LiT MAC: Addressing The Challenges of Effective Voice Communication in a Low Cost, Low Power Wireless Mesh Network

Vijay Gabale, Bhaskaran Raman, Kameswari Chebrolu, and Purushottam Kulkarni
Dept. of Computer Science, IIT Bombay
Mumbai, Maharashtra, India
vijaygabale@cse.iitb.ac.in, br@cse.iitb.ac.in, chebrolu@cse.iitb.ac.in,
puru@cse.iitb.ac.in

ABSTRACT

In this work, we consider the goal of enabling a local voice communication system, within a village, using a **low cost** and **low power** wireless mesh network. The design of an appropriate MAC is a major challenge in this context. Towards this goal, we present LiT: a **full-fledged** TDMA-based MAC protocol for real-time applications over such networks. We showcase the practicality of such a system through implementation-based evaluation of LiT on an inexpensive, low power 802.15.4 platform.

While there is plentiful literature on the use of TDMA for wireless mesh networks, a practical multi-hop TDMA system remains elusive. In this regard, LiT addresses several practical concerns. It has built-in support for time-synchronization, has a flexible interface with routing, and has a dynamic TDMA schedule dissemination mechanism. LiT is multi-channel capable and is centrally controlled. It achieves robustness in the face of wireless packet errors by making extensive use of soft-state mechanisms. With appropriate duty cycling, LiT can make nodes run for several weeks without power off the grid. Evaluation of LiT on outdoor testbed shows quick flow setup (latency $< 1s$), low packet delay ($< 240ms$) and negligible data path jitter (median $0ms$), essential for real-time applications.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Centralized networks, Wireless communication*

General Terms

Design, Experimentation

Keywords

802.15.4, TDMA-based multi-hop MAC, Voice applications

ACM DEV'10, December 17–18, 2010, London, United Kingdom.

1. INTRODUCTION

In this work, we consider the goal of enabling a local voice communication system in a village-like setting, in developing regions. The design of any such a commuication system poses unique challenges in terms of **cost** and **power** optimizations [4]. In this regard, *cellular technology* (GSM) is quite costly and power hungry (high base station cost ($10-100K), high power requirements (0.5-1kW)) requiring huge power backup and cooling systems. Specifically in India, this is evident from the fact that, the rural teledensity at the end of December 2009 was just 20%, even though 70% of 1.2 billion population resides in villages [1]. Hence, a question of importance is, how low can one *scale down* cost and power and still have an effective communication system? Recently, 802.11 WiFi is being explored as a popular choice for providing connectivity to rural areas. However, a comparison (Tab. 1) of 802.11 with 802.15.4 shows that 802.15.4 fits well to build a low-cost, low-power network.

In this respect, authors in [15] envision Lo^3: an 802.15.4-based **lo**w-cost, **lo**w-power, **lo**cal-voice multi-hop mesh network to provide voice applications in villages in developing regions. Such a network can enable applications like two-way interactive voice, stored voice messaging and community alerts. For rural settings in developing regions, where literacy levels are low, voice-based applications gain special importance in comparison to traditional web applications [5]. In this context, it is well known that CSMA-based multi-hop MAC gives poor throughput [14] and results in high delay and jitter, unsuitable for real-time applications. (Sec. 6.2 explicitly compares our MAC with CSMA to corroborate this.) Hence there is significant literature which has considered a TDMA-based approach. Past work has considered both TDMA scheduling algorithms ([17] and references thereof) as well as protocol design (e.g. [13, 19, 11]). However practical multi-hop TDMA systems are few and far between.

A multi-hop TDMA MAC, if it has to effectively support real-time applications as envisioned in Lo^3, has to address the following challenges. It has to: (a) achieve and maintain multi-hop time-synchronization, the basis for any TDMA mechanism, (b) disseminate and maintain TDMA schedules (output of scheduling algorithm), (c) interface with the routing mechanism (d) provide real-time end-to-end flow setup and flow maintenance mechanisms, (e) do all of the above well, despite wireless losses. To our knowledge, the above questions have not been addressed *comprehensively* by prior

Parameter	802.11	802.15.4
Cost of radio	~$5-10	~$5-10
Cost of off-the-shelf platform	~$50-100	~$30-40
Power consumption of radio	100-200mW	10-20mW
Power consumption of platform	6-10W	50-100mW
Duty cycling support	Poor	Very good
Radio capacity	1-54Mbps	250Kbps
No. of orthogonal channels	3	16

Table 1: Comparing 802.15.4 and 802.11

work on multi-hop TDMA MAC protocols. More importantly, the MAC should be light-weight to be implementable on low-cost platforms and should have intrinsic support for power savings without compromising on the performance of real-time applications.

To address these challenges, we present LiT, a **li**ght-weight (a) **T**DMA-based, (b) centralized (c) multi-channel (d) connection oriented MAC protocol, for operation in a wireless multi-hop mesh network. In this regard, this paper makes the following contributions.

• LiT has in-built support for time-sync over multi-hop networks. Our mechanism is simple, yet effective: we achieve and maintain low clock synchronization error of $\simeq 60\mu s \simeq 2$ clock-ticks per hop on an 802.15.4 platform.

• LiT makes extensive use of *soft-state mechanisms* to maintain schedule-state, network-state and flow-state. This achieves robustness in the face of wireless errors. The use of soft-state for such MAC level connection management is novel. Our evaluations show that the soft-state overheads are minimal, even in the presence of wireless packet losses. Simple time-sync mechanism coupled with soft-state makes the working of TDMA multi-hop MAC light-weight.

• Our implementation on 802.15.4-based Tmote platforms serves as a stringent test case for the MAC design since the radio is impoverished (250Kbps). Evaluation on an outdoor testbed shows that our prototype can handle dynamic flow setup (latency $< 1s$), and real-time flows effectively (median data path jitter of $\simeq 0ms$). This indicates that our MAC is suitable for real-time applications for Lo^3.

• We evaluate the scalability aspects of LiT through a custom-built simulator and show that control overheads (e.g schedule dissemination) are not bottlenecks for the MAC operation.

• Our simulation further shows that, with aggregate call duration of 2 hours per day and 20% duty cycling due to MAC, a node can effectively operate for 84 days with 4.5AH, 12 V battery.

To our knowledge, LiT is the first system to *implement* and extensively *evaluate* a multi-hop TDMA system for real-time applications. The rest of the paper is organized as follows. The next section (Sec. 2) compares prior work in MAC design for multi-hop mesh networks. Sec. 3 describes usage scenarios in Lo^3 system. Sec. 4 details the design of LiT. We extensively evaluate the LiT on an 802.15.4 prototype in Sec. 5, while Sec. 6 presents a simulation study. We brief future work in Sec. 7 and conclude the paper in Sec. 8.

2. RELATED WORK

Our design of multi-hop TDMA MAC, LiT, is an integral part of Lo^3 system [15]. Although MAC design for multi-hop wireless networks has received considerable attention so far, there are several differences between LiT MAC and these. A glance at Tab. 2 gives a comparison of LiT with four prominent types of MAC protocols in the literature.

