Enrico Gregori Giuseppe Anastasi
Stefano Basagni (Eds.)

Advanced Lectures on Networking

NETWORKING 2002 Tutorials

Springer

Table of Contents

Remarks on Ad Hoc Networking

Stefano Basagni

Department of Electrical and Computer Engineering
Northeastern University
basagni@ece.neu.edu

Abstract. This papers describes selected problems and solutions for *ad hoc networking*, namely, for networking in absence of a fixed infrastructure. *All* nodes of an ad hoc networks move freely and communicate with each other only if they are in each other transmission range (neighboring nodes). This implies that in case two nodes are not neighbors, in order for them to communicate they have to rely on the forwarding services of intermediate nodes, i.e., each node is a router and the communication proceeds in multi-hop fashion. In this paper we are concerned with three aspects of ad hoc networking. The problem of accessing the wireless channel, i.e., the problem of devising *Media Access Control (MAC)* protocols. The problem of grouping the nodes of the network so to obtain a hierarchical network structure (*clustering*). The problem of setting up an ad hoc network of *Bluetooth devices*, i.e., of forming a Bluetooth *scatternet*.

1 Introduction

The ability to access and exchange information virtually anywhere, at any time, is transforming the way we live and work. Small, handheld unthethered devices are nowadays at anybody's reach, thus allowing new forms of distributed and collaborative computation. Among the several examples of this new form of communication we can mention what is often referred to as *pervasive computing*. The essence of pervasive computing is the creation of environments saturated with computing and wireless communication, yet gracefully integrated with human users. Numerous, casually accessible, often invisible computing devices, which are frequently mobile or embedded in the environment, are connected to an increasingly ubiquitus network structure.

The possible network architectures that enable pervasive computing fall into two main categories: *Cellular networks* and *multihop wireless networks* or, as commonly termed recently, *ad hoc networks*. In the first case, some specialized nodes, called *base stations*, coordinate and control all transmissions within their coverage area (or *cell*). The base station grants access to the wireless channels in response to service requests received by the mobile nodes currently in its cell. Thus the nodes simply follow the instruction of the base station: For this reason, the mobile nodes of a cellular network need limited sophistication and can request and achieve all the information they need via the base station that is currently serving them.

E. Gregori et al. (Eds.): Networking 2002 Tutorials, LNCS 2497, pp. 101–123, 2002.

The primary characteristic of an *ad hoc network architecture* is the absence of any predefined structure. Service coverage and network connectivity is defined solely by node proximity and the prevailing RF propagation characteristics. Ad hoc nodes directly communicate with one another in a peer-to-peer fashion. To facilitate communication between distant nodes, each ad hoc node also acts as a router, storing and forwarding packets on behalf of other nodes. The result is a generalized wireless network that can be rapidly deployed and dynamically reconfigured to provide on-demand networking solutions.

In this work we are concerned about this second, more general, kind of network architecture, which is recently gaining more and more attention—both from the academia and industry.

While the generic architecture of ad hoc network certainly has its advantages, it also introduces several new challenges. All network control and protocols must be distributed. For any possible collaborative task, each ad hoc node must be aware of what is happening around them, and cooperate with other nodes in order to realize critical network services, which are instead realized by the base stations in a cellular environment. Considering that most ad hoc systems are fully mobile, i.e., each node moves independently, the level of protocol sophistication and node complexity is high. Power conservation is also of the essence, since most of the devices of upcoming ad hoc networks, such as handheld devices, laptops, small robots, sensors and actuators are battery operated. Finally, networks operations and protocols should be *scalable*, i.e., largely independent of the increasing number of networks nodes, or of their larger geographical distribution.

In this paper we describe some results that have been proposed in recent years on ad hoc networking. In particular here we focus on two main aspects, namely, *Media Access Control (MAC) protocols*, i.e., methods for successfully accessing the wireless channel, and *clustering* protocols, i.e., protocols for the grouping of nodes into clusters. As an application of clustering, we illustrate a protocol for the set up of an ad hoc network of *Bluetooth* devices, Bluetooth being a wireless technology that enables ad hoc networking. The important issue of routing and in general, of multipoint communication, are not dealt with in this paper. These issues have been widely covered in many comprehensive survey papers, such as [1] and [2].

The rest of the paper is organized as follows. In the next section we describe the problem of accessing the wireless channel, and the proposed MAC protocols that solve this problem. In Section 3 we give an example of a clustering algorithm which is well suited for mobile ad hoc networks. We finally illustrate an application of the presented clustering algorithm for setting up ad hoc networks of Bluetooth devices (Section 4). Section 5 concludes the paper.

2 Ad Hoc MAC Protocols

Ad hoc networks do not have the benefit of having predefined base stations to coordinate channel access, thus invalidating many of the assumptions held

by MAC designs for the centralized (cellular) architecture. In this section, we focus our attention on MAC protocols that are specifically designed for ad hoc networks. (This section is based on the research performed with Dr. Andrew Myers, and can be more thoroughly found in [3]).

We start by exploring the physical constraints of the wireless channel and discuss their impact on MAC protocol design and performance.

Radio waves propagate through an unguided medium that has no absolute or observable boundaries and is vulnerable to external interference. The signal strength of a radio transmission rapidly attenuates as it progresses away from the transmitter. This means that the ability to detect and receive transmissions is dependent on the distance between the transmitter and receiver. Only nodes that lie within a specific radius (the *transmission range*) of a transmitting node can detect the signal (carrier) on the channel. This location dependent carrier sensing can give rise to so-called *hidden* and *exposed* nodes that can detrimentally affect channel efficiency. A hidden node is one that is within range of a receiver but not the transmitter, while the contrary holds true for an exposed node. Hidden nodes increase the probability of *collision* at a receiver, while exposed nodes may be denied channel access unnecessarily, thereby under utilizing the bandwidth resources.

Performance is also affected by the signal propagation delay, i.e., the amount of time needed for the transmission to reach the receiver. Protocols that rely on carrier sensing are especially sensitive to the propagation delay. With a significant propagation delay, a node may initially detect no active transmissions when, in fact, the signal has simply failed to reach it in time. Under these conditions, collisions are much more likely to occur and system performance suffers. In addition, wireless systems that use a synchronous communications model must increase the size of each time slot to accommodate propagation delay. This added overhead reduces the amount of bandwidth available for information transmission.

A possible taxonomy of ad hoc MAC protocols includes three broad protocol categories that differ in their channel access strategy: *Contention protocols*, *allocation protocols*, and a combination of the two (*hybrid protocols*).

Contention protocols use direct competition to determine channel access rights, and resolve collisions through randomized retransmissions. Prime examples of this protocols are ALOHA and CSMA (for a brief discussion on "core" MAC protocols such as ALOHA, slotted ALOHA, CSMA, TDMA, FDMA and CDMA the reader is referred to [3]). With the exception of slotted ALOHA, most contention protocols employ an asynchronous communication model. Collision avoidance is also a key design element that is realized through some form of control signaling.

