame in a control and data portion. While such assumptions may hold for special cases (e.g. custom designs for military applications), they impose a major restriction for the deployment of ad hoc networks in general settings. In our approach each link uses a local time slot reference for communications provided by the hardware clock of one of the node endpoints. A distributed coordination mechanism is used to maintain the network TDMA schedule free of transmission conflicts while the nodes re-assign slots to reach the desired allocation.

The rest of the paper is structured as follows: Section II presents the asynchronous TDMA architecture used by the link scheduling scheme. Section III formulates the distributed dynamic link scheduling problem and elaborates on the issues involved in solving it. Section IV presents an optimal centralized asynchronous TDMA algorithm for trees. This algorithm forms the basis of a distributed dynamic algorithm presented in section V. Section VI elaborates on the integration of link scheduling with end-to-end bandwidth allocation. Section VII implements the bandwidth allocation framework over Bluetooth. Section VIII surveys related work. Section IX concludes the paper.

II. Asynchronous TDMA

A. Access Architecture

The wireless ad hoc network is represented as a directed graph $G(N, E)$. Each node has a unique identity (e.g. MAC address). A directed edge $(u, v) \in E$ signifies that nodes u and v are within range and have established a wireless link where u is assigned the role of master and v the role of slave. A link is a unique communication channel derived from the identity of the master. A channel can be implemented as a separate frequency or spread spectrum code. We assume that different channels provide the necessary orthogonality to tolerate simultaneous co-located transmissions. In other words, transmissions on a channel are correctly received by a node listening on that channel despite any in-range transmissions that may be happening at different channels. We also assume that there are no losses due to channel errors.

The system is slotted but not globally slot-synchronized. For each link, the master endpoint provides a local time slot reference for communication. Each slot supports full-duplex communication initiated by the master: During the first part of the slot the master polls a slave; during the second part a slave responds if polled by the master.

The access problem arises because each node has a single radio transceiver and can communicate (transmit or receive) to at most one link at a time. Thus, nodes need to coordinate their presence on links in mutual time intervals. Based on its own hardware clock, each node i divides time in fixed-size slots, each equal to the duration of a full-duplex communication slot. Transmissions on adjacent links are coordinated using a local link schedule $\boldsymbol{S}_i$ of period T_{system} slots. The local schedule determines communication action for the duration of a slot: the node can either be active on a single link (polling if master or listening for a poll if slave) or remain idle.

Due to the phase difference between the node hardware clocks the local schedules of different nodes are not slot-aligned. For conflict-free communication on τ_l consecutive slots on link l, the master must allocate τ_l slots in its local schedule for polling while the slave must allocate at least $\tau_l + 1$ time-overlapping slots for tuning to the channel and aligning to the time reference of this master.

The local schedules form the network asynchronous TDMA schedule. A *link slot allocation* $\boldsymbol{\tau} = [\tau_l]$ realized by the network TDMA schedule is the number of slots every link l transmits conflict-free during T_{system} slots and equals the number of slots allocated to the local schedule of the master endpoint. Figure 1 illustrates an example of an ad hoc network using a system period $T_{system} = 12$ slots. The slot allocation realized by the asynchronous TDMA schedule is $\boldsymbol{\tau} = (\tau_1, \tau_2, \tau_3, \tau_4) = (3, 3, 3, 4)$ slots.

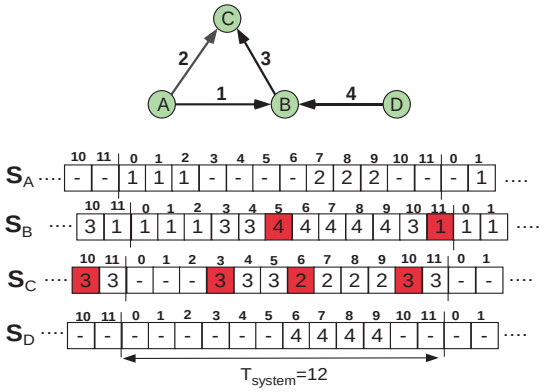

Fig. 1. (a) Network topology: Arrows denote master-slave relationships. Nodes A and D act as masters on all their adjacent links, B is slave on links 1 and 4 and master on link 3 while node C acts as slave on all its links. Links 2 and 4 can transmit simultaneously without conflict on different channels. (b) Asynchronous TDMA schedule of period $T_{system} = 12$ slots: Every slot supports full-duplex communication. Slots where slaves switch channel and time reference are marked in red. The realized slot allocation is $\boldsymbol{\tau} = (\tau_1, \tau_2, \tau_3, \tau_4) = (3, 3, 3, 4)$.

III. The dynamic link scheduling problem

At any point in time, the network operates using a conflict-free TDMA schedule that realizes a link slot allocation. In addition, every link is always characterized by a slot demand. In general, the current allocation and demand of a link may be different.

Network dynamics are viewed by the link layer as a higher-layer process that changes the link demands at asynchronous time instants. In section VI, we will instantiate this process to mechanisms that allocate bandwidth to multi-hop sessions. Mobility can be captured by viewing a link failure as zero demand and a link establishment as a transition from zero to a positive demand.

We assume that the higher-layer process alternates between two states: an active state where the link demands change and a quiescent state where no changes occur. The end of each active state corresponds to a link demand allocation to be realized by a TDMA schedule. The challenge is that nodes must reach such a schedule starting from the current TDMA schedule and using only local information.

The model of alternating states is necessary for the definition of convergence. The physical interpretation is that network topology and traffic dynamics must remain stable for a sufficient amount of time to allow realization of the desired allocation. However, the nodes are not aware of which of the two states the network is currently in. They can only detect demand changes on their adjacent links and must reach the desired TDMA schedule and allocation via local slot reassignments.

A. Local feasibility conditions

In order for convergence to occur, the higher layer process must provide the link layer with demand allocations that are feasible. A link demand allocation $\boldsymbol{\tau} = (\tau_1, ..., \tau_l, ..., \tau_{|E|})$ is feasible if there exists a TDMA schedule that can allocate τ_l conflict-free slots to every link l without exceeding T_{system} slots. In its general form, the problem of feasibility determination in multi-channel systems is NP-complete even if global slot synchronization and topology knowledge are available [6]. Since we consider distributed bandwidth allocation mechanisms, we are interested in identifying certain instances where feasibility can be characterized by a set of local conditions.

Let us first assume that global slot synchronization is supported. In this case, local conditions would require the demand sum of the links adjacent to each node not to exceed T_{system} slots. Due to the link scheduling interdependence, these local conditions cannot alone guarantee feasibility (see Fig. 2 for a counter-example). The additional non-local conditions require that, for every odd node subset Q ($|Q| > 1$) in the topology graph, the sum of the demands of all links adjacent to the nodes in Q must not exceed $\lfloor (|Q| - 1)/2 \cdot T_{system} \rfloor$ slots.

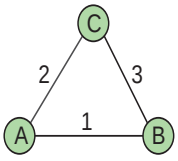

Fig. 2. Without loss of generality, assume that all nodes are slot-synchronized and T_{system} is even. There is no schedule that can allocate $T_{system}/2$ conflict-free slots per link, even if the local conditions $\tau_1 + \tau_2 \leq T_{system}$, $\tau_1 + \tau_3 \leq T_{system}$ and $\tau_2 + \tau_3 \leq T_{system}$ for nodes A, B, C, respectively allow this allocation. The non-local condition $\tau_1 + \tau_2 + \tau_3 \leq T_{system}$ is also needed.

There are two ways to guarantee feasibility using only local conditions: Restrict the network topology or underutilize the network.

If the network topology is bipartite, the entire set of feasible allocations can be captured only by local conditions. Topology control is inherent in multi-channel systems due to the need for assigning channels to the links before communication takes place. Bipartite topologies can be enforced using local information if every node is required to act only as master or slave to all its adjacent links, and the channel assigned to each link is derived from the (unique) id of the master node endpoint.

Alternatively, if no mechanism for topology control exists, it is possible to ensure feasibility by restricting the maximum number of slots each node provides to its adjacent links. For slot-synchronized multi-channel systems, feasibility is guaranteed by requiring the sum of link demands on every node be less than $\lfloor 2/3 \cdot T_{system} \rfloor$ slots at any time [20], [21]. Local conditions of this form are sufficient: they guarantee feasibility but only capture a fraction of the entire set of feasible allocations. This essentially means that the network is underutilized. We would like to note that the terms "underutilization" and "feasibility" are with respect to provision of guaranteed rates in the network. Thus, the maximum number of slots provided by the local feasibility conditions are for QoS traffic. The remaining number of slots in the nodes' local schedules can always be used for other purposes, such as control or best-effort traffic.

In an asynchronous TDMA system such as the one considered here, the region of feasible rates is further restricted. Due to the additional slots needed in the slaves' local schedules, the minimum period $L_{min}(\boldsymbol{\tau})$ realizing a demand allocation $\boldsymbol{\tau}$ is greater than the minimum period $L_{min}^{synch}(\boldsymbol{\tau})$ required by a perfectly synchronized system. Since feasibility is characterized by comparing the minimum period realizing $\boldsymbol{\tau}$ to the system

period T_{system}, certain allocations feasible by a synchronized system will not be feasible when asynchronicity is present. In [22] it has been shown that, for any given topology and demand allocation $\boldsymbol{\tau}$, $L_{min}(\boldsymbol{\tau}) \leq 2 \cdot L_{min}^{synch}(\boldsymbol{\tau})$ [22]. Consequently, a set of sufficient local feasibility conditions is for nodes to offer half the slots they would offer in the corresponding synchronized system: For bipartite topologies, feasibility is guaranteed if every node offers $\lfloor 1/2 \cdot T_{system} \rfloor$ slots while for arbitrary topologies $\lfloor 1/3 \cdot T_{system} \rfloor$ slots. These conditions imply further under-utilization–$1/2$ and $2/3$ of the total capacity of bipartite and arbitrary topologies, respectively cannot be used for QoS traffic.

