Distributed Dynamic Scheduling For End-to-end Rate Guarantees In Wireless Ad Hoc Networks

Theodoros Salonidis
Department of Electrical and
Computer Engineering
Rice University
6100 Main Street
Houston, TX, USA
thsalon@ece.rice.edu

Leandros Tassiulas
Department of Computer and
Communication Engineering
University of Thessaly
Argonafton Filellinon 38221
Volos, Greece
leandros@inf.uth.gr

ABSTRACT

We present a 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 Time Division Multiple Access (TDMA) protocol nodes adapt to the demand changes on their adjacent links by local, conflict-free slot reassignments. As soon as the demand changes stabilize, the nodes must incrementally converge to a TDMA schedule that realizes the global link (and session) demand allocation.

We first derive sufficient local feasibility conditions for certain topology classes and show that trees can be maximally utilized. We then introduce a converging distributed link scheduling algorithm that exploits the logical tree structure that arises in several ad hoc network applications.

Decoupling bandwidth allocation to multi-hop sessions from link scheduling allows support of various end-to-end Quality of Service (QoS) objectives. We focus on the max-min fairness (MMF) objective and design an end-to-end asynchronous distributed algorithm for the computation of the session MMF rates. Once the end-to-end algorithm converges, the link scheduling algorithm converges to a TDMA schedule that realizes these rates.

We demonstrate the applicability of this framework through an implementation over an existing wireless technology. This implementation is free of 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.

Categories and Subject Descriptors

C.2 [**Computer Communication Networks**]: Local and Wide Area Networks—*Access Schemes*; C.2 [**Computer Communication Networks**]: Network Architecture and Design—*Wireless Communication*; F.2 [**Analysis of algorithms and problem complexity**]: Miscellaneous

General Terms

Algorithms, Performance, Design

Keywords

Ad Hoc Networks, Distributed, Scheduling, Rate Control

1. 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, provision of bandwidth guarantees 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. Fulfilling 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 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 [10, 17, 29, 49]. Chen and Nahrstedt [10] and Gerla and Tsai [17] focus on mobility issues but assume that conflict-free slots have been preallocated 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 [49] and Lin [29], 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 a 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 re-computation 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 [2, 21], determining feasibility of a set of link rates in an ad hoc network of arbitrary topology is a NP-complete problem. Several centralized [28, 34, 41], semi-centralized [33], or distributed [3, 31] heuristics for TDMA link scheduling have been proposed, but they either do not possess well-defined performance guarantees or cannot be applied 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, an 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.

We first derive sufficient local feasibility conditions for certain topology classes and show that trees can be maximally utilized. Trees manifest in several ad hoc network applications as logical overlays over the physical topology defined by the node wireless proximities. In mesh networks, users share high speed internet access from a wired entry point through a low cost multi-hop wireless infrastructure [12, 15, 30]. In sensor networks data is reported to a single source over a tree structure [24, 48]. Trees are also used in Bluetooth scatternets [18, 47, 42] and mobile ad hoc networks (MANETs) for power-aware multicasting [44] or routing backbone structures [35]. Motivated by such applications, we present a distributed dynamic link scheduling algorithm that realizes all feasible link demand allocations in tree topology structures. According to this algorithm, nodes can start from any initial TDMA schedule and incrementally converge in a finite number of steps to a new schedule realizing any desired link demand allocation.

The distributed link scheduling algorithm focuses on realizing the link demands and is agnostic of the bandwidth allocation mechanisms running at higher layers. This allows realization of end-to-end service models where the session rates are not required to be known in advance. For example, a session may not have a specific rate requirement but may request the maximum possible bandwidth from the network. To this end, we consider end-to-end services where bandwidth must be shared to the sessions in a fair manner. While various fairness objectives have been considered for single-hop sessions (links) [19, 22, 32, 38, 43], fairness for multi-hop sessions has not yet been adequately addressed. Max-min fairness and utility-based fairness have been considered in [39] and [9, 14, 45], respectively. The authors propose distributed algorithms to compute the fair session rates subject to the wireless access constraints; however, so far no distributed link scheduling mechanism exists that can realize these rates. In [39], the rates are enforced using centralized TDMA link scheduling; in [14] a distributed random access mechanism is used but there are no analytical guarantees for the realization of the computed rates. We introduce a rate-based, asynchronous distributed algorithm for sharing bandwidth to sessions in a max-min fair (MMF) manner. Combined with the derived local feasibility conditions, it can compute MMF rates for any topology form. Coupled with the distributed link scheduling algorithm, it enforces the computed rates for wireless ad hoc networks employing tree structures.

Another important issue is that most TDMA implementations 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. We introduce a TDMA architecture where 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.

