TDM MAC Protocol Design and Implementation for Wireless Mesh Networks

Dimitrios Koutsonikolas[1], Theodoros Salonidis[2], Henrik Lundgren[2], Pascal LeGuyadec[2], Y. Charlie Hu[1], Irfan Sheriff[3]

[1]Purdue University, [2]Thomson, [3]UC Santa Barbara

ABSTRACT

We present the design, implementation, and evaluation of a Time Division Multiplex (TDM) MAC protocol for multi-hop wireless mesh networks using a programmable wireless platform. Extensive research has been devoted to optimal scheduling algorithms for multi-hop wireless networks assuming a perfect TDM MAC protocol. However, the problem of designing and implementing such a protocol has not received due attention. We introduce a design framework that addresses the three main challenges that comprise this problem: (i) How to calibrate and optimize the TDM MAC protocol parameters given a wireless platform, (ii) how to achieve network-wide synchronization with high accuracy, minimal overhead, and most importantly, bounded delay, and (iii) how to integrate the synchronization algorithm with the TDM MAC protocol state machine using minimal hardware resources. We apply our design framework to our platform and evaluate the resulting TDM MAC protocol through controlled experiments in a wireless mesh testbed. The results demonstrate the protocol's ability to provide fairness and graceful performance degradation under packet losses and multi-hop traffic patterns that arise in mesh network deployments.

1. INTRODUCTION

Mesh networks aim to provide high-speed/low-cost Internet access through 802.11-based multi-hop wireless backbones. Despite their rapid world-wide adoption, initial deployment experiences report various performance problems related to the 802.11 CSMA MAC protocol. Indeed, CSMA MAC protocols react poorly to the heavy traffic load encountered in the mesh backbone and to the losses or inconsistent channel state induced by the multi-hop wireless environment. In addition, due to their distributed nature, 802.11 CSMA MAC protocols make it notoriously hard to model the network's capacity and throughput even given the network topology and traffic pattern.

The alternative TDM MAC presents an almost opposite scenario to the CSMA MAC. On one hand, it assumes global knowledge of the multihop network, and hence a centralized scheduling can compute optimal or close-to-optimal schedules of transmissions for all nodes in the network. On the other hand, it poses three significant challenges to the design of an operational mesh network: First, the backbone mesh routers need to be tightly synchronized so they can perform local packet transmissions that follow the global transmission schedule. Second, the backbone routers need to agree upon and follow a time slot that is small but takes into account not only the synchronization error but also delays of incoming and outgoing packets while they traverse various stages of the network stack. Third, the integration of the TDM parameters with the synchronization algorithm, while also minimizing the hardware resource usage, is non-trivial.

In this paper, we systematically address the above three challenges and present the design, implementation, and evaluation of a Time Division Multiplex (TDM) MAC protocol for multi-hop wireless mesh networks using a programmable 802.11 wireless platform. Our contributions are as follows.

First, we identify the bottlenecks that govern the operation of a TDM MAC protocol. These bottlenecks are visible only at the lowest layers of the protocol stack and introduce delays in the form of clock drifts, slot processing and transmission/reception (Tx-Rx) overhead. These bottlenecks must be measured and taken into account in TDM MAC design to achieve correct and efficient protocol operation. We introduce a design framework that includes a set of experimental methods and design constraints. The experimental methods are used to measure platform communication bottlenecks. The design constraints relate these bottlenecks to the TDM MAC protocol parameters and allow to compute the optimal parameters for correct and efficient protocol operation. We then introduce a design procedure to apply our design framework. This procedure takes as input a mesh network topology, a programmable platform and a set of requirements and uses measurements and design constraints to yield optimized protocol parameters.

Second, we design and implement a novel in-band multihop clock synchronization algorithm to maintain the node clocks synchronized to a common time reference provided by the mesh gateway nodes. The algorithm addresses the TDM MAC requirements for network-wide synchronization,

ACM CoNEXT 2008, December 10-12, 2008, Madrid, SPAIN

bounded delay execution and high clock accuracy (at the microsecond level). It utilizes the TDM protocol's slotted structure to quickly broadcast the reference clock, minimizing execution time and synchronization overhead. Accurate clock synchronization can be achieved if nodes are synchronized to an external global clock using GPS devices [17]. However, GPS devices only work in outdoor settings with a clear sky view. Existing in-band clock synchronization algorithms for multi-hop wireless networks are focused on high accuracy and do not minimize execution time or communication overhead, a critical factor in a TDM system.

Third, we present an integration scheme that integrates the fundamental TDM parameters with the clock synchronization algorithm in the design of the slotted mechanism of any TDM protocol. We provide a general state machine for the implementation of this mechanism. Our implementation makes minimal hardware resource usage, by utilizing only two timers to handle the problem of imperfect clocks and the need for periodic re-synchronization.

Finally, we prototype a version of our TDM MAC protocol and evaluate it on a wireless testbed, in traffic scenarios that arise in gateway-centric mesh network applications. Our results show that our TDM MAC protocol achieves inter-flow fairness and graceful throughput to packet loss conditions typically encountered in mesh networks [2].

2. RELATED WORK

2.1 TDM MAC implementations

Recent work has leveraged off-the-shelf 802.11 hardware to design TDM MAC protocols for multi-hop wireless networks.

Overlay MAC Layer (OML) [13] provides an overlay TDM solution on top of 802.11 CSMA, in which time is loosely divided in slots. It focuses on slot allocation to nodes according to a WFQ policy to improve the fairness of 802.11 as opposed to issues in a native design of TDM protocols. As such, OML uses a relative large slot size equal to the time required to transmit 10 packets of maximum packet size to tolerate course-grained clock synchronization.

SoftMAC [9] and *MadMAC* [16] are two platforms that can be used to build experimental MAC protocols on top of the 802.11 PHY layer. As a demo, the authors in both cases showed how to implement a simple TDM protocol, and tested it for 2 nodes.

Similar to our work, RT-Link [15] also studies the design issues of a TDM protocol, but for energy constrained sensor networks. RT-Link resorts to an out-of-band, hardware-based synchronization, using either an AM transmitter-receiver module or a WWVB atomic clock broadcast. In contrast, our work focuses on a TDM design that uses in-band synchronization.

WiFi Based Long Distance (WiLD) networks [12, 10] faced similar issues as in a TDM MAC protocol design. These networks comprise of point-to-point wireless links that use high-gain directional antennas. Instead of network-wide synchronization, the major synchronization issue is to locally synchronize all interfaces of the same node to either receive or transmit at a given time, and to synchronize among neighbor nodes so they can transmit or receive together.

Finally, most of the above implementations are at the driver level using off-the-shelf 802.11 wireless cards. Our work presents the first comprehensive study of the challenges faced by multi-hop TDM MAC protocol design at firmware level. The design needs to take into account delays introduced as incoming and outgoing packets traverse the network stack and due to hardware and low-level software constraints, in addition to synchronization error.

2.2 Synchronization algorithms

Clock synchronization mechanisms are broadly classified as out-of-band and in-band. Out-of-band mechanisms like GPS clocks can provide a global time reference with nano-second level accuracy and incur zero overhead but only work in outdoor environments with clear sky view. In-band mechanisms synchronize nodes through beacon transmissions and are used in existing single-hop wireless TDM systems like cellular and IEEE 802.16. In these systems, clock synchronization is easily achieved since the base station can reach all client stations with a single broadcast transmission. However, in multi-hop wireless TDM networks, in-band clock synchronization is a challenging task.

Standard Internet synchronization algorithms such as NTP [8] yield low accuracy (milli-second scale) when applied to multi-hop wireless environments. Consequently, several in-band multi-hop synchronization algorithms have been designed, especially for sensor network applications [14]. From these algorithms, [4, 5, 7] aim at global time reference and have been implemented on sensor network platforms. These algorithms can achieve high accuracy (tens of μs) but have not been designed for TDM MAC protocols. Instead, they operate on top of CSMA MAC protocols which do not allow for bounded execution time.

