Implementation Based Evaluation of a Full-Fledged Multi-Hop TDMA-MAC for WiFi Mesh Networks

Vishal Sevani, Bhaskaran Raman, Piyush Joshi
Department of CSE, IIT Bombay, INDIA
{vsevani,br}@cse.iitb.ac.in, piyush28joshi@gmail.com

Abstract: Wireless mesh networks in general, and WiFi mesh networks in particular, offer a cost-effective option to provide broadband connectivity in sparse regions. Effective support for real-time as well as high throughput applications requires a TDMA-based approach. However, *multi-hop* TDMA implementations in wireless have been few and far-between, and for good reasons. These present significant issues in terms of time synchronization, TDMA schedule dissemination, multi-channel support, routing integration, spatial reuse, etc. And achieving these efficiently, in the face of wireless channel losses presents a formidable challenge.

In this work, we present an implementation of LiT MAC, a full-fledged multi-hop TDMA MAC, on commodity WiFi platforms. We undertake extensive evaluations using micro-benchmarks as well as application level performance, using outdoor as well as indoor testbeds. We also present an integration of LiT MAC with various routing metrics, and a routing stability study of recently proposed routing metrics (ROMA, SLIQ). Our results show that we can achieve μs granularity time synchronization across several hops, and TDMA slot size as small as 2ms. These imply low control overheads. Experiments over several days, on our 9-node outdoor testbed shows that LiT MAC's soft-state based approach is effective in robust operation even in the presence of significant external interference.

Index Terms: Wireless Mesh Networks, TDMA MAC Protocol, TDMA Mesh Networks

I. Introduction

It is well known that in the context of wireless mesh networks, a random access based MAC protocol such as CSMA/CA cannot provide QoS for applications such as real time voice, video streaming, etc. [1]. A TDMA-based approach is necessary for good throughput as well as low jitter & delay.

However a look at the history of wireless data standards reveals that more often than not, multi-hop TDMA has only been part of the standard on paper, without any implementation based evaluation. A classic example is that of Bluetooth, with scatternets never seeing the light of day. Even with the more recent WiMAX standard, we are not aware of any implementation based evaluation of the multi-hop mesh mode. A notable exception to this pattern is the case of 802.15.4 mesh networks [2]. However, this is intended only for low data rate sensing applications.

What makes multi-hop TDMA in wireless tough? The first is the issue of fine-grained *time synchronization* across multiple hops. This must be achieved and maintained in the face of wireless packet losses. Second, to achieve high performance, it must provide support for *spatial reuse* as well as *multiple channels*. Third, nodes in the network must agree upon a schedule of transmission. In the context of centrally determined time-slot schedules, this translates to *schedule dissemination*, from the central node to other nodes. This too must be done in face of wireless losses, as well as dynamic arrival and departure of network flows. And finally, the TDMA MAC must integrate easily with a routing mechanism.

It is this set of formidable challenges that have made most wireless data standards shy away from taking multi-hop TDMA to practice. Even recent research based efforts have only been piece-meal in nature. For instance, the multi-hop TDMA implementations in [3], [4] deal primarily with time synchronization, and have been evaluated only in relatively interference free (i.e. using 802.11a) indoor settings. And while Overlay MAC [5] considers a distributed mechanism for nodes to agree upon a schedule, it works under the restricted assumption of single-channel operation, and a 2-hop interference model. The TDMA schemes in [6], [7], [8] are applicable only to long-distance networks, with highly directional links, and not to generic mesh networks. (See Table I for a succinct comparison with other multi-hop TDMA wireless implementations).

In our recent prior work [9], we have designed *LiT MAC*, a full-fledged multi-hop TDMA mechanism, which addresses all the above-listed challenges. [9] reports an 802.15.4-based implementation of LiT MAC. The contributions of this paper are three-fold.

(1) First, we implement the various features of LiT MAC on *commodity WiFi hardware*, using modifications to the open-source MADWIFI driver. The use of commodity hardware is of practical significance since it allows us to ride on the low-cost benefits of these platforms[1]. Sec. IV-D highlights the salient features of our implementation technique in comparison with prior implementations. (2) Next, we evaluate the implementation along various dimensions: multi-hop synchronization error, time granularity of scheduling, achievable throughput for UDP & TCP flows, jitter for CBR flows, etc. We have used indoor as well as outdoor testbeds for such evaluation.

[1]Our earlier workshop publication [10] worked out some of the driver level implementation mechanisms, which we build upon in this work, to implement various MAC features, and to evaluate extensively.

(3) Third, we have integrated a routing mechanism with the LiT MAC and performed a routing stability evaluation on the outdoor testbed. The stability analysis includes three routing metrics: the popular ETX metric [11], and the more recent ROMA [12] & SLIQ [13] metrics. Such evaluation is of significance since ultimately application QoS depends not only on the MAC performance, but also on the stability of routes during packet flow.

Our highlight results are as follows. (A) By making use of hardware packet timestamping, we attain time synchronization accuracy with average error of only about $10\mu s$ even at 7 hops. This implies a low guard time overhead. The synchronization error of our technique is lower than that for prior techniques devised for commodity WiFi hardware and also our technique has been evaluated over larger number of hops (see Table II). (B) Using only Linux's software timers, we can achieve a slot size as small as 2ms, while CPU interrupt overheads prevent the use of smaller slot sizes. But a slot size of 2-5ms is often small enough to achieve low jitter & delay in practice. (C) The integration of LiT MAC with the SLIQ routing metric works well; SLIQ's stability is much better than that of ROMA, which in turn is more stable than ETX in realistic outdoor settings.

We believe that our implementation and evaluation is of practical significance to the mesh networking community. The rest of the paper is organized as follows. The next section (Sec. II) presents related work. Sec. III briefly describes LiT MAC, to setup the context for a description of its implementation over commodity hardware, in Sec. IV. Subsequently, Sec. V presents micro-benchmarks and parametric evaluation of the implementation, while Sec. VI describes overall evaluations of our implementation. Sec. VII concludes the paper with a few points of discussion.

II. Related Work

We now compare our work with prior related work.

Single-hop TDMA: Single hop wireless TDMA systems are aplenty. Significant such systems closely related to this work are as follows. Our methodology of modifying the MADWIFI open source driver to build a TDMA WiFi system on commodity hardware is similar to [14], [15], [16]. Apart from the primary difference that [14], [15], [16] are evaluated for single hop settings while we focus on multi-hop settings, there are other differences too. For instance, [14] makes use of out of band ethernet based time synchronization, which is not applicable for mesh networks. [15] does not evaluate the time synchronization accuracy of the proposed mechanism, but its extension [16] reports a time synchronization accuracy of about $25\mu s$. However, the slot size used by [16] is in the range of 20-60 ms, with a guard time of 4-12 ms. In comparison, we use slot sizes of 2-10 ms, with a guard time of $100\mu s$.

Multi-hop TDMA: Table I compares prior multi-hop TDMA MAC implementations with our work succinctly; the details are as below.

Overlay MAC [5] also modifies the MADWIFI driver to implement a TDMA MAC. It proposes a distributed mechanism for nodes to arrive at an agreement on slot usage. The design imposes the significant restriction that all nodes should use the same channel in all time slots. It also assumes that interference is limited to a 2-hop neighbourhood. Unlike this, due to its centralized approach, LiT MAC supports multiple channels as well as consideration of arbitrary interference patterns.

Another important difference is that [5] reports evaluation only in an interference free, 802.11a testbed. Now, a wireless TDMA system is stress tested only under conditions which cause significant packet loss. Hence our consideration of an 802.11g testbed in interference prone conditions is a more stringent test. A further difference is that while the testbed used in [5] is indoor, we consider an outdoor testbed too; this is significant since after all mesh networks are intended for outdoor deployment.

[3] outlines the design of a TDMA MAC protocol based on in-band time synchronization and takes into account hardware bottlenecks such as clock drift, processing delay, etc for calculating the guard interval, slot duration, etc. It also takes into account wireless errors for calculating the synchronization period, i.e. time interval after which the network needs to be re-synchronized. It outlines the implementation of the protocol on the proprietary WiLD MAC platform with time synchronization accuracy of the order of microseconds, and slot granularity of the order of milliseconds.

[4] proposes a pairwise synchronization algorithm similar to LiT MAC, and takes into account propagation delay to calculate clock difference between two nodes. It makes use of a guard band to account for processing delays encountered while packet transmission, and carries out measurements to empirically tune the value of the guard band. Based on the measurements carried out on an indoor testbed, it reports synchronization accuracy of the order of microseconds at three hops.

While [3] & [4] are closest to our work in terms of μs granularity of synchronization, the differences are many. Our implementation technique is different from [3], [4] and yields higher synchronization accuracy on commodity WiFi hardware over larger number of hops (see Sec. IV-D). Apart from implementation technique, other differences are that [3], [4] have primarily focused on time synchronization, while we have also considered a more comprehensive set of issues such as schedule dissemination (and the corresponding overhead), integration with a routing protocol, spatial reuse, and also an implementation on commodity WiFi hardware. Further, as in the case of [5], the evaluations in [3], [4] too have considered only interference free 802.11a settings, indoors. As mentioned earlier, it is important to test in interference prone (802.11g-based) realistic outdoor settings.

[6], [7], [8] outline TDMA-based MAC protocols for long-distance networks. Apart from the difference that these use loose time synchronization of the order of milliseconds, the primary difference is that these are applicable only for long-distance networks with highly directional antennas, while our work considers a generic mesh network with directional or omni-directional antennas.