2. ASYNCHRONOUS TDMA ARCHITECTURE

The wireless ad hoc network is represented as a graph $G(N, E)$. Each edge (u, v) signifies that nodes u and v are in range and have established a wireless link. Each node has a single radio transceiver and can communicate (transmit or receive) to at most one adjacent link at a time. We assume that interference among links exists only due to this constraint and that hidden terminals are avoided-there is no interference among any links that belong to distinct node pairs. This can be achieved by a set of orthogonal channels and a distributed signaling mechanism that assigns different channels to such interfering links. For example, if a distinct channel is used for each node, hidden terminals can be avoided by assigning to each link the channel of one of its node endpoints. Instead of N orthogonal channels, it is sufficient to use $D(D-1)+1$ channels, where D is the maximum number of intended recipients per node [16]. Channels can be implemented in the frequency domain or code domain (Direct Sequence codes or frequency hopping sequences). Hidden terminals can also be avoided using directional antennae pointing toward the intended recipients–this can be viewed as channelization in the space domain.

In such a multi-channel system, the set of links that can transmit simultaneously without conflict in the network is any matching in $G(N, E)$. This interference model has also been used in [20, 28, 38, 43, 46], among other works. As in every study of coordination problems at the MAC layer and above, we assume no losses due to channel errors.

Although the introduction of multiple channels mitigates hidden terminals, it introduces certain restrictions: Nodes need to coordinate their presence on common links (and tune to the corresponding channels) in mutual time intervals. To resolve this problem, we utilize an asynchronous TDMA architecture. Based on its own hardware clock, each node u divides time in fixed-size slots and coordinates transmissions on its adjacent links using a local periodic link schedule $\boldsymbol{S}_u$ of period T_{system} slots. Slot synchronization for each link is provided by the clock of one of its node endpoints, termed as the link master. Each slot on a link supports full-duplex communication initiated by the master: During the first part of the slot the master polls the slave; during the second part the slave responds.

The local schedule determines communication action for the duration of a full-duplex slot: the node can either be active on a single link (start polling if master or start 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 necessarily 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 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.

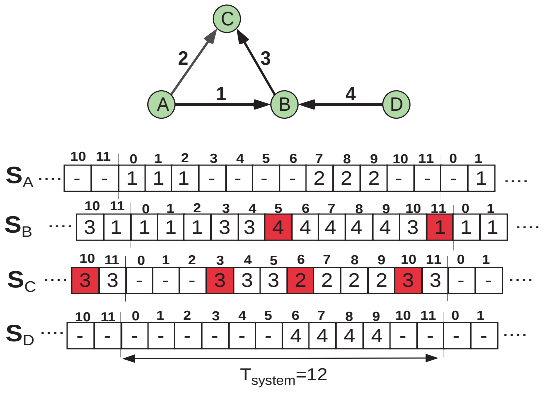

Figure 1: (a) Network topology: Arrows denote master-slave relationships. Links 2 and 4 can transmit simultaneously without conflict on different channels. (b) Network asynchronous TDMA schedule of period $T_{system} = 12$ slots: each slot supports full-duplex communication. Link slaves switch channel and time reference during the red slots. The realized slot allocation is $\boldsymbol{\tau} = (\tau_1, \tau_2, \tau_3, \tau_4) = (3, 3, 3, 4)$ slots.

The local schedules collectively form the network asynchronous TDMA schedule. A *link slot allocation* $\boldsymbol{\tau} = (\tau_1, .., \tau_l, ..., \tau_{|E|})$ 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 (see Fig. 1 for an example).

3. THE DYNAMIC LINK SCHEDULING PROBLEM

During network operation each link is characterized by its slot allocation (provided by the current network TDMA schedule) and by its slot demand. A higher-layer process may change the link demands at asynchronous time instants. In section 7, we will instantiate this process to an end-to-end bandwidth allocation mechanism. Mobility can also be captured by viewing a link failure as transition to zero demand and a link establishment as a transition from zero to a positive demand.

The higher-layer process alternates between two states: an active state where the link demands change and a quiescent state where no changes occur. This implies that network topology and traffic dynamics must remain stable for a sufficient time period to allow convergence. However, the nodes are not aware which of the two states the network is currently in. They can only detect the demand changes on their adjacent links and react by local slot reassignments. The end of each active state corresponds to a link demand allocation to be realized by a network TDMA schedule. The challenge: starting from the current TDMA schedule nodes must incrementally converge to such a schedule using only local information.

Our approach for the dynamic link scheduling problem consists of three components:

1) Local feasibility conditions: In order for convergence to occur, the higher layer process must provide the link layer with feasible link demands. 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. Feasibility determination of any demand allocation is a NP-complete problem even if global slot synchronization and topology knowledge are available [21]. However, in practice, the link demand changes will be due to end-to-end bandwidth allocation or hand-off mechanisms running locally at the nodes. Hence, we are interested in identifying certain instances where feasibility can be characterized by a set of local conditions.

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.

3) Distributed link scheduling algorithm: Determines how nodes should re-assign slots to reach a schedule realizing the desired demand allocation.