The contention protocols are simple and tend to perform well at low traffic loads, i.e., when there are few collision, leading to high channel utilization and low packet delay. However, protocol performance tends to degrade as the traffic loads are increased and the number of collisions rise. At very high traffic loads, a contention protocol can become unstable as the channel utilization drops. This

can result in exponentially growing packet delay and network service breakdown since few, if any, packets can be successfully exchanged.

Allocation protocols employ a synchronous communication model, and use a scheduling algorithm that generates a mapping of time slots to nodes. This mapping results in a transmission schedule that determines in which particular slots a node is allowed to access the channel. Most allocation protocols create collision-free transmission schedules, thus the schedule length (measured in slots) forms the basis of protocol performance. The time slots can either be allocated statically or dynamically, leading to a fixed and variable schedule length.

The allocation protocols tend to perform well at moderate to heavy traffic loads as all slots are likely to be utilized. These protocols also remain stable even when the traffic loads are extremely high. This is due to the fact that most allocation protocols ensure that each node has collision-free access to at least one time slot per frame. On the other hand, these protocols are disadvantaged at low traffic loads due to the artificial delay induced by the slotted channel. This results in significantly higher packet delays with respect to the contention protocols.

Hybrid protocols can be loosely described as any combination of two or more protocols. However, in this section, the definition of the term hybrid will be constrained to include only those protocols that combine elements of contention and allocation based channel access schemes in such a way as to maintain their individual advantages while avoiding their drawbacks. Thus the performance of a hybrid protocol should approximate a contention protocol when traffic is light, and an allocation protocol during periods of high load. (For details on hybrid protocols the reader is referred to [3].)

2.1 Contention Protocols

Contention protocols can be further classified according to the type collision avoidance mechanism employed. The ALOHA protocols make up the category of protocols that feature no collision avoidance mechanism, i.e., they simply react to collision via randomized retransmissions. Most contention protocols, however, use some form of collision avoidance mechanism.

The busy-tone multiple access (BTMA) protocol [4] divides the entire bandwidth into two separate channels. The main *data channel* is used for the transmission of packets, and occupies the majority of the bandwidth. The *control channel* is used for the transmission of a special *busy-tone signal* that indicates the presence of activity on the data channel. These signals are not bandwidth intensive, thus the control channel is relatively small.

The BTMA protocol operates as follows. When a source node has a packet to transmit, it first listens for the busy-tone signal on the control channel. If the control channel is idle, i.e., no busy-tone is detected, then the node may begin transmitting its packet. Otherwise, the node reschedules the packet for transmission at some later time. Any node that detects activity on the data channel immediately begins transmitting the busy-tone on the control channel. This continues until the activity on the data channel ceases.

In this way, BTMA prevents all nodes that are two hops away from an active source node from accessing the data channel. This significantly lowers the level of hidden node interference, and therefore reduces the probability of collision. However, the number of exposed nodes is dramatically increased. The consequence being a severely underutilized data channel.

The receiver initiated busy-tone multiple access (RI-BTMA) protocol [5] attempts to minimize the number of exposed nodes by having only the destination(s) transmit the busy-tone. Rather than immediately transmitting the busy-tone upon detection of an active data channel, a node monitors the incoming data transmission to determine whether it is a destination. This determination takes a significant amount of time, especially in a noisy environment with corrupted information. During this time, the initial transmission remains vulnerable to collision. This can be particularly troublesome in high speed systems where the packet transmission time may be short.

The wireless collision detect (WCD) protocol [6] essentially combines the BTMA and RI-BTMA protocols by using two distinct busy-tone signals on the control channel. WCD acts like BTMA when activity is first detected on the main channel, i.e., it transmits a *collision detect* (CD) signal on the BTC. RI-BTMA behavior takes over once a node determines it is a destination. In this case, a destination stops transmitting the CD signal, and begins transmitting a *feedback-tone* (FT) signal. In this way, WCD minimizes the exposed nodes while still protecting the transmission from hidden node interference.

These busy-tone protocols feature simple designs that require only a minimal increase in hardware complexity. Because of its unique characteristics, the WCD protocol is the overall performance leader followed by RI-BTMA and BTMA, respectively [6]. Furthermore, the performance of busy-tone protocols are less sensitive to the hardware switching time since it is assumed that a node can transmit and receive on the data and control channels simultaneously. However, wireless systems that have a limited amount of RF spectrum may not be able to realize a separate control and data channel. In such cases, collision avoidance using in-band signaling is necessary.

The multiple access with collision avoidance (MACA) protocol [7] uses a handshaking dialogue to alleviate hidden node interference and minimize the number of exposed nodes. This handshake consists of a *request-to-send* (RTS) control packet that is sent from a source node to its destination. The destination replies with a *clear-to-send* (CTS) control packet, thus completing the handshake. A CTS response allows the source node to transmit its packet. The absence of a CTS forces a node to reschedule the packet for transmission at some later time.

Notice that a hidden node is likely to overhear the CTS packet sent by a destination node, while an exposed node is not. Thus by including the time needed to receive a CTS and packet in the respective RTS and CTS packets, we reduce the likelihood of hidden node interference and the number of exposed nodes simultaneously.

The MACAW protocol [8] enhances MACA by including carrier sensing to avoid collisions among RTS packets, and a positive acknowledgement (ACK) to aid in the rapid recovery of lost packets. To protect the ACK from collision, a source node transmits a *data sending* (DS) control packet to alert exposed nodes of its impending arrival. Improvements are also made to the collision resolution algorithm to ensure a more equitable sharing of the channel resources.

The MACA with piggyback reservations (MACA/PR) protocol [9] enhances MACA by incorporating channel reservations. This allows the system to support QoS sensitive applications. Each node maintains a *reservation table* (RT) that is used to record the channel reservations made by neighboring nodes. A source node makes a reservation by first completing a RTS/CTS exchange. It then sends the first real-time packet whose header contains the time interval specifying the interval in which the next one will be sent. The destination responds with an ACK carrying the equivalent time interval. Other nodes within range note this reservation in their RT, and remain silent during the subsequent time intervals. Thus the source node can send subsequent real-time packets without contention. To ensure proper bookkeeping, the nodes periodically exchange their RTs.

The MACA by invitation (MACA-BI) protocol [10] reverses the handshaking dialogue of MACA. In this case, the destination node initiates packet transmission by sending a *request-to-receive* (RTR) control packet to the source node. The source node responds to this poll with a packet transmission. Thus each node must somehow predict when neighbors have packets for it. This requires each node must maintain a list of its neighbors along with their traffic characteristics. In order to prevent collision, the nodes must also synchronize their polling mechanisms by sharing this information with their neighbors.

These MACA based contention protocols minimize collisions by reducing the negative effect of hidden and exposed nodes through simple handshaking dialogues. However, the exchange of multiple control packets for each data packet magnifies the impact of signal propagation delay and hardware switching time. To some extent the MACA/PR and MACA/BI protocols alleviate these problems reducing the amount of handshaking, yet the amount of state information maintained at each node can be substantial.