The IEEE 802.16j relay task group has extended the IEEE 802.16 standard to multi-hop TDM networks of tree structure, where children nodes recursively synchronize to preambles transmitted by their parent nodes. This approach is specific to a tree topology structure and is not a complete synchronization algorithm. Chebrolu et al. [3] build a sensor network for a bridge monitoring application and use a synchronization mechanism where beacons are transmitted over lossless links of a routing tree. In Section 4, we show that synchronization over a tree may not be the best approach for the high-loss mesh network environment.

The ECMA standard [1] is based on UWB PHY and TDMA MAC and defines a distributed in-band clock synchronization mechanism during each TDM frame where each node synchronizes its clock to the clock value of its slowest one-hop neighbor. While in a single-hop network the slowest clock can be propagated to all nodes within a single TDM frame, in a multi-hop network it would require an unbounded number of TDM frames.

In contrast to previous work, our in-band synchronization algorithm is designed for a multi-hop TDM MAC protocol and uses an optimized transmission schedule to broadcast a synchronization beacon, within a single TDM frame and in bounded delay, from a set of gateway nodes to the mesh network. Furthermore, in Section 6 we show that the algorithm's simplicity allows efficient integration with the TDM MAC state machine.

3. TDM PROTOCOL ARCHITECTURE

A wireless mesh network consists of Internet Gateways (GWs) and mesh access points (MAPs). Both GWs and MAPs are equipped with wireless interfaces to route traffic within the mesh backbone and serve client devices. The GWs are equipped with additional Ethernet interfaces to connect to the Internet.

We consider a TDM MAC protocol for the wireless mesh backbone where the participating nodes are GWs and MAPs.[1] Fig. 1 depicts the protocol's structure. The GWs and MAPs are synchronized under a common time reference. Transmissions occur periodically through a sequence of multi-slot frames. Each frame consists of a synchronization control sub-frame (SCS) followed by a sequence of data sub-frames (DS). Both SCS and DS are of fixed duration and span several slots. During each DS slot, a set of GWs and MAPs transmit conflict-free according to a schedule that has been computed by a central entity. The GWs provide a natural location for such an entity. The basic parameters of a mesh

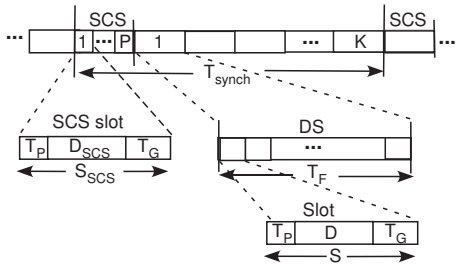


Figure 1: TDM MAC protocol slotted structure of the multi-hop wireless mesh backbone.

TDM MAC protocol are as follows.

Slot duration S**.** The minimum TDM MAC protocol communication time unit. During a slot a node must be able to successfully transmit (Tx) or receive (Rx) packets from its neighbors. The slot duration S should be minimized to enable provision of delay guarantees.

Slot packet duration D**.** The maximum MAC packet duration. It is determined by the minimum required data transmission rate and a range of required packet sizes. It should also enable full utilization of the useful part of each slot (slot duration minus overhead) with one or more packet transmissions.

Slot guard time T_G**.** Idle time appended at the end of each slot to enable correct protocol operation during Tx or Rx activity. It provides a time cushion for hardware-related communication delays and slot misalignments due to clock drifts. Guard time is a per-slot overhead, hence critical to minimize for efficient protocol operation.

SCS duration T_{SCS}**.** The execution time of the synchronization algorithm. It should be constant and known to all nodes during TDM MAC protocol operation. It should also be minimized to avoid disruption of delay-sensitive transmissions and minimize synchronization overhead.

Synchronization period T_{synch}**.** This parameter determines frequency of synchronization algorithm execution. It should be maximized to minimize the synchronization overhead and depends on losses and clock drifts.

The TDM MAC protocol parameters depend on application requirements, the hardware platform and the synchronization algorithm and also depend on each other. For example, a very large T_G can effectively mask all hardware bottlenecks and clock drifts but would also increase synchronization and slot overhead. It also presents challenges to the multi-hop synchronization algorithm to maintain low synchronization overhead and meet the T_{SCS} delay bounds. Increasing S reduces protocol overhead but also reduces the protocol ability to provide delay guarantees. In the next section we introduce a design framework to capture such dependencies and address the wireless mesh TDM MAC design problem.

[1]The clients are connected to the backbone through a different wireless interface and protocol.

4. DESIGN FRAMEWORK

Our TDM MAC protocol design framework consists of design constraints that relate measured bottlenecks of a programmable wireless platform to the protocol parameters. We distinguish between platform design constraints and synchronization design constraints. Platform design constraints ensure correct and efficient protocol operation for transmissions during both SCS and DS. They determine guard time, packet duration and slot duration as a function of the platform bottlenecks. Synchronization design constraints ensure correct and efficient operation of the synchronization algorithm during the SCS. They determine the synchronization period and SCS duration as a function of guard time, clock drift rate and synchronization failures due to packet losses.

We first identify the platform bottlenecks and then introduce the design constraints. We conclude with a design procedure that takes as input a programmable platform, a mesh network topology and a set of requirements, and uses measurements and the design constraints to yield optimized TDM MAC protocol parameters.

4.1 Platform bottlenecks

The platform bottlenecks that impact TDM MAC protocol design are (i) clock drift rate, (ii) packet preparation overhead, (iii) slot processing overhead, and (iv) communication turnaround overheads (RxRx, TxRx, RxTx, and TxTx). Bottlenecks (ii)-(iv) include hardware and low-level software processing delays that are not directly visible and cannot be controlled at the level where MAC design is performed. These bottlenecks can only be determined by measurements and should be defined and measured at the level where MAC design is performed.

Clock drift rate r_d**.** Clock drift rate r_d is the speed a MAP node clock deviates with respect to a reference clock. The drift rate of clocks based typical crystal oscillators can range from 1-100μs per second.

Slot processing T_P**.** Each slot involves processing the following actions before any Rx or Tx activity can begin: (i) schedule the next slot time instant (typically through a timer) (ii) determine the current slot type (Rx or Tx), and (iii) in case of Tx slot, prepare the hardware for transmission. Although hardware transmission preparation does not occur for Rx slots, T_P must take this action into account since slot duration is independent of slot type. We therefore define slot processing T_P as the duration from the start of a slot until the

first bit is in the air.

RxRx turnaround T_{RxRx}. Consecutive Rx slots require no change in hardware state. Therefore, no additional slot overhead is incurred by two or more consecutive Rx slots. We thus safely omit T_{RxRx} in further analysis.

TxRx turnaround T_{TxRx}. We define this turnaround time as the time from the last bit sent in the air until the time an incoming frame can be received.

RxTx turnaround T_{RxTx}. We define this bottleneck as the time from the reception of the last bit until the first bit of the outgoing frame is transmitted in the air. In addition to the delay for the hardware to change state from receive mode to transmit mode (turn on power amplifier, etc.), the RxTx turnaround includes the same hardware Tx preparation delay as in T_P.

Data packet preparation time T_{Dpp}. Each platform has a maximum rate at which incoming packets from higher layers can be prepared to enter the MAC protocol queue. This rate is typically controlled by lower level software processes and needs to be measured. This rate determines the packet preparation time T_{Dpp}, which is the minimum time for consecutive packet arrivals at the MAC queue.

TxTx turnaround T_{TxTx}. We define this bottleneck as the time from the last bit in the air for an outgoing packet until the first bit in the air of the next packet. This delay includes the delay from programming the hardware for transmission until the packet is transmitted in the air (part of T_P). It also depends on the platform's ability to continuously feed the MAC queue with prepared packets, i.e., it depends on T_{Dpp}.