Challenge	Sensor n/w MAC protocols (TRAMA, LMAC, SMAC etc.)	Multi-channel MAC protocols	RT-Link	WiMAX	***LiT***
TDMA schedule dissemination	Not considered	Not considered	Not considered	Has support	Supported, evaluated
Handling wireless errors	Yes	Yes	No explicit discussion	No explicit discussion	Using soft-state
Real-time support	No	None evaluated	Has support	Has support	Supported, evaluated
Implementation based evaluation	A few (e.g SMAC)	None for multi-hop	Yes	None yet for multi-hop	On 802.15.4 (& 802.11)

Table 2: Comparison of prior Work

LiT's significant contributions are: (1) the consideration of built-in time synchronization, (2) the use of a soft-state framework for dynamic schedule dissemination and other state maintenance; this handles wireless losses gracefully and without undue complexity, and (3) the consideration of real-time applications. Such a comprehensive consideration is absent in the prior work. For instance, TDMA MAC protocols proposed for sensor networks, such as TRAMA [13] or LMAC [19], do not consider real-time applications or other heavy network traffic. RT-Link [16] considers real-time applications, but employs an out of band time-sync mechanism and assumes support for schedule dissemination. The various multi-channel MAC protocols in [11] do not consider support for TDMA schedule dissemination, nor do they explicitly evaluate real-time application performance.

Comparing with WiMAX mesh standard [2], LiT has several crucial differences. First, WiMAX does not clearly specify what happens in the case of wireless errors in control packets. Second, state maintenance mechanisms are not discussed in-depth in WiMAX. For these two aspects, LiT has soft-state based mechanisms which function gracefully despite wireless losses. Third, WiMAX has a complex distributed election process for control slot selection, which is absent in LiT. Fourth, WiMAX allows data transmissions only along the (single) tree rooted at the central node. LiT on the other hand allows efficient data paths, and in fact there is flexibility of having *any* data path.

Finally, since currently there are no prototype-based evaluations of WiMAX mesh mode operation, we believe that lessons from our LiT prototype study would be valuable for future WiMAX systems too.

3. THE LO^3 SYSTEM

As envisioned in [15], the Lo^3 system can be viewed as a PABX (private automatic branch exchange), for use within a village. Fig. 1 depicts the Lo^3 mesh architecture and the usage model. There are two types of nodes in Lo^3: *infrastructure nodes* acting as relay nodes for data communication, and *client nodes* acting as data originators or terminators. The infrastructure nodes may be deployed on rooftops (5-10m height) and as reported in [6], a range of over 400m can be achieved for 0dBm transmit power and 8dBi omni-directional antenna mounted at a height of less than 3m. Given this, an area of about 3km diameter can be (hotspot) covered by a mesh of about 20-30 infrastructure nodes. The

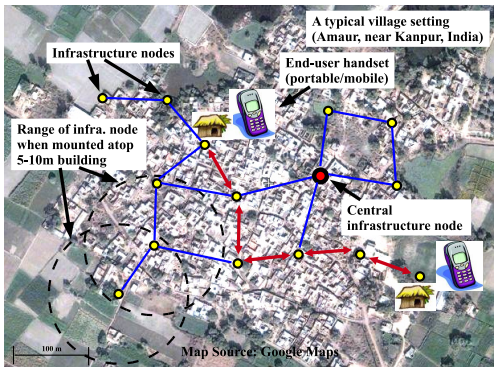

Figure 1: Lo^3 Architecture

use of mesh network helps reduce the cost of the network and the authors envision the overall infrastructure cost to be about U.S.$1-2K. In terms of usage volume, Lo^3 intends to support a user population of a 150-200, among a total village population of 1000+ with 5-10 simultaneous calls on an average. As the system usage grows, capacity can be increased by addition of more infrastructure nodes. Note that, *cellular technologies* do not *scale down* in terms of cost and power for operation on such a lower scale.

In the context of Lo^3, there is no known **full-fledged** design or implementation of a multi-hop TDMA MAC, especially tuned to support real-time applications and power savings, in wireless mesh networks. Thus, to enable effective support for such a set of applications, we have devised the LiT MAC protocol.

4. DESIGN OF THE LIT MAC

What is challenging in the design of LiT? We have to support real-time applications, in the face of several non-trivial requirements, as given below. (1) For TDMA-based operation, time-synchronization support must be built-into the MAC. (2) The MAC should support *dynamic routing*; i.e. nodes joining and leaving the system. (3) Similarly, it should support *dynamic flow setup* and *tear-down*. (4) The MAC must be flexible enough to allow the *data flow* for each connection to happen through an *"efficient" path*. The MAC must be independent of the scheduling algorithm itself. (5) The MAC mechanisms must be *robust to wireless errors*; any message loss should be handled gracefully, while at the same time keeping the protocol simple and low overhead. (6) Nodes should be able to *duty-cycle*; only a necessary set of nodes should be awake at any time.

We now describe how LiT is designed to address the above challenges. We first describe the overall design choices (Sec. 4.1), followed by the detailed design (Sec. 4.2).

4.1 Overall design choices

There are four important design choices in LiT. This *set* of design choices is subtle and this is what enables a practical implementation of LiT.

(1) TDMA versus CSMA: The default CSMA protocol in much of prior work allows for an easy distributed implementation, but it is known to suffer from poor performance in multi-hop networks [14], especially for real-time applications [16]. Self-interference, i.e. simultaneous transmissions across hops of a path, results in capacity reduction and delay unpredictability. In this respect, TDMA is known to perform better. In fact, our evaluation for 802.15.4 setting (Sec. 6.2) shows that CSMA performance degrades rapidly when number of bi-directional voice calls increase beyond just one call. While TDMA is suitable for operation in licensed spectrum (as WiMAX does), our evaluation shows that even in an interference-prone campus environment, in an unlicensed band, LiT performs well. Importantly, a TDMA-based approach also provides good support for duty-cycling and power savings in the network nodes. □

(2) Centralized versus distributed: For effective real-time application support, especially for features such as delay-aware scheduling or call-admission control, a centralized approach is intuitively better. While we do not rule out a distributed approach, we have currently used a centralized architecture for LiT; which gives significant leverage in effectively addressing the challenges listed earlier. For instance, the centralized coordination of time-slots and multiple channels lends itself to a simpler design, as compared to a distributed approach, in a multi-hop network. Although centralized approach is frowned upon for lack of scalability, we show in our evaluation of LiT; that the control overheads are not scaling bottlenecks in practice. □

(3) Multi-channel versus single-channel: 802.11b/g has 3 non-overlapping channels of operation, and 802.15.4 has 16 different channels in the 2.4GHz ISM band. Clearly, the use of multiple channels has better potential throughput, if the channel coordination can be done efficiently. Our centralized-TDMA-based approach eases the issue of multi-channel coordination significantly. The same slotting mechanism used by the TDMA frame is also used as the granularity of any channel switching (i.e. slot level switching). Channel switching does add a small but noticeable overhead, but the benefits of multi-channel operation far outweigh this. □

(4) Connection-oriented versus connection-less: A connection-oriented MAC means that a higher layer can use it only after a connection formation phase. Most MAC protocols are connection-less, with the notable exceptions of Bluetooth and WiMAX [2]. In LiT, the MAC provides a multi-hop connection-oriented interface to the layer above. This fits in well with our choice of using a TDMA-based multi-channel approach. With central control, the connection formation phase is used to specify the time-slot and channel of operation of each node in a data flow path. □

4.2 The LiT MAC mechanisms

We now discuss various terminologies and functions of LiT that address the MAC's requirements. In addressing the requirements, the MAC makes extensive use of *soft-state*. *Soft-state* is a technique for state maintenance where periodic messages are used to create as well as maintain state. The state is timed-out if not refreshed. In such a scheme, message loss, node/link failure, and other such problems require no special mechanisms: the periodic refresh, and the time-out are sufficient to handle all situations correctly. In the description below of the various aspects of the MAC, we explain where we use soft-state and the trade-offs involved.

Frame structure

Given our overall choice of TDMA-based operation, we have a time-slotted system. A *time-slot* (or simply, a *slot*) is a unit of resource allocation. In the context of LiT, central control implies that a central node, which we call the *root* node, decides: (a) who transmits in what slot, and to whom, and

(b) which channel is to be used for this transmission. A *frame* is a repeating pattern of slots. We have three kinds of slots: *control*, *contention*, and *data* slots. This is shown in Fig. 2.