4. LOCAL FEASIBILITY CONDITIONS

4.1 Synchronized TDMA

Let us first assume that global slot synchronization is supported. Local conditions 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). 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 [13].

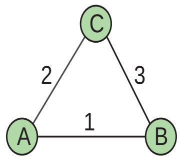

Figure 2: Without loss of generality, assume that all nodes are slot-synchronized and T_{system} is even. No schedule exists for allocating $T_{system}/2$ conflict-free slots per link, even if the local conditions $\tau_1 + \tau_2 \le T_{system}$, $\tau_1 + \tau_3 \le T_{system}$ and $\tau_2 + \tau_3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

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

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

Alternatively, if no mechanism for topology control exists, feasibility is ensured by requiring the sum of link demands on every node not to exceed $\lfloor 2/3 \cdot T_{system} \rfloor$ slots [40]. Local conditions of this form are sufficient: they guarantee feasibility but only capture a fraction of the entire set of feasible allocations. The network must be underutilized[1] in this case.

[1]The terms "underutilization" and "feasibility" are only with respect to the realization of a desired set of link rates in the network (QoS traffic). The remaining slots in the nodes' local schedules can always be used for control or best-effort traffic.

4.2 Asynchronous TDMA

In an asynchronous TDMA system such as the one considered in this paper, the region of feasible rates is further restricted. In [36] it has been shown that 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 underutilization.

Let $L_{min}(\boldsymbol{\tau})$ be the minimum period realizing link demand allocation $\boldsymbol{\tau}$. A lower bound on $L_{min}(\boldsymbol{\tau})$ 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,

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

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

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, the entire set of feasible allocations 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: Let $\boldsymbol{\tau}^*$ be an allocation satisfying eq. (3) but is not feasible. Since $\boldsymbol{\tau}^*$ is not feasible, the minimum period realizing it must be strictly greater than T_{system}: $L_{min}(\boldsymbol{\tau}^*) > T_{system}$. From 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 $L_{min}(\boldsymbol{\tau}) = LB(\boldsymbol{\tau})$ for every $\boldsymbol{\tau}$ in $G(N, E)$ we conclude $L_{min}(\boldsymbol{\tau}^*) \leq T_{system}$, i.e. $\boldsymbol{\tau}^*$ is feasible. This contradicts our initial hypothesis.

Proposition 1 states that topologies for which $L_{min}(\boldsymbol{\tau}) = LB(\boldsymbol{\tau})$ for all $\boldsymbol{\tau}$, can be fully utilized by distributed algorithms. Next, we show that trees are a topology class that satisfies this property:

Theorem 1: If $G(N, E)$ is a tree, any link demand slot allocation $\boldsymbol{\tau}$ can be realized by a periodic schedule of $LB(\boldsymbol{\tau})$ slots.

Sketch of Proof: Due to space limitations we provide sketch of proof for some results appearing in this paper. The detailed proofs appear in technical report [37]. The proof of Theorem 1 is constructive. Let each node be equipped with a local schedule $\boldsymbol{S}_u$ of period $T_{system} = LB(\boldsymbol{\tau})$ slots, indexed from 0 to T_{system}-1. Next, the following algorithm is executed: starting from an arbitrary slot s in $\boldsymbol{S}_r$ and proceeding towards the next (modulo T_{system}) slot, the root r schedules its child links in consecutive windows according to their demands. In addition, for each child link, time-overlapping windows are allocated to the local schedule of the corresponding child node endpoint. By the definition of T_{system} (eq. (1)), neither the root nor its children will run out of slots in their local schedules by these window assignments. The algorithm proceeds recursively: each child node starts from the end of its parent link window (assigned during the previous iteration) and schedules its own children links in a breadth-first manner. By induction on the tree levels it can be shown that no node runs out of slots during the algorithm execution.

To summarize the above discussion, we have arrived at the following local feasibility conditions for multi-channel, asynchronous TDMA ad hoc networks:

$$\sum_{l \in L(u)} \tau_l \leq T_u^R, \ \forall u \in N \tag{4}$$

where

$$T_u^R = \begin{cases} \lfloor 1/3 \cdot T_{system} \rfloor & \text{if } G \textit{ arbitrary} \\ \lfloor 1/2 \cdot T_{system} \rfloor & \text{if } G \textit{ bipartite} \\ T_{system} - \sum_{l \in L(u)} J_l^{(u)} & \text{if} G \textit{ tree} \end{cases} \tag{5}$$

The utilization factor T_u^R depends on the topology control mechanism used by the network (if any). In practice, each incoming node u queries its neighbors about the topology control algorithm used in the network and uses eq. (5) to set its T_u^R accordingly. This ensures that a distributed link scheduling algorithm will always seek a feasible allocation.