TSMP [2] describes an 802.15.4-based multi-hop TDMA MAC implementation. 802.15.4 not only has very different modulation scheme from 802.11, but also has much lower data

	Synch. Accuracy	Schedule Dissemination	Routing Integration	Spatial Reuse Support	Evaluation
Overlay MAC [5]	order of ms	distributed	No	1-chnl only	6 node, indoor, 802.11a
Wildnet[6]	order of ms	No	No	No	3 node, outdoor, 802.11b
2P [7]	loose	No	No	restricted	3 node, outdoor, 802.11b
TDM-MAC [3]	order of μs	No	No	No	8 node, indoor, 802.11a
Soft-MAC [4]	order of μs	No	No	No	4 node, indoor, 802.11a
TSMP [2]	order of μs	No evaln.	Yes	No evaln.	44 node, indoor, 802.15.4
LiT MAC on WiFi (our work)	order of μs	Yes	Yes	Yes	9 node, outdoor, 802.11g 15 node, indoor, 802.11g

TABLE I
COMPARISON WITH PRIOR TDMA MULTI-HOP IMPLEMENTATIONS

rate (250Kbps max), and is intended for low data rate sensing applications. We consider more traditional applications such as bulk data transfer, audio/video, etc. which require high data rates.

Routing stability studies: Our work includes an integration of LiT MAC with routing metrics and a routing stability study. [17], [18] are among earlier works to study routing instability due to link metric variations. [17] studies the extent of instability with the use of ETT metric in mesh networks. [18] on the other hand studies how do different factors such as link metric variations, channel conditions, time of the day, background traffic, etc affect routing stability. However, both these studies have carried out measurements for CSMA/CA based mesh network where the protocol behaviour and background traffic impacts the link metric computation. Our work is in context of TDMA mesh networks wherein link metric probes are transmitted in dedicated time slots. Further, our study includes the recently proposed ROMA [12] and SLIQ [13] metrics which offer much better stability compared to the ETX metric considered in [17], [18] (see Sec. VI-B for quantitative differences among the metrics).

[13] proposes the use of SLIQ metric to provide improved stability. However [13] presents only trace based analysis, of traces from a CSMA/CA based network. In our work, we have integrated SLIQ with the LiT MAC, and have quantified SLIQ's benefits in terms of routing stability, in a realistic outdoor testbed.

III. LIT MAC: PROTOCOL DESIGN

We now give a brief description of the salient features of LiT MAC [9]; specifically, time synchronization, schedule dissemination, and how these work robustly in the presence of wireless errors. For more detailed description of LiT MAC protocol design along with its comparison with other TDMA MAC protocols in literature reader is referred to [9].

Time synchronization & schedule dissemination: To facilitate time synchronization and TDMA schedule dissemination, LiT MAC follows a frame based mechanism, wherein a slot is a unit of allocation, and a frame consists of three different types of slots: control, contention and data slots (see Fig. 1). The use of these three types of slots is as follows.

Control slots: These are used for time synchronization as well as to convey resource allocation i.e. to disseminate the TDMA schedule. For the transmission of control packets, the network topology is arranged as a tree rooted at the root node. For time synchronization, LiT MAC makes use of a pairwise synchronization mechanism wherein a given node is time synchronized with its parent in the control tree. The control packet consists of three types of information: time synchronization information, the control schedule, and the data schedule. The control schedule is nothing but the allocation of control slots to each node in the network. The control slots for all the nodes need not be in a single frame: different nodes can be allotted slots in different frames; Fig. 1 shows such an example where the control schedule spans two successive frames.

Contention slots: In the contention slots the nodes transmit packets probabilistically. These slots are used to facilitate asynchronous events such as a new node joining the network, resource reservation for a new flow, or a routing metric update. A new node joins the network by sending node-join request packets toward the root node, using the contention slots.

Likewise, slot allocation for a new flow is accomplished by making use of flow request packets. The node that wants to initiate a flow transmits flow request packets in the contention slot, which then propagates up the control tree towards the root node. A scheduling algorithm then decides the slot allocation to the new flow. The MAC itself can support any centralized scheduling algorithm; it is responsible only for communicating the slot allocation given by the scheduler. This is communicated as the data schedule, as part of the control packets.

To facilitate dynamic routing, LiT MAC makes use of topology update packets that can be used by the nodes to convey routing metric information to the root node.

Data slots: Data slots are used for the actual transfer of the data. As mentioned above, a node is told about its data slots via the data schedule contained in the control packets. As with control slots, the data schedule may span across multiple frames.

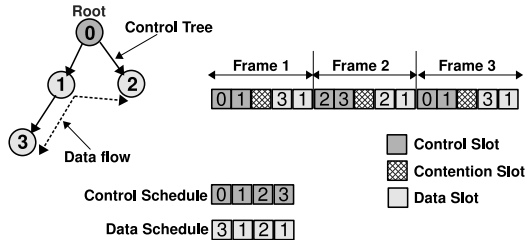


Fig. 1. LiT MAC: An example slot allocation

Fig. 1 shows a possible control schedule for an example topology and a data schedule for a bi-directional data flow between nodes 3 & 2. In this example, the control as well as the data schedules span two adjacent frames.

Handling wireless losses: Any realistic deployment will face wireless losses due to fading and interference. To handle

wireless losses, LiT MAC makes extensive use of soft-state mechanisms. For instance, soft-state based periodic transmission of the control & data schedules ensures that all nodes eventually follow the same schedule. Likewise nodes transmit topology update packets periodically to the root node to inform that they are alive. Soft-state is used to setup, maintain, and terminate flows too. That is, a flow-request is repeatedly sent until the corresponding slot allocation is given (or a timeout is reached): so the loss of flow-request packets are handled elegantly. Even after flow setup, flow-renewal packets are periodically sent, failing which the root node terminates a flow (i.e. deallocates the corresponding data slots).

Although such use of soft-state mechanism enables LiT MAC to elegantly handle wireless losses and maintain the network in consistent state, it has a cost. The overheads of the frame structure and the soft-state mechanism are quantified in our performance evaluation (Sec. VI). We now move on to the description of the LiT MAC implementation.

IV. Implementation of LiT MAC

Our goal is to implement a full-fledged multi-hop TDMA MAC on *commodity WiFi hardware*, to take advantage of the low-cost benefits of the same. This involves resolving some important issues, as we describe below. We have modified the MADWIFI driver v0.9.4-r3314 for our implementation. This driver works for the popular Atheros WiFi chipsets. We have used Ubiquiti XR2 (http://www.ubnt.com/xr2) and Mikrotik R52/R52H wireless cards, with Mikrotik RB 433AH/411AH/411AR (http://www.mikrotik.com/) single board computers. Some nodes run Openwrt Kamikaze v8.09, while others run Openwrt Backfire v10.03 (http://openwrt.org), both based on Linux v2.6.

A. TDMA mechanism

There are two important components of the TDMA mechanism (1) time synchronization i.e. how accurately the clocks across the different nodes are synchronized with each other, and (2) how accurately we can control packet transmission times, so that nodes transmit only in their allotted slot. We describe both below.

1) Time synchronization: As per the 802.11 standard, *beacon* packets contain the exact timestamp [2], in terms of WiFi NIC (Network Interface Card) clock, at which these packets get transmitted. Likewise at the receive side the hardware records the time, in terms of WiFi NIC clock, at which the packet is received. While we have disabled regular 802.11 beacons, we make the hardware treat LiT's control slot packets as beacons [10]. We then use the transmit and receive timestamps to calculate the difference in the WiFi NIC clock between the two nodes, i.e. the clock *offset*. Since LiT MAC makes use of pairwise synchronization along the control tree, every node propagates its clock offset with respect to the root node, down the control tree, and thereby every node is synchronized with the root node. As in our technique the nodes are synchronized with respect to WiFi NIC clock, it yields lower synchronization error compared to [4] as we elaborate in Sec. IV-D.

Discrepancy in WiFi hardware: We faced the following subtle issue in implementation. The hardware timestamp in beacon packets, mentioned above, is meant to correspond to the time at which the 25th byte of the beacon is transmitted on air. But the timestamp computation is done just before sending the packet. We observed that the WiFi NIC computes the timestamp assuming that the packet transmission rate is 1Mbps, even if the beacon were sent at another rate (say 6M). So for control packets transmitted at a data rate more than 1Mbps, we had to account for this discrepancy in the offset computation at the receiver, by appropriately taking into account the packet's transmit rate.

2) Controlling packet transmissions: The mechanism we use for packet transmission is: at the start of the node's transmission slot, the driver inserts the set of packets that can be sent in that slot, into the hardware queue of the WiFi NIC, which then transmits the packets. Now to accurately control the packet transmission time, there are two requirements: (a) The packet transmission event should be triggered exactly at the start of the slot, and (b) Once the packet is put in hardware queue, it should be transmitted immediately without waiting for the medium to be free. We describe these in turn.

Use of timers: To trigger the packet transmission event, we need to make use of a periodic timer. We have an option of using either Linux's 1ms granularity software timer or the hardware timer provided by WiFi commodity hardware. However, for the hardware timer we observed that under heavy load (i.e. under continuous packet reception) the timer accuracy is very low [10]. So we make use of the 1ms software timer, which is quite accurate for our purpose.