2.2 Allocation Protocols

There are two distinct classes of allocation protocols that differ in the way the transmission schedules are computed. *Static allocation protocols* use a centralized scheduling algorithm that statically assigns a fixed transmission schedule to each node prior to its operation. This type of scheduling is equivalent to the assignment of MAC addresses for Ethernet interface cards. *Dynamic allocation protocols* uses a distributed scheduling algorithm that computes transmission schedule in an on-demand fashion.

Since the transmission schedules are assigned beforehand, the scheduling algorithm of a static allocation protocols requires global system parameters as input. The classic TDMA protocol builds its schedules according to the maximum number of nodes in the network. For a network of N nodes, the protocol

uses a frame length of N slots and assigns each node one unique time slot. Since each node has exclusive access to one slot per frame, there is no threat of collision for any packet type (i.e., unicast or multicast). Moreover, the channel access delay is bounded by the frame length. Because of the equivalence between system size and frame length, classic TDMA performs poorly in large scale networks.

The time spread multiple access (TSMA) protocol [11] relaxes some of the strict requirements of classic TDMA to achieve better performance while still providing bounded access delay. The TSMA scheduling algorithm assigns each node multiple slots in a single frame, and permits a limited amount of collisions to occur. These two relaxations allow TSMA to obtain transmission schedules whose length scales *logarithmically with respect to the number of nodes*. Furthermore, TSMA guarantees the existence of a collision-free transmission slot to each neighbor within a single frame.

The source of this "magic" is the scheduling algorithm that makes use of the mathematical properties of finite fields. An excellent introduction to finite fields can be found in [12]. The scheduling algorithm is briefly outlined as follows. For a network of N nodes, the parameters q (of the form $q = p^m$, where p is a prime and m an integer) and integer k are chosen such that $q^{k+1} \geq N$ and $q \geq kD_{max} + 1$, where D_{max} is the maximum node degree. Each node can then be assigned a unique polynomial f over the Galois field $GF(q)$. Using this polynomial, a unique TSMA transmission schedule is computed where bit $i = 1$ if $(i \bmod q) = f(\lfloor i/q \rfloor)$, otherwise $i = 0$.

As shown in [11], that this TSMA scheduling algorithm provides each node with a transmission schedule with guaranteed access in each time frame. The maximum length of this schedule is bounded by:

$$L = O\left(\frac{D_{max}^2 \log^2 N}{\log^2 D_{max}}\right).$$

Notice that the frame length scales logarithmically with the number of nodes and quadratically with the maximum degree. For ad hoc networks consisting of thousands of nodes with a sparse topology (i.e., small D_{max}), TSMA can yield transmission schedules that are much shorter than TDMA. Table 1 compares the frame lengths of TDMA and TSMA for a network of $N = 1000$ nodes. For TSMA protocols a $\Omega(\log n)$ lower bound has been proved for L in [13]. We notice that there is still a gap between the TSMA upper bound and the mentioned logarithmic lower bound. Therefore, there is still room for improvements (more likely on the lower bound side). Protocols TSMA-like have also been deployed as a basis for implementing *broadcast* (i.e., one-to-all communication) in ad hoc networks. Upper and lower bound for deterministic and distributed TSMA-based broadcast can be found in [14,15] and [16], respectively.

With mobile ad hoc networks, nodes may be activated and deactivated without warning, and unrestricted mobility yields a variable network topologies. Consequently, global parameters, such as node population and maximum degree, are typically unavailable or difficult to predict. For this reason, protocols that use only local parameters have been developed. A local parameter refers to information that is specific to a limited region of the network, such as the number of

Table 1. Frame lengths of classic TDMA vs. TSMA.

	$D = 2$	$D = 5$	$D = 10$	$D = 15$
TDMA	1000	1000	1000	1000
TSMA	49	121	529	961

nodes within x hops of a reference node (referred to as an x-hop neighborhood). A dynamic allocation protocol then uses these local parameters to deterministically assign transmission slots to nodes. Because local parameters are likely to vary over time, the scheduling algorithm operates in a distributed fashion and is periodically executed to adapt to network variations.

Dynamic allocation protocols typically operate in two phases. Phase one consists of a set of reservation slots in which the nodes contend for access to the subsequent transmission slots. Lacking a coordinating base station, contention in this phase requires the cooperation of each individual node to determine and verify the outcome. Successful contention in phase one grants a node access to one or more transmission slots of phase two, in which packets are sent.

A great number of dynamic allocation protocols have been proposed. The protocols in [17]-[18] are just a few excellent examples of this two-phase design. The protocols in [17]-[19] use a contention mechanism that is based on classic TDMA. Essentially the nodes take turns contending for slot reservations, with the earliest node succeeding. This results in a high degree of unfairness which is equalized by means of a reordering policy. Although these protocols create transmission schedules that are specific to the local network topology, they still require global parameters.

In contrast, the five phase reservation protocol (FPRP) [18] is designed to be arbitrarily scalable, i.e., independent of the global network size. FPRP uses a complex frame structure that consists of two subframe types, namely *reservation frames* and *information frames*. A reservation frame precedes a sequence of k information frames. Each reservation frame consists of ℓ *reservation slots* that correspond to the ℓ *information slots* of each information frame. Thus, if a node wants to reserve a specific information slot, it contends in the corresponding reservation slot. At the end of the reservation frame, a TDMA schedule is created and used in the following k information frames. The schedule is then recomputed in the next reservation frame.

In order to accommodate contention, each reservation slot consists of m *reservation cycles* that contain a five round reservation dialogue. A reservation is made in the first four rounds, while the fifth round is mainly used for performance optimization. The contention is summarized as follows. A node that wishes to make a reservation sends out a request using p-persistent slotted ALOHA (round 1), and feedback is provided by the neighboring nodes (round 2). A successful request, i.e., one that did not involve a collision, allows a node to reserve the slot (round 3). All nodes within two hops of the source node are then notified of the reservation (round 4). These nodes will honor the reservation and make no further attempts to contend for the slot. Any unsuccessful reservation attempts

are resolved through a pseudo-Bayesian resolution algorithm that randomizes the next reservation attempt.

In [18], FPRP is shown to yield transmission schedules that are collision-free, however the protocol requires a significant amount of overhead. Each reservation cycle requires a number of hardware switches between transmitting and receiving modes. Each round of contention must also be large enough to accommodate the signal, propagation delay and physical layer overhead (e.g., synchronization and guard time). Add this together and multiply the result by m reservation cycles and ℓ reservation slots, and the end result is anything but trivial. Furthermore, the system parameters k, ℓ and m are heuristically determined through simulation and then fixed in the network. This limits the ability of FPRP to dynamically adapt its operation to suit the current network conditions which may deviate from the simulated environment.

3 Clustering for Ad Hoc Networks

Among the many ways to cope with the barriers and challenges posed by the ad hoc network architecture, here we describe a possible solution based on *grouping* the network nodes into *clusters*. This operation goes commonly under the name of *clustering*.