According to Proposition 1 and Theorem 1, all feasible allocations for tree structures can be captured only by the local conditions. Hence, trees can be maximally utilized by distributed algorithms. It is also interesting to note that, according to eq. (5), more restricted topologies allow higher per/node utilization, with trees allowing the maximum possible. This does not necessarily indicate that trees are the optimal topology structures. For certain traffic patterns, additional links could be useful to exploit shorter paths toward destinations. However, if a non-tree structure is to be used, the utilization of *every* node in the network must be reduced to ensure feasibility under distributed operation.

We now proceed to introduce a topology-independent distributed coordination mechanism and a distributed dynamic link scheduling algorithm that reaches any desired feasible allocation in a tree topology.

5. 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 $LOCK\_VEC$ (a local variable) 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 their $LOCK\_VEC$ variables) 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.

6. STABLE_TREE

We now introduce a distributed dynamic link scheduling algorithm, called STABLE_TREE, that operates within the state space defined by tree topologies and the corresponding conditions in eq. (4). 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 dynamic tree formation and maintenance protocol such as [6, 18, 42] runs in the network.

6.1 Notations and definitions

Before presenting the algorithm we introduce some notations and definitions. Slots in each local schedule $\boldsymbol{S}_u$ of node u are indexed from 0 to $T_{system}-1$. We denote by $W_l^{(u)}=[s_l^{(u)},e_l^{(u)}]$ a window of consecutively assigned slots to link l in $\boldsymbol{S}_u$, starting at slot $s_l^{(u)}$ and ending at slot $e_l^{(u)}$. Due to the periodicity of $\boldsymbol{S}_u$:

$$W_l^{(u)}=[s_l^{(u)},e_l^{(u)}]=\begin{cases} s_l^{(u)},...,e_l^{(u)} & \text{if } s_l^{(u)}\le e_l^{(u)} \\ s_l^{(u)},..,0,...,e_l^{(u)} & \text{otherwise}\end{cases} \qquad (6)$$

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

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

6.2 Operation

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) Let $W_{l_p}^{(u)}=[s_{l_p}^{(u)},e_{l_p}^{(u)}]$ be the window in $\boldsymbol{S}_u$ satisfying the parent link l_p of node u. 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 further action takes place. Otherwise, the highest priority unstable child 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, u computes the closest slot position to $s_{l_p}^{(u)}$ for which the stability conditions for l_c will hold:

$$s_{max}=s_{l_p}^{(u)}\ominus\sum_{k\in CH(u):p_k<p_{l_c}}(\tau_k+J_k^{(u)}) \qquad (7)$$

  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. Let $W_{l_m}^{(u)}=[s_{l_m}^{(u)},e_{l_m}^{(u)}]$ be the window satisfying the demand of 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

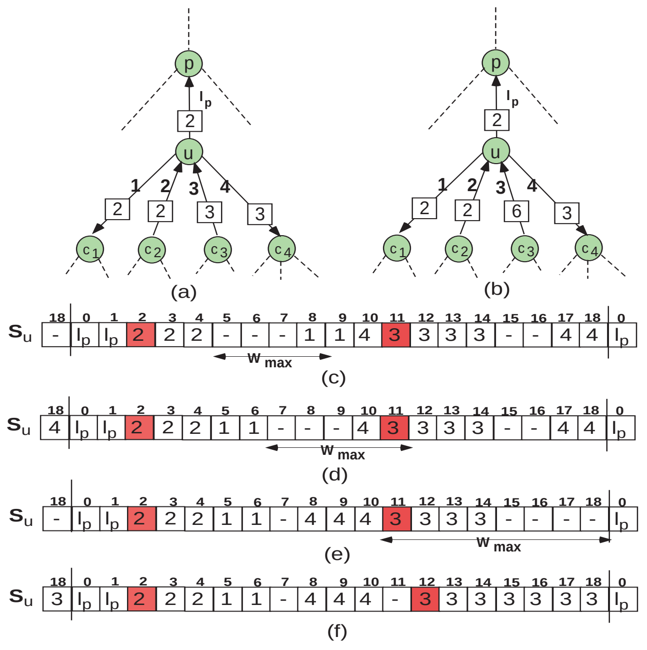
p
l_p
u
1
2
3
4
c_1
c_2
c_3
c_4
(a)
(b)
S_u
18 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 0
- l_p l_p 2 2 2 - - - 1 1 4 3 3 3 3 - - 4 4 l_p
W_{max}
(c)
4 l_p l_p 2 2 2 1 1 - - - 4 3 3 3 3 - - 4 4 l_p
(d)
- l_p l_p 2 2 2 1 1 - 4 4 4 3 3 3 3 - - - - l_p
(e)
3 l_p l_p 2 2 2 1 1 - 4 4 4 - 3 3 3 3 3 3 3 l_p
(f)

Figure 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.

$W_{l_c}^{(u)}$ 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 issues SC_UPD packets to its affected neighbors. The coordination mechanism (Section 5) 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.

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 sends a STABLE_REQ packet to its parent. The parent will respond in one of two possible ways: either 1) it replies with a STABLE_ACK packet as permission for node u to continue sampling and rescheduling its children or 2) it initiates a rate adjustment on this link via a SC_INFO packet.