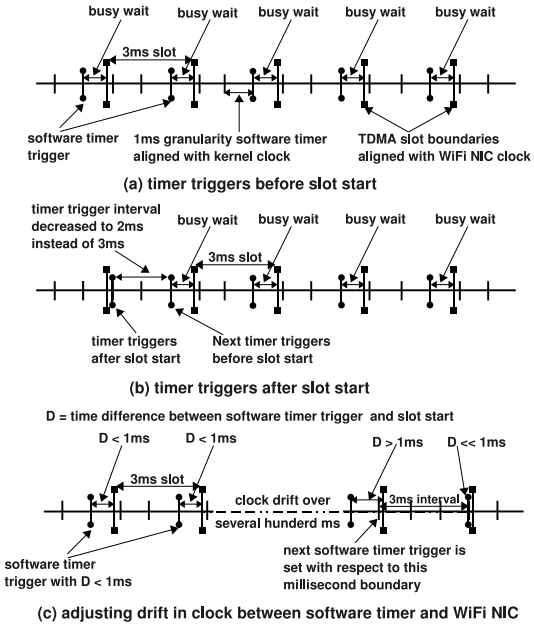


Fig. 2. Triggering packet transmission event

However, for making use of software timer we need to resolve the following issue. The linux software timer is of 1ms granularity, whereas the hardware clock of the WiFi NIC is of $1\mu s$ granularity. So it is likely that millisecond boundary

[2]beacon timestamp is the exact time in terms of WiFi NIC clock when 25th byte of beacon packet is sent on air

of the software timer is not aligned with the slot boundary of the TDMA i.e. the software timer can trigger at-most 1ms before or after the start of the TDMA slot. The event wherein the timer triggers before the slot start can be handled easily by doing busy wait till the start of the slot as shown in Fig. 2(a). However the event when the timer triggers after the slot start is undesirable, as no packets are transmitted between the slot start and triggering of the timer. So in case the timer triggers after the start of a slot, we decrease the next timer trigger interval by 1ms so that for the next TDMA slot (as well as a few subsequent slots) the timer triggers before the start of the slot, as shown in Fig. 2(b).

Furthermore, another subtle issue that our measurements revealed is that there is drift between the software timer and WiFi NIC clock. This is because they use different clock sources within the same node. We observed that the software timer typically drifts behind the WiFi NIC clock by $1\mu s$ in 400ms. So the time difference between the software timer trigger and slot start (busy wait period in Fig. 2(a)) slowly increases.

However our scheme ensures that this difference does not increases beyond 1ms. This is since we use the WiFi NIC clock for time synchronizing the nodes and also to identify TDMA slot boundaries: the software timer is used only as an approximation to the slot boundary. To elaborate, when the difference between software timer trigger and slot start becomes more than 1ms, we trigger next software timer event, after interval of slot size, with respect to the millisecond boundary of software timer just before the slot start and not with respect to the time of current software timer trigger as shown in Fig. 2(c). This ensures that for the next software timer trigger the difference with respect to TDMA slot start again decreases to a small value as shown in Fig. 2(c).

Disabling CCA: Once the packet is inserted into the hardware queue by the driver, it is essential that the the packet is transmitted on-air immediately. However, the default WiFi hardware uses clear channel assessment (CCA) and transmits the packet only when the medium is free. So in presence of interference the packet transmission can get delayed. We have disabled the CCA, alongwith 802.11 backoff, as per the mechanism given in [19]. The evaluation of the accuracy with which the packet transmission time can be controlled with CCA disabled is presented in Sec. V.

B. Hop-by-hop error recovery

In wireless, especially in multi-hop settings, it is beneficial to have hop-by-hop error recovery for lost packets. We considered three options for this: adaptive FEC (forward error recovery), bulk ACKs (acknowledgments), and per-packet ACKs. Wildnet [6] has implemented the first two options, whereas we have used per-packet ACKs. These ACKs are transmitted immediately after the packet transmission ends in the same slot itself thereby preventing the need to schedule ACK transmissions separately. While we postpone a quantitative comparison to Sec. V, it is intuitive to note that per-packet ACKs are efficient if we can get to use WiFi's in-built immediate ACK mechanism, since these are generated by the hardware without any driver involvement.

However, the use of WiFi's in-built ACKs is tricky for the following reason. If we let the default hardware decide the number of retransmissions, such retransmissions may cross our TDMA slot boundaries. So we do the following. We *turn-on* per-packet hardware ACKs, but *turn-off* hardware retransmissions. Now, the hardware at the sending node notifies the driver about whether or not an ACK was received. The driver uses this notification to determine whether or not to retransmit, while also respecting slot boundaries, and also implementing a maximum retransmission count.

Another subtle issue is that, for the hardware we use, the above mentioned notification is received at the driver after a delay of about 200-300 μs. Waiting for such a long time to retransmit the lost packet can be inefficient. So what we do is: we transmit *n* distinct packets in a given slot, where *n* is the number of the packets that can be transmitted in that slot. Of these *n* packets, if any of the packets is lost then we retransmit it in the next data slot allotted for the node. Assuming that the next allotted data slot is after the maximum ACK notification delay (i.e. about 300 μs), the notification would have been received by the time the next data slot starts and hence the node can retransmit the lost packet if needed.

C. Routing integration and spatial reuse

In multi-hop wireless networks, routing is inextricably linked with the MAC, since the same neighbourhood set determines the next-hop (i.e. routing) as well as transmission opportunity (i.e. MAC). LiT MAC, being a centralized MAC, supports centralized dynamic routing. To incorporate various routing metrics, two aspects are required: (a) the computation of the routing metric, and (b) conveying the routing metric to the root node.

Routing metrics such as ETX [11], ROMA [12], or SLIQ [13] are computed by measuring the packet delivery ratio over a given link. For this, we need to decide if separate probe packets should be sent or existing traffic may be used; and if existing traffic is to be used, then packets in which type of slots may be used.

We decide not to use the contention slot packets as they have low periodicity and may also have collision-induced losses. We also decide not to use data slots as these are allotted only to nodes that have data traffic to send; and explicitly allotting data slots to nodes for sending probe packets seems inefficient. Control packets, on the other hand are broadcasted periodically by each node. We can use these for routing metric computation, at no additional overhead.

For the second aspect of conveying the computed routing metric information to the root node, we use the periodic topology update packets sent in the contention slots toward the root node.

Spatial reuse: In parallel work, we have also integrated LiT MAC with a spatial reuse mechanism, which builds an interference map using the signal to interference ratio (SIR). The SIR is in turn computed using the received signal strength (RSS) profile of neighbours. Like the routing metrics, this information is also collected using control packets, and conveyed to the root as part of the topology update packets.

D. Comparison with prior implementation techniques

We now compare in detail our TDMA MAC implementation technique with two of the prior techniques in literature [4], [3] that have also reported micro-second level time synchronization accuracy. Table II presents the detailed comparison.

As can be seen from Table II, for [4] the nodes in the network are time synchronized to system clock [3]. To achieve time synchronization the software module timestamps the packet with current time before it hands over the packet to WiFi hardware for transmitting it on air. The drawback of this technique is that it does not account for the delay in the transfer of the packet from linux kernel to WiFi hardware, which can be variable. As against this our technique makes use of WiFi hardware beacon timestamps, for synchronization. As our technique makes use of WiFi hardware clock, it does not suffer from synchronization error that can arise due to variable delay in transfer of packets from linux kernel to WiFi hardware as in [4]. Likely due to this reason the maximum synchronization error for our technique is also much less than that for [4] as can be seen from Table II.

For triggering slot boundaries, like [4], we make use of software timers provided by linux kernel. But since in our technique we make use of WiFi hardware clock for higher time synchronization accuracy, it is essential to account for the drift in the WiFi hardware clock and system clock in order to make use of linux software timers that are based on system clock. Our technique outlines a mechanism to resolve this non-trivial issue of drift between WiFi hardware clock and system clock.

Another important contribution of our technique, in comparison with [4], is that it outlines a mechanism to make use of 802.11 per-packet hardware acknowledgements for error recovery which is more efficient than bulk acknowledgement/FEC based error recovery used in [6], as we elaborate later in Sec. V-C.

Though [3] also incorporates per-packet acknowledgement based error recovery, however [3] makes use of proprietary programmable hardware and implements the TDMA MAC protocol by carrying out firmware level code changes. As [3] makes use of custom hardware, it is not evident if their technique is applicable to WiFi commodity hardware in general and if yes then how does their technique perform for commodity hardware.

Table I and II together illustrate the primary contribution of our work in comparison with prior work. Having described the various implementation aspects, we now move on to evaluation of various metrics in the next two sections (Sec. V & Sec. VI).

V. Micro-Benchmarks & Parametric Evaluation

In this section, we first look at the micro-benchmark of time synchronization accuracy and packet timing accuracy. We then present parametric studies for the slot size, and the number of per-hop retransmissions.

A. Time synchronization & packet timing accuracy

Time synchronization error: To measure the synchronization error, we use the setup shown in Fig. 3. We have 8 nodes (in range of each other), and enforce a 7-hop linear topology in LiT MAC. A separate laptop node is also kept in the vicinity. The laptop transmits packets every 100ms. As every LiT MAC node is synchronized with the root node, it records the sequence number and the global time (time at the root node) at which each packet from the laptop is received. Now, ignoring propagation delay, all the LiT nodes should receive a given packet from the laptop at the same time. We hence measure synchronization error for a given node as global time recorded by the root node minus the global time recorded by the given node for a given packet from the laptop. We run the experiment for an hour and take the average synchronization error across the laptop's packets received in that duration. We also have continuous backlogged UDP traffic running from the root to node-7, to verify if synchronization accuracy is affected under the presence of heavy load.