In this section, we describe a protocol for clustering set up and clustering maintenance in presence of node mobility. The cluster are characterized by a node that coordinates the clustering process (a *clusterhead*) and possibly few non-clusterhead nodes, that have direct access to only one clusterhead (one hop, non-overlapping clusters).

In the following description of the clustering protocol we consider an *ad hoc* network as an undirected graph $G = (V, E)$ in which V, $|V| = n$, is the set of (wireless) nodes and there is an edge $\{u, v\} \in E$ if and only if u and v can mutually receive each others' transmission. In this case we say that u and v are neighbors. The set of the neighbors of a node $v \in V$ will be denoted by $\Gamma(v)$. Due to mobility, the graph can change in time.

Every node v in the network is assigned a unique identifier (ID). For simplicity, here we identify each node with its ID and we denote both with v. Finally, we consider weighted networks, i.e., a weight w_v (a real number ≥ 0) is assigned to each node $v \in V$ of the network. For the sake of simplicity, here we stipulate that each node has a different weight. (In case two nodes have the same weight, the tie can be broken arbitrarily, e.g., by using the nodes' ID.)

In this section, clustering an ad hoc network means partitioning its nodes into *clusters*, each one with a *clusterhead* and (possibly) some *ordinary nodes*. (Clusterhead and ordinary node ar the *roles* that each node may assume.) The choice of the clusterheads is here based on the *weight* associated to each node: the bigger the weight of a node, the better that node for the role of clusterhead. Each node constantly computes its weight based on what is most critical to that node for the specific network application (e.g., node mobility, its remaining battery life, and its *connectivity degree*, i.e., the number of its neighbors). For

instance, as introduced in [20,21,22], we assume the following expression for the computation of node v's weight:

$$w_v = \sum_{i \in I} c_i P_i$$

where the c_is are the (constant) weighing factors for the $|I|$ system parameters of interest P_i.

The protocol described in this section is a generalization of the Distributed Mobility-Adaptive Clustering (DMAC) originally presented in [23]. DMAC is a distributed algorithm for clustering set-up and maintenance in presence of node mobility that partition the nodes of the network into "one hop" clusters in such a way that no two clusterheads can be neighbors and so that whenever a "better clusterhead" moves into the neighborhood of an ordinary nodes, the ordinary node must affiliate to the new clusterhead. Here we relax these conditions allowing clusterheads to be neighbors and allowing ordinary nodes to choose whether to switch to a new neighboring clusterhead or not.

The process of cluster formation/maintenance is continuously executed at each node, and each node decides its own role so that the following three requirements (that we call "ad hoc clustering properties") are satisfied:

1. Every ordinary node always affiliates with only one clusterhead.
2. For every ordinary node v there is no clusterhead $u \in \Gamma(v)$ such that $w_u > w_{Clusterhead} + h$, where *Clusterhead* indicates the current clusterhead of v.
3. A clusterhead cannot have more than k neighboring clusterheads.

Requirement number 1. ensures that each ordinary node has direct access to at least one clusterhead (the one of the cluster to which it belongs), thus allowing fast intra- and inter-cluster communications. This is the property that insures that this protocol is a "single hop" kind of clustering protocol. Also, since an ordinary node affiliates only to one clusterhead, the obtained clusters are not overlapping. The second requirement guarantees that each ordinary node always stays with a clusterhead that can give it a "guaranteed good" service. By varying the threshold parameter h it is possible to reduce the communication overhead associated to the passage of an ordinary node from its current clusterhead to a new neighboring one when it is not necessary. Finally, requirement number 3. allows us to have the number of clusterheads that can be neighbors as a parameter of the algorithm. This, as seen for requirement number 2. allows us to consistently reduce the communication overhead due to the change of role of nodes.

The following description of the algorithm is based on the following two common assumptions:

- A message sent by a node is received correctly within a finite time (a *step*) by all its neighbors.
- Each node knows its own ID, its weight, its role (if it has already decided its role) and the ID, the weight and the role of all its neighbors (if they have already decided their role). When a node has not yet decided what its role is going to be, it is considered as an ordinary node.

The algorithm is executed at each node in such a way that at a certain time a node v decides (to change) its role. This decision is entirely based on the decision of the nodes $u \in \Gamma(v)$ such that $w_u > w_v$.

Except for the initial procedure, the algorithm is message driven: a specific procedure will be executed at a node depending on the reception of the corresponding message. We use three types of messages that are exchanged among the nodes: CH(v), used by a node $v \in V$ to make its neighbors aware that it is going to be a clusterhead, JOIN(v, u), with which a node v communicates to its neighbors that it will be part of the cluster whose clusterhead is node $u \in \Gamma(v)$, $v, u \in V$, and RESIGN(w) that notifies a clusterhead whose weight is $\leq w$ that it has to resign its role. In the following discussion and in the procedures we use the following notation:

- v, the generic node executing the algorithm (from now on we will assume that v encodes not only the node's ID but also its weight w_v);
- *Cluster*(v), the set of nodes in v's cluster. It is initialized to $\emptyset$, and it is updated only if v is a clusterhead;
- *Clusterhead*, the variable in which every node records the (ID of the) clusterhead that it joins. It is initialized to **nil**;
- *Ch*$(-)$, boolean variables. Node v sets *Ch*(u), $u \in \{v\} \cup \Gamma(v)$, to **true** when either it sends a CH(v) message ($v = u$) or it receives a CH(u) message from u ($u \neq v, u \in \Gamma(v)$).

Furthermore:

- Every node is made aware of the failure of a link, or of the presence of a new link by a service of a lower level (this will trigger the execution of the corresponding procedure);
- The procedures of the algorithm (M-procedures, for short) are "atomic," i.e., they are not interruptible;
- At clustering set up or when a node is added to the network its variables *Clusterhead*, *Ch*$(-)$, and *Cluster*$(-)$ are initialized to **nil**, **false** and $\emptyset$, respectively.

The following two rules define how the nodes assume/change their roles adapting to changes in the network topology.

1. Each time a node v moves into the neighborhood of a clusterhead u with a bigger weight, node v switches to u's cluster only if $w_u > w_{\mathit{Clusterhead}} + h$, where *Clusterhead* is the clusterhead of v (it can be v itself) and h is a real number ≥ 0. This should happen independently of the current role of v. With this rule we want to model the fact that we incur the switching overhead only when it is really convenient. When $h = 0$ we simply obtain that each ordinary nodes affiliates to the neighboring clusterhead with the biggest weight.
2. We allow a clusterhead v to have up to k neighboring clusterheads, $0 \leq k < n$. We call this condition the k-neighborhood condition. Choosing $k = 0$ we obtain that no two clusterheads can be neighbors (maximum degree of independence: In graph-theoretic terms, the resulting set of clusterhead is an *independent set*).

The parameters h and k can be different from node to node, and they can vary in time. This allows the algorithm to self-configure dynamically in order to meet the specific needs of upper layer applications/protocols that requires an underlying clustering organization. At the same time, different values of h and k allow our algorithm to take into account dynamically changing network conditions, such as the network connectivity (related to the average nodal degree, i.e., to the average number of the neighbors of the nodes), variations in the rate of the mobility of the nodes, etc.