A lower bound on the minimum period $L_{min}(\boldsymbol{\tau})$ of an asynchronous TDMA schedule realizing a demand allocation $\boldsymbol{\tau} = (\tau_1, ..., \tau_l, ..., \tau_{|E|})$ is given by:

$$LB(\boldsymbol{\tau}) = \max_{u \in N} \sum_{l \in L(u)} (\tau_l + J_l^{(u)}) \tag{1}$$

where $L(u)$ is the set of links adjacent to node u and,

$$J_l^{(u)} = \begin{cases} 1 & \text{if } u \, is \, slave \, on \, link \, l \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The term τ_l in the sum of the RHS of (1) exists because each node can communicate to only a single link at every slot of its local schedule. The term $J_l^{(u)}$ is due to the need for (at least) an additional slot for time-slot reference alignment on every link a node acts as slave. The lower bound on the minimum period is not tight, but can be used to identify instances where the entire set of feasible allocations can be captured by a set of local conditions. This is summarized by the following proposition:

Proposition 1: Consider an asynchronous TDMA ad hoc network $G(N, E)$. If for every demand slot allocation $\boldsymbol{\tau}$, $L_{min}(\boldsymbol{\tau}) = LB(\boldsymbol{\tau})$, then, *all* feasible allocations for $G(N, E)$ can be captured by the following set of local conditions:

$$\sum_{l \in L(u)} \tau_l \leq T_{system} - \sum_{l \in L(u)} J_l^{(u)}, \; \forall u \in N \tag{3}$$

Proof: We will use contradiction. Let $\boldsymbol{\tau}^*$ be a demand allocation which is allowed by the local conditions of eq. (3) but is not feasible. Since $\boldsymbol{\tau}^*$ is not feasible, the minimum period for realizing it must be strictly greater than T_{system} slots: $L_{min}(\boldsymbol{\tau}^*) > T_{system}$. The demand allocation $\boldsymbol{\tau}^*$ obeys the local conditions of eq. (3):

$$\sum_{l \in L(u)} (\tau_l^* + J_l^{(u)}) \leq T_{system}, \; \forall u \in N \Rightarrow$$

$$\max_{u \in N} \sum_{l \in L(u)} (\tau_l^* + J_l^{(u)}) \leq T_{system} \Rightarrow$$

$$LB(\boldsymbol{\tau}^*) \leq T_{system}$$

Since $L_{min}(\boldsymbol{\tau}) = LB(\boldsymbol{\tau})$, $\forall \boldsymbol{\tau} \in G(N, E)$, we reach the conclusion that $L_{min}(\boldsymbol{\tau}^*) \leq T_{system}$, i.e. $\boldsymbol{\tau}^*$ is feasible. This contradicts our initial hypothesis.

Proposition 1 states that classes of topologies or specific topologies for which $L_{min}(\boldsymbol{\tau}) = LB(\boldsymbol{\tau})$ for all $\boldsymbol{\tau}$, can be fully utilized by distributed algorithms. In the next section we show trees are a topology class that satisfies this property.

IV. Optimal link scheduling for tree networks

Let the ad hoc network topology $G(N, E)$ be a tree. Without loss of generality, assume that an arbitrary node in the tree is designated as root. The root provides a reference for parent-child relationships between the node endpoints of every link in the network. The parent-child relationship between the node endpoints is independent of their master-slave relationship. We define the *level of a node* to be its hop distance from the root (the root has a level equal to zero). The *level of a link* equals the level of its child node endpoint.

Let $\boldsymbol{\tau}$ be a demand allocation. Given $\boldsymbol{\tau}$, every node is equipped with a local schedule of period $T_{system} = LB(\boldsymbol{\tau})$ slots (eq. (1)). The slot positions in each local schedule are indexed from 0 to $T_{system} - 1$. A set of consecutively assigned slots to link l in the local periodic schedule $\boldsymbol{S}_u$ of node u, forms a (circular) window $W_l^{(u)} = [s_l^{(u)}, e_l^{(u)}]$:

$$[s_l^{(u)}, e_l^{(u)}] = \begin{cases} s_l^{(u)}, ..., e_l^{(u)} & \text{if } s_l^{(u)} \leq e_l^{(u)} \\ s_l^{(u)}, .., 0, ..., e_l^{(u)} & \text{otherwise} \end{cases} \tag{4}$$

where $s_l^{(u)}$ and $e_l^{(u)}$ denote the start and end slot positions assigned to link l in $\boldsymbol{S}_u$, respectively. We denote by $|W_l^{(u)}|$ the number of slots in $W_l^{(u)}$. Modulo-T_{system} addition and subtraction are denoted by "$\oplus$" and "$\ominus$", respectively.

We now describe the operation of CENTRAL_TREE, a link scheduling algorithm that realizes $\boldsymbol{\tau}$ using a period of $LB(\boldsymbol{\tau})$ slots. Initially all local schedules are empty. Links are scheduled in a breadth-first manner. In the first iteration, the root node r starts from slot 0 in its local schedule $\boldsymbol{S}_r$ and schedules its children links (level-1 links) until their total demand is satisfied. The links are scheduled non-preemptively in successive windows: For each child link $l = (r, c)$, r allocates in $\boldsymbol{S}_r$ a window of $\tau_l + J_l^{(r)}$ consecutive slots, immediately succeeding the window of the previously scheduled child link. After l has been scheduled in $\boldsymbol{S}_r$, the child node c assigns $\tau_l + J_l^{(c)}$ time-overlapping slots to link l in its own local schedule $\boldsymbol{S}_c$.

In the next iteration, the root children (level-1 nodes) schedule their own children links (level-2 links). For each such node u, its parent link l_p has already been satisfied by a window $W_{l_p}^{(u)} = [s_{l_p}^{(u)}, e_{l_p}^{(u)}]$ in $\boldsymbol{S}_u$, during the previous iteration. Node u starts from slot position $e_{l_p}^{(u)} \oplus 1$ and schedules its children links non-preemptively in successive windows by filling $\boldsymbol{S}_u$ towards slot position $s_{l_p}^{(u)}$ in a circular fashion. For each link $l = (r, c)$ scheduled in $\boldsymbol{S}_u$, the child node endpoint c assigns time-overlapping slots in $\boldsymbol{S}_c$.

The scheduling process is repeated recursively until the highest-level links have been scheduled and the leaf nodes have updated their local schedules.

Theorem 1: If the network topology is a tree, any demand allocation $\boldsymbol{\tau}$ can be realized by algorithm CENTRAL_TREE using a period of $LB(\boldsymbol{\tau})$ slots.

Proof: By induction on the tree levels, we show that no node runs out of slots during the algorithm execution. See Appendix X-B for the detailed proof.

Algorithm CENTRAL_TREE cannot be used in practice because it requires global information and a-priori knowledge of a static demand allocation for which it computes an optimal schedule. However, it is important because it establishes that the entire set of feasible allocations for tree topologies can be captured by a set of local conditions, even for the case of asynchronous TDMA. In addition, it forms the basis for a distributed algorithm that operates in settings where link demands or topology may change over time.

V. Distributed algorithm

The distributed algorithm for trees fits to a general algorithmic framework that consists of three components:

1) Topology and capacity conditions: First, feasibility must be guaranteed via a combination of local topology and capacity conditions. In our case, the topology must be a tree at any time. This can be accomplished using existing distributed algorithms for dynamic tree formation and maintenance [14], [15], [23]. According to Proposition 1 and Theorem 1, when the topology is a tree, local conditions can capture the entire set of feasible allocations. Therefore, each node u uses the following capacity conditions:

$$\sum_{l \in L(u)} \tau_l \leq T_{system} - \sum_{l \in L(u)} J_l^{(u)} \tag{5}$$

2) Distributed coordination mechanism: Since nodes have access only to local information they will respond independently to the demand changes on their adjacent links. The coordination mechanism ensures that the network TDMA schedule remains free of transmission conflicts despite the simultaneous slot reassignments on different links. The coordination mechanism is not bound to a specific topology structure.

3) Distributed link scheduling algorithm: Determines how nodes should re-assign slots to reach a schedule realizing a desired link allocation. This algorithm may be topology-specific.

A. Distributed coordination mechanism

During network operation, several links may be asynchronously triggered in parallel for rate adjustment. Rate adjustment on a link occurs when the node endpoints re-assign concurrent slot positions for this link in their local schedules. The criteria to trigger adjustment may depend on local traffic load on the nodes and the current communication needs of the link. The coordination mechanism is also used to assign an initial number of conflict-free slots on a link that has just been established.

Each node can be involved at only one link rate adjustment at a time. It conveys its current busy status to its neighbors using an internal one-bit variable called BusyBit. This bit is copied to the corresponding field of every outgoing packet (be it data or control one). Rate adjustment on a link l can be initiated when none of its endpoints are currently busy on a rate adjustment of other links. Upon initiation, both endpoints set their BusyBits to one. Then they exchange their current local schedules using SC_INFO control packets. This information aids one of the endpoints to determine a new set of slot positions to be assigned to this link. Some of these slots may be currently assigned to the other links adjacent to the endpoint nodes and need to be canceled.

Each endpoint stores the new positions in a variable called $LOCK\_VEC$ and signals schedule modifications to all its affected neighbors using SC_UPD packets. An SC_UPD packet transmitted on a link, contains new slot positions to refresh the old ones for this link in the recipient's local schedule. After *all* affected neighbors acknowledge schedule modifications, the endpoints assign the new positions (stored in $LOCK\_VEC$) to link l in their own local schedules. Then, they become available for rate adjustment on other links by clearing their BusyBit and $LOCK\_VEC$ variables.

Communications are not suspended during the rate adjustment process. The control packets are transmitted using the conflict-free slots in the old TDMA schedule until the endpoints modify their local schedules once they have received all acknowledgments from their neighbors. The coordination mechanism keeps the network free of transmission conflicts at all times. Conflicts would arise if the same slots were simultaneously assigned on adjacent links to the same node or nodes re-assigned slots on links without notifying the corresponding neighbors. The first case cannot arise because the BusyBit precludes all one-hop neighbors to initiate rate adjustment with the endpoints. The second case cannot arise because the endpoints modify their schedules only after having received acknowledgements from all their affected neighbors.

B. A distributed link scheduling algorithm for trees

The distributed scheduling algorithm operates within the state space defined by tree topologies and the capacity conditions of eq. (5). Nodes are only aware of the parent/child relationship and the current demands on their adjacent links. The algorithm is self-stabilizing: it may start from any initial TDMA schedule and converge to a new schedule realizing a desired allocation $\boldsymbol{\tau}$. Mobility can also be supported as long as a tree formation and maintenance protocol [14], [15], [23] runs in the network.