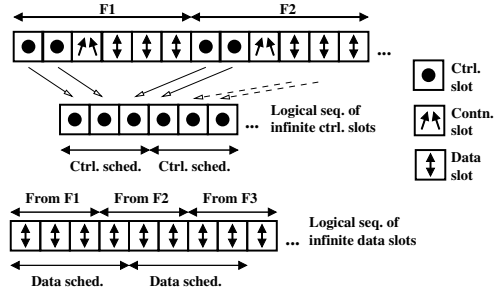

Figure 2: Frame structure

Control slots: We designate a *tree* rooted at the *root* node to transmit packets in control slots. These are slots *assigned* by the root, for control packets to flow *down* the tree. There are three kinds of information which flow down the tree: (1) time-synchronization information, (2) information about the tree structure, so that each node is aware of its children and parent on the tree, and (3) the data schedule itself. □

Contention slots: These slots are for control packets to flow *up* the tree. The packets are forwarded from child to parent so that it eventually reaches root. They are mainly used for (a) node join requests, (b) new flow requests, and (c) any routing status messages. Such use is intended to be relatively infrequent and the transmissions may not be collision-free. For contention slot transmissions, we do not use carrier-sense or back-off mechanisms, since such mechanisms would be difficult to accommodate within a limited sized-slot[1]. Instead, each node simply transmits probabilistically within its contention slot. Within a contention slot, we have time included for an immediate ACK. If no ACK is received, then the node retries probabilistically in the next contention slot until it gives up after threshold number of retries. □

Data slots: These are used for the actual data flow; they are allocated for each link along the flow from source to destination. Note that, the path taken by the flow is the concern of the routing module and the MAC is independent of it. □

Channel usage

The control slot is always in a pre-decided *default* channel. This acts as a fail-safe fall-back mechanism in case anything goes wrong: such as a node going out of synchrony, or a parent node failure, etc. The contention slots are also used in the default channel. That is, all nodes in the network potentially contend for it. Any channel reuse happens naturally in "far-away" parts of the network. The data slots use non-default channels to allow efficient data transport. Since we expect the bulk of the slots to be data slots, this reflects in overall efficient operation.

The control schedule

The control information flows from the root, down the tree, along the tree edges. An important point to note is that we do not intend that the information flow from the root to all the nodes be completed within a frame. Quite to the contrary, the nodes in the network take turns in transmitting in their respective assigned control slots. We call the order in which the nodes take turns in using the control slots, as the *control schedule*. The notion of the *control schedule* is easily understood by visualizing a logical, infinite stream of control slots, from successive frames as shown in Fig. 2. The root node decides the control schedule in terms of a routing tree, represented as an array of parent-child relation. The order in which nodes appear in this array is implicitly taken to be the control schedule.

In LiT, all *infrastructure* nodes (nodes which serve as intermediate hop, usually static) transmit control packets, including those which do not have children. This is necessary, since even nodes which do not currently have a child in the tree, may have new nodes join them as children. However, the *client* nodes (which can be mobile), which will never serve as intermediate hop, need not transmit control packets.

The data schedule

The data schedule computation itself is the concern of the scheduler module, and is independent of the LiT MAC. The data schedule specifies the usage of the data slots. The MAC has a notion of a logical infinite sequence of data slots from consecutive frames. And the data schedule is specified as a repeating pattern in this sequence of data slots (Fig. 2).

A subtle aspect to note is that the above abstraction allows us to keep the following two parameters independent of one another: (a) the number of data slots in a frame (dictated by overhead of control slots and delay due to long frame lenght), and (b) the length of the data schedule (dictated by number of flows, their paths and interference graph etc).

Rx	Tx	Chnl.	e2e dst	e2e src	flow id	slot(s) spec.

Figure 3: The data scheduling element

The data schedule is specified in terms of *scheduling elements*. Each scheduling element (Fig. 3) consists of (a) the slot(s) of operation (slot spec.), (b) the transmitting node, the receiving node, and the channel of operation, and (c) the end-to-end destination, the end-to-end source, and a the flow-id; these three uniquely identify a flow, and are present in the header of all data packets. Note that the flow-id by itself is not globally unique; this is so that a flow-id can be chosen by the flow's source, while originating the flow request.

Soft-state for control and data schedules

How should the control and data schedule be disseminated and maintained? Should we disseminate the schedule only when there is an update? In a network of several nodes over several hops, it is difficult to design an acknowledgement mechanism for the root to learn that *all* nodes have indeed learnt the schedule (control or data). This is especially true in a wireless network where losses can happen frequently.

To tackle this, in LiT, the control and data schedule information is maintained based on soft-state. That is, the root and infrastructure nodes, broadcast schedule periodically even when the schedule does not change. Each node

[1]In case of 802.15.4 radio, it takes $128\mu s$ to sample the channel. Further, accomodating a timeout and a back-off mechanism in a $6ms$ slot is inefficient.

has a time-out associated with the schedule it has learnt. After time-out, nodes stop following the schedule, become orphan assuming the parent is down and start network-join procedure. This timeout is set to be much larger than the time-period with which the nodes broadcast the schedule information. Although this approach has the overhead of having to send the information periodically, we consciously avoided a non-soft-state based approach due to the fundamental problem of designing the appropriate acknowledgement mechanism.

Soft-state for network management

How should the network connectivity be managed in a centralized architecture? In LiT, the connectivity information of nodes is maintained as a soft-state at the root by using periodic topology updates. This implies that if the root does not receive a topology update from a particular node within timeout, the root removes that node from the tree and flushes it's network state. Thus, the usage of topology updates at the root is two-fold: (1) periodically refresh the routing state of the node (2) periodically refresh the neighborhood information of the node for the network and the interference graph data structures. A topology update can also be used to convey node-specific parameters like remaining battery-power for energy-aware routing.

Soft-state for flow management

What should be the bandwidth request mechanism to maintain an ongoing flow? Whether the root should poll the nodes or the nodes should periodically renew the flows? In LiT, we use soft-state for flow maintenance too.

When a node wants to start a flow (e.g. voice call request), it sends a flow request to the root. Forwarded along the tree edges, when the root receives this request, it consults the scheduling module to see if the flow can be accommodated. The modified data schedule, if any, serves as an implicit ACK for the flow request originator. The flow request has no explicit ACK from the root. In fact, if the flow were rejected, there is no explicit negative-ACK either!

In our soft-state mechanism, the flow-requests are sent periodically *even after* the flow is established. These serve the purpose of *renewing* the request at the root. The root maintains a flow only so long as it keeps getting flow-request renewals. If it stops receiving such renewals, the root times out the flow, and deallocates the relevant data slots in the subsequent data schedule. As an optimization, the original flow-request originator may send an explicit flow termination (using contention slots). But we have no special mechanism to handle the loss of the flow termination message: we simply fall back on the time-out at the root.

As compared polling mechanism, the above soft-state mechanism makes flow maintenance very easy, but at the cost of the overhead of sending the periodic flow request renewals. There is also the additional overhead of extra time for which a flow's data slots remain allocated, if the flow termination were to be lost. However our evaluation shows that the soft-state mechanism is not a performance bottleneck.