In Fig. 3, node u must perform a STABLE_REQ/STABLE_ACK handshake with its parent p for every child link it reschedules. If, meanwhile, link l_p becomes unstable, the parent will respond to STABLE_REQ 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 STABLE_TREE is described in Fig. 7 and Fig. 8 in the Appendix.

6.3 Properties

In this section we establish the convergence properties of STABLE_TREE.

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 corresponding capacity condition of eq. (4). STABLE_TREE will converge to a new TDMA schedule realizing $\boldsymbol{\tau}$ in a finite number of link rate adjustments.

Sketch of Proof: In general, nodes re-assign slots using SampleReschedule() when their adjacent links are detected "unsatisfied". We show that, as soon as the changes in link demands stabilize, convergence is guaranteed to occur progressively from the root downward. The detailed proof appears in [37].

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 downward, 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 downward, multiple links will be scheduled in parallel. The convergence behavior of STABLE_TREE in practice is investigated in Section 8, together with end-to-end bandwidth allocation mechanisms (addressed next).

7. END-TO-END FRAMEWORK

We now introduce a framework for integrating link scheduling with end-to-end bandwidth allocation. The asynchronous TDMA ad hoc network 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 is unidirectional.

Let the maximum radio transmission rate be R bps. To support a rate of ρ_i ($\leq R$) bps for session i 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 over the session 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:

$$\sum_{i \in F(u)} \delta_i^{(u)} \cdot \tau_i \leq T_u^R, \; \forall u \in N \tag{8}$$

where T_u^R is given by eq. (5) and

$$\delta_i^{(u)} = \begin{cases} 1 & \text{if } u\, is\, source\, or\, destination\, of\, session\, i \\ 2 & \text{otherwise} \end{cases} \tag{9}$$

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 integrated framework provides end-to-end bandwidth guarantees using three components:

Session rate allocation: Sessions are allocated feasible rates according to eq. (8).

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

$$\tau_l = \sum_{i \in F(l)} \tau_i, \; \forall l \in E \tag{10}$$

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.

Session rate enforcement: Once link scheduling converges, every link has been allocated enough bandwidth (conflict-free slots) to support the session demands. The slots of each link can be shared to its sessions using Weighted Round Robin (WRR), Weighted Fair Queuing (WFQ) [11] or other single-server queuing disciplines. Another possibility is to combine FIFO queuing at intermediate links with explicit control of the transmission rates at the sources.

Decoupling end-to-end bandwidth allocation from link scheduling, allows realization of various end-to-end service models. For example, in certain applications sessions arrive with specific rate requirements that need to be satisfied by the network. In this case it is possible to perform admission control using eq. (8). We consider a service model where sessions request maximum rate and the network bandwidth must be shared to the sessions in a fair manner. We focus on the max-min fairness (MMF) objective and introduce a distributed algorithm to compute the session MMF rates. This algorithm integrates the access constraints (given by eq. (8)) with algorithms for wireline networks. (A similar integration can be used to realize other fairness objectives such as utility-based max-min fairness [7] or proportional fairness [27]).

7.1 End-to-end max-min fairness

For convenience, we use normalized rates instead of slots to represent bandwidth allocations. 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 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 $\boldsymbol{r} = (r_1, .., r_{|F|})$ is MMF if it satisfies the following property with respect to another feasible rate allocation $\boldsymbol{r}' = (r_1^{'}, ..., r_{|F|}^{'})$: if there exists a session i such that $r_i < r_i'$, then there exists another session j such that $r_j \leq r_i$ and $r_j^{'} < r_j$.

Nodes use local feasibility conditions derived by dividing both sides of eq. (8) 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 $C_u^R = T_u^R/T_{system}$. Node u is defined to be a *bottleneck* for session i if it is fully utilized (with respect to C_u^R) and session i has maximum rate over all sessions in $F(u)$. 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, offline centralized algorithm similar to the algorithm of Bertsekas and Gallager in wireline networks [4]. 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

```
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
```

Figure 4: Update algorithm for session rate, link demands and MMF rate estimate ϕ_u.

rate from the node available bandwidth). Any node and link whose sessions have been removed 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 MMF rates.

We have implemented a distributed version of the centralized algorithm, similar in spirit with algorithms proposed for wireline ATM networks [8, 25, 26]. This is a rate-based approach 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 over the session path.

Each node u maintains 1) a subset $FC(u)$ of its sessions $F(u)$, currently seen as "constrained" by other nodes and 2) an estimate ϕ_u for the MMF rate it provides to its currently unconstrained sessions. When each session control packet is about to be forwarded, $FC(u)$, ϕ_u together with the link demands and the "rate" and "constrained" fields of the session control packet are updated by procedure MMF_UpdateState() (see Fig. 4).

Upon return of a control packet the source adjusts its transmission rate according to the value r_i in the rate field. New session i control packets are sent out with rate field set to the new rate and the constrained bit field set to zero.