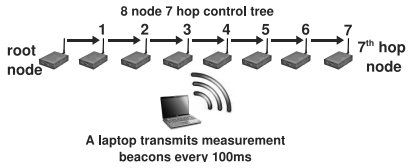


Fig. 3. Set-up to measure time-sync error

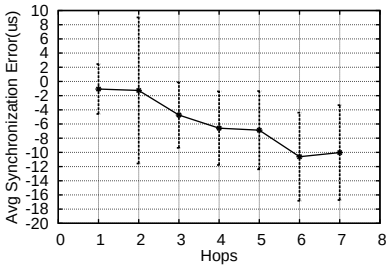


Fig. 4. Time synchronization error (in μs)

Fig. 4 shows the average synchronization error, as a function of the number of hops from the root node; the error bars show standard deviation. As can be seen the synchronization error increases with hop count from the root node, but the average error is only about $-10\mu s$ even at 7 hops. The negative sign indicates that the global time as recorded by a node is more than that at the root node. The standard deviation of synchronization error can be seen to be about 4-6 μs except for the node at second hop for which the standard deviation is about 10 μs. The maximum synchronization error was also small, about -31 μs for the seventh hop node and decreased as the hop count decreased.

Thus the benefit of our implementation technique is small synchronization error even at number of hops as large as seven. In comparison the prior work outlined in Table I has been tested over small number of hops (mostly 3-4). Higher synchronization accuracy over large hops is desirable for mesh networks used for extending connectivity over several kilometers. Though, in such networks, number of hops can be reduced by making use of directional antennas mounted on high rise towers (high rise towers are essential to prevent ground reflection and provide fresnel zone clearance). However the cost of high rise towers can be quite high [20]. So to achieve a range of a few kms without high rise towers, the

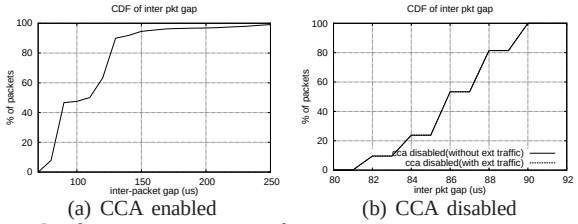
[3]system clock is the clock used by operating system

	Time synchronization	Max hops for time-sync evaln	Max time-sync error	Triggering slot boundaries	Error recovery	Hardware used
Soft-Mac [4]	Nodes synchronized to system clock	Three	84μs	Use of system timers	None	802.11 commodity hardware
TDM-MAC [3]	Nodes synchronized to hardware clock	Two	10μs	Use of hardware timers	Hardware per-packet ACKs	Proprietary programmable hardware
LiT MAC (our work)	Node synchronized to WiFi hardware clock	Seven	-31μs	Use of system timers	WiFi Hardware per-packet ACKs	802.11 commodity hardware

TABLE II
DETAILED COMPARISON WITH PRIOR TDMA MAC IMPLEMENTATION TECHNIQUES

number of hops necessarily has to be large, since each link range will be a few hundreds of metres at most, with the use of 10-12m masts; such masts can be cheaper than high rise towers by up to 2 orders of magnitude [20].

Effectiveness of disabling CCA: It is essential to quantify how accurately the packet transmission time can be controlled with CCA disabled for which we carried out the following experiment. (While [19] gives the mechanism to disable CCA, it does not quantify the accuracy). In a 2-node topology with a root node and a hop-1 node, we have backlogged UDP traffic running from root node to the hop-1 node. The hop-1 node then measures the inter-arrival time of the packets that it receives from the root node, from which it computes the inter-packet gap. We measure the inter-packet gap for 10,000 packets from the root node. We carry out this experiment for the two cases of CCA enabled and disabled. And for the case of CCA disabled, we do experiments both with and without external interference. To produce external interference, we place a laptop in the vicinity of the nodes, and make it connect to WiFi access point, and have continuous data transfer between the laptop and WiFi access point.



(a) CCA enabled (b) CCA disabled

Fig. 5. Gap between successive packets

Fig. 5(a) shows the CDF of the inter-packet gap for the case of CCA enabled (in presence of external interference traffic), while Fig. 5(b) shows the CCA of inter-packet gap for the case of CCA disabled, both in absence as well as presence of external traffic. For the case of CCA enabled, as can be seen from Fig. 5(a), only about 50% of the packets have inter-packet gap less than 90 μs, whereas rest of the packets have inter-packet gap more than 90 μs. While for the case of CCA disabled inter-packet gap is in the range 80 to 90 μs for both the cases of with and without external traffic. For the case of CCA disabled also we observed that the maximum inter-packet gap was about 200 μs, but the inter-packet gap was more than 90μs for only a tiny fraction (less than 0.001%) of the total of ten thousand packets.

Now the expected inter-packet gap is,

$$= SIFS + ack\ transmission\ time\ +\ DIFS$$
$$= 10\ +\ 38.6\ +\ 30\ =\ 78.6\mu s$$

As can be seen from Fig. 5, the measured inter-packet gap is only about 5-10 μs more than the expected time in most cases. Thus, these results indicate that with CCA disabled it is possible to precisely control packet transmission times over commodity WiFi hardware.

Guard time: The guard time for TDMA slots is dependent on three factors: the synchronization error, clock drift between nodes, and the software+hardware delay in transmitting the packet. We have empirically found that a guard time of $100\mu s$ accounts for these factors, and works well in practice [10].

B. Effect of slot size

An important parameter in a TDMA implementation is the slot size, the smallest unit of allocation. Intuitively, a smaller slot size is good for lower delay/jitter, but bad for throughput efficiency due to implementation overheads. To study this quantitatively, we used a 4-hop (5-node) linear topology in indoor setup. In LiT MAC, we set: *num. control slots* = 5, *num. contention slots* = 2, and *num. data slots* = 96. As there is no interference in 802.11a in our lab, we carried out this experiment on channel 149 at the data rate of 6M of 802.11a.

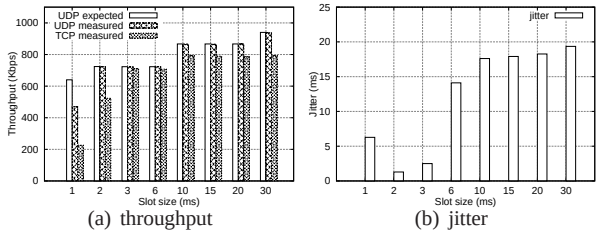
Using iperf v2.0.4 tool, we generate backlogged uni-directional traffic from the root to the 4^{th} hop node. We measure the throughput and jitter for UDP, as well as TCP throughput. (UDP and TCP are in separate experiments). We vary the slot size from 1ms to 30ms. Each experiment duration is 30 seconds and the packet size with MAC layer overheads is 1586 bytes for slot sizes of 3ms or more, while the packet size is 520 and 1120 bytes for slot sizes of 1 and 2ms, as the slot sizes of 1 and 2ms cannot accommodate 1586 byte packet at data rate of 6M. We report the average across 5 experiments in each case.

In these experiments, we do not have any spatial reuse, and use a per-node slot allocation scheme: each of the 5 nodes get a data slot in turn: root, followed by the hop-1 node, then the hop-2 node, and so on[4].

Fig. 6 shows the throughput and jitter for different slot sizes. Fig. 6 also shows the expected UDP throughput, computed by taking into account the LiT MAC header as well as the 802.11 header that we attach over the LiT MAC header to enable per packet acknowledgement.

As can be seen from Fig. 6(a), the UDP throughput increases as the slot size increases. There are two reasons for this. First, the fixed guard time of $100\mu s$ between the slots, constitutes lower overhead for larger slot sizes: the guard time overhead for a 1ms slot size is about 10%, while for slot size of 30ms it is about 0.3%. The second reason is wastage internal to the slot, which arises since we do not (at this time) support MAC layer fragmentation in our implementation. To

[4]We also tried a slot allocation where the order of slots allotted is hop-4 node, followed by the hop-3 node, ..., root. The results were not very different from what we report here.



(a) throughput (b) jitter

Fig. 6. Effect of slot size on throughput and jitter

elaborate, the packet transmission time for a 1586 byte packet at 6M, including acknowledgement, is about $2224\mu s$. In a 3ms slot, only one packet is transmitted, the idle time being $(3000 - 100 - 2224) = 676\mu s$, which leads to inefficiency of about 22%. Whereas for 30ms slots, 13 packets are transmitted leading to inefficiency of only about 3% and thereby the throughput for the 30ms slot size case is higher.

Importantly, the throughput results show that the measured UDP throughput is close to the expected value, thereby implying that the driver-based implementation mechanism for TDMA outlined in Sec. IV is working properly, without CPU or memory bottlenecks.

The TCP throughput also increases with the slot size, but the increase in TCP throughput is lesser than that for UDP throughput. This is since for TCP we transmit both the TCP data packet as well as the TCP ACK in the same data slot. So for a particular slot size the overhead of TCP ACK transmission can cause fewer number of TCP data packets to be transmitted in that slot than the number of UDP packets, leading to lesser TCP throughput.

As can be seen from the Fig. 6(b), the jitter value is very small for slot sizes upto 3ms, whereas for other slot sizes the jitter is about 15-20ms. This can be explained as follows. For slot size up to 3ms, only one packet is transmitted per slot, and the inter-packet delay is fixed: equal to the schedule length i.e. 5 slots. This leads to low jitter. But consider the case of slot size of 6ms, wherein two packets are transmitted in a slot. Now the delay between receiving the two consecutive packets that are transmitted in the same slot is very small, corresponds to the inter-packet gap of $90\mu s$. But the delay between the second packet of a slot and the first packet of the next slot is equal to schedule length, which is much larger. Thus the variation in delay between receiving the consecutive packets (i.e. jitter) is more for such scenarios. We however note that jitter values of 15-20ms are well within the tolerable limits for real-time applications.