Before presenting the algorithm we introduce the notions of satisfied and stable links. Let τ_l be the current demand for link $l = (u, v)$, and $t_l^{(u)}$ be the number of conflict-free slots currently assigned to l in the local schedule $\boldsymbol{S}_u$ of node u. Link l is called *satisfied* by node u if the following conditions hold:

STF1: The link is scheduled in a single window $W_l^{(u)} = [s_l^{(u)}, e_l^{(u)}]$ in $\boldsymbol{S}_u$.

STF2: The current demand is exactly satisfied by the current assignment: $t_l^{(u)} = \tau_l + J_l^{(u)}$.

where $J_l^{(u)}$ is given by eq. (2).

Let the parent link $l_p = (u, p)$ of node u be satisfied by a window $W_{l_p}^{(u)} = [s_{l_p}^{(u)}, e_{l_p}^{(u)}]$ in $\boldsymbol{S}_u$. Also, let the children links $l_c = (u, c)$ of u be assigned distinct priorities p_{l_c}.

A child link l_c of u is *stable* if it is satisfied and the position of window $W_{l_c}^{(u)} = [s_{l_c}^{(u)}, e_{l_c}^{(u)}]$ in $\boldsymbol{S}_u$ provides enough room for scheduling all links of lower priority according to their current demands. More formally, a child link l_c is called stable by node u if the following conditions hold:

STBL1: Link l_c is satisfied.

STBL2: $|[e_{l_c}^{(u)} \oplus 1, s_{l_p}^{(u)} \ominus 1]| \geq \sum_{k \in CH(u): p_k < p_{l_c}} (\tau_k + J_k^{(u)})$

where $CH(u)$ is the set of children links of u and "$\oplus$" and "$\ominus$" are Modulo-T_{system} addition and subtraction, respectively.

Central to the algorithm operation is procedure SampleReschedule(). This procedure is asynchronously triggered for execution at a node either when the higher layer process changes the demand of an adjacent link or after an adjacent link is rescheduled. When either of these events occurs, a non-root node u proceeds in execution of SampleReschedule() only if its parent link l_p is satisfied; the root proceeds in execution unconditionally.

During execution of SampleReschedule() at node u the following actions are performed:

1) If u is not the root, let $W_{l_p}^{(u)} = [s_{l_p}^{(u)}, e_{l_p}^{(u)}]$ be the window in $\boldsymbol{S}_u$ satisfying its parent link l_p. First, u assigns decreasing priorities to its children links in the (circular) order that they currently appear in $\boldsymbol{S}_u$, starting at slot $e_{l_p}^{(u)}$ and ending at $s_{l_p}^{(u)}$. (The root node assigns priorities using 0 and $T_{system} - 1$ as start and end slots, respectively).

2) By inspecting $\boldsymbol{S}_u$, node u samples its children in decreasing priority for violation of the stability conditions. If all links are found stable SampleReschedule() terminates and no action takes place. Otherwise, the highest priority unstable child link l_c is found. Let l_m be the stable link of immediately higher priority than l_c. If l_c is the highest priority child link, l_m is defined to be the parent link l_p. In either case, link l_m is satisfied. Link l_c needs to be rescheduled and stabilized.

3) Node u initiates rate adjustment on l_c by exchanging SC_INFO packets with the child endpoint c. After the exchange, u erases from $\boldsymbol{S}_u$ all slots currently allocated to l_c and considers a fresh allocation for a window W_{l_c} of $\tau_{l_c} + J_{l_c}^{(u)}$ slots. The position of W_{l_c} in $\boldsymbol{S}_u$ is determined as follows:

- First, node u computes the closest slot position to $s_{l_p}^{(u)}$ for which the stability conditions for l_c will hold:

$$s_{max} = s_{l_p}^{(u)} \ominus \sum_{k \in CH(u): p_k < p_{l_c}} (\tau_k + J_k^{(u)}) \qquad (6)$$

  Let $W_{l_m}^{(u)} = [s_{l_m}^{(u)}, e_{l_m}^{(u)}]$ be the window satisfying the demand of link l_m in $\boldsymbol{S}_u$. Link l_c will be stable if window W_{l_c} is scheduled within the window $W_{max}^{(u)} = [e_{l_m}^{(u)} \oplus 1, s_{max} \ominus 1]$.
- Node u decides on the position of $W_{l_c}^{(u)}$ within W_{max}: The new position of $W_{l_c}^{(u)}$ may cancel slots of lower-priority children links in $\boldsymbol{S}_u$. Also, the position of $W_{l_c}^{(u)}$ will be enforced to the local schedule of the child node c and may cancel slots on some of the children links of c. Using the local schedule of c (provided in the SC_INFO packet) the position of $W_{l_c}^{(u)}$ is selected within W_{max} such that the total number of affected links at both node endpoints is minimized.

4) Once u determines the position of $W_{l_c}^{(u)}$, it initiates the distributed coordination mechanism of section V-A by issuing SC_UPD packets to its affected neighbors. The coordination mechanism ensures that the local schedules of endpoint nodes u and c, as well as the local schedules of their affected neighbors, will be free of transmission conflicts after the update.

After l_c has been scheduled, node u must restart sampling from the highest priority child link for violation of the stability conditions. This is because the demands of links of higher priority than l_c may have changed while the rate adjustment was taking place. If the demands stop changing, repetitive invocation of procedure SampleReschedule() will reschedule and stabilize the unstable links in decreasing priority. The sampling-rescheduling loop terminates when all child links are found stable.

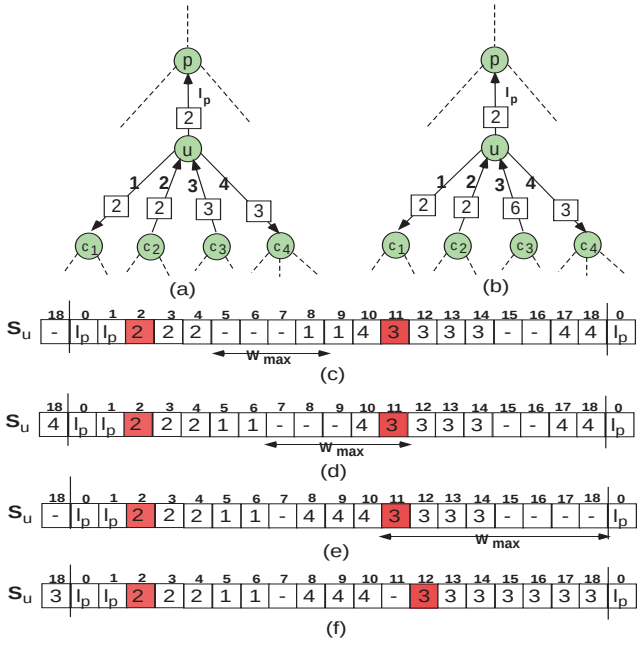

Fig. 3. (a) Arrows denote master-slave relationships and red slots denote switching slots of links where u is slave. (b) Demand of link 3 changes from 3 to 6. (c) The highest priority child link (2) is satisfied and the distance of slot 5 to slot 18 ($|[5, 18]| = 14$) is greater than the current demand sum of the lower priority child links ((2+0)+(6+1)+(3+0)=12)–link 2 is stable. The next priority link 1 is satisfied but not stable ($|[10, 18]| = 9 < (6 + 1) + (3 + 0) = 10$). To satisfy condition **STBL2**, window W_1 ($\tau_1 + J_1^{(u)} = 2 + 0 = 2$ slots) must be within $W_{max} = [5, 8]$. (d) $\boldsymbol{S_u}$ after link 1 has been rescheduled. The position was decided after the link coordination mechanism with node c_1 and consulting with S_{c_1}. Link 4 is not satisfied (**STF1** does not hold); it needs to be rescheduled within $W_{max} = [7, 10]$ to become stable. (e) $\boldsymbol{S_u}$ after link 1 has been rescheduled. Link 3 is not satisfied; it can be rescheduled within $W_{max} = [11, 18]$. (f) All links are now stable–the sampling-rescheduling loop is complete.

An example of SampleReschedule() is shown in Fig. 3. According to the initial local schedule $\boldsymbol{S}_u$ (Fig. 3(c)), the allocations on adjacent links of node u are $(t_{l_p}^{(u)}, t_1^{(u)}, .., t_4^{(u)}) = (2, 2, 3, 4, 3)$ and corresponding demands are $(\tau_{l_p}, \tau_1, ..., \tau_4) = (2, 2, 2, 3, 3)$. In Fig. 3(b) the demand of link 3 changes from 3 to 6 slots. Since the parent link l_p is satisfied ($t_{l_p}^{(u)} =$

$\tau_{l_p} + J_{l_p}^{(u)} = 4$), node u initiates SampleReschedule(). Using the window $[0, 1]$ assigned to its parent link l_p, u assigns decreasing priorities to its children links in the cyclic order they appear in $\boldsymbol{S}_u$, starting from slot 1 towards slot 0. The links in decreasing priority are $2, 1, 4, 3$. Figures 3(c)-(f) illustrate a sequence of steps and modifications of S_u that stabilize the links.

The above description corresponds to the desired operation of SampleReschedule() at a node u. However, the fact that nodes may be busy at any time makes things more complicated. For example, when the highest priority unstable child link is sampled, it may be currently busy scheduling a child of its own and, therefore, unavailable for re-scheduling. Hence, a need exists for coordinating parent and children to allow proper operation of the sampling re-scheduling loop. This is accomplished by the STABLE_REQ/STABLE_ACK packet exchange. Before executing SampleReschedule() node u requests permission from its parent by sending a STABLE_REQ packet. The parent will respond in one of two possible ways: 1) it replies with a STABLE_ACK packet as a permission for u to continue sampling and rescheduling its children. 2) It initiates a rate adjustment on this link via an SC_INFO packet.