4.2 Platform Design Constraints

We make the following assumptions for the turnaround overheads: (i) changing state from Rx to Tx (or vice versa) includes a small hardware-related delay to change the hardware state, (ii) Tx slots include delay to prepare and program the hardware for transmission, and for Tx slots which are preceded by a Rx slot, the delay of (i) is added. This leads to the following relationships for the turnaround overheads: $T_{RxRx} < T_{TxRx} < T_{RxTx}$. Furthermore, since an Rx activity is assumed to finish within its slot, the T_{RxTx} delay starts at latest at the slot boundary. This implies that $T_{RxTx} \leq T_P$.

We thus use the remaining turnaround bottlenecks T_P and T_{TxTx}, and T_{Dpp} to derive the platform design constraints.

The slot processing overhead T_P is unavoidable at the start of each slot. This provides a design constraint that relates T_P to S, D and T_G:

$$S = T_P + D + T_G \tag{1}$$

We now derive a constraint for the TDM MAC protocol ability to transmit a stream of packets in consecutive TX slots. Incoming packets from higher layers are queued at a packet preparation module that has service time T_{Dpp}. Prepared packets enter the TDM MAC protocol queue. For consecutive Tx slots, the TDM MAC protocol generates transmission requests every S seconds. The first prepared packet in the MAC queue (if any) begins transmission after a delay equal to the slot processing overhead T_P. Given a stream of incoming packets to the packet preparation queue, the service rate of the MAC queue should be at least equal to the service rate of the packet preparation queue:

$$S \geq T_{Dpp} \tag{2}$$

If this constraint is satisfied, the MAC protocol is always able to serve packets incoming to the packet preparation queue at each Tx slot. In this case, the MAC queue will be full and each prepared packet will be transmitted after T_P, as expected. However, if this condition is not satisfied, the protocol cannot operate correctly for consecutive Tx slots. Its behavior depends on whether the implementation allows transmissions of prepared packets that arrive at the MAC queue after the beginning of a slot or waits for the next Tx slot. In the first case, packets may exceed the slot boundaries resulting in incorrect operation. In the second case, Tx slots are wasted resulting in performance degradation.

If constraint (2) is satisfied, consecutive prepared packet transmissions are separated by T_{Dpp} or higher depending on the packet duration D. For D less than $T_{Dpp} - T_P$, T_{TxTx} equals $T_{Dpp} - D$; otherwise it equals T_P. Since T_{TxTx} only depends on T_{Dpp}, D and T_P, it does not need to be measured or be explicitly taken into account in the platform design constraints.

Using Eq. (1) and (2) we reach a platform design constraint for guard time T_G:

$$T_G \geq max(T_{Dpp} - T_P - D, 0) \tag{3}$$

The max() operation covers the case $D > T_{Dpp} - T_P$, where the platform can continuously prepare packets for transmission within the duration of slot processing and packet duration. In this case, the guard time due to platform bottlenecks can be set to zero. Otherwise, the guard time should be such that the T_{Dpp} minimum packet separation is satisfied. The constraints on T_G due to the clock drift rate r_d are captured by the synchronization design constraints described next.

4.3 Synchronization Design Constraints

We determine the design constraints for the synchronization parameters T_{SCS} and T_{synch}. Our analysis is independent of the mechanics of a particular synchronization algorithm. The algorithm uses beacons to synchronize all MAP clocks to a common time reference clock provided by the GWs. It is executed during P slots of the SCS and executions occur every T_{synch} slots. During each execution, some MAPs may not receive a synchronization beacon and need to wait until the next execution. We say that the network *de-synchronizes* if the clock drift between at least one node pair (MAP-MAP or MAP-GW) exceeds the guard time T_G. Let rd_{max} be the maximum clock drift rate in the network. Then, after T_G/rd_{max} the clock offset of the node pair with maximum drift rate will exceed T_G and the network will de-synchronize. We now derive the constraints for each parameter that ensure the synchronization algorithm operates correctly, synchronization overhead is minimized and the network does not de-synchronize.

SCS duration T_{SCS}. The synchronization algorithm is executed during the SCS. Therefore, T_{SCS} is determined by P and the duration of each SCS slot:

$$T_{SCS} = P(T_P + D_{SCS} + T_G) \tag{4}$$

For real-time applications like video and voice, T_{SCS} should not exceed a delay bound T_{SCS}^{max}. Also, since synchroniza-

tion beacons are transmitted during the SCS, the node clock offsets should not exceed T_G during the algorithm execution. Therefore, T_{SCS} should be less than T_G/rd_{max} for correct algorithm operation. However, T_{SCS} is by definition less than T_{synch}, which should also be less than T_G/rd_{max}. Therefore, the main constraint for T_{SCS} is the delay constraint T_{SCS}^{max}.

$$T_{SCS} < T_{SCS}^{max} \tag{5}$$

Synchronization period T_{synch}. From Fig. 1, T_{synch} should be greater than T_{SCS} plus the duration of at least one DS.

$$T_{synch} > T_{SCS} + T_F^{max} \tag{6}$$

where T_F^{max} is an upper bound on delay of the DS.

To avoid network de-synchronization T_{synch} should be less than T_G/rd_{max}. To minimize synchronization overhead, T_{synch} should be maximized. However, executing the synchronization algorithm every T_G/rd_{max} would only apply to a lossless network. In the practical case where losses exist, not all nodes will receive a beacon in a single algorithm execution. To capture the effect of losses, we assume that each synchronization algorithm execution has a constant failure probability p that is independent of previous executions. The execution fails if at least one node does not receive a beacon. Under this assumption, the probability of network de-synchronization $P_{desynch}$ is:

$$P_{desynch} = p^{\lfloor \frac{T_G/rd_{max}}{T_{synch}} \rfloor} \tag{7}$$

Given a reliability upper bound ϵ on $P_{desynch}$, (7) yields an upper bound for T_{synch} that takes losses into account.

$$T_{synch} < \frac{T_G}{rd_{max}} \frac{\log p}{\log \epsilon} \tag{8}$$

Combining Eqs. (6) and (8), we reach the design constraints for the synchronization period.

$$T_{SCS} + T_F^{max} < T_{synch} < \frac{T_G}{rd_{max}} \frac{\log p}{\log \epsilon} \tag{9}$$

The ratio $\frac{\log p}{\log \epsilon}$ is a loss penalty because it restricts T_{synch} to be less than the lossless bound T_G/rd_{max}. Fig. 2 plots this ratio within the $[1\% - 99\%]$ range of the algorithm execution failure probability p, for $\epsilon = 10^{-6}$ and $\epsilon = 10^{-4}$. Link packet losses in mesh networks are in the range 10% and above [2]. This implies even higher p for a multi-hop synchronization algorithm. For $\epsilon = 10^{-6}$, the range $p > 10\%$ corresponds to more than 85% reduction for $\frac{T_G}{rd_{max}}$.

Recall that our analysis assumes p is constant and independent across algorithm executions. This assumption applies to a synchronization algorithm where the clock reference is propagated from a GW to all MAPs through a tree structure. Since the beacon must reach all nodes in the tree, p is determined by the maximum-loss path in the tree. This leads to two observations. First, tree-based synchronization approaches are not the best solution for the loss rates observed in mesh networks. Second, the ratio $\frac{\log p}{\log \epsilon}$ yields a conservative upper bound to tune T_{synch} when designing a clock synchronization algorithm. Our synchronization algorithm in Section 5 is inspired by both observations.

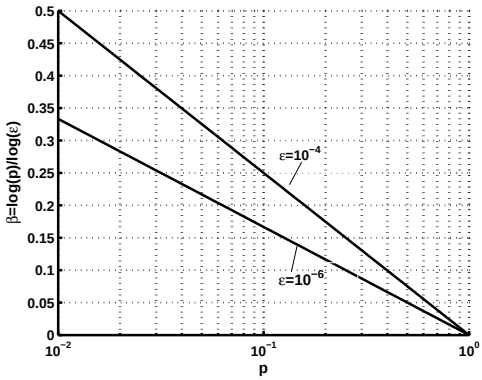


Figure 2: T_{synch} **loss penalty as a function of synchronization algorithm failure probability** p**.**

4.4 Design Procedure

We now present a design procedure that takes as input a hardware platform, a mesh network topology and a set of application requirements and yields TDM MAC protocol parameters that minimize protocol overhead.