Temporary schedule inconsistencies

During the time of multiple frame durations it takes for a schedule (control or data) to reach from the root to the remaining nodes, over several hops, the network is in a temporarily inconsistent state. That is, some nodes are following the old schedule while some others have learnt the new schedule. Inconsistency in the data schedule may potentially cause unintended collisions (and packet losses) in the data slots. To avoid such inconsistencies, the data schedule has a *valid-from* field (much like the use of sequence numbers in DSDV) that specifies a time instant from which the new schedule is valid. The *valid-from* value is an expectation of the time taken for the new schedule to reach all the infrastructure nodes.

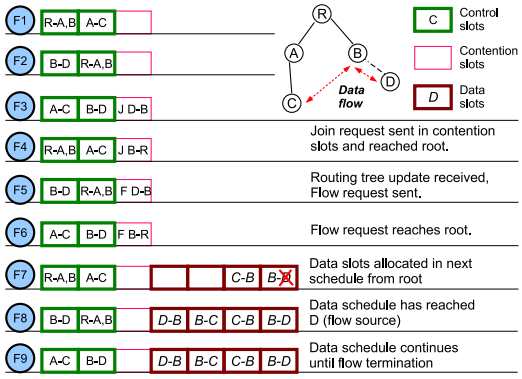

Figure 4: LiT MAC operation: an illustration

We could still have inconsistencies if there were any message losses in propagating the schedule information. But because of periodic schedule dissemination, such a inconsistent state lasts only for a few frame durations. For the worst case scenario (e.g repeated collisions), we fall back on oursoft-state mechanism where the nodes simply timeout their current control schedule if they do not continue receiving periodic renewals of that schedule. Eventually all nodes learn the new and correct schedule.

4.3 Example of LiT MAC operation

We now present a simple example to illustrate how the LiT MAC operates. The scenario we consider is shown in Fig. 4. The framing structure we have assumed is: 2 control slots, 1 contention slot, and 4 data slots in each frame. R is the root node, and we start our example after nodes A, B, and C have joined the network. The control schedule at this point is $[R, A, B]$. In this example, we have assumed that C is a client node, and hence does not transmit control packets. Note how the control schedule operates independent of the frame boundaries; that is, in some frames R transmits in the first control slots, while in others it transmits in the second control slot. The data schedule is initially empty.

D is a node which wishes to join the network. Right after boot up, it enters the *orphan* state where it is not-synchronized with the network and does not have parent in the tree. It waits until it hears control packets from some node; it happens to hear from B. From the control packet, D knows the time-synchronization information as well as the slotting structure. It then uses B to send a join request (marked J) toward R. We also assume that D is a client node. Thus, after getting the join request from D, R hence does not have to change the control schedule. If D had been an infrastructure node, R would have changed the control schedule to include D. After joining, it goes into the *joined* state. D then originates a flow request (marked F). This flow is a bi-directional flow between D and C. In the example, we have assumed that the chosen path for the data flow

is $D - B - C$. After getting the flow request, R computes an appropriate data schedule. In this case, four slots are assigned to each of the links, in both directions.

The data schedule reaches A, B, and C in frame-7, while it reaches D only in frame-8. D then goes to the *called* state and once the call is teared down, it goes back to the *joined* state. In frame-7, D does not yet schedule and hence cannot transmit or receive data packets. This is indicated with a cross in the last data slot of frame-7. From frame-8 onwards, the data schedule continues until tear-down (or timeout). In the *joined* and the *called* states, nodes send periodic topology updates and particularly, in the *called* state, call originator sends flow-renewal requests in contention slots (not shown in the Fig. 4 for clarity).

4.4 Further Details of LiT

Centralized routing : LiT's approach of central control for the MAC leads naturally to a centralized routing mechanism. As the part of the join request or topology update, a node sends its neighborhood information to the root, which is then used by the routing module at the root to decide the node's forwarding tables. The keen reader may have noted that our data scheduling elements in fact carry *implicit forwarding table* information. The scheduling element specifies the next-hop for each flow: this is the "rx" (receiver) field in Fig. 3. The periodic topology updates in the design of the LiT aid the centralized scheduler to maintain network connectivity graph and to make scheduling decisions. □

Control and data schedule fragmentation: During schedule dissemination, it could happen that the amount of information to be sent exceeds what can fit in these set of slots. In such a scenario, the information is simply fragmented across the control slots in successive control schedule rounds. Such fragmentation can happen for whatever is conveyed in the control packets: this could be the control schedule information itself, or it could be the data schedule information. □

Hop-by-hop ACKs: For contention slots, we enable hop-by-hop ACKs as a mechanism for improving efficiency in the presence of wireless channel errors. To allow this, we ensure that the slots used by a node have enough time to include an immediate ACK from the receiver. It is worth noting that the MAC *does not* have any special slots to accommodate potential retransmissions. Such an approach would waste slots in the common case when there is no wireless error. Instead, any retransmission is attempted only on the *next available* transmission opportunity; i.e. the next contention slot for contention slot loss, or the next designated data slot for data slot losses. [2] □

Fault tolerance: In a centralized approach, if the central node fails, the network cannot operate. We can however overcome this by having a pre-designated root node backup, or even a dynamic root selection algorithm, like that in Ethernet switched networks. This is part of future work. □

4.5 A scheduler for LiT

As mentioned earlier, LiT itself is independent of the scheduler module, and any centralized scheduler can be used. To evaluate LiT, we have implemented a simple greedy scheduler. This scheduler is custom-fit for voice applications, which we envision to be one of the important applications for LiT-MAC based networks.

The scheduler takes four different inputs: the connectivity graph, the interference graph, the data schedule for the current set of flows, and the new flow request. The path between the source and destination is computed by shortest-hop metric over the network connectivity graph. As output, it gives a schedule including the new, or it rejects the flow. The flow request is assumed to be bi-directional.

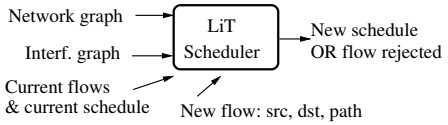

Figure 5: The LiT scheduler

The above scheduler works under the constraint that the schedule corresponding to the current set of flows is *not* modified. That is, their slot allocation does not change. Such an approach ensures that new flows do not disturb the old flows in any way; not even due to the introduction of any temporary inconsistency during the propagation of the new schedule. Our scheduler also simplifies things by considering the number of data slots in a frame to be the same as the data schedule length, although this does not have to be the case in a generic scheduler. For a path with l hops, $2l$ links have to be scheduled (each link in both directions). For each link, we consider the transmitter-receiver pair, and the scheduling involves two steps:

(1) Allocation of time-slot: Since an intermediate node needs to receive and transmit in each direction, it requires 4 slots per frame for single radio nodes. As pointed out in [8], the sequence in which links are scheduled decides the end-to-end delay, crucial for real-time applications. Hence, for a path A-B-C, considering the scheduling of slots for node B, we schedule A-B link before B-C (similarly for reverse direction). During scheduling, we check each of the slots in the data schedule to find one in which both the transmitter and the receiver node are free. If we cannot find such a common free slot, the scheduler reports failure and does not admit the flow.

(2) Allocation of channel: Once a time-slot is chosen, a channel has to be chosen. Now, at the link's transmitter, a set of channels are not allowed; the channels that would cause interference to a nearby, already admitted flow. Similarly, at the link's receiver, a set of channels are not allowed; the channels in which transmissions are happening for the already admitted flows in the interference neighborhood of the receiver. The scheduler seeks to find a common allowed channel between the transmitter and the receiver. If it cannot, it reports failure and rejects the flow.

Although simple, extensive simulations show that the above scheduler has good performance in practical settings. It is intuitive to see why this is the case, in the context of channel allocation in 802.15.4. Suppose that the degree of a node X in the interference graph is d_X. Then, in the above channel allocation step, at most d_T channels are disallowed at the transmitter T, and at most d_R channels are disallowed at the receiver R. So if $d_T + d_R < N_{Chnl}$ where N_{Chnl} is the number of available channels, then the channel allocation step will not fail. In practice, $N_{Chnl} = 16$ for 802.15.4, and is large enough to exceed $d_T + d_R$ in most cases.