CPU bottleneck for slot size of 1ms: As can be seen from Fig. 6(a) the throughput for a slot size of 1ms is about 200 Kbps less than the expected throughput, with a packet loss of 4%. The jitter value for 1ms slot size is also slightly more than that for 2ms and 3ms. Unexpectedly, we observed a packet loss of about 4% irrespective of enabling or disabling per-hop retransmissions.

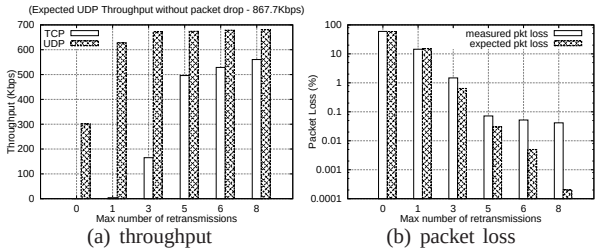
To understand this, we first checked if the reason for the packet loss could be loss of synchronization or slot overlaps. We measured the time-sync error as well as the error in triggering the packet transmission event, but we observed that these errors were similar to that as observed for slot sizes of 2ms or more.

On more detailed investigation, we observed that the packets were being dropped at the MAC driver level, as the receive interrupts for the packet were being missed! Now since the packet loss is similar for the cases of packet retransmissions enabled or disabled, it is likely that the CPU acts as a bottleneck for processing of the received packets, since the timer trigger frequency for the slot size of 1ms is relatively high. To further verify this, we carried out measurements at a data rate of 24M, with slot size as 1ms, under two cases: (a) transmitting only one packet of 1586 bytes per slot, and (b) 3 packets of 350 bytes in a slot. We observed that for the latter case the packet loss was about three times (12-15%) that of the former case (4-5%). This indeed indicates that, at 1ms slot size, the CPU is not able to handle both the timer trigger events and packet reception events leading to packet loss. It is worth noting here that the CPU speed of the Mikrotik RB433AH platform we used was 680MHz; it had 256MB of RAM, but as we have seen here, memory is not a bottleneck.

While we have identified this CPU bottleneck to smaller slot sizes, as noted earlier, the jitter values of 15-20ms seen for larger slot sizes are acceptable for real-time applications.

C. Number of per-hop retransmissions

It is essential to understand the effect of varying the maximum number of per-hop retransmissions, on throughput and jitter. To study this, we use the same 4-hop (5-node) linear topology as above, and the same LiT MAC parameters as earlier. We use a slot size of $10ms$. We emulate various degrees of packet losses, by probabilistically changing the destination MAC address of a packet to a "black hole" (non-existent) MAC address just before sending to the hardware. We then introduce backlogged traffic, as earlier, from the root node to the 4^{th} hop node.



(a) throughput (b) packet loss

Fig. 7. Effect of increasing per-hop retx limit

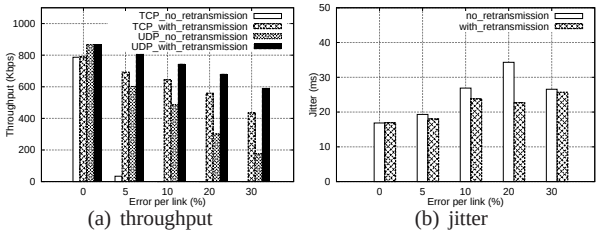
Fig. 7 shows the measured UDP & TCP throughputs as well as measured and expected packet loss, for different values of the maximum number of per-hop retransmissions. Here we emulated a loss rate of 20% in each of the 4 links. As can be seen, the TCP throughput increases significantly as the maximum number of retransmissions is increased. And, as is expected the UDP throughput value does not change much by increasing the number of retransmissions. Also, the measured packet loss is quite close to the expected value as can be from Fig. 7(b) (note that y-axis in Fig. 7(b) uses log-scale). For the case of number of retransmissions 8, the measured packet loss is slightly more but it is still under 0.05% of the expected packet loss. The measured packet loss is slightly more for this case likely due to hardware imperfections. For e.g. we observed that sometimes the notification by hardware,

at the driver level, about whether ACK was received or not was missed causing the packet to be dropped.

The expected UDP throughput without packet drop is 867.7Kbps. With retransmissions enabled, for the case of 3 or more retransmissions when the measured packet loss is small, the measured UDP throughput should be 20% less than 867.7Kbps i.e. 694.2Kbps. This is since each node retransmits about 20% of the packets, the drop in throughput is 20%. As can be seen, the measured UDP throughput for the cases of 3 or more retransmissions is about 675Kbps which is only about 20Kbps less than the expected value of 694.2Kbps. Thus these results indicate that our scheme of per-packet acknowledgement with retransmission of the lost packet in the next data slot is efficient in controlling packet loss without affecting the throughput. And choosing a higher per-hop retransmission limit results in higher TCP throughput along with lower packet loss.

While we do not show the results here, but we observed that jitter values do not change much by increasing the number of retransmissions.

To study the effectiveness of the retransmission scheme with increasing (emulated) error rate, we carried out similar experiments, while fixing the retransmission limit to 8. Fig. 8 shows the UDP/TCP throughput as well as jitter for different (emulated) error rates per link, for the both cases of with and without per-hop retransmissions. As can be seen, the benefit of incorporating retransmissions is particularly apparent in case of TCP throughput wherein without retransmissions TCP throughput is practically 0 for link error rates of 5% or more, whereas with retransmissions even at 30% error rate in each link, the TCP throughput is slightly more than half of the TCP throughput at 0% emulated error rate. Jitter values too are slightly better, with retransmissions as compared to without per-hop retransmissions.



(a) throughput (b) jitter

Fig. 8. Effect of increasing error rates per link

Comparison with Wildnet: [6] employs bulk ACKs and adaptive FEC based error recovery schemes. In the bulk ACK scheme, at the end of a given data slot, the receiver transmits a single ACK packet for all the data packets received in that particular slot. This bulk ACK can be piggybacked with data packets of the receiver. But if the receiver does not have any data packets to send, a dedicated slot allocation is needed at the end of each data slot to transmit bulk ACK itself. This constitutes significant overhead as compared with our per-packet ACK scheme which incurs an per-packet overhead of only about 48 μs (SIFS + ack tx time) at the 6M data rate.

FEC based scheme results in lower end to end delay compared with bulk ACKs or per-packet ACKs, but incurs higher throughput overhead. For e.g. to reduce packet loss from 10% to 1%, FEC incurs about 30% throughput overhead, while to reduce the packet loss from 30% to 1% it incurs 100% throughput overhead [6]. In comparison, for our scheme, as can be seen from Fig. 8, to reduce the packet loss from 10% to $\simeq$0% the throughput overhead is about 24%, while to reduce the packet loss from 30% to $\simeq$0% the overhead is only about 40%.

VI. Overall Performance Evaluation

We now study the overall performance of LiT MAC in a realistic setting of an outdoor testbed. We also evaluate the latency and control overheads associated with the use of soft-state in LiT MAC, using a 15-node indoor testbed. The hardware and software specifications for the experimental set-up is same as that given in Sec. IV, except that for our outdoor testbed, we use 15dBi omni-directional antennas installed on building rooftops, whereas for the indoor testbed we use 2dBi omni-directional internal antennas connected to the nodes. We used 802.11g in both testbeds, and there was significant and variable external interference in many instances. Such external interference causes packet losses in LiT MAC's control, contention, and data slots, and this stress tests protocol operation in realistic scenarios. Fig. 9 shows the outdoor testbed. The indoor testbed is deployed in our department building and is spread across three floors.

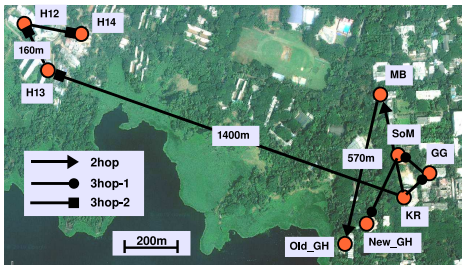


Fig. 9. Nine node outdoor testbed

A. LiT MAC evaluation

Long-term evaluation: We carried out measurements, spread over a long duration (7 days), to study how effectively LiT MAC can fulfill application QoS requirements in a realistic setting of the outdoor testbed. The long duration is significant since we wish to check if the TDMA system develops problems such as loss of synchronization over time, or is able to perform stably.

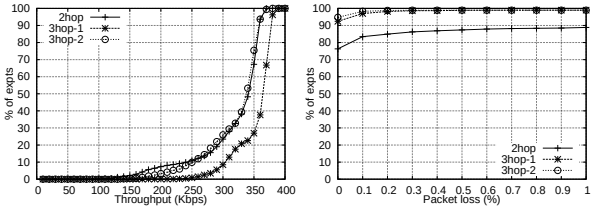
The data traffic we used was: we started three simultaneous uni-directional CBR UDP flows of 400 Kbps source rate, for a duration of one minute, across the following three paths: KR-MB-Old_GH (2hop), KR-GG-SOM-New_GH(3hop-1) and KR-H13-H12-H14 (3hop-2). For each of these flows we measured the throughput, packet loss and jitter. This one minute experiment was then repeated after an interval of five minutes. This process was repeated for a duration of 7 days.