The application requirements consist of maximum SCS duration T_{SCS}^{max}, maximum DS duration T_F^{max}, synchronization reliability ϵ, and bounds on DS packet duration D^{min}, D^{max} and SCS packet duration $D_{SCS}^{min}, D_{SCS}^{max}$. The platform may support multiple data rates that correspond to different PHY modulation schemes. The synchronization algorithm operation during the SCS is characterized by execution time of P slots and a separate failure probability p for each data rate.

The design procedure determines the MAC protocol parameters $T_G, T_{synch}, D, D_{SCS}, S, T_F$ which minimize protocol overhead, meet the above requirements and satisfy the platform and synchronization design constraints. The design procedure consists of the following steps.

Step 1. Perform bottleneck measurements on the platform to determine T_P, T_{Dpp} and rd_{max}.

Step 2. Determine T_G, T_{synch}, D and D_{SCS}, that minimize protocol overhead, by solving the following optimization problem:

MIN_OVERHEAD:

$$\text{Minimize } \frac{P(T_P + D_{SCS} + T_G)}{T_{synch}} + \frac{T_P + T_G}{T_P + D + T_G}$$

Subject to:

$$max(0, T_{Dpp} - T_P - D) \leq T_G < \frac{T_{SCS}^{max} - P(T_P + D_{SCS})}{P}$$

$$T_F^{max} + P(T_P + D_{SCS} + T_G) < T_{synch} < \frac{T_G \cdot \log p}{rd_{max} \cdot \log \epsilon}$$

$$D^{min} \leq D \leq D^{max}$$

$$D_{SCS}^{min} \leq D_{SCS} \leq D_{SCS}^{max}$$

Step 3. Compute S using T_G, D, and T_P in (1). Compute T_F as an integer multiple of S. Finally, adjust T_{synch} as T_{SCS} plus an integer multiple of T_F:

$$T_F = S\lfloor \frac{T_F^{max}}{S} \rfloor \tag{10}$$

$$T_{synch}^{final} = T_{SCS} + TF\lfloor\frac{T_{synch} - T_{SCS}}{T_F}\rfloor \quad (11)$$

The design procedure is flexible and has low complexity. It uses a few parameters to characterize the platform and synchronization algorithm and incorporates multiple packet sizes and multiple data rates. $MIN\_OVERHEAD$ uses linear constraints and only a few variables (T_G, T_{synch}, D and D_{SCS}). Hence, it can be solved efficiently using a standard optimization solver like MATLAB or CPLEX. The impact of different synchronization algorithms can be compared by solving $MIN\_OVERHEAD$ for each parameter pair (P,p).

Before the design procedure can be applied to our system, the synchronization algorithm that yields input parameters (P, p) must be specified. We describe our synchronization algorithm in the next section.

5. SYNCHRONIZATION ALGORITHM

In this section, we present an in-band synchronization algorithm for TDM multi-hop wireless networks that maintains the clocks of all MAPs accurately synchronized to a common time reference provided by the GWs. The GWs are synchronized to this time reference through out-of-band synchronization like GPS clocks. The algorithm is executed during the SCS and utilizes the TDM slotted structure to broadcast a synchronization beacon from the GWs to the MAPs in bounded time. For simplicity, we present an algorithm version that uses a single GW.

5.1 Algorithm Description

During the SCS, all nodes use a pre-computed schedule of P slots. During each slot, a single node transmits and the rest are in receive mode. The first slot is assigned to the GW. Each node transmits at most once during the SCS. Fig. 3 shows the schedule that determines the node transmission sequence during the SCS.

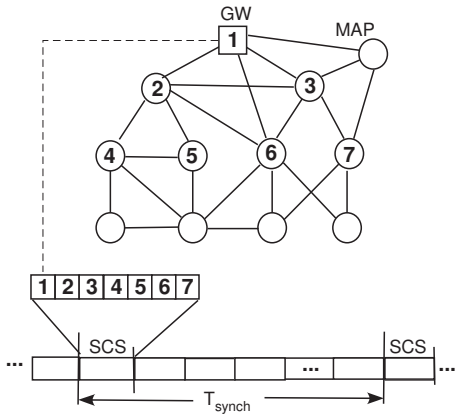


Figure 3: An example 7-slot SCS for a 12-node mesh network (1 GW and 11 MAPs). Links denote ability to communicate (potentially with losses). Numbers in nodes denote their assigned Tx slot in the SCS.

The algorithm begins when the GW broadcasts a beacon containing a timestamp of its current clock value at the first slot of the SCS. Each node that receives the beacon, provided it is the first beacon it has received in the current SCS, computes an estimate of its clock offset to the GW clock and determines the beginning of the next Data Subframe (DS) based on the GW clock. It then re-broadcasts the beacon (unmodified) during its assigned slot in the SCS. The node discards other beacons it receives during the SCS. At the end of the SCS all nodes update their clocks based on their updated offset estimates.

We now describe in detail the offset computation at each node upon reception of a beacon and the clock update actions performed by all nodes at the end of the SCS when the algorithm terminates.

Offset computation. Suppose node n receives its first beacon at SCS slot i from node i. Let $T_n(L)$ be the local clock value of node n when the beacon is received. Node n first computes an estimate $T_i(L)$ of the time when node i sent the beacon:

$$T_i(L) = T_n(L) - \delta \quad (12)$$

where δ is an estimate of the total delay for a beacon transmission from node i to node n. We measure δ offline using an experimental method that emulates a sender-receiver synchronization handshake protocol similar to [8]. The estimation uncertainty of δ is minimal due to the following reasons. First, we leverage our TDM implementation to remove non-deterministic medium access delay. Second, we timestamp the beacon at the MAC layer, bypassing uncertainties due to higher layers of the network stack at the sender. Third, propagation delay is negligible for distances encountered in typical wireless mesh networks. For example, in our implementation over 802.11 hardware the clock granularity is $1\mu s$ which equals the propagation delay at 300m.

Given $T_i(L)$, node n computes its estimate $T_s(L)$ of the time when the GW node sent the beacon:

$$T_s(L) = T_i(L) - (i-1)S_{SCS} \quad (13)$$

where S_{SCS} is the SCS slot duration. Finally, node n uses the beacon timestamp value $T_s(R)$ to estimate its offset with respect to the GW clock.

$$Offset = T_s(R) - T_s(L) \quad (14)$$

Algorithm termination (clock updates). At the end of slot P of the SCS, each node n updates its local clock based on the newly computed offset. The reason for all nodes to simultaneously update their clocks at the end and not during the SCS is to avoid de-synchronization due to nodes updating their clocks with new clock values before others.

5.2 SCS slot sequence computation

Our synchronization algorithm requires each node to transmit at most once during the SCS. This property helps minimize the algorithm execution time. However, it introduces the problem of computing a sequence of node transmissions to disseminate the synchronization beacon to all nodes in the most reliable manner. The problem is combinatorial. In a mesh network of one GW and N MAPs, the optimal solution can be found by enumerating and testing N! sequences– complexity becomes prohibitive as the network size increases.

In this section we provide a heuristic to construct the SCS slot sequence. The algorithm uses a set of measured packet

loss rates of all links in the mesh network. Such measurements can be performed at $O(N)$ complexity using broadcast packets [6, 11]. Our algorithm uses the link loss rates p_l to construct a tree rooted at the GW that yields a maximum reliability path (or minimum loss path) toward each MAP. The reliability of each path s is the product of packet success rates of its intermediate links l: $\prod_{l \in s}(1-p_l)$. Maximizing path reliability from GW to each MAP is equivalent to maximizing log path reliability. Log path reliability results to addition of log success rates over the path. Hence, the tree can be constructed by running N times Dijkstra's shortest path algorithm between the GW and each MAP. We estimate the synchronization algorithm failure probability p as $1 - \prod_{l \in s_m}(1-p_l)$, where s_m is the minimum-reliability (maximum-loss) path in the tree.