The following is the description of the six M-procedures.

• *Init.* At the clustering set up, or when a node v is added to the network, it executes the procedure *Init* in order to determine its own role. If among its neighbors there is at least a clusterhead with bigger weight, then v will join it. Otherwise it will be a clusterhead. In this case, the new clusterhead v has to check the number of its neighbors that are already clusterheads. If they exceed k, then a RESIGN message is also transmitted, bearing the weight of the first clusterhead (namely, the one with the $(k+1)$th biggest weight) that violates the k-neighborhood condition (this weight is determined by the operator $\min_k$). On receiving a message RESIGN(w), every clusterhead u such that $w_u \leq w$ will resign. Notice that a neighbor with a bigger weight that has not decided its role yet (this may happen at the clustering set up, or when two or more nodes are added to the network at the same time), will eventually send a message (every node executes the *Init* procedure). If this message is a CH message, then v could possibly resign (after receiving the corresponding RESIGN message) or affiliate with the new clusterhead.

```
PROCEDURE Init;
begin
    if {z ∈ Γ(v) : w_z > w_v  ∧  Ch(z)} ≠ ∅
        then begin
                 x := max_{w_z > w_v}{z : Ch(z)};
                 send JOIN(v,x);
                 Clusterhead := x
             end
        else begin
                 send CH(v);
                 Ch(v) := true;
                 Clusterhead := v;
                 Cluster(v) := {v};
                 if |{z ∈ Γ(v) : Ch(z)}| > k then
                     send RESIGN(min_k{w_z : z ∈ Γ(v) ∧  Ch(z)})
             end
end;
```

• *Link_failure.* Whenever made aware of the failure of the link with a node u, node v checks if its own role is clusterhead and if u used to belong to its cluster. If this is the case, v removes u from $Cluster(v)$. If v is an ordinary node, and u was its own clusterhead, then it is necessary to determine a new role for v. To this aim, v checks if there exists at least a clusterhead $z \in \Gamma(v)$ such that

$w_z > w_v$. If this is the case, then v joins the clusterhead with the bigger weight, otherwise it becomes a clusterhead. As in the case of the *Init* procedure, a test on the number of the neighboring clusterheads is now needed, with the possible resigning of some of them.

```
PROCEDURE Link_failure(u);
begin
    if Ch(v) and (u ∈ Cluster(v))
        then Cluster(v) := Cluster(v) \ {u}
        else if Clusterhead= u then
                if {z ∈ Γ(v) : w_z > w_v ∧ Ch(z)} ≠ ∅
                    then begin
                            x := max_{w_z > w_v}{z : Ch(z)};
                            send JOIN(v,x);
                            Clusterhead:= x
                         end
                    else begin
                            send CH(v);
                            Ch(v) := true;
                            Clusterhead:= v;
                            Cluster(v) := {v};
                            if |{z ∈ Γ(v) : Ch(z)}| > k then
                               send RESIGN(min_k{w_z : z ∈ Γ(v) ∧ Ch(z)})
                         end
end;
```

• *New_link*. When node v is made aware of the presence of a new neighbor u, it checks if u is a clusterhead. If this is the case, and if w_u is bigger than the weight of v's current clusterhead plus the threshold h, than, independently of its own role, v affiliates with u. Otherwise, if v itself is a clusterhead, and the number of its current neighboring clusterheads is $> k$ then the operator $\min_k$ is used to determine the weight of the clusterhead x that violates the k-neighborhood condition. If $w_v > w_x$ then node x has to resign, otherwise, if no clusterhead x exists with a weight smaller than v's weight, v can no longer be a clusterhead, and it will join the neighboring clusterhead with the biggest weight.

```
PROCEDURE New_link(u);
begin
    if Ch(u) then
        if (w_u > w_Clusterhead + h)
            then begin
                    send JOIN(v,u);
                    Clusterhead:= u;
                    if Ch(v) then Ch(v) := false
                 end
            else if Ch(v) and |{z ∈ Γ(v) : Ch(z)}| > k then
                        begin
                            w := min_k{w_z : z ∈ Γ(v) ∧ Ch(z)};
                            if w_v > w then send RESIGN(w)
```

```
                                    else begin
                                           x := max_{w_z > w_v}{z : Ch(z)};
                                           send JOIN(v,x);
                                           Clusterhead := x;
                                           Ch(v) := false
                                         end
                            end
    end;
```

• *On receiving* CH(u). When a neighbor u becomes a clusterhead, on receiving the corresponding CH message, node v checks if it has to affiliate with u, i.e., it checks whether w_u is bigger than the weight of v's clusterhead plus the threshold h or not. In this case, independently of its current role, v joins u's cluster. Otherwise, if v is a clusterhead with more than k neighbors which are clusterheads, as in the case of a new link, the weight of the clusterhead x that violates the k-neighborhood condition is determined, and correspondingly the clusterhead with the smallest weight will resign.

```
On receiving CH(u);
begin
    if (w_u > w_Clusterhead + h) then begin
                                        send JOIN(v,u);
                                        Clusterhead := u;
                                        if Ch(v) then Ch(v) := false
                                    end
                                    else if Ch(v) and |{z ∈ Γ(v) : Ch(z)}| > k
                                          then begin
                                             w := min_k{w_z : z ∈ Γ(v) ∧ Ch(z)};
                                             if w_v > w then send RESIGN(w)
                                                        else begin
                                                          x := max_{w_z > w_v}{z : Ch(z)};
                                                          send JOIN(v,x);
                                                          Clusterhead := x;
                                                          Ch(v) := false
                                                        end
                                          end
    end;
```

• *On receiving* JOIN(u,z). On receiving the message JOIN(u,z), the behavior of node v depends on whether it is a clusterhead or not. In the affirmative, v has to check if either u is joining its cluster ($z = v$: in this case, u is added to $Cluster(v)$) or if u belonged to its cluster and is now joining another cluster ($z \neq v$: in this case, u is removed from $Cluster(v)$). If v is not a clusterhead, it has to check if u was its clusterhead. Only if this is the case, v has to decide its role: It will join the biggest clusterhead x in its neighborhood such that $w_x > w_v$ if such a node exists. Otherwise, it will be a clusterhead. In this latter case, if the k-neighborhood condition is violated, a RESIGN message is transmitted in order for the clusterhead with the smallest weight in v's neighborhood to resign.

```
On receiving JOIN(u, z);
begin
    if Ch(v)
        then if z = v then Cluster(v) := Cluster(v) ∪ {u}
                      else if u ∈ Cluster(v) then Cluster(v) := Cluster(v) \ {u}
        else if Clusterhead = u then
                    if {z ∈ Γ(v) : w_z > w_v ∧ Ch(z)} ≠ ∅
                    then begin
                             x := max_{w_z > w_v}{z : Ch(z)};
                             send JOIN(v,x);
                             Clusterhead := x
                         end
                    else begin
                             send CH(v);
                             Ch(v) := true;
                             Clusterhead := v;
                             Cluster(v) := {v};
                             if |{z ∈ Γ(v) : Ch(z)}| > k then
                                 send RESIGN(min_k{w_z : z ∈ Γ(v) ∧ Ch(z)})
                         end
end;
```