Using arguments similar to [8] we can prove that the distributed algorithm converges in a finite number of iterations to the session MMF rate values. This holds for any topology form given the appropriate fractional capacities C_u^R that ensure feasibility in each case. The distributed MMF algorithm differs from its wireline counterparts [8, 25, 26] in three aspects. First, the available rate of each node must be divided over the *session parts* instead of the sessions sharing it. Second, every node in a session path–including source and destination–must update their session MMF rate estimate. Third, the link demands must be updated and passed to the link scheduling algorithm.

End-to-end rate computation and link scheduling occur in parallel. 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, allowing the link scheduling algorithm to converge.

8. BLUETOOTH IMPLEMENTATION

Bluetooth [5] is a multi-channel asynchronous TDMA system with a constraint that a node can be master to at most seven adjacent links. Channels are implemented as frequency hopping sequences and termed as *piconets*. Transmissions on each link occur in the piconet defined by the unique Bluetooth address of the master node endpoint. For example, if the asynchronous TDMA ad hoc network in Figure 1 were to be realized by Bluetooth, links {1,2}, {3}, and {4} would be using the time slot references and distinct piconets defined by masters A, B and D, respectively. A Bluetooth ad hoc network is termed as *scatternet*.

Figure 5 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 supports all lower-level functions related to link establishment (discovery, master-slave assignment and computation of the piconet frequency hopping sequence and time slot reference) and link maintenance (maintaining slot synchronization and full-duplex communication between master and slave, switching to low power modes, etc).

The Bluetooth Baseband Specification does not define how nodes in a scatternet should divide their time among their adjacent links and piconets. We have therefore implemented the periodic local schedule structure (Section 2), coordination mechanism (Section 5) and link scheduling algorithm (Section 6) at the application layer. More specifically, we use the Bluetooth sniff mode to instruct the Baseband to transmit according to the local schedule maintained at the application layer. Sniff mode is a low power mode where a slave can listen to a master for a window of $N_{sniff\_attempt}$ slots within a period of T_{sniff} slots. The Bluetooth Host Controller Interface (HCI) exports a function from Baseband to higher layers 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 u is modified at the application layer, we instruct the hardware to start sniff mode on link l on that offset by setting $N_{sniff\_attempt} = \tau_l + J_l^{(u)}$.

The Bluetooth L2CAP layer supports both unidirectional and bidirectional logical channels between two node endpoints. Each session consists of multiple L2CAP bidirectional channels, one for each link over the session path. Bluetooth supports half-duplex slots of duration $D_{slot} = 0.625ms$, each carrying up to $B = 216$ payload bits. Each Bluetooth full-duplex slot may consist of $(1, 1)$, $(1, 3)$ or $(1, 5)$ half-duplex slots. Here, we use $(1, 1)$ configuration, yielding a maximum transmission rate of $R = B/2D_{slot} = 172.8$ Kbps per direction.

When a source receives an end-to-end control packet with rate r_i, it adjusts its transmission rate to $r_i \cdot R$ bps. To enhance perfor-

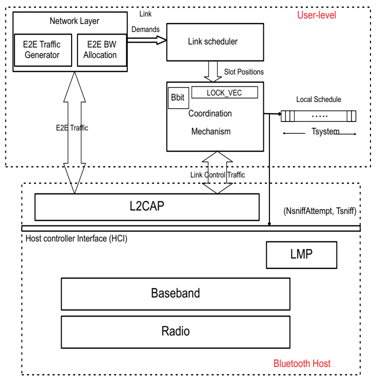

Figure 5: Implementation of the end-to-end bandwidth allocation framework over the Bluetooth protocol stack

mance, in addition to source rate control, each node uses a packet scheduler on each link to decide the type of packet to 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, WRR is used to schedule packets of outgoing sessions for this link.

9. EXPERIMENTS

We use BlueHoc [23], the IBM Bluetooth extensions to the *ns* simulator [1]. 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 *ns* modules. We have performed experiments on various topology and session configurations. Due to space limitations, here we present and analyze a representative scenario in detail. Our aim is to investigate the effect of various factors on the joint performance of end-to-end and link scheduling algorithms during transience and at steady state.

We consider the 10-node configuration of Fig. 6. The node positions are kept fixed. Topology changes would appear to the link scheduling algorithm as changes from zero to a positive number (link establishment) or transitions to zero (link failure). While computing the MMF rates, the end-to-end algorithm creates rich link demand dynamics to be tracked by the link scheduling algorithm.

We run 10 experiments, each corresponding to the selection of a different node as root. In all experiments a period of $T_{system} = 50$ slots is used. Nodes start with an arbitrary conflict-free TDMA schedule; Initially, all sources transmit at maximum rate (172.8 Kbps) and adjust it based on the values of the returning session control packets. Time is measured with respect to the time slot reference of the root node. Each simulation lasts 20000 slots (or $20000 \times 1.25ms = 25sec$).