The slots of the SCS sequence are assigned to nodes by traversing the tree in a breadth-first manner. For each node's slot, this assignment ensures at least one neighbor node transmitted a beacon at a previous slot. The leaf nodes do not need to be assigned an SCS slot because they are covered by the rest of the nodes. As an example, the SCS sequence in Fig. 3 has been constructed assuming certain packet loss rates on the links. We emphasize that the tree is constructed only to determine the SCS slot sequence. During the algorithm execution the beacon is disseminated through broadcasting.

5.3 Discussion

The algorithm execution time P may vary from 1 slot (when all MAPs are within range of the GW) to N slots (when there is only a single path from GW to all MAPs). In large networks execution time can be decreased by running the synchronization algorithm in parallel at multiple network regions, where each region has been assigned to a different GW and channel. Further reduction in execution time can be achieved by modifying the algorithm to exploit spatial reuse. This would require a more sophisticated slot sequence computation and an interference model that predicts packet losses of links transmitting in parallel; such models can be built and still use O(N) measurements [6, 11]. Finally, if the delay bound on T_{SCS} allows, the synchronization algorithm reliability can also be increased by running multiple copies of the SCS sequence within a single algorithm execution. We plan to investigate these optimizations in our future work.

Our implementation on an 8-node wireless testbed demonstrates that this algorithm synchronizes the network with a very low execution time and negligible overhead (Section 8). Apart from low execution time, the algorithm simplicity allows efficient integration and implementation with the TDM MAC protocol state machine. We proceed to describe this integration in the following section.

6. SYSTEM INTEGRATION

In this section, we integrate our synchronization algorithm with the mesh TDM MAC protocol architecture of Section 3. We aim at a design of low-complexity, using a minimum number of timers. We first describe the two timers we used and then present the integrated protocol flow using a simplified flow diagram of the TDM MAC protocol state machine.

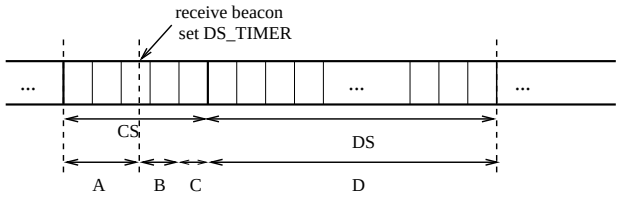


Figure 4: Sequence of phases within a TDM frame.

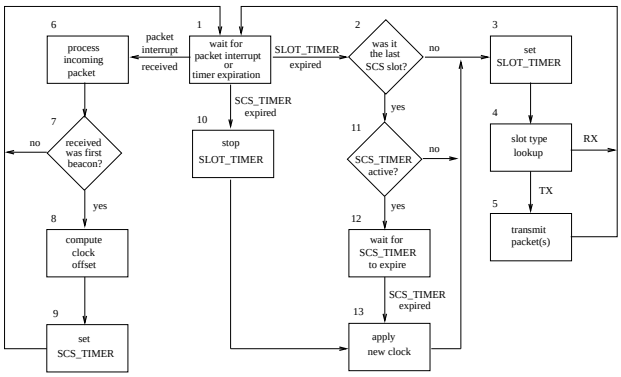


Figure 5: State diagram of the integrated TDM protocol operations.

6.1 Protocol Timers

A system with perfect clocks could implement the TDM protocol operation through synchronization using a single slot timer. However, due to imperfect clocks, additional mechanisms are needed to keep nodes synchronized. We provide a simple design of the integrated system using only two timers. The *SLOT_TIMER* is the basic timer. It is set at the start of each slot and its timeout is equal to the slot duration. Since we need to periodically compute the clock offset and update the clocks, we use an additional timer called *SCS_TIMER*. This timer is set at beacon reception and expires at the end of the SCS according to the GW's clock. This timer ensures that all nodes are synchronized at the end of the SCS.

6.2 Integrated protocol operation

During each SCS, each TDM node transitions through four protocol phases, as illustrated in Figure 4.

- **Phase A:** Node is in SCS and it has not yet received a beacon.
- **Phase B:** Node is in SCS and it has received a beacon. This phase might not exist in a cycle, if the node receives no beacon due to low link quality.
- **Phase C:** Node is in the last slot of SCS.
- **Phase D:** Node is in a DS.

The MAC state diagram in Figure 5 illustrates the flow of protocol transitions. We proceed to describe the flow of each of the phases A-D. In all four phases, the node always starts from state 1, waiting for a timer to expire.

Phase A. In this first phase, a node is in the SCS and it has not yet received a synchronization beacon. For each SCS slot until it receives a beacon, a node's *SLOT_TIMER* expires and the only node action is to set a new *SLOT_TIMER* to expire in the next SCS slot. A node thus follows the flow $[1, 2, 3, 4, 1]$ in Fig. 5. Upon the first beacon reception in the SCS, the node performs the clock offset computation relative to the GW's clock. It then uses this offset to set the *SCS_TIMER* to expire at the end of the last SCS slot. Note, however, that for the rest of the SCS, the node will still use its local (possibly drifted) clock. Since the *SCS_TIMER* is set using the new clock offset, it will expire simultaneously at all nodes (see Phase C). The flow for the beacon reception is $[1, 2, 3, 4, 1, 6, 7, 8, 9, 1]$ in Fig. 5.

Phase B. In this second phase, a node is in an SCS slot other than the last SCS slot and it has already received a synchronization beacon. In all the remaining SCS Rx slots, a node will simply ignore any additional synchronization beacons it may receive (following the flow $[1, 2, 3, 4, 1, 6, 7, 1]$). Upon its own SCS Tx slot, a node will re-broadcast the (first) synchronization beacon (containing the GW's timestamp) that it received in a previous SCS slot. This corresponds to the flow $[1, 2, 3, 4, 5, 1]$ in Fig. 5.

Phase C. In this third phase, the SCS ends and all nodes must apply the new clock and then simultaneously start the first DS slot. At the beginning of the last SCS slot, the *SLOT_TIMER* timer is set to expire according to the node's local clock.

If the node had received a beacon during the SCS, the SCS_TIMER was set to expire in Phase A according to the node's old clock. If a node's clock is faster than the GW clock, the *SLOT_TIMER* will expire before the *SCS_TIMER*. In this case, the node will wait for the *SCS_TIMER* to expire and apply the new clock. If a node's clock is slower than the GW clock, the *SCS_TIMER* expires first and the node then immediately cancels the *SLOT_TIMER* and applies the new clock. Applying the new clock based on the *SCS_TIMER* ensures that, regardless of the individual nodes' clock drifts, all nodes are now synchronized and simultaneously start the DS. The two different flows described above are $[1, 2, 11, 12, 13, 3, ..., 1]$ and $[1, 10, 13, 3, ..., 1]$ in Fig. 5, respectively. After the application of the new clock, the first DS slot starts. The handling of a DS slot is identical independent of whether or not it is the first DS slot. We refer to Phase D for a description of the flows in DS.

If the node had not received a beacon during the SCS, its *SCS_TIMER* was not set in Phase A. Thus, the node will trigger the end of SCS using the *SLOT_TIMER*. This is flow $[1, 2, 11, 3, ..., 1]$ in Fig. 5. The node thus uses its local clock until the next SCS when it receives a beacon.

Phase D. In this final phase a node is in a DS. In this phase, each new slot is triggered by the expiration of the *SLOT_TIMER*. The node sets a new *SLOT_TIMER* and then determines whether the current slot is of Tx or Rx type. Since nodes know their schedule in advance, a fast table lookup for the current slot ID is enough to provide the required information about the current slot[2]. In case of a Tx slot, the node transmits any data packets that are readily available in its outgoing queue and can fit within the current slot. In case of a Rx slot, the node processes any data packet(s) received during the slot. The flows in this phase are $[1, 2, 3, 4, 5, 1]$ and $[1, 2, 3, 4, 1, 6, 7, 1]$ in Fig. 5 for Tx slots and Rx slots, respectively.