4.6 Analysis of LiT MAC performance

[2]For real-time applications, having data slot retransmissions may be inefficient and is left as a policy decision to the higher layer.

We now examine LiT with respect to some important performance metrics. This serves as a basis for our prototype evaluation. Tab. 3 summarizes our notation.

Notation	Description
h	The max. node depth (from root)
N_s	Number of static (infrastructure) nodes
N_m	Number of mobile (client) nodes
S	Slot duration = $S_{used} + S_{guard}$
n_{ctl}	Num. control slots in a frame
n_{con}	Num. contention slots in a frame
n_{dat}	Num. data slots in a frame
F	Frame durn. = $(n_{ctl} + n_{con} + n_{dat}) \times S$
n_f	Number of flows in the system
h_f	Number of hops in a flow

Table 3: Notation for various LiT parameters

Flow-setup and node-join delay: These involve two steps: (S1) the request has to go up the tree, and (S2) the modified data or control schedule has to come down the tree. S1 involves a delay of h frames, assuming no collisions in the contention slots. And, assuming no wireless losses, and assuming that the data/control schedule involve no fragmentation, S2 involves $\frac{N_s}{n_{ctl}}$ frames in the worst case. So the worst-case flow-setup and node-join delays are: $(h + \frac{N_s}{n_{ctl}})F$.

Delay in the data path: In our scheduler (Sec. 4.5), the worst case data path delay can be bounded. In each direction, each intermediate node gets at least two slots: one each for reception and transmission. Thus a packet can traverse at least two hops in each data schedule round, even in the worst case. Under the special condition that the data schedule length is the same as the number of data slots in a frame, the above worse case data path delay would be $(h_f/2) \times F$.

Duty-cycling: By design, a node in LiT is required to be operational only during the control and contention slots, unless there is a data flow through that node. So we have a duty cycle of $(n_{ctl} + n_{con})/(n_{ctl} + n_{con} + n_{dat})$, with any additional requirement arising only due to data flow load.

The next three sections evaluate LiT extensively using prototype implementations as well as simulations.

5. PROTOTYPE-BASED EVALUATION

We have implemented a prototype of the LiT MAC on an 802.15.4 platform based on the CC2420 chip. We have chosen the Tmote Sky platform, which uses an MSP430 microcontroller. Our software platform is TinyOS v2.1.0. We use this 802.15.4 platform in the 2.4GHz frequency, which gives a PHY data rate of 250 Kbps. **Note:** In the Tmote Sky platform, one clock *tick* $\simeq 30.5\mu s$, since it uses a 32 KHz clock. We report many of our measurements in terms of these ticks.

5.1 Micro-benchmarks: Guard Time Determination

In any TDMA system, we need a guard time for each transmission. In LiT, we define *guard* time as the duration given as leeway prior to each transmission. There are several components which contribute to the guard time. (1) Time synchronization error: (1a) error right after synchronization, and (1b) further error due to drift between successive time-synchronization events. (2) Inaccuracy in timer fire: we may set a timer to fire at time t, but it fires at $t \pm \delta$. (3) Processing jitter: we expect an operation to complete within a certain time, but it could take longer. (4) Finally, the channel switching time.

Component	Measured value
Time-sync error	
(a) right after sync	± 1 tick, per hop
(b) drift error	< 1.5 ticks per sec
Inaccuracy in timer fire	1-2 ticks
Processing jitter	1-2 ticks
Channel switching time	$\simeq 10$ ticks

Table 4: Components of guard time

Table. 4 summarizes the above four values, as measured on our prototype. The time synchronization error values are consistent with those reported in [6, 18]. If we wish to support a network depth of say up to 5 hops (network diameter of 10 hops), then we can compute the guard time for this as follows. (1a) In the worst case, two adjacent nodes in the data path could both be 5 hops from the root, and the maximum synchronization error is $2 \times 5 = 10$ ticks. (1b) As we shall see shortly, the duration between two successive time-sync events is about 1-2 sec in practice; so the drift error is about 3 ticks. Adding the other components, we get a guard time of about 27 ticks.

5.2 Parameters to support real-time voice

To examine the parameter settings to support real-time voice, we consider the *G.723.1* codec. This codec needs 24 bytes to be transmitted every 30 ms. We choose a slot-duration of 6 ms by empirical measurements. With a guard time of 27 ticks, we have $S_{used} \simeq 5.18ms$, which can accommodate about 66 bytes at 250 Kbps along with system overheads. This means that we can easily accommodate 60 ms worth of codec data (i.e. 48 bytes) within a slot.

What should be the frame duration F? A large F is good for efficiency, but bad for the above computed delays. If we choose $F = 60$ ms, we have 10 slots/frame. A voice flow will generate 48 bytes in each direction every frame. We have already mentioned that an intermediate node needs to receive and transmit in each direction, a flow will require 4 slots per frame. Thus, if we have $n_{ctl} = 1$, and $n_{con} = 1$, we are left with $n_{dat} = 8$, which can support $8/4 = 2$ voice calls through a bottleneck intermediate node. For $F = 60$ ms, the data path delay in an 8-hop path would only be $(8/2) \times F = 240$ ms (this is because, due to our scheduling algorithm, a data packet traverses two hops, in each frame), which is good for real-time voice. In the absence of data flows, the above set of parameters results in a duty cycle of $(1 + 1)/(1 + 1 + 8) = 20\%$.

5.3 Testbed Evaluation Setup

With the Tmote-based prototype, we used two different environments for evaluation: (1) an indoor setup, and (2) an *outdoor* setup around a residential area. The outdoor setting had some nodes placed near trees and buildings. The routing tree for control and contention packet in the outdoor setting had two variations—*fixed* and *dynamic*. In the former case, the routing tree was imposed and the parent-child relationships between nodes were fixed. In the dynamic case, these relationships were formed using an RSSI-threshold based shortest hop metric: shortest path to root using links with RSSI greater than -84dBm.

In the indoor setup, we used 19 nodes within a circular

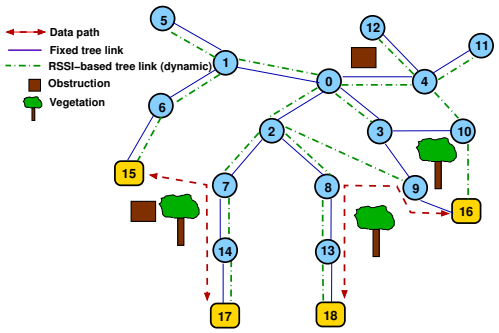

Figure 6: Outdoor network topology.

room about 20m in diameter. The room was an active lab with people using it for work; and there was also WiFi in the vicinity which we did not attempt to control. The *outdoor* setup, as shown in Fig. 6, had 19 nodes, each placed on a wooden stand, at a height of about 4 feet from the ground. For our setup, we used a transmit power -7 dBm, which gives a transmission range of about 15-20m. This was largely for ease of experimentation. The nodes were spread over an area of about 50mx50m. In the outdoor setup too, there was noticeable WiFi interference. Each unique run of the experiment lasted for around 20 minutes. We use two-way CBR traffic to represent a voice-call (without silence suppression). In the testbed setup, of the 19 nodes, 15 were used as infrastructure nodes, and 4 were used as portable end clients, as end-points for voice-calls.