9.1 Transience

The quantities of interest during transience are convergence delay and the control overhead required by both algorithms. The transient behavior of the algorithms is affected by the location of the root. Table 1 contains a summary of the results.

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 due to the link scheduling algorithm convergence. The delay component D_S depends on the root location and the transient states of the TDMA schedule. Since slots are shared by session control, session data and link control packets, the

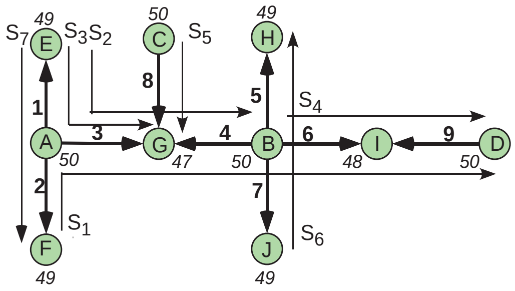

Figure 6: Arrows on links denote master-slave relationships. Italicized numbers on each node u denote $T_u^R = T_{system} - \sum_{l \in L(u)} J_l^{(u)}$, where T_{system} = 50 slots. The normalized capacities are $C_u^R = T_u^R / T_{system}$; Sessions S_1, S_2, S_3 and S_7 first receive the lowest MMF 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. These MMF rates correspond to a slot allocation $(\tau_{S_1}, .., \tau_{S_7}) = (6, 6, 6, 10, 15, 10, 6)$ within $T_{system} = 50$ slots.

circulation of some session control packets may be delayed. As illustrated in Table 1, this phenomenon was observed for the maximum D_S case (3145 slots), when the root was node B. Link 3 was allocated 5 slots or less until slot 904. Since link 3 is in the control path of first-level sessions S_1 and S_2, overall end-to-end algorithm convergence was slowed down. Such behavior did not arise in other runs. Minimum D_S (1523 slots) occurred when the root was A, the first-level bottleneck node.

	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

Table 1: Convergence delay and control overhead in the configuration of Fig. 6, for different choices of the root node.

The delay component D_L depends on the order link demands stabilize, and the location of the root with respect to this order. According to Table 1 , maximum D_L (733 slots) occurred when the root was D. The last demand to stabilize was of link 9, adjacent to the root. Although the demands at lower tree levels had already already been stabilized and satisfied, the entire tree was rescheduled from the root downwards. This worst-case scenario did not always occur: root I was also adjacent to the slowest converging demand (link 6); however, in this case, the tree was partially re-scheduled in $D_L = 361$ slots. Incidentally, minimum D_L (178 slots) occurred for the root being B, the case of maximum D_S. In all experiments D_L is less than $2T_{system}(N - 1) = 900$ slots, the convergence delay bound of STABLE_TREE.

Another quantity of interest during convergence is the control overhead–the fraction of slots used for link coordination and session control packet transmissions. According to Table 1 , the link control overhead is greater during D_S (maximum $O_S = 17.05\%$) due to the continuous changes in link demands. After the link demands stabilize, the link control overhead O_L is about 10% on the average. 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%.

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

Table 2: Steady state performance (root is node A): Session throughput (T), goodput(G) and average delay (D_{avg}) with 95% confidence intervals (d_{95}) for the configuration in Fig. 6, measured at each session destination after convergence.

9.2 Steady state

We evaluate performance at steady state in terms of control overhead and achieved session throughputs/inter-packet delays measured at the receivers. After convergence, only end-to-end control overhead exists–the circulation of session control packets is needed to track the MMF rates in presence of network dynamics. All choices of root node result in similar steady state behavior, as expected. Table 2 depicts the session throughput and goodput as well as average delay between data packet arrivals measured at the session destination after convergence (root was node A). The throughput (goodput) of a session in bps is the number of bits due to data+control packets (data packets) the destination receives for this session from the time of convergence ($D_S + D_L$) until the end of the simulation. 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}); the short delay is due to the TDMA schedule periodicity and the WRR link schedulers over each session path.

10. 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 asynchronous distributed algorithm aiming at end-to-end max-min fairness. This algorithm can operate for any topology form and compute the session MMF rates with respect to a fraction of the network capacity provided by the local feasibility conditions. We showed that tree topologies can be maximally utilized and introduced a link scheduling algorithm that can enforce the computed end-to-end rates for this case. 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 per/node utilization). Such algorithms are the subject of our future research efforts.

11. ACKNOWLEDGMENTS

This work was supported by U.S. Army Research Office (ARO) grant W911NF-04-1-0306 and by Greek General Secretariat for Research and Technology (GSRT) grant ”Cooperation with countries outside Europe”.

12. REFERENCES

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

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

[3] D. 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.

[4] D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, London, U.K., 1992.

[5] BluetoothSIG. Specification of the Bluetooth system, version 1.2. In *www.bluetooth.com*.