• *On receiving* RESIGN(w). On receiving the message RESIGN(w), node v checks if its weight is $\leq w$. In this case, it has to resign and it will join the neighboring clusterhead with the biggest weight. Notice that since the M-procedures are supposed to be not interruptible, and since v could have resigned already, it has also to check if it is still a clusterhead.

```
On receiving RESIGN(w);
begin
    if Ch(v) and w_v ≤ w then begin
                                  x := max_{w_z > w_v}{z : Ch(z)};
                                  send JOIN(v,x);
                                  Clusterhead := x;
                                  Ch(v) := false
                              end
end;
```

The correctness of the described protocol in achieving the ad hoc clustering properties listed above can be found in [21] along with simulation results that demonstrate the effectiveness of the protocol in reducing the overhead of role switching in presence of the mobility of the nodes.

4 Forming Ad Hoc Networks of Bluetooth Devices

In this section we illustrate the use of clustering as described in the previous section to define a protocol for *scatternet formation*, i.e., the formation of an ad hoc network of Bluetooth devices. The protocol outlined in this section is joint

research with Professor Chiara Petrioli and has been described more thoroughly in [24] and [25].

Bluetooth Technology (BT) [26] is emerging as one of the most promising enabling technologies for ad hoc networks. It operates in the 2.4GHz, unlicensed ISM band, and adopts frequency hopping spread spectrum to reduce interferences.

When two BT nodes come into each others communication range, in order to set up a communication link, one of them assumes the role of *master* of the communication and the other becomes its *slave*. This simple "one hop" network is called a *piconet*, referred in the following as a *BlueStar*, and may include many slaves, no more than 7 of which can be active (i.e., actively communicating with the master) at the same time. If a master has more than seven slaves, some slaves have to be "parked." To communicate with a parked slave a master has to "unpark" it, while possibly parking another slave.

All active devices in a piconet share the same channel (i.e., a frequency hopping sequence) which is derived from the unique ID and Bluetooth clock of the master. Communication to and from a slave device is always performed through its master.

A BT device can timeshare among different piconets. In particular, a device can be the master of one piconet and a slave in other piconets, or it can be a slave in multiple piconets. Devices with multiple roles will act as gateways to adjacent piconets, resulting in a multihop ad hoc network called a *scatternet*.

Although describing methods for device discovery and for the participation of a node to multiple piconets, the BT specification does not indicate any methods for scatternet formation. The solutions proposed in the literature so far ([27], [28], and [29]), either assume the radio vicinity of all devices ([27] and [29]), or require a designated device to start the scatternet formation process, [28]. Furthermore, the resulting scatternet topology is a tree, which limits the efficiency and robustness of the resulting scatternet.

In this paper we present BlueStars, a new scatternet formation protocol for multi-hop Bluetooth networks, that overcomes the drawbacks of previous solutions in that it is fully distributed, does not require each node to be in the transmission range of each other node and generates a scatternet whose topology is a mesh rather than a tree.

The protocol proceeds in three phases:

1. The first phase, *topology discovery*, concerns the *discovery of neighboring devices*. By the end of this phase, neighboring devices acquire a "symmetric" knowledge of each other.
2. The second phase takes care of BlueStar (piconet) formation. By the end of this phase, the whole network is covered by disjoint piconets.
3. The final phase concerns the selection of *gateway devices* to connect multiple BlueStars so that the resulting *BlueConstellation* is connected.

These three phases are described in the following sections.

4.1 Topology Discovery

The first phase of the protocol, the topology discovery phase, allows each device to become aware of its one hop neighbors' ID and weight. According to the BT specification version 1.1, discovery of unknown devices is performed by means of the *inquiry procedures*.

The problem of one-hop neighbors discovery in Bluetooth has been dealt with extensively in [27] (for "single hop" networks, i.e., networks in which all devices are in each other transmission range) and [30] (for multihop networks). The BT inquiry and paging procedures are used to set up two-node temporary piconets through which two neighboring devices exchange identity, weight and synchronization information needed in the following phases of the scatternet formation protocol. This information exchange allows a "symmetric" knowledge of one node's neighbors, in the sense that if a node u discovers a neighbor v, node v discovers u as well.

4.2 BlueStars Formation

In this section, we describe a distributed protocol for grouping the BT devices into piconets. Given that each piconet is formed by one master and a limited number of slaves that form a star-like topology, we call this phase of the protocol BlueStars formation phase.

Based on the information gathered in the previous phase, namely, the ID, the weight, and synchronization information of the discovered neighbors, each device performs the protocol locally. The rule followed by each device is the following: A device v decides whether it is going to be a master or a slave depending on the decision made by the neighbors with bigger weight (v's "bigger neighbors"). In particular, v becomes the slave of the first master among its bigger neighbors that has paged it and invited it to join its piconet. In case no bigger neighbors invited v, v itself becomes a master. Once a device has decided its role, it communicates it to all its (smaller) neighbors so that they can also make their own decision.

Let us call *init devices* all the devices that have the biggest weight in their neighborhood. If two nodes have the same weight, the tie can be broken by using the devices unique ID. Init devices are the devices that initiate the BlueStars formation phase. They will be masters. As soon as the topology discovery phase is over, they go to page mode and start paging their smaller neighbors one by one. All the other devices go in paging scan mode.

The protocol operations in this phase are described by the initOperations() procedure described below.

```
initOperations() {
    if (for each neighbor u: myWeight > uWeight) {
        myRole = 'master';
        go to page mode;
        send page(v, master, v) to all smaller neighbors;
        exit the execution of this phase of the protocol; }
```

```
    else
        go to page scan mode;
}
```

The following procedure is triggered at a non-init device v by the reception of a page. The parameter of the page are the identity of the paging device u, its role (either 'master' or 'slave') and, in the case the paging device u is a slave, the identity of the device to which it is affiliating. (In case u is a master this parameter is irrelevant and can be set to u itself.)

```
onReceivingPage(deviceId u, string role, deviceId t) {
record that u has paged;
record role(u);
if (role(u) == 'slave')
    master(u) = t;
if (myWeight < uWeight) {
    if (role(u) == 'master')
        if (myRole == 'none') {
            join u's piconet;
            myMaster = u;
            myRole = 'slave'; }
        else
            inform u about myMaster's ID;
    if (some bigger neighbor has to page yet)
        exit and wait for the following page;
    else {
        switch to page mode;
        if (all bigger devices are slaves) {
            myRole = 'master';
            send page(v, master, v) to each neighbors
            (smaller neighbors first);
            exit the execution of this phase of the protocol; }
        else {
             send page(v, slave, myMaster) to each neighbors;
             switch to page scan mode; } } }
else
    if (all neighbors have paged)
        exit the execution of this phase of the protocol;
    else
        exit and wait for the next page;
}
```

The procedure of recording the role of a device u includes all the information of synchronization, addressing, etc., that enable v to establish a communication with u at a later time, if needed.