[2]The same lookup is performed during SCS in Phases A-C.

7. PROTOTYPE IMPLEMENTATION

In this section, we first overview the architecture of our 802.11 wireless programmable platform. We then highlight the required modifications to its IEEE 802.11 CSMA MAC implementation to realize our TDM MAC protocol on this platform.

7.1 WiLD MAC Platform

WiLD MAC is a standard-compliant IEEE 802.11a/b/g platform. Figure 6 gives an overview of the WiLD MAC architecture. The MAC protocol is divided in Higher MAC and Lower MAC. The Higher MAC handles low-priority tasks such as configuration and management. The Lower MAC is the part where we have implemented the TDM MAC protocol and measured the platform bottlenecks.

The Lower MAC performs time critical tasks such as frame transmission and reception. It primarily consists of the Packet Processing (PP) and the Tx Rx Coordination (TRC) modules. The PP module receives packets from the network layer and performs encryption/decryption, before delivering them to the TRC. The TRC is the last step traversed by a data packet in the data path before it is delivered to the PHY layer. The TRC is also the last point where we can have access to a packet. The TRC implements time-critical 802.11 functions that typically require response time of a few microseconds. It is interrupt-driven and communicates with the Burst Processor (BuP) for packet transmission and reception.

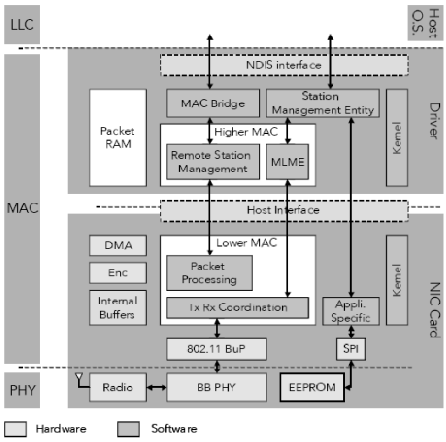


Figure 6: WiLD MAC platform architecture.

The BuP is a dedicated processor for packet transmission and reception, realized in hardware. It implements the most time-critical 802.11 functions (e.g., frame acknowledgement), and contains registers for timers and counters (e.g., backoff and NAV). The TRC decides the appropriate values for these counters and timers, writes the registers and then the BuP performs the countdown. The BuP signals through an interrupt to the TRC when a value has reached zero.

The WiLD MAC firmware is implemented in ANSI-C. In contrast to existing off-the-shelf 802.11 wireless cards, WiLD MAC platform provides access to the complete firmware code. We proceed to describe our modification to the firmware required to realize our protocol.

7.2 TDM MAC Protocol Implementation

In this section, we describe our modifications to the WiLD MAC to realize the TDM MAC protocol in Section 6.

We disabled native 802.11 CSMA protocol functions such as carrier sensing, NAV, backoff, and RTS/CTS. The carrier sensing is disabled/enabled through manipulation of a hardware register. With all the above functions disabled, data packets can be transmitted at a given time instant that we control through the TRC. We modify the TRC to program the BuP to set a "pending transmission flag" as soon as it finished preparing a packet. Then, at the start of each Tx slot the TRC checks this flag. If the flag is set, the TRC programs the BuP for transmission. The transmission will start after the inter-frame spacing requirement has timed out. We were not able to completely disable inter-frame spacing, but we modified it to always take the minimal allowed value, which in our case is equal to SIFS.

Both DS and SCS Tx slots are scheduled by setting hardware timers in the BuP. The *SLOT_TIMER* is set at the start of each slot based on the local clock. The *SCS_TIMER* is set upon reception of a synchronization beacon, using the GW's clock in the beacon. Upon *SCS_TIMER* expiration, the local clocks are synchronized by adjusting their clock offset.

In addition to data packet transmissions and to cope with the high loss rates encountered in mesh networks we implemented an ACK-based retransmission mechanism similar to 802.11. The main difference is that our mechanism does not perform exponential backoff when a packet fails. Depending on the slot duration that we configure, it can retransmit the packet in the same slot or in the next slot.

We proceed to measure our design and protocol parameters and to evaluate our TDM MAC on a testbed.

8. DESIGN AND PERFORMANCE EVALUATION ON A WIRELESS TESTBED

We design and evaluate our TDM MAC protocol on a wireless testbed of 8 nodes deployed in an office building of two floors and a mezzanine (Fig. 7). The first floor contains cubicles with thin separations. The second floor contains multiple rooms separated by both thin and thick walls. Each node consists of a laptop connected via Ethernet to a WiLD board, equipped with an integrated omni-directional antenna. All boards are configured to operate in ad hoc mode with transmission power set to 30mW.

We design our system in an environment where external interference is mitigated. All experiments are performed in channel 60 of 5GHz band (802.11a) during nights and weekends. As an extra measure, before each experiment we used sniffers to ensure that no other 802.11a networks existed in this channel.

8.1 Application of Design Framework

We apply our design framework in Section 4 to determine the TDM MAC protocol parameters for our wireless platform and testbed deployment. We follow the design procedure described in Section 4.4 to account for design constraints due to platform bottlenecks and synchronization algorithm operation under packet losses.

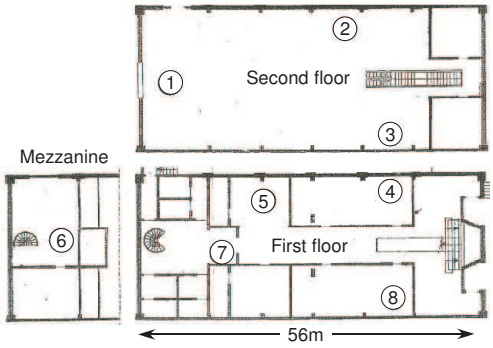


Figure 7: Wireless testbed.

8.1.1 Packet loss measurements and SCS sequence computation

Figure 8 depicts measured packet loss rates for all links in our testbed at 54 Mbps data rate. In each measurement experiment, nodes were sequentially scheduled to continuously transmit broadcast packets for 60 seconds each, while all other nodes recorded the received packets. Each data point is the average for each link across 10 experiments on multiple days. This profile contains links with 0 or 1 packet loss rate but also links of intermediate quality. Similar profiles have been observed in outdoor 802.11 mesh networks [2]. We used these measurements to seed the SCS construction

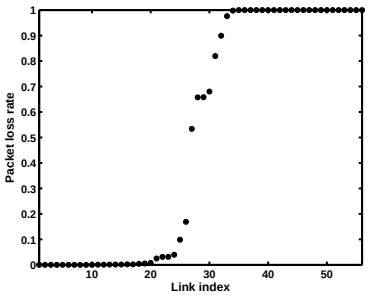


Figure 8: Average link packet loss rates at 54 Mbps in our testbed.

algorithm in Section 5.2. Table 1 shows number of SCS slots P and synchronization algorithm failure probability estimates p for all GW choices. We selected as GW node 5 which provides minimum P and relatively low p.

GW	1	2	3	4	5	6	7	8
P	3	3	3	2	2	4	3	3
p	0.23	0.43	0.33	0.41	0.30	0.62	0.51	0.42

Table 1: SCS construction algorithm number of slots P and failure probability estimates p for all GW choices.

8.1.2 TDM MAC Protocol Parameter Determination

We now apply the procedure of Section 4.4 to design the TDM protocol with the requirements of Table 2, which also depicts the procedure steps and their outcomes. Below we describe our measurement methodologies and findings during each step of the design procedure.

Step 1. We first measure the platform bottlenecks rd_{max}, T_P, and T_{Dpp} on the WiLD boards, before the testbed deployment. The measurement methodologies were implemented at the Lower MAC (where the protocol is implemented) and used high-accuracy timers and interrupts.