In the example of Fig. 3, node u must perform a STABLE_REQ/STABLE_ACK handshake with its parent p for every child link it reschedules. If link l_p becomes unstable during this process, the parent will respond with an SC_INFO packet and link l_p will be rescheduled. Based on the new stable window l_p, node u will reassign priorities and resume the sampling-rescheduling loop. The detailed operation of the asynchronous protocol, called STABLE_TREE, is described in Figure X-A, Appendix X-A

Convergence Theorem: Consider an initial tree topology and network TDMA schedule. Assume that a set of arbitrary demand and topology changes occur that eventually stabilize to a new tree topology and demand allocation $\boldsymbol{\tau}$ obeying the capacity condition (5). The asynchronous distributed algorithm will converge to a new TDMA schedule realizing $\boldsymbol{\tau}$ in a finite number of link rate adjustments.

Proof: In general, nodes re-assign slots on their adjacent links via SampleReschedule() when they detect demands that are not satisfied. We show that, as soon as the changes in demands stabilize, convergence is guaranteed to happen progressively from the root downwards. We provide the detailed proof in Appendix X-C.

The convergence delay of STABLE_TREE depends on the tree depth and T_{system}. For a worst-case analysis, let us assume that all links have become unsatisfied due to the changes. Since convergence is guaranteed from the root downwards, in the worst-case scenario all links will need to be rescheduled in this order. Also, the worst tree topology is a line starting at the root node–in this case all $(N-1)$ links will be scheduled sequentially in time.

According to the link coordination mechanism, the endpoints wait for acknowledgements from all the affected neighbors before updating their local schedules. In the worst case, all neighbors are affected and acknowledgments will arrive within T_{system} slots. Therefore, each link activation for rate adjustment has a maximum duration of T_{system} slots.

When a node samples the highest priority unstable link, it will wait at most T_{system} slots in case the child node is busy. Thus each link on the line will be scheduled in at most $2T_{system}$ slots. We conclude that once link demands have stabilized, STABLE_TREE will converge within $2T_{system}(N-1)$ slots.

The worst-case analysis assumes that all links become unsatisfied and rescheduling will happen in the order that guarantees convergence–starting from the root downwards. Since nodes continuously detect changes and reassign slots locally, convergence may occur faster in practice. In addition, demands may be changing locally at lower tree levels; only part of the tree will need to be rescheduled in this case. Existing tree topology control algorithms strive to maintain balanced structures. In this case, even if links will need to be scheduled from the root downwards, multiple links will be scheduled in parallel. Also, during a link rate adjustment, not all neighbors are always affected and acknowledgements may arrive in less than T_{system} slots. The convergence behavior of STABLE_TREE in practice will be investigated in the experiments section, jointly with end-to-end bandwidth allocation mechanisms addressed next.

VI. End-to-end rate guarantees

We now introduce a framework for integrating link scheduling with end-to-end bandwidth allocation. The asynchronous TDMA ad hoc network $G(N, E)$ is shared by a set of unicast multi-hop sessions. Without loss of generality, we assume that half-duplex parts of a slot assigned to a link have equal duration D_{slot} and are used by the same session. Although bidirectional transfer is supported over a path, we assume that data traffic flows in a single direction.

Each node can transmit at a maximum rate of R bps on a link. To support a rate of $\rho_i (\leq R)$ bps for session i over a path, the network must be able to allocate $\tau_i = \lceil (\rho_i / R) \cdot T_{system} \rceil$ conflict-free slots for i to all links in the path.

Since each slot assigned to a link can be used only by a single session, the total bandwidth consumed by the sessions $F(u)$ sharing node u must obey the local feasibility conditions:

$$\sum_{i \in F(u)} \delta_i^{(u)} \cdot \tau_i \leq T_u^R, \; \forall u \in N \tag{7}$$

where

$$\delta_i^{(u)} = \begin{cases} 1 & \text{if } u \; is \; source \; or \; destination \; of \; session \; i \\ 2 & \text{otherwise} \end{cases}$$

and

$$T_u^R = \begin{cases} \lfloor 1/3 \cdot T_{system} \rfloor & \text{if } G \; arbitrary \\ \lfloor 1/2 \cdot T_{system} \rfloor & \text{if } G \; bipartite \\ T_{system} - \sum_{l \in L(u)} J_l^{(u)} & \text{if} G \; tree \end{cases} \tag{8}$$

The term $\delta_i^{(u)}$ indicates that, in order to support allocation τ_i for session i, an intermediate node u must be able to communicate for τ_i slots on both upstream and downstream links of the session.

The utilization factor T_u^R depends on the topology control mechanism used by the network (if any) and equals the maximum number of slots each node can provide in each case

to ensure feasibility via local conditions. According to 8, more restricted topologies can be better utilized and trees allow maximum utilization. The integrated framework provides end-to-end bandwidth allocations and guarantees using three independent components:

End-to-end rate allocation: Conditions (7) are used for allocating feasible rates to the multi-hop sessions.

Link scheduling: The session rates are translated to (feasible) link demands:

$$\sum_{i \in F(l)} \tau_i = \tau_l, \; \forall l \in E \tag{9}$$

where $F(l)$ is the set of sessions crossing link l. The link demands are realized by a distributed dynamic link scheduling algorithm. STABLE_TREE is such an algorithm for tree topologies.

End-to-end rate enforcement: Once the link scheduling algorithm converges, every link has been allocated enough bandwidth (conflict-free slots) to support the session demands. The slots allocated to each link can be shared to its sessions according to their demands, using Weighted Round Robin (WRR), Weighted Fair Queuing (WFQ) [24] or other single-server queuing disciplines. Since the asynchronous TDMA architecture uses fixed-size slots, WRR would be a reasonable choice in this case. Another possibility is to combine First-Come-First-Serve (FCFS) queuing at intermediate links with explicit control of the transmission rates at the source nodes. The choice will depend on the target environment and application requirements.

Decoupling end-to-end bandwidth allocation from link scheduling, allows definition and realization of various end-to-end QoS objectives. In the following sections we introduce end-to-end rate allocation mechanisms for Constant Bit Rate (CBR) and Available Bit Rate (ABR) services.

A. Constant Bit Rate (CBR) Service

According to the CBR service model, sessions have fixed rate requirements that need to be satisfied by the network. A typical application is packetized voice. For each session arriving at a source node, a path supporting the requested rate to the destination must be determined.

Session i with rate demand ρ_i bps can be admitted on a path $u_1, u_2, ..., u_p$ if the corresponding demand allocation $\tau_i = \lceil (\rho_i / R) \cdot T_{system} \rceil$ does not exceed the minimum available node capacity over the path. Therefore, session i is admitted if:

$$\tau_i \leq \min_{k \in 1,..,p} \left\lfloor \frac{T^R_{u_k} - \sum_{j \in F(u_k)} \delta_j^{(u_k)} \tau_j}{\delta_i^{(u_k)}} \right\rfloor \tag{10}$$

A similar admission control rule is used in wireline networks. The difference here is that the shared resources over the path are nodes instead of links. The rule in eq. (10) admits sessions without taking into account the arrangement of slots in the current TDMA schedule. This is possible due to the underlying dynamic link scheduling algorithm. If the session i is admitted, the demands of all links on the selected path are increased by τ_i slots. As soon as the nodes in the path detect the demand changes on their adjacent links, they use the link scheduling algorithm to re-assign transmission slots and converge to a new TDMA schedule realizing the new link (and end-to-end) demand allocation. In case the session is admissible by multiple paths, a path selection criterion similar to ones used for wireline networks can be used (see [25] and references therein). The admission control rule over a single path and the path selection criterion, together constitute a QoS routing algorithm.

QoS routing in ad hoc networks via TDMA has also been considered in [3] for multi-channel systems and [4] for single-channel systems. The main difference of these algorithms with our approach is that they do not allow slot re-assignments for accommodation of incoming sessions. Instead, they keep the slot positions of existing sessions fixed and seek to allocate available slots to incoming sessions subject to the current state of the TDMA schedule. Finding the maximum number of available slots on a path subject to the slot positions of the existing sessions is an NP-complete problem, even if global topology information is available. The authors propose distributed heuristics for available path bandwidth estimation and slot assignment.

In exchange to the more complex admission control, [3], [4] operate in arbitrary topologies, while our approach currently provides rate allocation and enforcement for tree topologies. However, [3], [4] assume that global slot synchronization exists. In addition, no flexibility can be provided for scheduling the sessions on the links: the slot positions for an incoming session are (uniquely) determined during the path bandwidth calculation.

Due to the heuristic nature of the available path bandwidth estimation in [3], [4], sessions that could be accepted are blocked, i.e. the network is underutilized. Underutilization is also unpredictable: given a set of session arrivals, the number of admitted sessions depends on the order of arrivals. For tree topologies, our approach will admit the maximum possible number of sessions irrespective of the order of session arrivals. However, the ability to admit more sessions comes with the penalty that some existing sessions may not receive their requested service while the TDMA schedule is reorganized to accommodate incoming sessions. Thus, provision of continuous CBR service requires a detailed experimental study of convergence delay under various traffic loads.

Although it would be interesting to experimentally compare the two approaches in terms of their strengths and weaknesses, we refrain from doing so here on account of space. Instead, we focus on end-to-end Available Bit Rate (ABR) service, not currently provided for multi-hop wireless networks. According to ABR, sessions do not arrive with specific bandwidth requirements but agree to comply with what is available by the network. Such a setting necessitates provision of fair access to the end-to-end sessions. The approach of [3], [4] cannot be applied in this case, because it is specific to the path bandwidth allocation mechanism which is closely tied to the underlying slotted structure.

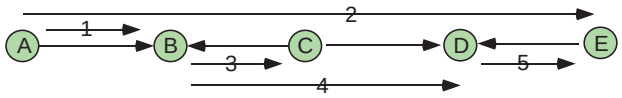

Fig. 4. For ease of illustration, we compute the MMF session rates with respect to fractional capacities $C_u^R = C = 1 - \frac{2}{T_{system}}$. The MMF rate in the first iteration is $C/5$ (bottlenecks are B and C). Sessions 1,2,3,4 are allocated C/5 and they are removed from the network, along with bottleneck nodes B,C. Node A is also removed since all sessions crossing it have been removed. The bottleneck in the second iteration is node D providing all its remaining bandwidth $(2/5 \cdot C)$ to session 5. The session MMF normalized rates are $(r_1, r_2, r_3, r_4, r_5) = (1/5, 1/5, 1/5, 1/5, 2/5) \cdot C$

B. Available Bit Rate (ABR) service

In the ABR framework, optimality is understood as allocating bandwidth to sessions in a maxmin fair manner. For convenience and ease of illustration, we will use normalized rates instead of slots to represent the sharing of bandwidth to the end-to-end sessions. Given slot allocation $\boldsymbol{\tau}$, the corresponding normalized rate allocation is $\boldsymbol{r} = \boldsymbol{\tau}/T_{system}$. Conversely, the slot allocation corresponding to rate allocation $\boldsymbol{r}$ is $\boldsymbol{\tau} = \lfloor \boldsymbol{r} \cdot T_{system} \rfloor$.