Upon receiving a page from a device u, device v starts checking if this is a page from a bigger neighbor or from a smaller one. In the former case, it checks

if the sender of the page is a master. If so, and v is not part of any piconet yet, it joins device u's piconet. If instead device v has already joined a piconet, it informs device u about this, also communicating the name of its master. Device v then proceeds to check if all its bigger neighbors have paged it. If this is not the case, it keeps waiting for another page (exiting the execution of the procedure).

When successfully paged by all its bigger neighbors, device v knows whether it has already joined the piconet of a bigger master or not. In the first case, device v is the slave of the bigger master that paged it first. In the latter case device v itself is going to be a master. In any case, device v goes to page mode, and communicates its decision to all its smaller neighbors.

At this point, a master v exits the execution of this phase of the protocol. If device v is a slave, it returns to page scan mode and waits for pages from all its smaller neighbors of which it still does not know the role. Indeed, some of a slave's smaller neighbors may not have decided their role at the time they are paged by the bigger slave. As soon as a device makes a decision on its role, it therefore pages its bigger slaves and communicates whether it is a master or a slave, along with its master ID (if it is a slave). This exchange of information is necessary to implement the following phase of gateway selection for obtaining a connected scatternet (see Section 4.3).

Notice that the outermost else is executed only by a slave node, since once it has paged all its neighbors, a master has a complete knowledge of its neighbors role and of the ID of their master and thus it can quit the execution of this phase of the protocol.

Implementation in the Bluetooth Technology. The protocol operations of this phase all rely on the standard Bluetooth paging procedures. However, the paging and paging scan procedure described above assume the possibility of exchanging additional information, namely, a device role and for slaves, the ID of their masters. These information cannot be included in the FHS packet which is the packet exchanged in the standard paging procedures.

Our proposal is to add an LMP protocol data unit (PDU), including fields to record the role of the sending device and the ID of its master, to easily exchange the information needed for scatternet formation while possibly avoiding a complete set up of the piconet.

Of course, whenever a slave joins a non-temporary piconet, a complete piconet set up has to be performed, after which the slave is put in park mode to allow it to proceed with the protocol operation (e.g., performing paging itself).

4.3 Configuring BlueStars

The purpose of the third phase of our protocol is to interconnect neighboring BlueStars by selecting inter-piconet gateway devices so that the resulting scatternet, a *BlueConstellation*, is connected whenever physically possible. The main task accomplished by this phase of the protocol is gateway selection and interconnection.

Two masters are said to be *neighboring masters* (*mNeighbors*, for short) if they are at most three hops away, i.e., if the shortest path between them is either a two-hops path (there is only one slave between the two masters) or a three-hops path (there are two slaves).

A master is said to be an *init master*, or simply an *iMaster*, if it has the biggest weight among all its mNeighbors. Therefore, the set of masters that results from the BlueStars formation phase is partitioned into two sets, the iMasters and the non-iMasters devices.

The connectivity of the scatternet is guaranteed by a result, first proven in [31], that states that given the piconets resulting from the BlueStars formation phase, a BlueConstellation—a connected BT scatternet—is guaranteed to arise if each master establishes multihop connections to all its mNeighbors. These connections are all needed to ensure that the resulting scatternet is connected, in the sense that if any of them is missing the scatternet may be not connected.

This result provides us with a criterion for selecting gateways that ensures the connectivity of the resulting scatternet: all and only the slaves in the two and three-hops paths between two masters will be gateways. If there is more than one gateway device between the same two masters they might decide to keep only one gateway between them, or to maintain multiple gateways between them.

Upon completion of the previous phase of the protocol a master v is aware of all its mNeighbors. It directly knows all its neighboring slaves which in turn are aware of (and can communicate to the master v) the ID of their master and of the master of their one-hop slave neighbors.

Establishment of a connected scatternet. We are finally able to establish all the connections and the needed new piconets for obtaining a BlueConstellation, i.e., a connected scatternet.

This phase is initiated by all masters v by executing the following procedure.

```
mInitOperations() {
if (for each mNeighbor u: myWeight > uWeight) {
    myRole = 'iMaster';
    instruct all gateway slaves about which neighbors to page;
    go to page mode;
    page all the slaves which belong to a different piconet
        and have been selected as interconnecting devices;
    exit the execution of this phase of the protocol; }
else {
    tell all gateway slaves to bigger mNeighbors
        to go to paging scan mode;
    if (there are bigger mNeighbors' slaves in my neighborhood
        which will interconnect the two piconets)
            go to page scan mode;
    tell all gateway to smaller mNeighbors to go to paging mode
```

```
        when the links to bigger mNeighbors are established;
    if (there are smaller mNeighbors' slaves in my neighborhood
        which will interconnect the two piconets)
        go to page mode when the links to bigger mNeighbors are up; }
}
```

Every master v starts by checking whether it is an iMaster or not. If it is an iMaster, then it instructs each of its gateway slaves to go into page mode and to page (if any):

- Its two-hop mNeighbors. In this case, as soon as v's slave has become the master of an mNeighbor u, they perform a switch of roles, as described in the BT specification, so that v's slave become also a slave in u's piconet. In this case, no new piconet is formed and the slave in between u and v is now a slave in both their piconets, as desirable.
- The slaves of its three-hop mNeighbors (that are two-hops away from v). In this case v's slave becomes also a master of a piconet whose slaves are also slaves to the three-hop mNeighbors, i.e., a new piconet is created to be the *trait d'union* between the two masters.

The iMaster v itself can then go into paging mode to recruit into its piconet some of those neighboring slaves (if any) that joined some other piconets, so that these slaves can be the gateway to their original masters.

Notice that, given the knowledge that every master has about its "mNeighborhood," an iMaster v instructs each of its gateway slaves about exactly who to page, and the resulting new piconet composition. If, for instance, a slave is gateway to multiple piconets, iMaster v knows exactly to which of the neighboring piconet its slave is going to be also a slave, and if it has to be master of a piconet that can have, in turn, multiple slaves.

When the gateway slaves of a non-iMaster device v have set up proper connections toward bigger mNeighbors, they will go into page mode and page those of its two-hop mNeighbors and of the the slaves of its three-hop mNeighbors with which they have been requested by v to establish a connection.

Implementation in the Bluetooth Technology. The mechanism described above can be easily implemented by means of the BT standard procedures for parking and unparking devices, and those for link establishment. In particular, upon completion of the second phase of the protocol, a slave asks its master to be unparked. The master will then proceed activating (unparking) different groups of slaves, and collecting from them all the information required for configuring the BlueConstellation. Based on this information, the master will then make a decision on which links to establish to connect with its mNeighbors, and will unpark the gateways in groups of seven to inform them of the piconets to which they are gateway. Each gateway will then run the distributed procedure for interconnecting neighboring piconets described in the previous section, at the end of which it will issue to the master a request for being unparked in order to communicate the list of links successfully established.