Clock drift rate, rd_{max}. We measured the clock drift rate for each node pair A and B by periodically sending timestamped packets from A to B. For each received packet, B computed and stored (i) its local clock (ii) the offset equal to the difference of its local clock and the timestamp in the packet. The clock drift rate of B with respect to A, was computed offline as the slope of the offset evolution over time. This procedure was performed at different locations and times for each pair A and B. Fig. 9 depicts the measured average drift rates for all node pairs. For each node pair, the drift rate has low variability over time and locations. Across node pairs the drift rate ranges from 0.1 to 5.5 $\mu s/s$. In our design, we use the maximum observed drift rate $rd_{max} = 5.5\mu s/s$. .

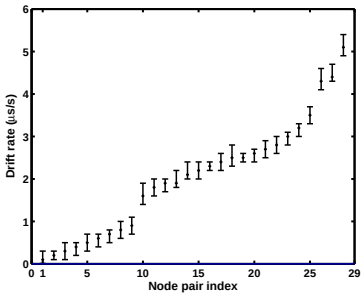


Figure 9: Drift rates for all node pairs AB in our testbed. Only the positive drift rates rd_{AB} are shown (the negative drift rates are $rd_{BA} = -rd_{AB}$)

Slot processing, T_P. We measured T_P on each board as the difference between the *SLOT_TIMER* interrupt signaling the start of the TDM slot and the $T_{txStart}$ interrupt signaling the start of the transmission by the BuP. Extensive measurements on each board and across boards, yielded $T_P = 17 \pm 1\ \mu s$.

Data packet preparation time, T_{Dpp}. We measured T_{Dpp} through back-to-back transmissions of minimal-duration data packets timstamped at sender A and stored by receiver B. The time difference between consecutive timestamps included the slot processing overhead T_P plus packet duration D at node A. Minimal packet duration D exposed the packet preparation bottleneck T_{Dpp} (the system operated under $T_{Dpp} > T_P + D$). Extensive measurements at each board and across boards, yielded $T_{Dpp} = 104 \pm 1\ \mu s$.

Step 2. In our implementation, we use 802.11 beacons ($L_{SCS} = 52$ bytes) and the maximum 802.11 MAC packet (1530 bytes) for each TDM slot. According to 802.11a OFDM PHY, these sizes correspond to durations $D_{SCS} = 28\mu s$ and $D = 300\ \mu s$ at 54Mbps data rate. These two values and the values $P = 2, p = 0.3$ (corresponding to GW node 5 in Table 1) were used in $MIN\_OVERHEAD$ optimization problem to yield T_G and T_{synch}.

Step 3. Application of (1), (10) and (11) determined the final values of S, T_{synch} and T_F as shown in Table 2.

Requirements	T_{SCS}^{max} $5000\mu s$	T_F^{max} $5000\mu s$	ϵ 10^{-4}	Datarate 54 Mbps
Synchronization algorithm	P 2	L_{SCS} 52 bytes	p 0.3	
Step 1	r_d 5.5 $\mu s/s$	T_P 17 μs	T_{Dpp} 104 μs	
Step 2	D_{SCS} 28 μs	D 300 μs		
	T_G 6 μs	T_{synch} 95064μs	T_{SCS} 102 μs	
Step 3	S 323 μs	T_F 4845 μs	T_{synch} 92157μs	

Table 2: Design procedure steps and outcomes.

The resulting TDM MAC protocol is efficient. Its overhead is 7.22%, consisting of slot overhead $\frac{T_P+T_G}{S} = 7.12\%$ and negligible synchronization overhead $\frac{T_{SCS}}{T_{synch}} = 0.1\%$. Furthermore, the algorithm execution time T_{SCS} of 102 μs is well below the T_{SCS}^{max} delay bound of 5000 μs.

8.2 Synchronization Algorithm Validation

In this section, we validate our synchronization algorithm and evaluate its accuracy in terms of maximum clock drift over all node pairs. In these experiments only synchronization beacons are transmitted (during the SCS) at 54Mbps according to the designed synchronization period T_{synch}. Instead of the designed protocol's beacon slot duration we used the larger data slot duration $S = 323\mu s$ This provides a large guard time to observe potentially high clock drifts without de-synchronizing the network.

During each experiment all nodes store their gateway offset at the end of each synchronization period. The maximum clock drift over all nodes that received a beacon at each beacon period is computed offline based on their offset differences, in a similar manner to our clock drift rate measurement methodology. Fig. 10 depicts the cdf of 24000 clock drift samples, spanning several 2-minute measurement intervals across four days.

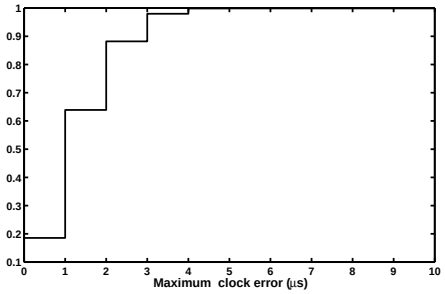


Figure 10: CDF of maximum clock drift (error) for $T_{synch} = 92157\mu s$, across four days.

The maximum observed clock drift was $10\mu s$ and only 7 out of 24000 samples exceeded the designed guard time

$T_G = 6\mu s$. This is the same order of magnitude as the reliability requirement $\epsilon = 10^{-4}$. The majority of the clock drift samples do not exceed $4\mu s$, which is less than the designed guard time $T_G = 6\mu s$. This overestimation is due to the synchronization design constraints that use the conservative analysis of Section 4.3. We conclude that the synchronization algorithm achieves high accuracy and uses a slightly conservative guard time.

8.3 Performance evaluation

We focus on two aspects that have not yet been explored in a mesh TDM MAC protocol implementation: the impact of slot duration S and the impact of packet losses on the operation of a mesh TDM MAC protocol. We use throughput, mean delay, and delay variance (jitter) as performance metrics. All experiments use the TDM MAC parameters in Table 2. Each reported data point is the average of ten experiments.

8.3.1 Impact of slot duration

Existing TDM MAC implementations operate with slot duration S of tens to hundreds of milli-seconds. We investigate performance at smaller time scales, enabled by our implementation. Our setup involves two bidirectional flows between the GW and a MAP in our testbed, configured to transmit in alternate slots. We have implemented and enabled multi-packet transmissions within a slot. To isolate impact of slot duration, we selected a link with good quality in both directions. We consider both TCP and backlogged UDP traffic.

S	Aggregate Throughput (Mbps)	Delay (ms)	Jitter (ms)
$323\mu s$	34.8	0.6	0.1
1ms	33.0	0.7	0.6
10ms	29.2	0.8	3.9

Table 3: Impact of slot duration: UDP results.

S	Aggregate Throughput (Mbps)	Delay (ms)	Jitter (ms)
$323\mu s$	21.7	1.5	11.7
1ms	18.7	1.7	9.4
10ms	18.4	2.1	16.9

Table 4: Impact of slot duration: TCP results.

As expected, the UDP results in Table 3 show that S does not affect a lot throughput and average delay. However, as S increases, the jitter increases because at large slot durations packets experience small delays within a slot but large delays across slots, when the other side is transmitting.

The TCP results in Table 4 show throughput degradation and increased jitter for large S. This is due to the TCP self-clocking and backoff mechanisms. When S is large, the TCP ACKs are delayed and this affects maximum TCP sending rate. The increased jitter results from TCP erroneously activating congestion control and backoff due to delayed TCP ACKs instead of packet loss. Also note that, depending on the TCP state, in some slots there might no TCP traffic or only TCP ACKs to send. This further contributes to the TCP throughput degradation.

We conclude that S should be minimized to the extent possible. Note that these measurements were performed with only two nodes. As the number of nodes sharing the TDM frame increase, performance can further degrade.

Loss rate (%)	Aggregate Throughput (Mbps)	Delay (ms)	Jitter (ms)
0	27.8	0.8	0.8
20	23.5	1.0	2.0
60	4.3	5.2	12.0

Table 5: Impact of packet loss on 802.11 MAC UDP performance.