A session rate allocation $\boldsymbol{r} = (r_1, .., r_{|F|})$ is *feasible*, if for every session i, each link in the path $L(i)$ can support $\tau_i = \lfloor r_i \cdot T_{system} \rfloor$ slots, that is, the induced demand slot allocation on the network links is feasible. A feasible rate allocation is *maxmin fair (MMF)*, if the rate of a session cannot be increased without decreasing the rate of another session of equal or lower rate. More formally, a feasible rate allocation $(r_1, .., r_{|F|})$ is MMF if it satisfies the following property with respect to another feasible rate allocation $(r_1^{'}, ..., r_{|F|}^{'})$: if there exists a session i such that $r_i < r_i'$, then there exists a j such that $r_j \leq r_i$ and $r_j^{'} < r_j$.

Determining feasibility of a session demand allocation requires determining feasibility of the corresponding link slot allocation. According to [5], [6] this problem is NP complete for arbitrary topologies. Since MMF allocations are by definition feasible, finding or detecting them for arbitrary topologies becomes problematic. We will thus assume that only part of the overall network capacity is utilized for ABR, and seek the MMF rates with respect to this fraction. The fraction depends on the degree of topology control and is determined by the local feasibility conditions (written in terms of normalized rates by dividing both sides of (7) with T_{system}):

$$\sum_{i \in F(u)} \delta_i^{(u)} \cdot r_i \leq C_u^R, \; \forall u \in N \tag{11}$$

where $C_u^R = T_u^R / T_{system}$. Note that, for tree topologies, it is possible to compute the absolute MMF rates since (11) captures the entire set of feasible allocations in this case.

Node u is defined to be a *bottleneck* for session i if u is fully utilized (with respect to C_u^R) and session i has been allocated maximum rate over all sessions $F(u)$ sharing u. The definition of bottleneck node yields a criterion for determining whether a given session allocation is MMF:

MMF criterion: A session rate allocation $\boldsymbol{r} = (r_1, ..., r_i, ..., r_{|F|})$ is MMF if and only if every session has at least one bottleneck node.

The session MMF rates can be computed using an iterative, off-line centralized algorithm similar to the algorithm of Bertsekas and Gallager for wireline networks [26]. The modification has to take into account that, in our case, the resources are nodes instead of links and that sessions in intermediate nodes need to consume twice the bandwidth than their allocated rate due to the slots needed at both incoming and outgoing links.

During each iteration of the centralized algorithm, each node equally divides its available bandwidth over the total number of sessions on its adjacent links. The bottlenecks of the current iteration are the nodes for which this division is minimum; the minimum ratio is the MMF rate for this iteration and is allocated to the sessions crossing the bottleneck nodes. We then remove the bottleneck nodes and their sessions from the network and reduce the available bandwidth of the remaining nodes by the amount consumed by the removed sessions (for each intermediate node in the path of each removed session, we must subtract twice the MMF rate from the node available bandwidth). Any node whose available bandwidth becomes zero is also removed. We then consider the next level bottleneck nodes of the reduced network and repeat the procedure. We continue until all sessions have been allocated their rates. The algorithm operation is described in the example of Fig. 4.

We have implemented an asynchronous distributed version of the centralized algorithm. The distributed algorithm is similar in spirit with algorithms proposed for wireline ATM networks [27][28][29]. This is a rate-based approach for flow control, where each source adjusts its transmission rate based on values seen in returning control packets, previously injected and circulated over the session path. The returning values are the most recent estimates of the session MMF rate, as computed by all nodes in the session path.

Every node u maintains a subset $FC(u)$ of its sessions $F(u)$, currently seen as "constrained" by other nodes. It also maintains an estimate ϕ_u for the MMF rate it currently provides to its unconstrained sessions. The MMF rate estimate ϕ_u is updated locally by procedure MMF_UpdateState(). The source or an intermediate node of a session invokes MMF_UpdateState() when a control packet is about to be sent to the downstream link (forward direction); the destination node invokes MMF_UpdateState() when a session control packet is about to be sent to the upstream link (reverse direction). In each case, when a control packet p of session i is about to be sent on link l, procedure MMF_UpdateState() at node u involves the following actions (Fig. 5):

Step 1: Node u updates the rate r_i of session i as the minimum of ϕ_u and the value in the rate field of packet p.

Step 2: The demand of link l is updated to reflect the change in r_i. If u is an intermediate node of session i, the demand of the other adjacent link k shared by i is updated in a simlilar fashion. The new link demand(s) are passed to the link scheduling algorithm.

Step 3: If ϕ_u is less than or equal to the value carried by the packet, it is copied to the packet rate field. Also a bit in the packet is set to indicate that the session is constrained by a node in the path. Otherwise, session i is added to $FC(u)$ and

the packet contents are not modified.

Step 4: Node u updates the MMF estimate ϕ_u by subtracting the bandwidth taken by the currently constrained sessions and equally dividing the rest of the bandwidth to the unconstrained sessions.

Step 5: The rates of some sessions in $FC(u)$ may be greater than the new ϕ_u. If this is the case, these sessions are removed from $FC(u)$ and step 4 is repeated. After the second iteration, it is guaranteed that no sessions in $FC(u)$ will have rate greater than ϕ_u.

Upon return of a control packet, the source adjusts the transmission rate according to the packet rate field. If the field indicates a value of r_i, the source adjusts its sending rate to $r_i \cdot R$ bps, where R is the maximum transmission rate of the radio in bps. The new control packets for session i are sent out with the packet rate field set to r_i and the constrained bit field set to zero.

Using arguments similar to those in [27], it can be proven that the asynchronous distributed algorithm converges in a finite number of iterations to the end-to-end MMF rate values. This holds for any topology form, given the appropriate fractional capacities C_u^R that ensure feasibility in each case. The main difference of the distributed algorithm with the wireline versions lies in the update of the MMF estimated rate that divides available rate of each node to its session parts (instead of sessions), and in that *every* node in the path–including the source and destination nodes–must update the MMF rate estimate. According to step 2 of MMF_UpdateState(), the demands of adjacent links are updated and passed to the link scheduling algorithm. Viewed globally, the end-to-end computation and link scheduling processes happen in parallel. The link scheduling is not aware of whether the end-to-end process is complete; it simply reacts to the link demand updates. As soon as the end-to-end bandwidth allocation converges to the MMF rates, the link demands stabilize and the link scheduling algorithm can converge.

Procedure `MMF_UpdateState`

Update algorithm at node u for a control packet p of session i to be forwarded on link l

1 $r_i = \min(\phi_u, p.rate)$ /*update the session rate*/;
$\tau_i = \lfloor r_i \cdot T_{system} \rfloor$;
2 $\tau_l = \sum_{j \in F(l)} \tau_j$ /*update demand of link l*/;
if ($\delta_i^{(u)} == 2$) /*u is intermediate node of i*/ **then**
  $\tau_k = \sum_{j \in F(k)} \tau_j$ /*update demand of the other link k adjacent to u where session i belongs*/;
end
3 **if** ($\phi_u \leq p.rate$) **then**
  $p.rate = \phi_u$; $p.constrained = 1$;
end
if ($\phi_u \geq p.rate$) **then**
  $FC(u) = FC(u) \bigcup \{i\}$;
end
4 **if** ($|FC(u)| == |F(u)|$) **then**
  $\phi_u = C_u^R - \sum_{j \in F(u)} r_j + \max_{j \in F(u)} r_j$;
else
  $\phi_u = \frac{C_u^R - \sum_{j \in FC(u)} \delta_j^{(u)} \cdot r_j}{\sum_{j \in F(u)} \delta_j^{(u)} - \sum_{j \in FC(u)} \delta_j^{(u)}}$;
end
5 **if** *exists* j *in* $FC(u)$ *such that* $r_j \geq \phi_u$ **then**
  for *all* j *in* $FC(u)$ *such that* $r_j \geq \phi_u$ **do**
    FC(u)=FC(u)-{j};
  end
  repeat step 4;
end

Fig. 5. Update algorithm for session rate, link demands and MMF rate estimate ϕ_u.

VII. Bluetooth Implementation

A. Design

Bluetooth [30] is an instance of a multi-channel asynchronous TDMA system with a special constraint that a node can be master to at most seven adjacent links. Channels are implemented as frequency hopping sequences and termed as *piconets*. A Bluetooth ad hoc network is termed as a *scatternet*.

Figure 6 depicts the implementation of the end-to-end bandwidth allocation algorithm, the link scheduling algorithm and the coordination mechanism over the Bluetooth protocol stack. The Bluetooth Baseband layer operates according to the asynchronous TDMA scheme presented in section II. The Bluetooth Link Manager Protocol (LMP) is used for exchange of baseband control packets. Ideally, the link scheduling protocol and coordination mechanism would be implemented in the Baseband with the control packets being LMP messages. The current Bluetooth specification does not offer periodic scheduling at the Baseband layer. We have therefore implemented the link scheduling and coordination mechanisms in software, at the application layer.

We use the Bluetooth sniff mode to instruct the Baseband to transmit according to the schedule maintained at the application layer. Sniff mode is a low power mode where a slave can listen to a master for only a window of $N_{sniff\_attempt}$ slots within a period of T_{sniff} slots. Before entering sniff mode the nodes must agree on a slot offset within the period where they will commmunicate. The Bluetooth Host Controller Interface (HCI) exports a function where a node (either master or slave) can initiate sniff mode on a link. We can thus directly map T_{system} to T_{sniff}. Each node will impose different non-overlapping sniff windows to its neighbors. When, during the execution of the coordination mechanism, the local schedule of a node is modified at the application layer, we instruct the hardware to start sniff mode on link l on that offset by setting $N_{sniff\_attempt} = \tau_l + J_l^{(u)}$. Sniff mode has also been used in other approaches specifically targeted for scatternet scheduling [31][32].