The values of the TDMA MAC parameters that we used are given in Table III. Now, the application level throughput depends to a large extent on the scheduling algorithm used. Since the focus of the current work is to evaluate the protocol performance, and since the MAC itself can support any centralized scheduling algorithm, we use the following flow

Slot duration	5ms
Num. control slots	2
Num. contention slots	1
Num. data slots	33
Max. num. retransmissions	8
Control overhead	$\frac{2+1}{2+1+33} = 8.33\%$
Data rate	6M (802.11g)
Channel	8
Testbed	Outdoor

TABLE III
TDMA MAC PARAMETER VALUES

dependent slot allocation scheme. Since all the three flows start at the root node (i.e. node KR), the root node is allotted the first three slots in the data schedule. Then all the first hop nodes are allotted one slot each in the data schedule, then the second hop nodes and so on. Now the schedule length (i.e. number of slots after which the schedule is repeated) for this slot allocation scheme is 11 slots, so the end to end delay incurred by each flow is about 11 x 5 = 55ms (in the absence of losses).



(a) Throughput CDF (b) Packet loss CDF

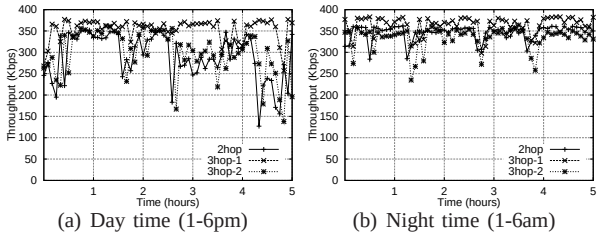
(c) Jitter CDF

Fig. 10. Application performance for CBR traffic

Fig. 10 shows the CDF of throughput, packet loss and jitter for the 1 minute experiments carried out for the period of 7 days. Now, based on the TDMA MAC parameter values, the expected throughput for each of the flow is 392 Kbps. As can be seen from Fig. 10, for the flow 3hop-1, about 90% of the experiments, while for the other two flows about 75% of the experiments have throughput over 300 Kbps. The packet loss for over 95% of the experiments is less than 0.1% for the flows 3hop-1 and 3hop-2, while for the flow 2hop, the packet loss is less than 1% for about 90% of the experiments. The 80th %-ile jitter is about 40ms, 70ms, and 100ms for the flows 3hop-1, 2hop, and 3hop-2 respectively.

A good fraction of flows 2hop show high packet loss despite the per-hop retransmissions. Closer observation showed that this was because the flow 2hop experienced bursty packet losses, especially for experiments during the day time when the external interference was generally more.

To ascertain the reason for relatively lesser throughput for the flows 2hop and 3hop-2 (as can be seen in Fig. 10(a)), we analyzed the measurements at various times of the day. Typically during day time the presence of external WiFi interference is more in the campus. So we observed that for the experiments carried out during day time the throughput for two flows was relatively less compared with the flow 3hop-1, as can be seen from Fig. 11 which shows the measured throughput for the experiments carried out during day and night time, for the 5^{th} day of the 7 day experiment. As against this, for the experiments carried out during night time, the throughput for the three flows is closer to one another, as the presence of external interference is less. We note that it is difficult to devise a metric that can accurately quantify amount of external interference present in the network [5]. But since the performance during night time, when external interference is generally less, is much better than that during day time, it indicates that less than expected throughput especially for flows 2hop and 3hop-2 is largely due to presence of external interference rather than other factors.



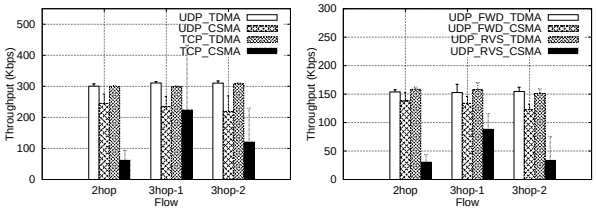
(a) Day time (1-6pm) (b) Night time (1-6am)

Fig. 11. Throughput variation across time of day

The important conclusion from the above study is that despite periods of heavy interference and the corresponding packet losses, LiT MAC operated stably, without nodes losing sync with each other. In most situations, the throughput is high, as well as packet loss and jitter values are low. The merit of TDMA performance for application QoS is further highlighted below, in comparison with CSMA.

Comparison with CSMA: [9] presents detailed performance comparison of LiT MAC with 802.11 CSMA/CA for voice traffic. In this work we evaluate the application level benefits of LiT MAC over 802.11 CSMA/CA in terms of throughput, packet loss and jitter for CBR UDP and TCP traffic.

We carry out measurements across the three paths 2hop, 3hop-1 and 3hop-2, mentioned above. We keep the values of different LiT MAC parameters as shown in Table III, except the slot duration which we change to 3ms. Accordingly, Fig. 12 shows the throughput values for the three flows. The error bars in the graph represent the standard deviation.



(a) uni-directional (b) bi-directional UDP

Fig. 12. Comparison of TDMA and CSMA

Now the expected throughput for TDMA for uni-directional UDP is 326 Kbps, whereas for bi-directional UDP it is 163 Kbps for each flow. As can be seen from Fig. 12 the measured throughputs for TDMA are quite close to the expected value. However for CSMA the throughput values are typically less than that for TDMA. The difference between TDMA and CSMA is particularly apparent in case of uni-directional TCP and bi-directional UDP transfers. Not only is the TCP and

[5]External interference is largely due to Internet traffic which is unpredictable [21] and hence so is external interference.

bi-directional UDP throughput for CSMA less than that for TDMA, but the variation in throughput across different flows is also much more. For e.g. for TCP, for the flow 3hop-1 the CSMA throughput is about 200 Kbps, while for the flow 2hop the CSMA throughput is only about 60 Kbps.

Flow	Average Packet loss (%) (stddev)					
	UDP_UNI		UDP_BI_FWD		UDP_BI_RVS	
	CSMA	TDMA	CSMA	TDMA	CSMA	TDMA
2hop	17.7 (9.8)	0(0)	13.5 (9.4)	0(0)	81 (8.2)	0(0)
3hop-1	19.7 (10.4)	0.1 (0.1)	14.2 (8.0)	0.1 (0.1)	44.4 (17.5)	0.4 (0.5)
3hop-2	25.8 (14.9)	0(0)	20.2 (6.4)	0(0)	78.2 (26.9)	0(0)

TABLE IV
TDMA VS CSMA: PACKET LOSS STATISTICS

Slot duration	5ms
Num. control slots	3
Num. contention slots	2
Num. data slots	45
Max num. retransmissions	5
Probability of transmission in contention slot	$\frac{\text{maxlevel}-\text{nodelevel}+1}{\text{numnodes}}$
Flow renewal interval	10 frames
Topology update interval	20 sec
Control overhead	$\frac{3+2}{3+2+45} = 10\%$
Data rate	6M (802.11g)
Channel	8
Testbed	Indoor

TABLE V
TDMA MAC PARAMETER VALUES

Apart from the throughput results above, we observed that average jitter values for TDMA were about 15-30ms, while the same for CSMA were 2-3 times higher. More importantly, the difference in average packet loss statistics are stark, as shown in Table IV. While the packet loss for CSMA can be as high as 80%, TDMA has negligible packet loss, under 0.5% in most cases. Thus these results highlight the application level benefits of using TDMA for mesh networks, in comparison with CSMA. CSMA was never designed for such use in mesh networks and shows poor performance.

Latency metrics and soft-state overheads: LiT MAC uses soft-state based mechanisms for aspects such as node-join, flow-setup, etc. (Sec. III). How does this affect metrics such as flow-setup latency, control overhead, etc.? To study this, we carried out experiments in our 15-node indoor testbed.

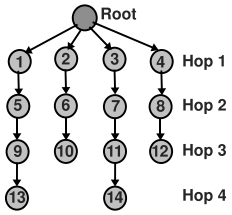


Fig. 13. Control tree for indoor testbed

To analyze various latency related metrics, in the 15-node network, we enforce the control tree as shown in Fig. 13. The control tree is enforced by ensuring that a given node selects only a particular other node as a parent during node join. After a node boots and joins the network, it then initiates a flow request after a random interval of a few seconds. This flow continues throughout the experiment. We run the experiment for half an hour and measure the following latency related metrics: node join latency, flow set-up latency, flow renewal and topology update latency.

The node join latency is the time taken by the newly started node to become part of the network, after it hears the first control packet. The flow set-up latency is the time between when the first flow request packet is sent on air and when its response is received. Flow renewal latency is the time between when the flow renewal request is sent on air and when it reaches the root node; topology update latency is defined likewise.

The TDMA MAC parameter values that we have used are shown in Table V. As can be seen, every node sends a flow renewal request 10 frames after the last flow renewal request was sent. Topology update packets are sent with a period of 20 seconds. Since the nodes that are closer to the root forward more flow renewal and topology update packets, the probability of transmission that we have used for contention slots takes into account the node level (i.e. hop count from the root node). The simulation based analysis in [9], shows that this approach for computing the probability of transmission results in low contention slot collisions.

Though [9] has also carried out evaluation for above mentioned metrics the essential difference of our evaluation, apart from the type of hardware used, is that we do not assume anything about the type of traffic, while [9] has carried out evaluation specifically for real time voice. Due to these reasons the TDMA MAC parameter values in our setting are different. Another difference is that [9] has carried out evaluation for sparse traffic wherein only few voice calls are active simultaneously while we assume worst case traffic wherein all the nodes in the network are active simultaneously.

The results of our experiment are shown in Table VI. We show the latencies in terms of number of frames rather than in seconds. This is since, for the same number of data, control and contention slots in a frame if the slot duration is changed, frame duration will change and so will the latencies.

As can be seen from Table VI, node join and flow set-up latency increases as the node level (i.e. number of hops from root node) increases. For the 4^{th} hop nodes (nodes 13 and 14), the node join latency is slightly more than 25 frames, while flow set-up latency for nodes 13 & 14 are about 11 & 18 frames respectively. Also the flow renewal and topology update latencies are about 12-13 frames for 4^{th} hop nodes.