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

[7] Z. Cao and E. Zegura. Utility Max-Min: An Application-Oriented Bandwidth Allocation Scheme. In *Proc. IEEE INFOCOM*, New York, NY, March 1999.

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

[9] L. Chen, S. Low, and J. Doyle. Joint Congestion Control and Media Access Control Design for Ad Hoc Wireless Networks . In *Proc. IEEE INFOCOM*, Miami, FL, USA, 2005.

[10] 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.

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

[12] R. Draves, J. Padhye, and B. Zill. Routing in Multi-radio, Multi-hop Wireless Mesh Networks. In *Proc. ACM MOBICOM*, Philadelphia, PA, USA, September 2004.

[13] J. Edmonds. Maximum matching and a polyhedron with 0,1 vertices. *In Proc. Journal of Research National Bureau of Standards*, 69(B), 1965.

[14] Z. Fang and B. Bensaou. Fair Bandwidth Sharing Algorithms based on Game Theory Frameworks for Wireless Ad-hoc Networks. In *Proc. IEEE INFOCOM*, Hong Kong, March 2004.

[15] V. Gambiroza, B. Sadeghi, and E. Knightly. End-to-End Performance and Fairness in Multihop Wireless Backhaul

Networks. In *Proc. ACM MOBICOM*, Philadelphia, PA, USA, September 2004.

[16] J. Garcia-Luna-Aceves and J. Raju. Distributed Assignment of Codes for multi-hop Packet Radio Networks. In *Proc. MILCOM*, Monterey, CA, USA, October 1997.

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

[18] 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.

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

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

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

[22] 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.

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

[24] 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.

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

[26] 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.

[27] F. Kelly, A. Maulloo, and D. Tan. Rate control in communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49:237–252, 1998.

[28] M. Kodialam and T. Nandagopal. Characterizing the Achievable Rates in Multihop Wireless Networks. In *Proc. ACM MOBICOM*, San Diego, CA, USA, September 2003.

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

[30] MeshNetworks. Fostering Disruptive Technologies. In *www.meshnetworks.com*, Maitland, FL, USA, January 2002.

[31] U. K. 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.

[32] 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.

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

[34] 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.

[35] V. B. R. Sivakumar, B. Das. Spine Routing in Ad hoc Networks. *ACM/Baltzer Publications Cluster Computing Journal, Special Issue on Mobile Computing*, 3, June 1998.

[36] T. Salonidis and L. Tassiulas. Asynchronous TDMA ad hoc networks: Scheduling and Performance. *Proc. European Transactions In Telecommunications (ETT)*, 3, May-June 2004.

[37] T. Salonidis and L. Tassiulas. Distributed dynamic scheduling for end-to-end rate guarantees in wireless ad hoc networks. Technical report, TR 2004-7, Institute of Systems Research (ISR), University of Maryland, College Park, MD, USA, 2004.

[38] T. Salonidis and L. Tassiulas. Distributed on-line schedule adaptation for balanced slot allocation in wireless ad hoc networks. In *Proc. IEEE International Workshop on Quality of Service (IWQoS)*, Montreal,Canada, June 2004.

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

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

[42] 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.

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

[44] J. Wieselthier, G. 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.

[45] K. N. Y. Xue, B. Li. Price-based Resource Allocation in Wireless Ad Hoc Networks. In *Proc. 11th International Workshop on Quality of Service (IWQoS)*, Monterey, CA, USA, June 2003.

[46] Y. Yi and S. Shakkottai. Hop-by-hop Congestion Control over a Wireless Multi-hop Network. In *Proc. IEEE INFOCOM*, Hong Kong, March 2004.

[47] G. 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.

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

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

APPENDIX

A. ALGORITHM PSEUDOCODES

```
Procedure SampleReschedule
    begin
SR-1    PrioritizeLinks();
SR-2    $l_c$ = GetMaxUnstableChildLink();
        if ($l_c \neq -1$) then
            if (busybit_==0 AND BusyBit(v)==0) then
SR-3            busybit=1;
SR-4            send SC_INFO packet to $v$;
            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 = $e_{l_p}^{(u)} \oplus 1$;
        repeat
            $l_c$ = local_schedule[slot];
            if ($l_c \in LINKSET$) then
                $p_{l_c} = p$ /*set the priority of $l_c$ to p*/;
                p=p-1;
                LINKSET = LINKSET - $\{l_c\}$ ;
            end
            slot = slot $\oplus$ 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
                $lpsum = \sum_{k \in CH: p_k < p_l} (\tau_k + J_k)$;
                if ($lpsum > |[e_{l_c} \oplus 1, s_{l_p} \ominus 1]|$)) then
                    return $l_c$;
                end
            end
        end
        return -1;
    end
```

Figure 7: Procedure SampleReschedule()

```
Algorithm 1: STABLETREE
    Data    : Asynchronous events at node u 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
```

Figure 8: The asynchronous distributed link scheduling algorithm