The Bluetooth L2CAP layer provides connection-oriented and connectionless services to upper layer protocols. It can support both unidirectional and bidirectional logical channels between two nodes. For the exchange of the link coordina-

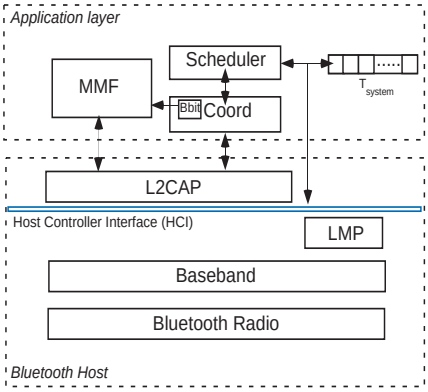

Fig. 6. Implementation of the end-to-end bandwidth allocation framework over the Bluetooth stack

tion mechanism control packets we use a bidirectional L2CAP channel. Each session consists of multiple L2CAP bidirectional channels, one for each link in the path. Thus, a session at an intermediate node is mapped on two L2CAP bidirectional channels, one to the upstream and the other to downstream link. Session data packets or control packets flowing in the forward direction are sent on the downstream L2CAP connection while session control packets returning to the source to the upstream L2CAP connection.

When a source receives feedback control packet with normalized rate r_i, it adjusts its transmission rate to $r_i \cdot B/D_{slot}$ bits/sec, where B/D_{slot} is the ratio of maximum payload bits per direction over the duration of a full-duplex slot. The Bluetooth baseband layer supports half-duplex slots of duration $0.625ms$. Each half-duplex slot can support up to $B = 216$ bits for payload data (Bluetooth DH1 packets). Slots can be combined in full duplex configurations of $(1,1)$, $(1,3)$, $(1,5)$ half duplex slots. In the experiments we use $(1,1)$ configuration. Thus, a full-duplex slot has duration equal to $2 \cdot D_{slot} = 1.25ms$ and a maximum rate $R = B/2D_{slot} = 172.8$ Kbps per direction can be supported.

To enhance performance, in addition to rate adjustment at the sources, a packet scheduler is used on every link l to decide the type of packet that will be transmitted on a conflict-free slot. To expedite convergence of the link scheduling algorithm, link control packets are given highest priority. When the link control packet queue is empty, Weighted Round Robin (WRR) is used to share the bandwidth among the outgoing sessions on this link. When the demand of link l changes during the the end-to-end algorithm execution (step 2 of MMF_UpdateState() in Fig. 5), the WRR weight $W_i(t)$ for each session i in the set of outgoing sessions $OUTF(l)$ of this link is updated as $W_i(t) = \frac{r_i(t)}{\min\limits_{j \in OUTF(l)} r_j(t)}$. Then, a new WRR cycle is constructed that will schedule sessions in proportion to their new relative weights. All WRR weights stabilize when all link demands stabilize; the target rates will be enforced when the desired TDMA link schedule has been reached.

B. Experiments

To test the system in complex configurations we use BlueHoc[33], the IBM Bluetooth extensions to the NS simulator [34]. We have further extended BlueHoc to support scatternets and the sniff mode. The link scheduling algorithm, the end-to-end algorithm and the coordination mechanism have been implemented as separate modules. We have performed experiments on various topology and session configurations. Due to space limitations, we will present and analyze a representative case here.

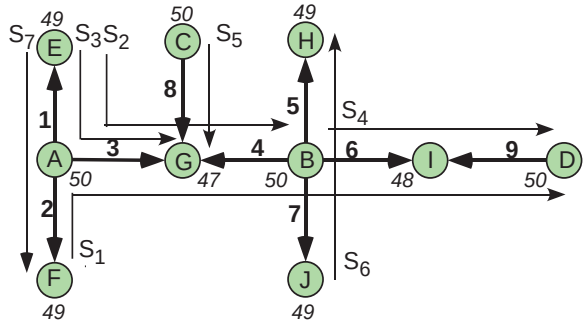

Fig. 7. Arrows on links denote master-slave relationships. Italicized numbers on each node u denote $T_u^R = T_{system} - \sum_{l \in L(u)} J_l^{(u)}$, where T_{system} = 50 slots. The normalized capacities are $C_u^R = T_u^R / T_{system}$; the (normalized) MMF rates are $(r_{S_1}, .., r_{S_7}) = (0.125, 0.125, 0.125, 0.208, 0.315, 0.208, 0.125)$. These rates correspond to $(\tau_{S_1}, .., \tau_{S_7}) = (6, 6, 6, 10, 15, 10, 6)$ slots within $T_{system} = 50$ slots.

We consider the configuration shown in Fig. 7. A period of $T_{system} = 50$ slots is used. Nodes start with an arbitrary conflict-free TDMA schedule; all sources start transmitting at maximum rate (172.8 Kbps) and subsequently adjust it based on the values of the received end-to-end control packets. Time is measured with respect to the time slot reference of the root node. Each simulation lasts 20000 slots (or $20000 \times 1.25ms = 25sec$).

Convergence delay is determined by D_S, the time until the link demands stabilize due to the end-to-end algorithm convergence, and D_L, the additional delay needed by the distributed link scheduling algorithm to converge to a TDMA schedule realizing these demands. The table in Fig. 8 includes D_S and D_L

	Convergence Delay (slots) and Overhead (%)				
Root	D_S	D_L	$D_S + D_L$	$O_S(\%)$	$O_L(\%)$
A	1523	469	1992	11.5	9.7
B	3145	178	3323	8.5	7.2
C	1995	282	2277	17.05	11.80
D	1718	733	2451	12.54	10.2
E	2529	196	2725	15.8	8.78
F	2765	327	3092	11.25	10.3
G	2836	392	3228	8.74	7.05
H	1943	436	2379	12.56	9.9
I	1982	361	2343	16.31	11.86
J	2225	543	2768	15.44	9.2

Fig. 8. Convergence delay and control overhead in the configuration of Fig. 7, for different choices of the root node.

that resulted from different choices of the root node. Both delay components depend on the location of the root and the order with which the end-to-end algorithm satisfies the sessions–in

increasing order of MMF rates. According to Fig. 7, sessions S_1, S_2, S_3 and S_7 first receive the lowest MMF rate (0.125) due to the first-level bottleneck node A. Then the MMF rates of S_5 (0.315) and S_4,S_6 (0.208) will be allocated by the second-level bottleneck nodes G and B, respectively.

In addition to the location of the root, the delay component D_S depends on the transient states of the TDMA schedule. During the TDMA schedule modifications, some links may be occasionally allocated a few slots. Since slots are shared by link control packets, as well as control and data packets of various sessions, this may delay the circulation of the control packets of some sessions and, consequently, increase the convergence delay of the end-to-end algorithm. This phenomenon was observed in the case of maximum D_S (3145 slots) where node B is the root. Link 3 was allocated 5 slots or less until slot 904; links 4 and 8 were constantly being rescheduled at the expense of link 3 during this period. Link 3 is in the control path of sessions S_1 and S_2, which belong to the set of sessions whose MMF rates must be computed first; this slowed down the end-to-end algorithm convergence. This behavior did not arise for all other choices of the root. The minimum D_S (1523 slots) was observed when the root was selected as the first-level bottleneck node A.

The delay component D_L depends on the order link demands have stabilized, and the location of the root with respect to this order. The link demands stabilize in the order they are "removed" (along with bottleneck nodes and sessions) during the end-to-end algorithm execution. In Fig. 7, the first demands to stabilize will be of links 1-4 because these links are crossed by the first-level sessions S_1,S_2,S_3 and S_7 (and only those sessions). Then, links 5-9 will follow, due to the second-level sessions S_4-S_6. This reasoning provides the order for demand stabilization among groups of links. The order within a group depends on the specific experiment run. According to Fig. 8, maximum D_L (733 slots) occurred when node D was selected as root. In this run, the last demand to stabilize (at slot $D_S = 1718$) was link 9, the only link adjacent to the root. We observed that, although at slot D_S the link demands at lower tree levels had already been stabilized and satisfied, the entire tree was rescheduled from the root downwards. This worst-case global re-scheduling did not occur for similar scenarios; for example, when node I was selected as root, it was adjacent to the slowest converging demand (link 6 at slot $D_S = 1982$) but in this run the tree was partially re-scheduled and convergence occurred within 361 slots. Incidentally, the minimum D_L was observed for the root being B, the case that yielded maximum D_S. In all experiments D_L is less than $2T_{system}(N-1) = 900$ slots, the delay bound of the tree link scheduling algorithm.

Another quantity of interest during convergence is the control overhead, expressed as the fraction of slots used for control packet transmissions. The control overhead consists of the overhead due to the link coordination mechanism (SC_INFO, SC_UPD, SC_UPD_ACK, STABLE_REQ and STABLE_ACK packets) and the overhead due to the circulating end-to-end control packets. According to Fig. 8, the link control overhead is greater during D_S (maximum $O_S = 17.05\%$) because the link demands change constantly during this period. After the link demands stabilize, the link control overhead O_L is in the order of 10% on the average. Link control overhead is dominated by the STABLE_REQ/STABLE_ACK packet exchanges– a node must request permission from its parent for each unstable child link it needs to reschedule. Similar to ATM networks, the end-to-end control overhead is regulated at the source by sending 1 control for every P data packets. The parameter P can be adjusted to trade-off increased speed of convergence for increased overhead. In the experiments we use $P = 19$; this yields a fixed overhead of 5%.

After convergence, only end-to-end control overhead exists because the sources are never aware that the MMF rates have been reached. Constant flow of end-to-end control packets is needed for dynamic recomputation of the session MMF rates in presence of network dynamics (session additions and removals). The table in Fig. 9 depicts the session throughput and goodput as well as average delay between data packet arrivals measured at the session destination after convergence. The throughput (goodput) of a session in bps is the number of bits due to data or control packets (data packets only) the destination receives for this session from the time of convergence ($D_S + D_L$) until the end of the simulation run. The session throughputs exactly match the MMF rates; as expected, the goodput of every session is approximately 5% less than the throughput on account of the end-to-end control overhead. Sessions within the same MMF group experience similar average delay (D_{avg}) within a small 95% confidence interval (d_{95}); this is due to the TDMA schedule periodicity and the WRR link schedulers employed over each session path.