These results can guide in the setting of appropriate frame duration, depending on the application requirement. For e.g. if the network needs to support a large number of short lived flows, then frame duration can be kept small. For a frame duration of 50ms and considering the flow set-up latency for 4^{th} hop node to be 20 frames, the flow set-up latency for 4^{th} hop nodes is about 1s.

Now, average flow renewal latency for 4^{th} hop nodes is about 13 frames; if we take the timeout for terminating a flow as 3 x 13 = 39 frames, then for a flow that has ended and its flow termination request is lost, the root has to wait for 39 x 50ms $\approx$ 2 sec before de-allocating data slots for that particular flow. This is the cost of using soft-state mechanism for flow termination. We see that this inefficiency due to data slots remaining allotted to a flow that has terminated, is quite small.

Our measurement results show that with a 10% control

Node Num	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Node level	1	1	1	1	2	2	2	2	3	3	3	3	4	4
Join latency (# frames)	2.4	1.8	5.3	5.4	7.7	3.6	4.8	9.6	18.3	11.8	10.8	12.3	27.2	26.4
Flow set-up latency (# frames)	2.9	3.0	1.9	3.9	8.9	10.0	5.9	6.0	7.9	13.0	8.0	24.0	10.9	17.9
Avg. flow renewal latency (# frames) (std. deviation)	0.7 (2.0)	0.4 (1.3)	0.2 (1.0)	1.0 (2.3)	2.7 (2.9)	3.6 (3.5)	3.4 (3.2)	3.3 (3.2)	6.1 (4.0)	6.9 (4.3)	7.2 (4.8)	6.1 (4.1)	12.4 (6.7)	12.7 (7.9)
Avg. topology update latency (# frames) (std. deviation)	0.6 (1.8)	0.4 (1.5)	0.4 (0.8)	1.0 (2.1)	2.9 (3.2)	3.5 (3.2)	3.4 (3.5)	2.7 (3.1)	5.8 (3.9)	7.5 (4.6)	7.2 (4.7)	6.7 (5.1)	12.3 (6.8)	11.6 (5.8)

TABLE VI
VARIOUS LATENCY METRICS AS MEASURED IN THE INDOOR TESTBED

overhead for a 15-node mesh network, the various latency metrics are within a few seconds even at four hops. These latencies are acceptable for long duration flows that last for few minutes or more. However, for shorter duration flows that last for say 5-10 seconds, such as HTTP transfer, latency of the order of seconds for setting up a flow may not be acceptable. But we note that such short flows usually will not require slot reservations, and hence can be handled through other means. For instance, one may use a pre-setup best-effort flow-id, which can accommodate such short flows.

B. Routing stability evaluation

As mentioned in Sec. I, routing stability is a key factor in application performance; LiT MAC's TDMA slot allocations will lose meaning if there is high route instability. Having integrated routing metric computation with LiT MAC (Sec. IV), we have compared the performance of the following three routing metrics: ETX [11], ROMA [12] and SLIQ [13] as regards routing stability.

We carry out measurements for 33 links in our 9-node outdoor testbed for this study, with links operating at the 802.11g 6M data rate on channel 8. The values for the different TDMA MAC parameters that we used are *num. control slots* = 9, *num. contention slots* = 3, and *num. data slots* = 8. We keep the slot duration as 5ms, thus forming a 100ms frame, while the experiment duration was 2000 seconds.

As mentioned in Sec. IV, we use control packets of the LiT MAC protocol as probe packets. The essential difference of SLIQ compared with ETX and ROMA is that SLIQ is computed over 200 probe packets while ETX and ROMA are computed over 10 probe packets sent at a periodicity of 1sec. Thus for computing SLIQ metric we use the control packet transmitted in every frame (i.e. every 100ms) thereby updating the SLIQ metric value every 20 seconds. For computing ETX and ROMA we use the control packets transmitted every 10th frame. Each node, in our experiment, logs the packet delivery information of the control packets received from the neighbouring nodes and computes the value of the three metrics accordingly.

We now compare the performance of the three metrics in terms of four aspects: link trigger rate, dominant route prevalence, route persistence, and route flaps.

(i) Link trigger rate: A trigger is generated whenever the current link metric differs from the previous value by at least a threshold percentage. Using a 10% threshold, Fig. 14(a) shows the CDF of trigger rate computed over all links. As can be seen, the trigger rate for ROMA and SLIQ is significantly smaller than that of ETX. For ETX, only about 10% of the links show trigger rate less than 10 per 1000s, while for ROMA and SLIQ, about 80% and 100% of links have trigger rate less than 10 per 1000s, respectively. We observed that the mean trigger rate for SLIQ is only 3 per 1000s whereas it is 13 and 42 respectively for ROMA and ETX. Although we do not show results here, we note here that we also compared the performance at lower threshold values (i.e. $< 10\%$). We observed that in such cases ROMA performs worse than ETX, while SLIQ continues to perform well in absolute terms, and best in relative terms.

(ii) Dominant route prevalence: Among all the routes between a given source destination pair, the route which is prevalent for the longest percentage of time is the dominant route. The CDF of the prevalence of the dominant route, across all source-destination pairs, is shown in Fig. 14(b). As can be seen, for about 91% of the cases with SLIQ, the dominant route prevalence is 90% or more. For ETX only about 73% whereas for ROMA only about 63% of routes have dominant route prevalence of over 90%. Thus these results further indicate that SLIQ is more stable than ETX or ROMA.

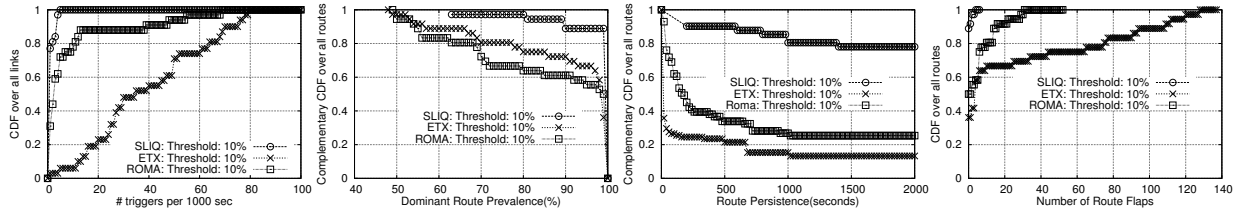
(iii) Route persistence: We now analyze the mean of the route persistence, which is measured as the duration of time a given route lasts. Fig. 14(c) shows the mean route persistence for the three metrics at 10% trigger threshold. We can see that in case of SLIQ, more than 91% of the routes persist longer than 500 seconds, whereas for ROMA and ETX only about 36% and 22% of routes persist longer than 500 seconds. Also the mean route persistence for SLIQ is 1319 seconds, whereas for ROMA and ETX it is 219 and 66 seconds respectively.

(iv) Number of route flaps: We also evaluated the number route flaps for the three metrics, over the duration of 2000 seconds. Route flap refers to a change in the route between the given source-destination pair. As can be seen from Fig. 14(d), the number of route flaps for SLIQ, for all the source-destination pairs is less than 10. On the other hand for ROMA about 20% of the routes have more than 10 route flaps, whereas in case of ETX about 40% of the routes have more than 10 route flaps over the 2000 seconds.

On detailed analysis of routes selected, we observed that most route flaps occur for the routes that involve links with *intermediate* quality. *Intermediate* quality links are the links with PDR (Packet Delivery Ratio) between 10 and 90%. From Fig. 14(c), it is evident that SLIQ is capable of masking the link variability and hence can select stable routes.

Summary of routing stability results: The above results uniformly show that the SLIQ metric performs significantly better than ETX and ROMA as regards routing stability.

SLIQ performs better than ETX, even if we directly compare with the results in [17]. For instance, at the 10% threshold, [17] reports a median value (i.e. 50%-ile) of 30min



(a) Trigger rate CDF (b) Dominant route prevalence (c) Mean route persistence (d) Num. route flaps in 2000s

Fig. 14. Comparison of routing metrics at the 10% link trigger threshold

(i.e. 1,800s) for the persistence of the dominant route, in the Roofnet outdoor testbed. Whereas as Fig. 14(c) indicates, SLIQ's mean route persistence itself is about 2,000s for about 80% of the routes (the dominant route persistence can only be expected to be higher).

The reason for higher stability of SLIQ, as highlighted in [13], is that it uses more number of packets (200 packets) than ETX and ROMA (10 packets each) to compute the metric value and so can factor out the short term variations in link quality, resulting in higher stability. Though in ignoring short term variations in link quality, SLIQ can also ignore high throughput paths that can occur for short time. Tracking such short term changes is possible through approaches such as Opportunistic Routing [22], but this we believe is not appropriate for real-time applications like voice/video which have strict delay/jitter requirements. For such applications routing stability offered by SLIQ, can enable LiT MAC to give good and stable performance [23].

C. Spatial reuse

Spatial reuse is a key technique for improving performance in mesh networks; this refers to the use of the same TDMA slot, in the same channel, by two links in the network. Such spatial reuse requires interference mapping: i.e. information on which links interfere with which other links. We have integrated LiT MAC with an SIR-based interference mapping [24] as described in Sec. IV. While the evaluation of the interference mapping technique itself is the topic of parallel work, beyond the scope of this paper, here we briefly present results to demonstrate that LiT MAC can support spatial reuse.

In our outdoor testbed, we identify four 3-hop paths which support spatial reuse across the first and last hops (within each path). That is, two of these links can be allotted the same time slot. In a given experiment, we use one of the four 3-hop paths, and setup a uni-directional or bi-directional UDP flow. We use a slot size of 10ms, *num. control slots = 4*, *num. contention slots = 2*, *num. data slots = 74*, and a MAC layer pkt size of 1100 bytes. We fix all links at the 6M 802.11g data rate, using channel 8.