5.4 Prototype Parameters

The primary parameters of concern are the slot and frame durations. Now, in the Tmote Sky platform, apart from the radio, there is another bottleneck, as identified in [12]: the SPI (Serial Peripheral Interface) bus. Every packet needs to go through the SPI, for transfer from radio chip to the CPU, or from the CPU to the radio chip. In a multi-hop setting, every intermediate node has to transfer the entire packet back and forth from/to the radio chip. The work in [12] describes how to eliminate this bottleneck, by pipelining SPI and radio operations. We have implemented such a pipeline in TinyOS (which gives a data slot of $6ms$), but have not yet integrated it with the LiT MAC code.

Although in our protocol description in Sec. 4, we have used equal slot lengths, we have found it convenient to use different slot lengths for control, contention, and data slots. This is to accommodate different sized packets in these 3 kinds of slots. This does not change the protocol operation in any way.

	SPI (ticks)	Radio (ticks)	Processing (ticks)
TX	93	154	$\simeq$20-35
RX	94	138	17

Table 5: Components of slot time for 126 byte packet

Tab.5 lists the various delays involved in the control slot calculation. With the SPI bottleneck, a control slot has to accommodate: (a) time to transfer a 126-bytes (size of radio transmit buffer is 128-bytes) control packet from the CPU to radio (93 ticks) (b) radio transmission time (154 ticks), including 16 ticks for sending a "strobe" signal to radio to commence transmission; radio reception at the receiving node happens in parallel with transmission (c) time to transfer the packet from the radio to the CPU at the receiving node (94 ticks), and (d) any processing jitter involved ($\simeq$52 ticks). Thus, we chose control slot duration to be $393(93 + 154 + 94 + 52) + 27 = 420$ ticks, considering 27 ticks for the guard band as given earlier. Similarly, we compute the slot time for contention and data slots as 320 ticks (contention packet size of 40 bytes) and 330 ticks ($\simeq 10ms$, data packet size of 48 bytes) respectively. Further, we have chosen $n_{ctl} = n_{con} = 1$ and $n_{dat} = 4$, to have a frame length of 2060 ticks (approximately $63ms$).

Although we use $10ms$ as slot duration in prototype, we have already noted that with pipeline optimization [12], we can get slot duration of $6ms$.

5.5 802.15.4 Evaluation Results

For the results in this section, unless mentioned otherwise, we have used the *fixed* topology in *outdoor* setting. We have very similar results for the *dynamic* version.

Time-synchronization: *What is the worst-case clock drift error of LiT's time-synchronization mechanism?* In LiT, during the transmission of control packets by infrastructure nodes, we timestamp the packet with global clock (i.e. root's clock) at the hardware level by capturing the SFD (Start Frame Delimiter) interrupt. The receiving node also timestamps the packet at the time of SFD reception. Now, the receiver node uses the information from its parent to update its own notion of the global clock (by calculating *clock offset*). Thus with each control packet received, there is an associated *clock correction* value at each node. This measures the relative *clock drift* between the sending and receiving nodes. This is a measure of the effectiveness of the multi-hop synchronization mechanism: if this value is large, then it means that the receiving node had been working with a large error so far.

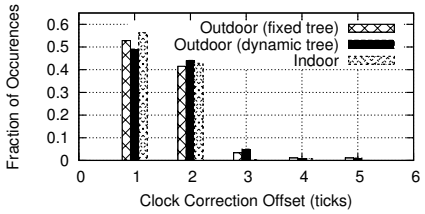

Figure 7: PDF of clock correction values

Fig. 7 plots the PDF of the various clock correction values, in units of clock ticks. We can see that most of the clock corrections are within 1-2 ticks. So the light-weight multi-hop synchronization mechanism is indeed effective, and we are able to make do without any sophisticated message exchange based synchronization [7] or drift estimation [10].

Soft-state performance: *What is the overhead and effectiveness of LiT's soft-state based mechanisms?* In our implementation, we have used the contention slot (one slot/frame) for periodic topology updates and flow renewal requests, to maintain soft-state, along with node-join and flow-setup requests. All information flow is from a node towards the root. So a natural concern is the usage of this contention slot since nodes nearer the root could get congested.

In the absence of flow setup requests, the primary usage of the contention slots is for topology updates. In our implementation, we have used a topology update period of 20 sec. The root times out a node X if it does not hear a topology update from X for over 100 sec and removes it from the

routing tree. This choice of values works stably in practice. In this setting we log the number of topology updates *forwarded* (or originated) by each node toward the root, in each 30-sec period. This is shown in Tab. 6 for a subset of nodes.

Node num.	1	2	3	6	8	9	14	15	16	17	18
Node-level	1	1	1	2	2	2	3	3	3	4	4
Num. topo updates fwded	273	456	263	132	187	129	129	62	57	61	56
Join latency (ms)	62	62	126	317	381	318	764	1147	1211	1147	1211
% ctl pkts lost from parent	1.9	1.5	1.9	3.1	1.6	2.3	1.4	1.3	2.3	1.5	1.3

Table 6: Node stats: samples at various tree-levels

The average number of updates forwarded by level-1 nodes (nodes 1, 2, 3 and 4) is 297, whereas that sent by level-4 nodes (nodes 16, 17 and 18) is 58. The number of available contention slots are $1000/64 \times 30 \simeq 470$ slots per 30-sec interval and $470 \times \frac{20min}{30sec} = 18800$ for the 20-minute duration. Hence, the usage of the contention slots for level-4 and level-1 nodes is $\frac{58}{18800} \simeq 0.3\%$ and $\frac{297}{18800} \simeq 1.6\%$ respectively, in both cases quite low; this includes any potential retransmissions required. Further, we observed very few (1 or 2) node-disconnections at the root; even these were confirmed to be due to a weak wireless signal, and not due to LiT's control overhead. The above results not only indicate that sufficient contention slots are available for flow requests and for scaling the network but also that the overhead of topology update messages, required for *network soft-state*, is negligible.

During active calls, the flow-renewal requests are sent at the period of 30 sec with 90 sec time-out. For **none** of the calls, termination occured at the root due to non-receipt of flow-renewal requests. This shows that *flow soft-state* works effectively.

Node join latency, as shown in Tab. 6, varied between a best case of 1 frame duration to a worst case of 19 frames (about 1.2s) which is very much tolerable in practical setting.

Data flow measurements: *What is the flow-grant latency of a voice call?* The call originator sends a flow request toward the root, the root runs the scheduler, and if successful, sets up the flow by disseminating the updated data schedule. All the relevant nodes in the chosen data path then follow the stipulated schedule, i.e., using the stipulated time-slot and the channel. Other parameters are as follows: exponentially distributed call duration with mean of 2 min, exponential inter-call time with mean of 2 min. Thus in the network, at any given time, we have 0, 1, or 2 flows active.

As shown in Tab. 7, the flow setup latency is mostly under a second, $0.3 - 1.1$ seconds. The CBR flow generates one data packet per frame (in each direction). So the number of packets shown is also indicative of the flow duration, in terms of number of frames. The packet loss percentage, in next column, is quite small, which indicates that there are no collisions due to loss of time synchronization, or undue data schedule losses.