5 Conclusions

In this paper we have described some issues and solutions proposed for ad hoc networking. In particularly, we have illustrated leading MAC protocols, clustering protocols and we have shown how these protocols can be applied to networks of Bluetooth devices for the formation of Bluetooth scatternet.

References

1. E. M. Royer and C.-K. Toh. A review of current routing protocols for ad hoc mobile wireless networks. *IEEE Personal Communications*, 6(2):46–55, April 1999.
2. S. Giordano, I. Stojmenovic, and L. Blazevic. Position based routing algorithms for ad hoc networks: A taxonomy. http://www.site.uottawa.ca/ ivan/wireless.html, 2002.
3. A. D. Myers and S. Basagni. *Wireless Media Access Control*, chapter 6, pages 119–143. John Wiley and Sons, Inc., 2002. I. Stojmenovic, Ed.
4. F. A. Tobagi and L. Kleinrock. Packet switching in radio channels. ii. the hidden terminal problem in carrier sense multiple-access and the busy-tone solution. *IEEE Transactions on Communications*, COM-23(12):1417–1433, December 1975.
5. C. Wu and V. O. K. Li. Receiver-initiated busy tone multiple access in packet radio networks. *ACM Computer Communication Review*, 17(5):336–342, August 1987.
6. A. Gummalla and J. Limb. Wireless collision detect (WCD): Multiple access with receiver initiated feedback and carrier detect signal. In *Proceedings of the IEEE ICC 2000*, volume 1, pages 397–401, New Orleans, LA, June 2000.
7. P. Karn. Maca—a new channel access protocol for packet radio. In *Proceedings of ARRL/CRRL Amateur Radio 9th Computer Networking Conference*, pages 134–140, 1990.
8. V. Bharghavan and al. MACAW: A media access protocol for wireless LANs. *ACM Computer Communication Review*, 24(4):212–225, October 1994.
9. C. Lin and M. Gerla. Real-time support in multihop wireless networks. *ACM/Baltzer Wireless Networks*, 5(2):125–135, 1999.
10. F. Talucci and M. Gerla. MACA-BI (MACA by invitation): A wireless MAC protocol for high speed ad hoc networking. In *Proceedings of IEEE ICUPC'97*, volume 2, pages 913–917, San Diego, CA, October 1997.
11. I. Chlamtac and A. Faragó. Making transmission schedule immune to topology changes in multi-hop packet radio networks. *IEEE/ACM Transactions on Networking*, 2(1):23–29, February 1994.
12. R. Lidl. *Introduction to Finite Fields and Their Applications*. Cambridge University Press, 1994.
13. S. Basagni and D. Bruschi. A logarithmic lower bound for time-spread multiple-access (TSMA) protocols. *ACM/Kluwer Wireless Networks*, 6(2):161–163, May 2000.
14. S. Basagni, D. Bruschi, and I. Chlamtac. A mobility transparent deterministic broadcast mechanism for ad hoc networks. *ACM/IEEE Transactions on Networking*, 7(6):799–807, December 1999.
15. S. Basagni, A. D. Myers, and V. R. Syrotiuk. Mobility-independent flooding for real-time, multimedia applications in ad hoc networks. In *Proceedings of 1999 IEEE Emerging Technologies Symposium on Wireless Communications & Systems*, Richardson, TX, April 12–13 1999.

16. D. Bruschi and M. Del Pinto. Lower bounds for the broadcast problem in mobile radio networks. *Distributed Computing*, 10(3):129–135, April 1997.
17. I. Cidon and M. Sidi. Distributed assignment algorithms for multihop packet radio networks. *IEEE Transactions on Computers*, 38(10):1353–1361, October 1989.
18. C. Zhu and S. M. Corson. A five-phase reservation protocol (FPRP) for mobile ad hoc networks. In *Proceedings of IEEE Infocom'98*, volume 1, pages 322–331, San Francisco, CA, March/April 1998.
19. A. Ephremides and T. V. Truong. Scheduling broadcasts in multihop radio networks. *IEEE Transactions on Communications*, 38(4):456–460, April 1990.
20. S. Basagni. Distributed clustering for ad hoc networks. In A. Y. Zomaya, D. F. Hsu, O. Ibarra, S. Origuchi, D. Nassimi, and M. Palis, editors, *Proceedings of the 1999 International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN'99)*, pages 310–315, Perth/Fremantle, Australia, June 23–25 1999. IEEE Computer Society.
21. S. Basagni. Distributed and mobility-adaptive clustering for multimedia support in multi-hop wireless networks. In *Proceedings of the IEEE 50th International Vehicular Technology Conference, VTC 1999-Fall*, volume 2, pages 889–893, Amsterdam, The Netherlands, September 19–22 1999.
22. M. Chatterjee, S. K. Das, and D. Turgut. An on-demand weighted clustering algorithm (WCA) for ad hoc networks. In *Proceedings of IEEE Globecom 2000*, San Francisco, CA, November 27–December 1 2000. To appear.
23. S. Basagni. A note on causal trees and their applications to CCS. *International Journal of Computer Mathematics*, 71:137–159, April 1999.
24. S. Basagni and C. Petrioli. Scatternet formation protocols. Technical report, Università di Roma "La Sapienza", Roma, Italy, September 2001.
25. S. Basagni and C. Petrioli. A scatternet formation protocol for ad hoc networks of Bluetooth devices. In *Proceedings of the IEEE Semiannual Vehicular Technology Conference, VTC Spring 2002*, Birmingham, AL, May 6–9 2002.
26. http://www.bluetooth.com. *Specification of the Bluetooth System, Volume 1, Core.* Version 1.1, February 22 2001.
27. T. Salonidis, P. Bhagwat, L. Tassiulas, and R. LaMaire. Distributed topology construction of Bluetooth personal area networks. In *Proceedings of the IEEE Infocom 2001*, pages 1577–1586, Anchorage, AK, April 22–26 2001.
28. G. Záruba, S. Basagni, and I. Chlamtac. Bluetrees—Scatternet formation to enable Bluetooth-based personal area networks. In *Proceedings of the IEEE International Conference on Communications, ICC 2001*, Helsinki, Finland, June 11–14 2001.
29. C. Law, A. K. Mehta, and K.-Y. Siu. Performance of a new Bluetooth scatternet formation protocol. In *Proceedings of the 2001 ACM International Symposium on Mobile Ad Hoc Networking & Computing, MobiHoc 2001*, pages 183–192, Long Beach, CA, 4–5 October 2001.
30. S. Basagni, R. Bruno, and C. Petrioli. Device discovery in Bluetooth networks: A scatternet perspective. In *Proceedings of the Second IFIP-TC6 Networking Conference, Networking 2002*, Pisa, Italy, May 19–24 2002.
31. I. Chlamtac and A. Faragó. A new approach to the design and analysis of peer-to-peer mobile networks. *Wireless Networks*, 5(3):149–156, May 1999.