	Rates (Kbps) and Delay (ms)				
Root	MMF	T	G	D_{avg}	d_{95}
S_1	20.73	20.73	19.69	10.65	$\pm$ 1.25
S_2	20.73	20.73	19.66	10.71	$\pm$ 1.22
S_3	20.73	20.73	19.66	10.69	$\pm$ 1.20
S_4	34.56	34.56	32.83	6.54	$\pm$ 0.73
S_5	51.84	51.84	49.24	4.284	$\pm$ 0.78
S_6	34.56	34.56	32.83	6.54	$\pm$ 0.73
S_7	20.73	20.73	19.68	10.63	$\pm$ 1.21

Fig. 9. Session throughput (T), goodput(G) and average delay (D_{avg}) with 95% confidence intervals (d_{95}) for the configuration in Fig. 7, measured at each session destination after convergence.

VIII. Related work

Provision of end-to-end fairness via distributed link scheduling has not been addressed yet for wireless ad hoc networks. Weighted fairness, utility-based fairness and maxmin fairness have been defined and addressed for single-hop sessions (links) in [35], [36] and [37], respectively. Distributed random access MAC protocols are used to coordinate transmissions– fairness can be realized only in a probabilistic sense. Tassiulas and Sarkar [38] define single-hop maxmin fairness in slotted TDMA systems. A distributed token generation mechanism is used to compute the MMF rates for the network links. However, the scheduling needs global topology information and a maximum weighted matching computation at every slot to enforce these rates. In a sequel paper [39], the token generation mechanism is coupled with back-pressure flow control to compute the

maxmin fair rates for end-to-end sessions. The centralized link scheduling component is still required.

The TDMA link scheduling in [38], [39] can only achieve long-term fair rates because operation is considered in a slotted system of infinite horizon. Short term rate guarantees, as well as bounded delay (to the extent of the system period) can be achieved by periodic TDMA schedules. Our approach uses a fixed system period T_{system} combined with a set of local feasibility conditions. Another line of research has focused on link scheduling algorithms for systems of variable period[8], [9], [12]. These algorithms try to reach a TDMA link schedule with short period realizing a given link demand slot allocation (finding the minimum period and TDMA schedule is an NP-complete problem). In [8], a centralized heuristic is introduced. A subsequent semi-centralized implementation [9] requires that information about each change in topology or demand to be distributed to all nodes; then each node uses an identical copy of the centralized algorithm [8] to compute the allocation on its adjacent links. This approach would not be practical for settings of frequent changes in topology or traffic demands. DSSA [12], a link scheduling algorithm for Bluetooth scatternets, uses only local information. Nodes start with knowledge of the demands of their adjacent links and use a heuristic to construct a TDMA link schedule realizing the demands in a short period. However, the algorithm termination is not distributed; this renders implementation harder in dynamic settings. In addition to the difficulties specific to each algorithm of [8], [9], [12], variable-period systems by nature need to suspend communications while a new TDMA schedule is computed in response to changes in topology or traffic demands. Once the computation is complete, a signal must be broadcast to the entire network before operation is resumed.

Distributed TDMA scheduling schemes of lower complexity have been proposed for both slot-synchronized [10], [11] and asynchronous TDMA systems [40], [41], [32]. These schemes are dynamic but do not target specific link demand allocations. Hence, they cannot be used for provision of deterministic bandwidth guarantees to links or end-to-end sessions.

IX. Conclusions

We presented a framework where end-to-end bandwidth allocation algorithms currently available for wireline networks can be used with certain modifications for wireless ad hoc networks if we can find a set of appropriate local feasibility conditions and an underlying distributed, self-stabilizing link scheduling algorithm. The link scheduling is based on an asynchronous TDMA protocol that does not rely on global slot synchronization or knowledge of the number of nodes in the network.

Using this framework, we proposed an algorithm for provision of end-to-end ABR service to multi-hop sessions. This algorithm can operate for any topology and compute the session MMF rates with respect to a fraction of the network capacity provided by the local feasibility conditions. We showed that, in the case of tree topologies, the network can be fully utilized and a link scheduling algorithm that can enforce the computed end-to-end rates exists. We presented an implementation of this framework over Bluetooth, an existing asynchronous TDMA wireless technology.

A natural extension for the link scheduling component of the framework is the design of converging algorithms that provide rate enforcement in more general topologies than trees (at the inevitable expense of reduced utilization). Such algorithms are the subject of our future research efforts.

References

[1] S. Chen and C. Nahrstedt. Distributed Quality of Service Routing in ad hoc networks. *Proc. IEEE Journal on Selected Areas in Communications*, 17, August 1999.

[2] M. Gerla and T. Tsai. Multicluster, mobile multimedia radio network. *Proc. ACM Baltzer Journal of Wireless Networks*, 1:255–65, August 1995.

[3] C.R. Lin. On-demand QoS routing in Multihop mobile networks. In *Proc. IEEE INFOCOM*, Anchorage, AK, April 2001.

[4] C. Zhu and M.S. Corson. QoS routing for mobile ad hoc networks. In *Proc. IEEE INFOCOM*, New York, NY, June 2002.

[5] E. Arikan. Some complexity results about packet radio networks. *Proc. IEEE Transactions on Information Theory*, 30:681–685, July 1984.

[6] I. Holyer. The NP-completeness of edge coloring. *Proc. SIAM Journal of Computing*, 10:169–197, 1981.

[7] J. Silvester. Perfect Scheduling in Multihop Broadcast Networks. In *Proc. International Conference on Computer Communications (ICC)*, London, England, Sepmteber 1982.

[8] M. Post, P. Sarachik, and A. Kershenbaum. A Biased Greedy Algorithm for Scheduling Multihop Radio Networks. In *Proc. Annual Conference on Information Sciences and Systems (CISS)*, Johns Hopkins Univ., March 1985.

[9] M. Post, A. Kershenbaum, and P. Sarachik. A Distributed Evolutionary Algorithm for Reorganizing Network Communications. In *Proc. MILCOM*, Boston, MA, October 1985.

[10] D.J. Baker and A. Ephremides. The architectural organization of a packet radio network via a distributed algorithm. *Proc. IEEE Transactions on Communications*, 29:1694–1701, 1981.

[11] A. Kershenbaum and M. Post. Distributed Scheduling of CDMA Networks with Minimal Information. *Proc. IEEE Transactions on Communications*, 39, January 1991.

[12] U. Korner N. Johansson and L. Tassiulas. A distributed scheduling algorithm for a Bluetooth scatternet. In *Proc. International Teletraffic Congress (ITC)*, Salvador da Bahia, Brazil, September 2001.

[13] G.V. Záruba, S. Basagni, and I. Chlamtac. Bluetrees - scatternet formation to enable Bluetooth-based ad hoc networks. In *Proc. International Conference on Computer Communications (ICC)*, St. Petersburg, Russia, June 2001.

[14] G. Tan, A. Miu, J. Guttag, and H. Balakrishnan. An Efficient Scatternet Formation Algorithm for Dynamic Environments. In *Proc. IASTED Communications and Computer Networks (CCN)*, Cambridge, MA, November 2002.

[15] R. Guerin, J. Rank, S. Sarkar, and E. Vergetis. Forming Connected Topologies in Bluetooth Adhoc Networks. In *Proc. International Teletraffic Congress (ITC)*, Berlin, Germany, September 2003.

[16] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proc. ACM MOBICOM*, Boston, MA, USA, 2000.

[17] W. Zhang and G. Cao. Optimizing Tree Reconfiguration for Mobile Target Tracking in Sensor Networks. In *Proc. IEEE INFOCOM*, Hong Kong, March 2004.

[18] J. Wieselthier, G.D. Nguyen, and A. Ephremides. On the Construction of Energy-Efficient Broadcast and Multicast Trees in Wireless Networks. In *Proc. IEEE INFOCOM*, Tel Aviv, Israel, April 2000.

[19] P. Sinha, R. Sivakumar, and V. Bharghavan. CEDAR: Core extraction distributed ad hoc routing. In *Proc. IEEE INFOCOM*, New York City, NY, USA, June 1999.

[20] B. Hajek and G. Sasaki. Link Scheduling in Polynomial Time. *Proc. IEEE Transactions on Information Theory*, 34:910–917, September 1988.

[21] C. Shannon. A theorem on colouring lines of a network. *J. Math. Phys.*, 39:148–151, 1948.

[22] T. Salonidis and L. Tassiulas. Performance issues of Bluetooth scatternets and other asynchronous TDMA ad hoc networks. In *Proc. International Workshop on Mobile Multimedia Communications (MoMuC)*, Munich, Germany, October 2003.

[23] I.A. Cimet C. Cheng and P.R. Kumar. A protocol to maintain a minimum spanning tree in a dynamic topology. In *Proc. ACM SIGCOMM*, Stanford, CA, August 1988.

[24] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Proc. ACM SIGCOMM*, Austin, TX, USA, September 1989.

[25] G. Apostolopoulos, R. Guerin, S. Kamat, A. Orda, T. Przygienda, and D. Williams. QoS routing mechanisms and OSPF extensions, 1998.

[26] D. Bertsekas and R. Gallager. *Data networks*.

[27] A. Charny. An algorithm for rate allocation in a packet switching network with feedback, M.Sc. Thesis, May 1994.

[28] L. Kalampoukas. *Congestion Management in High Speed Networks*. PhD thesis, University of California Santa Cruz, September 1997.

[29] S. Kalyanaraman, R. Jain, S. Fahmy, R. Goyal, and B. Vandalore. The ERICA switch algorithm for ABR traffic management in ATM networks. *TON*, 8(1):87–98, 2000.

[30] Bluetooth Special Interest Group. Specification of the Bluetooth system, version 1.2. In *www.bluetooth.com*.

[31] P. Johansson, R. Kapoor, M. Kazantzidis, and M. Gerla. Rendezvous Scheduling in Bluetooth Scatternets. In *Proc. International Conference on Computer Communications (ICC)*, New York, NY, April 2002.