Data path delay and jitter: *Is the data path delay and jitter tolerable to effectively support a voice call?* Data path delay, as mentioned earlier, is $(h_f/2) \times F$ (Sec. 4.6), which is $128ms$ for 4-hop flows. This is quite acceptable for real-time voice. For the G.723.1 codec, a delay of 128ms, a packet loss of 1%, and a jitter of 1.5ms represents a MOS [3]

Flow	Hops	Setup Latency (ms)	#packets origintated	% packet loss	Max Jitter (ms)	Avg Jitter (ms)
1, I	3	445.12	378	0.26	64	0.17
2, I	3	764.06	360	0.56	64	0.36
3, I	3	891.64	1074	0.09	64	0.06
4, I	3	445.09	687	0.15	64	0.09
5, I	3	572.55	938	0.32	128	0.2
1, O	3	1147.11	378	0.53	64	0.34
2, O	3	572.64	360	1.39	128	0.89
3, O	3	891.58	1074	0.74	128	0.48
4, O	3	381.4	687	0.29	64	0.19
5, O	3	764.06	938	0.85	128	0.55

Table 7: Data flow statistics (I=indoor, O=outdoor)

(Mean-Opinion-Score) of 3.8, which is considered "good" in practice. Note that, our scheduler has notion of delay constraint (maximum hops up to 10 corresponding to 300 ms), and thus rejects those calls which exceed the end-to-end delay limit.

The last columns in Tab. 7 report the measured jitter values: maximum and average. The median jitter was always 0ms (i.e. the TDMA slotting is working well). Jitter is caused when there is a packet loss in a particular data-slot. The average jitter is quite small (1-2ms); this demonstrates suitability of MAC for real-time voice. The maximum jitter scenario happens when there is a train of wireless losses in a node's data slot; this happens quite rarely.

6. SIMULATION-BASED EVALUATION

To answer the questions of scalability and control-overhead, we simulated LiT using a custom discrete event simulator. We evaluate LiT for voice based wireless mesh setting envisioned in [15] over an area of 1.5 km x 1.5 km, with a total of 100 nodes: 25 infrastructure nodes and 75 client nodes. The duration of one simulation run is 12 hrs. The evaluation parameters considered are: 250m transmission and 350m interference range, 16 channels, calls between random source-destination pairs, exponential inter-call duration with mean of 1 hour, exponential call duration with mean of 2 min with parameters to support real-time voice as mentioned in Sec.5.

6.1 LiT Performance

Tab. 8 shows the performance of scheduler in terms of number of accepted and rejected calls for different wireless loss rates. The loss rate was simulated on each link in the network. We note that the MAC's control messages are getting through, and the soft-state based flow maintenance is not a bottleneck. This is indicated by the fact that there are very few calls falsely dropped (last column). In fact, even when the loss rate of all links in the network is 10%, only about 7% of the calls are falsely dropped.

Error Rate (%)	Calls Originated	Calls Admitted	Calls Rejected	Dropped due to Ctrl. Overhead
1	891	887	4	0
2	885	882	3	3
5	894	890	4	4
10	902	898	4	58

Table 8: Call statistics

Now, there are three possible reasons why a flow request could be rejected: (1) the flow request did not reach the root (2) the scheduler could not accommodate the flow either due to no free scheduling slots or no overlapping free slots at two nodes or no free channel and (3) the acceptance information

did not reach the relevant nodes in the network. We observed that in all the cases of rejected flows, it was always the scheduler which was rejecting the flow. Thus, the MAC related control overhead was *not* the bottleneck. Also, even with a high call generation rate (mean call time of 0.5 hrs) from each client, the scheduler rejects 1.8% (32 out of 1725) calls which shows that they system is **scalable** and that the voice-aware multi-channel scheduling is quite effective.

In terms of latency involved, for more than 95% flows, the flow setup latency is $< 5s$ (median 2.4 s, for 2% error rate) which is quite tolerable. The median, minimum and maximum path length (number of hops) is 6, 2 and 10 which gives worst case delay of $300ms$, a tolerable value in practice.

Further, we observed that the scheduler could only accept 634 calls for single channel as compared to 1693 calls for 16 channels when mean call time was 0.5 hrs, which shows advantage of multi-channel MAC. In terms of soft-state, even for such a large network, when we simulated a 5% error rate on all links, we neither observed any tree disconnections nor any accidental flow revocations. The errors due to schedule inconsistencies in data slots were less than 2%. This supports the design choice of *soft-state* for MAC level connection management.

Power consumption: Consider a aggregate call duration of 2 hours a day and a 20% effective duty-cycling. The Tmote platform consumes about 60mW while transmitting and receiving (0dBm). Thus assuming a node power consumption of 100mW when it is active, the power consumption per day is: 100mW × 2H + 20% × 100mW × 22H = 640mWH. If we use 4.5AH, 12V battery at each infrastructure nodes, a node can effectively operate for 4.5AH × 12V/640mWH ≈ 84 days. Thus the system can operate without relying on the power off the grid for several weeks.

6.2 LiT vs. CSMA

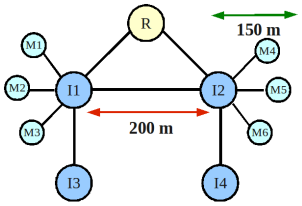

Figure 8: CSMA vs TDMA comparison topology

What would have been the performance of voice calls, had we used the default distributed CSMA MAC protocol (for the 802.15.4 multi-hop network)? To answer this question and thus to gauge the performance benefits of LiT, we simulate the distributed CSMA MAC for a multi-hop network. The topology instance used for simulation is shown in Fig. 8. For CSMA operation, 802.15.4 standard specifies following parameters which we use in our simulations: symbol period = 16 us, max backoff exponent = 5, max retry limit = 3, SIFS of 12 symbols, LIFS of 20 symbols, back off unit (slot) of 20 symbols, CCA detection in 8 symbols. We also simulate the Tmote Sky platform environment which has SPI speed (with DMA enabled) 350kbps, bottleneck in packet transmissions and receptions, as seen in our implementation. We use the 6.3 kbps codec (same as used in the LiT implementation) which generates a 48 byte packet every 60 ms (CBR flow). We send 1000 packets (60 sec worth data) for a flow from source handset to destination handset. We keep

Row No.	Calls active	CSMA				TDMA			
		% Packet Loss	Avg. Delay	Avg. Jitter	MOS	% Packet Loss	Avg. Delay	Avg. Jitter	MOS
1	One uni-direct	0	25	0	3.95	0	60	0	3.95
	One bi-direct								
2.1	Forward flow 1	0	42	0.89	3.95	0	60	0	3.95
2.2	Backward flow 1	0	29	0.54	3.95	0	60	0	3.95
	Two bi-direct								
3.1	Forward flow 1	24	80	50	1.58	0	60	0	3.95
3.2	Backward flow 1	26	82	47	1.58	0	60	0	3.95
3.3	Forward flow 2	25	83	47	1.58	0	60	0	3.95
3.4	Backward flow 2	26	82	49	1.58	0	60	0	3.95
	Three bi-direct								
4.1	Forward flow 1	52	228	106	1.02	0	60	0	3.95
4.2	Backward flow 1	53	234	115	1.03	0	60	0	3.95
4.3	Forward flow 2	51	232	108	1.02	0	60	0	3.95
4.4	Backward flow 2	54	228	136	1.03	0	60	0	3.95
4.5	Forward flow 3	55	242	117	1.03				
4.6	Backward flow 3	52	230	111	1.03				

Table 9: CSMA vs TDMA comparison for voice calls

a buffer of 3 packets at the source (without this buffer, the packets will get dropped if the previous packet is pending to be transmitted) and a buffer of 3 packets at the destination (this is the playback buffer). Further we maintain transmission range of 250 meters and interference range of 350 meters, just like in LiT MAC simulator.