Table VII shows the throughput results for the uni-directional and bi-directional UDP flows for the four different paths. In our experiments of Sec. VI-A, we scheduled three flows simultaneously, which represents the heavy load condition. As is well known, CSMA performs poorly under heavy load [1]. But in this case we schedule the flows across the four paths individually, so it represents a relatively low load condition. But even in this condition, TDMA after incorporating the spatial reuse, performs much better than

Path	CSMA Thghput (Kbps)		TDMA Thghput (Kbps)		Thghput improvement(%)	
	uni	bi	uni	bi	uni	bi
KR-SOM-H12-H13	1520.2	394.5	1735.1	637.1	14.1	61.5
H13-H12-SOM-KR	1465.1	401.6	1585.2	650.6	8.2	62.0
GG-SOM-H12-H13	1280.4	397.0	1590.3	604.0	24.2	52.1
H13-H12-SOM-GG	1522.1	399.6	1517.1	628.3	-0.3	57.2

TABLE VII
COMPARISON OF CSMA AND TDMA WITH SPATIAL REUSE USING UNI-DIRECTIONAL (UNI) AND BI-DIRECTIONAL (BI) UDP TRANSFER

CSMA. The benefit is particularly apparent in case of bi-directional UDP transfer wherein the throughput improvement of TDMA with spatial reuse is as much as 60%.

These results quantify the performance improvement due to spatial reuse compared with CSMA and further highlight the benefit of LiT MAC. In other experiments, detailed in parallel work, we observed that TDMA with spatial reuse gives close to the expected throughput improvement, compared with TDMA without spatial reuse.

VII. DISCUSSION AND CONCLUSION

Despite the popularity of WiFi mesh networks among the research community as well as in deployments, providing QoS in such networks remains an elusive goal. We take the stance that a TDMA-based approach is necessary to achieve this. But multi-hop wireless TDMA presents several challenges. Time synchronization across multiple hops, in presence of wireless channel losses is the first issue. Next, any TDMA schedule has to be reliably and consistently disseminated to the network nodes. Supporting the use of multiple channels of operation in this context is a further challenge, and so is integration of aspects such as spatial reuse and dynamic routing.

In this work we have implemented and evaluated a full-fledged TDMA MAC, LiT MAC [9]. LiT MAC takes a centralized approach, and makes extensive use of soft-state mechanisms to address the above challenges. Our implementation is on top of commodity WiFi hardware, which lets us leverage the low-cost benefits of the same. We have evaluated various aspects of the implementation, using indoor as well as outdoor testbeds.

Our results indicate that LiT MAC can achieve time synchronization accuracy of a few μs across several hops. TDMA slot sizes can be as small as 2-5ms. However, for slot sizes smaller than 2ms the CPU becomes a bottleneck affecting performance, due to the high rate of interrupts. We believe that for most applications, a slot size of 2ms should not be an issue.

Outdoor evaluation over long durations has shown that the TDMA MAC design as well as the implementation are stable enough for use in practice. The network nodes are able to

maintain synchronization even in the presence of high internal load and high external interference conditions. This shows the robustness of the soft-state based design.

The soft-state overhead is manageable, and the latencies involved are a few seconds for operations such as a new node joining the network, or a new flow being admitted. The latency of a few seconds is usually acceptable for long-duration flows. Fortunately, short lasting flows such as HTTP downloads lasting 5-10 sec or less typically do not require slot reservation; and these can be handled through a pre-setup "best-effort" flow.

Our routing stability evaluations show that the use of the SLIQ metric results in stable routes, which in combination with the TDMA slotted approach means that applications can expect to get stable performance. And finally, we have also demonstrated that LiT MAC can support spatial reuse, to further enhance network capacity.

REFERENCES

[1] S. Xu and T. Saadawi, "Revealing the problems with 802.11 medium access control protocol in multi-hop wireless ad hoc networks," *Computer Networks*, vol. 38, no. 4, 2002.

[2] K. S. J. Pister and L. Doherty, "TSMP: Time Synchronized Mesh Protocol," in *Distributed Sensor Networks (DSN 2008)*, Nov 2008.

[3] D. Koutsonikolas, T. Salonidis, H. Lundgren, P. LeGuyadec, Y. Hu, and I. Sheriff, "Tdm mac protocol design and implementation for wireless mesh networks," in *CoNEXT 2008*.

[4] P. Djukic and P. Mohapatra, "Soft-TDMAC: A Software TDMA-based MAC over Commodity 802.11 hardware," in *INFOCOM 2009*.

[5] A. Rao and I. Stoica, "An Overlay MAC Layer for 802.11 Networks," in *Mobisys*, Apr 2005.

[6] R. Patra, S. Nedevschi, S. Surana, A. Sheth, L. Subramanian, and E. Brewer, "Wildnet: Design and implementation of high performance wifi based long distance networks." NSDI, 2007.

[7] B. Raman and K. Chebrolu, "Design and evaluation of a new MAC protocol for long-distance 802.11 mesh networks," in *MOBICOM 2005*.

[8] S. Nedevschi, R. K. Patra, S. Surana, S. Ratnasamy, L. Subramanian, and E. Brewer, "An Adaptive, High Performance MAC for Long-Distance Multihop Wireless Networks," in *MobiCom*, Sep 2008.

[9] V. Gabale, B. Raman, K. Chebrolu, and P. Kulkarni, "Lit mac: Addressing the challenges of effective voice communication in a low cost, low power wireless mesh network," in *ACM DEV*, 2010.

[10] A. Dhekne, N. Uchat, and B. Raman, "Implementation and evaluation of a tdma mac for wifi-based rural mesh networks," in *NSDR 2009*.

[11] D. Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," *Wireless Networks*, vol. 11, no. 4, 2005.

[12] A. Dhananjay, H. Zhang, J. Li, and L. Subramanian, "Practical, distributed channel assignment and routing in dual-radio mesh networks," *SIGCOMM*, 2009.

[13] K. Chebrolu and A. Mishra, "Loss behavior analysis and its application in design of link quality metrics," in *COMSNETS*, 2011.

[14] A. Sharma and E. Belding, "Freemac: framework for multi-channel mac development on 802.11 hardware," in *ACM PRESTO*, 2008.

[15] M. Neufeld, J. Fifield, C. Doerr, A. Sheth, and D. Grunwald, "Softmac-flexible wireless research platform," in *Proc. HotNets-IV*, 2005.

[16] A. Sharma, M. Tiwari, and H. Zheng, "Madmac: Building a reconfiguration radio testbed using commodity 802.11 hardware," in *IEEE SDR'06*.

[17] K. Ramachandran, I. Sheriff, E. Belding, and K. Almeroth, "Routing stability in static wireless mesh networks," *Passive and Active Network Measurement*, 2007.

[18] S. Das, H. Pucha, K. Papagiannaki, and Y. Hu, "Studying wireless routing link metric dynamics," in *IMC*, 2007.

[19] E. Anderson, C. Phillips, G. Yee, D. Sicker, and D. Grunwald, "Challenges in deploying steerable wireless testbeds," in *TridentCom*, 2010.

[20] S. Sen and B. Raman, "Long distance wireless mesh network planning: problem formulation and solution," in *World Wide Web*. ACM, 2007.

[21] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the self-similar nature of Ethernet traffic (extended version)," *IEEE/ACM Transactions on Networking (ToN)*, 1994.

[22] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft, "Xors in the air: practical wireless network coding," in *SIGCOMM*, 2006.

[23] P. Joshi, "Design, Implementation and Evaluation of Stability Aware Routing for WiFi Mesh Networks," Master's thesis, IIT Bombay, 2011.

[24] B. Raman and R. Jain, "SIR-Based Interference-Maps for TDMA-Based Outdoor Mesh Networks," in *Invited paper, LANMAN 2010*, 2010.

Vishal Sevani Vishal Sevani received his B.E in Electronics Engineering and M.E in Electronics and Telecommunication Engineering from Mumbai University (India) in 2002 and 2006 respectively. After completing his Masters, he worked as a Software Engineer for two years. He then joined Computer Science and Engineering Dept. of Indian Institute of Technology, Bombay (India) in 2008 to pursue his PhD. His PhD topic is on optimizing performance in TDMA based wireless mesh networks. His research interests are broadly in the area of communication networks, specifically wireless networks.

Bhaskaran Raman Bhaskaran Raman received his B.Tech in Computer Science and Engineering from Indian Institute of Technology, Madras in May 1997. He received his M.S. and Ph.D. in Computer Science from University of California, Berkeley, in 1999 and 2002 respectively. He was a faculty in the CSE department at Indian Institute of Technology, Kanpur (India) from June 2003. Since July 2007, he is a faculty at the CSE department at Indian Institute of Technology, Bombay (India). His research interests and expertise are in communication networks, wireless/mobile networks, large-scale Internet-based systems, and Internet middleware services. His current focus and specific interest is in appropriate technology in the domain of communication and computing for rural use.

Piyush Joshi Piyush Joshi received his B.Tech in Computer Science and Engineering from Shri Guru Gobind Singhji Institute of Enginnering and Technology (India) in 2009 and his M.Tech in Computer Science and Engineering from Indian Institue of Technology, Bombay (India) in 2011. His research interest include wireless communication, sensor and ad-hoc networks and distributed systems. He is presently working as a Network Software Engineer with Intel Technologies Pvt Ltd and his current focus area is in providing connectivity solutions for mobile platforms.