[32] S. Baatz, M. Frank, C. Kuhl, P. Martini, and C. Scholz. Bluetooth Scatternets: An Enhanced Adaptive Scheduling Scheme. In *Proc. IEEE INFOCOM*, New York, NY, June 2002.

[33] IBMResearch. BlueHoc: Bluetooth Performance Evaluation Tool. In *http://oss.software.ibm.com/bluehoc/*.

[34] NS notes and documentation. In *http://www.isi.edu/vint/nsnam*.

[35] S. Lu H. Luo and V. Bharghavan. A new model for packet scheduling in multihop wireless neworks. In *Proc. ACM MOBICOM*, Boston, MA, USA, August 2000.

[36] T. Nandagopal, T. Kim, X. Gao, and V. Bharghavan. Achieving MAC layer fairness in Wireless Packet Networks. In *Proc. ACM MOBICOM*, Boston, MA, USA, October 2000.

[37] X. Huang and B. Bensaou. On Max-min Fairness and Scheduling in Wireless Ad-Hoc Networks: Analytical Framework and Implementation. In *Proc. ACM MOBIHOC*, Long Beach, CA, USA, October 2001.

[38] L. Tassiulas and S. Sarkar. Maxmin Fair Scheduling in Wireless Networks. In *Proc. IEEE INFOCOM*, New York, NY, USA, June 2002.

[39] S. Sarkar and L. Tassiulas. End-to-end bandwidth guarantees through fair local spectrum share in wireless ad-hoc networks. In *Control and Decision Conference (CDC)*, Maui, HI, USA, December 2003.

[40] N. Johansson, F. Alriksson, and U. Jonsson. JUMP mode - a dynamic window-based scheduling framework for Bluetooth scatternets. In *Proc. ACM MOBIHOC*, Long Beach CA, October 2001.

[41] A. Racz, G. Miklos, F. Kubinszky, and A. Valko. A Pseudo Random Coordinated Scheduling algorithm for Bluetooth Scatternets. In *Proc. ACM MOBIHOC*, Long Beach CA, October 2001.

X. Appendix

A. Algorithm Pseudocodes

Procedure `SampleReschedule`

```
begin
    PrioritizeLinks();                                          SR-1
    l_c = GetMaxUnstableChildLink();                            SR-2
    if (l_c ≠ −1) then
        if (busybit_==0 AND BusyBit(v)==0) then                 SR-3
            busybit=1;
            send SC_INFO packet to v;                           SR-4
        end
    end
end
```

Procedure `PrioritizeLinks`

```
Assign priorities to children links in order of appearance
after slot e_{l_p}^{(u)}
local   : CH = set of children links, LINKSET, p, slot
begin
    p = |CH|; LINKSET = CH; slot = e_{l_p}^{(u)} ⊕ 1;
    repeat
        l_c = local_schedule[slot];
        if (l_c ∈ LINKSET) then
            p_{l_c} = p /*set the priority of l_c to p*/;
            p=p-1;
            LINKSET = LINKSET - {l_c} ;
        end
        slot = slot ⊕ 1;
    until LINKSET is empty;
end
```

Function `GetMaxUnstableChildLink`

```
Return the maximum priority unstable child link or -1 otherwise
local   : CH = set of my children links, J_k equals 1 if I
          am slave on child link k and zero otherwise
begin
    for p=|CH| down to 1 do
        l_c = the child link of priority p;
        if (not satisfied(l_c)) then
            return l_c;
        else
            lpsum = Σ_{k∈CH:p_k<p_l} (τ_k + J_k);
            if (lpsum > |[e_{l_c} ⊕ 1, s_{l_p} ⊖ 1]|)) then
                return l_c;
            end
        end
    end
    return -1;
end
```

Fig. 10. Procedure SampleReschedule()

B. Proof of Theorem 1

Proof: Since T_{system} is set to $LB(\boldsymbol{\tau})$ slots, it suffices to show that no node runs out of slots in its local schedule during the algorithm execution. We use induction on the link levels.

Level 1: Starting at slot 0 in its local schedule $\boldsymbol{S}_r$, the root r schedules $\sum_{l\in L(r)} \tau_l + J_l^{(r)}$ slots, which, by definition, is less than or equal to $LB(\boldsymbol{\tau})$. In addition, each child node c on child link l allocates $\tau_l + J_l^{(c)}$ time-overlapping slots in its local schedule, which does not exceed $LB(\boldsymbol{\tau})$.

Level k-1: Assume that no node has ran out of slots after all level $k-1$ links have been scheduled. Due to the breadth-first recursion, each of the level $k-1$ nodes that are children of the same (level $k-2$) parent has been assigned a single window, the windows of such nodes being mutually exclusive.

Therefore, without loss of generality, we can consider a link l_p of level $k-1$ and its child node u in isolation. During the previous iteration u has been allocated $\tau_{l_p} + J_{l_p}^{(u)}$ consecutive slots in $\boldsymbol{S}_u$.

Let $CH(u)$ be the set of children links of node u. Node u will need $\sum_{l\in CH(u)}(\tau_l + J_l^{(u)})$ slots in $\boldsymbol{S}_u$

Algorithm 5: STABLETREE

Data : Asynchronous events at node u 1 Parent node(link): $p(l_p)$ (or none if root), Child node (link): c (l_c)

Result : Corresponding actions

E1 **Events: e1: Any adjacent link becomes non-satisfied**; **OR e2: Scheduling of a link just completed**;

begin

  if *(event e2 occured)* **then**

E1-1 busybit_=0;

  end

  if *(busybit_==0)* **then**

    if *(I am root)* **then**

      SampleReschedule();

    else

      if *(wait_parent_==0))* **then**

E1-2 wait_parent_=1;

E1-3 send STABLE_REQ packet to parent p;

      end

    end

  end

end

E2 **Event: STABLE_REQ packet received from child** c;

begin

  if *(I am root OR satisfied(l_p))* **then**

    if *(stable(l_c))* **then**

E2-1 send STABLE_ACK packet to child c;

    else

      if *(busybit_==0 AND wait_parent_==0)* **then**

E2-2 SampleReschedule();

      end

    end

  end

end

E3 **Event: STABLE_ACK packet received from parent** p;

begin

E3-1 wait_parent_=0;

E3-2 SampleReschedule();

end

E4 **Event: SC_INFO packet received from node** v;

begin

  if *(I am child of v)* **then**

E4-1 busybit_=1;

E4-2 wait_parent_=0;

E4-3 send SC_INFO packet to v;

  else

E4-4 AssignSlots(l_v) /*Determine new slot positions for l_v*/ ;

E4-5 Initiate distributed coordination mechanism by updating v and affected neighbors with SC_UPD packets.

  end

end

Fig. 11. The asynchronous distributed link scheduling algorithm

to schedule its children links in mutually exclusive windows. Thus node u will have assigned a total of $\sum_{l\in CH(u)}(\tau_l + J_l^{(u)}) + \tau_{l_p} + J_{l_p}^{(u)} = \sum_{l\in L(u)}(\tau_l + J_l^{(u)})$ slots at the end of iteration k, which, by definition does not exceed $LB(\boldsymbol{\tau})$. Also, for each link $l = (u, c) \in CH(u)$, the child node c will allocate $\tau_l + J_l^{(c)}$ in $\boldsymbol{S}_c$, which does not exceed $LB(\boldsymbol{\tau})$. Therefore no node runs out of slots at the end of iteration k. The induction step is complete.

C. Proof of Convergence Theorem

We assume that changes on a link demand are detected by both node endpoints (not necessarily at the same time instant) and that control messages are not lost due to channel errors. Mobility is a special case of link demand changes with a link failure being transition to zero demand and a link establishment being a transition from zero to a positive demand satisfying the local feasibility conditions.

Given an arbitrary set of changes that have stabilized, let K_{min} be the link level such that all links of level K_{min} or less have not been affected by the changes. We will prove convergence by induction on the link levels $k > K_{min}$ that have been affected by the change. We distinguish two cases for K_{min}:

Case A $K_{min} = 0$**:** This is the case where at least one of the child links of the root has been affected by the changes.

Step 1: Level 1: Link level 1 includes the root and its children. Upon detection of any unsatisfied link, the root will run SampleReschedule() only if it is not busy or after it has finished scheduling its current link. Let l_c the highest-priority unstable child link. We distinguish two cases for the child node endpoint c of l_c:

Case 1: Node c not busy: the root initiates scheduling of l_c by sending an SC_INFO packet to c (line SR-4, Fig. 10).

Case 2: Node c currently busy: the root exits SampleReschedule(). When node c finishes scheduling, it will send a STABLE_REQ packet to the root (line E1-3, Fig. 11). It also becomes unavailable for rescheduling its own children until it receives a response from the root (line E1-2, Fig. 11). Upon reception of the STABLE_REQ packet, the root executes SampleReschedule(). Since the are no more changes, the highest priority child will be again node c and it is guaranteed not to be busy this time (due to line E1-3 node c will not enter SampleReschedule() upon reception of STABLE_REQ packets from its children.). Then the root initiates scheduling on l_c (line SR-4, Fig. 10). In a similar fashion, the root will eventually schedule all level-1 unstable links in decreasing order of their priority.

Step k: Level k**:** Assume that all links up-to and including level k have been scheduled and stabilized. We will show that all level $k+1$ unstable links will be scheduled and stabilized in a finite number of iterations .

Since every level-k node u has been independently assigned a stable parent link window $W_{l_p} = [start_{l_p}, end_{l_p}]$, it suffices to consider one such node in isolation. Each time node u needs to execute SampleReschedule(), it asks permission from its parent node p by sending a STABLE_REQ packet. Since l_p is stable, the parent p will always reply with a STABLE_ACK packet (line E2-1, Fig. 11).

As soon as u receives permission to run SampleReschedule(), we have the same case of the root node and the level-1 links. Therefore, all unstable children links of node u will eventually be re-scheduled and stabilized. Since this will happen for all level-k nodes and their level $k+1$ children links, the induction step is complete.

Case B $K_{min} > 0$**:** This case can be proven using as initial inductive step $k = K_{min}$. The initial step holds since it is similar to the Level-k inductive step of the case $K_{min} = 0$. For level $k > K_{min}$ to $k+1$, a similar argument is applicable. Q.E.D.