Loss rate (%)	Aggregate Throughput (Mbps)	Delay (ms)	Jitter (ms)
0	34.8	0.6	0.1
20	28.3	0.8	0.3
60	11.9	1.9	2.6

Table 6: Impact of packet loss on TDM MAC UDP performance.

8.3.2 Throughput comparison to 802.11 MAC

Fig. 11 depicts a scatterplot comparing throughput for several one-hop and two-hop bi-directional flows in our testbed. Each point corresponds to the throughput of each uni-directional flow under TDM (y-axis) and 802.11 (x-axis). Points of the same shape correspond to throughput of uni-directional flows that belong to the same bi-directional flow. We observe that the TDM MAC provides higher throughput per-direction (most points are above the y=x line). In addition, the TDM MAC distributes throughput more fairly to the uni-directional parts of each bi-directional flow: two points of the same shape typically have much less vertical distance than horizontal distance.

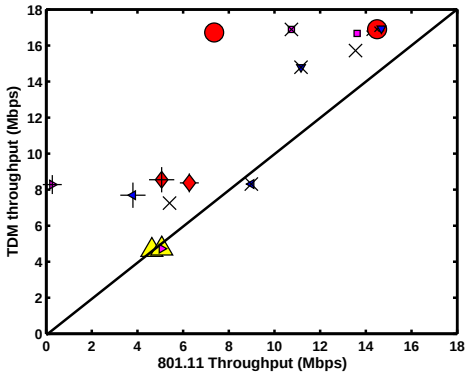


Figure 11: Throughput comparison of TDM MAC and 802.11 for one-hop (upper right region) and two-hop (lower left region) bi-directional flows.

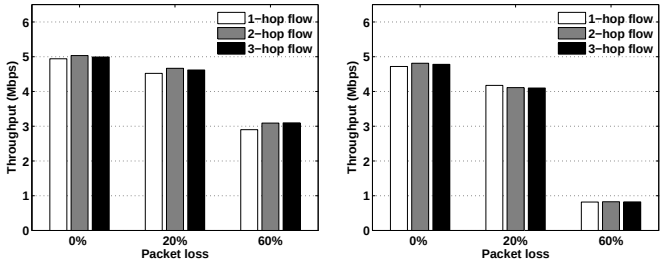
8.3.3 Impact of losses

In this section we study how packet loss impacts the performance of the TDM protocol and how this performance compares to 802.11 MAC. We investigate the impact of three different link characteristics in our testbed (c.f. Fig. 8): low loss, medium loss (average 20% loss) and high loss (average 60% loss) links.

Single link, multi-flow performance. We first examine single-link bidirectional communication between different node pairs in our testbed. Table 5 and Table 6 show the UDP performance resulting from packet loss experiments

for 802.11 MAC and our TDM MAC, respectively. The 802.11 performance degrades rapidly with increased packet loss. The main reason is that 802.11 MAC performs retransmissions using binary exponential backoff. Even if eventually a retransmission succeeds, a significant amount of time has been lost. On the other hand, the TDM MAC performs retransmissions at each assigned slot. According to Table 6 this results in graceful performance degradation for the TDM protocol even in the high loss scenario.

Multi-hop, multi-flow performance. We consider a chain topology of four nodes and evaluate the performance of three UDP flows operating in parallel from the gateway node to the three downstream nodes. Figures 12(a) and 12(b) show the throughput performance of the three flows with different packet loss behavior on the link from the gateway to the first node on the chain. We observe that for all loss rates both protocols TDM achieves higher throughput than 802.11. Furthermore, as loss increases, 802.11 performance degrades rapidly due to binary exponential backoff while TDM experiences graceful performance degradation.



(a) TDM (b) 802.11

Figure 12: TDM and 802.11 downstream performances.

9. CONCLUSIONS

We presented the design, implementation, and evaluation of a slotted TDM MAC protocol for multihop wireless mesh networks using a programmable 802.11-based platform. We introduced a design procedure that optimizes the protocol parameters given platform communication bottlenecks and synchronization constraints. We designed a novel multi-hop clock synchronization algorithm that utilizes the protocol's slotted structure to achieve accurate micro-second level synchronization with low overhead and bounded execution time. We integrated this synchronization algorithm with the TDM MAC protocol state machine, using minimal hardware resources. We prototyped this protocol on our programmable platform and applied our design framework to optimize its operation. We experimentally validated its correct operation, including the chosen protocol parameters and the synchronization algorithm, and demonstrated its performance under realistic network configurations and traffic patterns.

To the best of our knowledge, this is the first comprehensive design and implementation of a mesh TDM MAC protocol at firmware level which enables communication and control at microsecond granularity. In summary, we believe that the design guidelines presented in this paper along with their experimental validation on a real platform make up the missing piece of the TDM MAC design and implementation as a contender for high-performance MAC in mesh networks.

10. REFERENCES

[1] ECMA 368, High Rate Ultra Wideband PHY and MAC standard, 2007.

[2] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. Link-level Measurements from an 802.11b Mesh Network. In *Proc. ACM SIGCOMM*, Portland, OR, Aug. 2004.

[3] K. Chebrolu, B. Raman, N. Mishra, P. Valiveti, and R. Kumar. Brimon: a sensor network system for railway bridge monitoring. In *Proc. ACM MobiSys*, Breckenridge, CO, USA, Jun. 2008.

[4] J .Elson. Time synchronization in wireless sensor networks. PhD Thesis, UCLA, 2003.

[5] S. Ganeriwal, R. Kumar, and M. Srivastava. Timing-sync Protocol for Sensor Networks. In *Proc. ACM SenSys*, Los Angeles, CA, USA, Nov. 2003.

[6] A. Kashyap, S. Ganguly, and S. Das. A Measurement-based Approach to Modeling Link Capacity in 802.11-based Wireless networks. In *Proc. ACM MobiCom*, Montreal, Canada, Oct. 2007.

[7] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi. The Flooding Time Synchronization Protocol. In *Proc. ACM SenSys*, Baltimore, MD, USA, Nov. 2004.

[8] D. Mills. Internet time synchronization: The network time protocol. *IEEE Transactions on Communications*, 39:1482–1493, 1991.

[9] M. Neufeld, J. Fifield, C. Doerr, A. Sheth, and D. Grunwald. SoftMAC: - Flexible Wireless Research Platform. In *HotNets*, College Park, MD, USA, Nov. 2005.

[10] 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. In *Proc. NSDI*, Cambridge, MA, USA, Apr. 2007.

[11] L. Qiu, Y. Zhang, F. Wang, M. Han, and R. Mahajan. A general model of wireless interference. In *Proc. ACM MobiCom*, Montreal, Canada, Oct. 2007.

[12] B. Raman and K. Chebrolu. Design and Evaluation of a New MAC Protocol for Long-Distance 802.11 Mesh Networks. In *Proc. ACM MobiCom*, Cologne, Germany, Aug. 2005.

[13] A. Rao and I. Stoica. An Overlay MAC Layer for 802.11 Networks. In *Proc. ACM MobiSys*, Seattle, WA, USA, Jun. 2005.

[14] K. Romer, P. Blum, and L. Meier. *Time Synchronization and Calibration in Wireless Sensor Networks*. Book chp., Handbook of Wireless Sensor Networks, Wiley Series on Parallel and Distributed Computing, 2005.

[15] A. Rowe, R. Mangharan, and R. Rajkumar. Rt-link: A Time-Synchronized Link Protocol for Energy-Constrained Multi-hop Wireless Networks. In *Proc. IEEE SECON*, San Diego, CA, USA, Jun. 2006.

[16] A. Sharma, M. Tiwari, and H. Zheng. MadMac: Building a Reconfigurable Radio Testbed Using Commodity 802.11 Hardware. In *IEEE Workshop on Networking Technologies for Software Defined Radio (SDR) Networks*, Portland, OR, USA, Nov. 2006.

[17] USCG Navigation Counter GPS page. http://www.navcen.uscg.nil/gps/default.html.