Now, with this set of parameters, a packet takes approximately 8ms to get transmitted and received over a single link. All packets flow through the intermediate nodes I1 and I2 (Fig. 8). Now, we start a single uni-directional flow from handset M1 to handset M4 (we refer to the flow from M1 to M4 as forward flow). The first row in Tab.9 shows the packet loss percentage, the average delay and the average jitter values for the flow, which is 0%, 25ms, 0ms respectively. Now, we start the traffic from M4 to M1 (backward flow, through same intermediate nodes, I1 and I2). This models a real-time bi-directional voice call. As the rows 2.1 and 2.2 show, both the forward flow and the backward flow of the call experience 0% packet loss with the average delay of less than 30 ms & 42 ms respectively and the average jitter of less than 1ms. This is conceivable as both the forward and the backward flows are getting sufficient channel time (overall 60 ms) to transmit packets end-to-end from source to destination. Next, we start one more bi-directional flow from M2 to M5. As we expected, this overwhelmed the limited channel capacity and as the rows from 3.1 to 3.4 show, both the calls suffer heavily with 25% packet loss and the average jitter of 45ms. To stress test, we started a third bi-directional voice call between M3 and M6 through I1 and I2. The performance of this call was not only poor due to already 'congested' channel but it also degraded the quality of existing voice calls. This shows the requirement of admission control in multi-hop network. While admission control is not ruled out in CSMA, it requires more complex capacity estimation techniques [9].

Now, we apply the same setting to LiT MAC. We have already noted that with the pipeline optimization (which gives us slot duration of 6ms), we can support two simultaneous calls in 60ms frame of LiT MAC. By the very unsynchronized nature of packet clocking, such a pipelining optimization is not possible in CSMA MAC. As we can see in right-half of Tab.9, the average delay of two bi-directional calls is 60ms with the average jitter of 0ms. Due to the time-slotted scheduling of transmissions in LiT, the calls do not experience any packet loss due to the packet collisions. For two simultaneous calls, the MOS score for CSMA MAC is very poor whereas it is quite satisfactory for TDMA MAC. In fact, the drop in MOS from 5 to 3.95 is primarily due to the choice of 6.3Kbps. The table also shows (rows 4.5

and 4.6) that the centralized admission control of LiT MAC rejects the third call due to non-availability of channel resources. This justifies the choice of TDMA multi-channel access and centralized admission control in LiT for a mesh network setting, especially for capacity limited network.

7. FUTURE WORK

We have developed a portable 802.15.4-based handset by interfacing TI's C5505 USB stick module with CC2520 radio. We have integrated the license-free *Speex* codec with the C5505 USB stick module to encode and decode the voice samples. We established successfully a voice call over a 4-hop network running on LiT MAC.

Going forward, we are working on deploying a backbone network of 20 nodes in the CSE department to run for a period of several days. The 802.15.4-based handsets will be used to establish the voice calls. In such a setting, our goal is to measure and quatify the performance of the network in terms of the power efficiency (i.e. longevity of the network) and voice call quality. Such a deployment will give us insights into an operational mesh network. As the subsequent step, we want to deploy the Lo^3 system in an outdoor setting in a nearby village with few handsets. We envision that the users will be able to communicate with each other over a network which would run for several days without battery replacement. Apart from bi-directional real-time voice, there are several applications that can be built in Lo^3 system. Such applications include community radio and broadcasting for real-time information dissemination. Building such applications is also part of future work. To support text-based applications, we can reserve a part of network capacity and can have a scheduler which schedules both text and voice-based flows, in an opportunistic fashion.

8. CONCLUSION

In [15], authors envisioned Lo^3: a low-cost, low-power, local-voice communication system to support real-time applications in a village-like setting, in developing regions. In this work, we described an end-to-end design, implementation and evaluation of an 802.15.4-based prototype to enable voice communication in wireless mesh networks. This prototype is a significant step to deploy Lo^3 in practice. To achieve the goal of supporting real-time voice applications, we devised LiT: a **li**ght-weight **T**DMA based multi-hop MAC protocol. LiT is novel in its use of soft-state for dynamically maintaining the topology, and various MAC-level connections. The MAC has a built-in multi-hop time-synchronization mechanism, dynamic schedule dissemination, and flexible support for scheduling algorithms. The control overheads in LiT are minimal even under realistic wireless loss scenarios. We achieve low flow setup delays, and data path delay/jitter are small too. This bodes well for real-time application support envisioned in Lo^3. Our simulation based study strengthens our confidence that LiT's soft-state based approach, or the centralized architecture, do not present scaling bottlenecks in practice.

Acknowledgement

This project is being carried out under the IU-ATC project funded by the Department of Science and Technology (DST), Government of India and the UK EPSRC Digital Economy Programme. This work is supported in part by IBM faculty award (2008). The Speex implementation on C5505 platform was provided to us by CouthIT, Hyderabad. Thanks to Swanand and Victor for their help during outdoor experiments.

9. REFERENCES

[1] Department of telecommunication, http://www.dot.gov.in/.

[2] IEEE 802.16 WirelessMAN, http://www.ieee802.org/16/.

[3] Mean Opinion Score Calculator, http://www.davidwall.com/MOSCalc.htm.

[4] P. Bhagwat, B. Raman, and D. Sanghi. Turning 802.11 Inside-Out. In *HotNets-II*, Nov 2003.

[5] K. Chebrolu and B. Raman. FRACTEL: A Fresh Perspective on (Rural) Mesh Networks. In *NSDR*, Sep 2007. A Workshop in SIGCOMM 2007.

[6] K. Chebrolu, B. Raman, N. Mishra, P. K. Valiveti, and R. Kumar. BriMon: A Sensor Network System for Railway Bridge Monitoring. In *MobiSys*, Jun 2008.

[7] P. Djukic and P. Mohapatra. Soft-TDMAC: A Software TDMA-based MAC over Commodity 802.11 hardware. In *INFOCOM'09*, Apr 2009.

[8] P. Djukic and S. Valaee. Delay aware link scheduling for multi-hop TDMA wireless networks. *IEEE/ACM Transactions on Networking*, 2009.

[9] A. Kashyap, S. Ganguly, S. Das, and S. Banerjee. Voip on wireless meshes: Models, algorithms and evaluation. In *INFOCOM*, 2007.

[10] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi. The Flooding Time Synchronization Protocol. In *SenSys*, Nov 2004.

[11] J. Mo, H.-S. W. So, and J. Walrand. Comparison of Multichannel MAC Protocols. *IEEE Transactions on Mobile Computing*, May 2007.

[12] F. Osterlind and A. Dunkels. Approaching the Maximum 802.15.4 Multi-hop Throughput. In *HotEmNets*, Jun 2008.

[13] V. Rajendran, K. Obraczka, and J. GarciaLunaAceves. EnergyEfficient, CollisionFree Medium Access Control for Wireless Sensor Networks. In *SenSys*, Nov 2003.

[14] B. Raman and K. Chebrolu. Design and Evaluation of a new MAC Protocol for Long-Distance 802.11 Mesh Networks. In *11th Annual International Conference on Mobile Computing and Networking paper (MOBICOM)*, Aug/Sep 2005.

[15] B. Raman and K. Chebrolu. Lo^3: **Lo**w-cost, **Lo**w-power, **Lo**cal Voice and Messaging for Developing Regions. In *NSDR'09*, Oct 2009.

[16] A. Rowe, R. Mangharam, and R. Rajkumar. RT-Link: A Time-Synchronized Link Protocol for Energy-Constrained Multi-hop Wireless Networks. In *SECON*, 2006.

[17] A. Sen and M. L. Huson. A New Model for Scheduling Packet Radio Networks. In *INFOCOM*, 1996.

[18] H.-S. W. So, G. Nguyen, and J. Walrand. Practical Synchronization Techniques for Multi-Channel MAC. In *Mobicom*, Sep 2006.

[19] L. van Hoesel and P. Havinga. A Lightweight Medium Access Protocol (LMAC) for Wireless Sensor Networks. In *INSS*, 2004.