TSTP MAC: A Foundation for the Trustful Space-Time Protocol

Davi Resner and Antônio Augusto Fröhlich
Software/Hardware Integration Lab
Federal University of Santa Catarina
PO Box 476, 88040-900 - Florianópolis, SC, Brazil
{davir,guto}@lisha.ufsc.br

***Abstract*—The *Trustful Space-Time Protocol* (TSTP) is an application-oriented protocol designed to resource-efficiently deliver authenticated, encrypted, timed, georeferenced, SI-compliant data communication support to IoT devices interacting with an IoT gateway, effectively defining a new API and paradigm for programming WSNs and the IoT. In this work, we describe TSTP's MAC in detail and show how it is a fundamental part of TSTP's design, orchestrating the interaction between several integrated components across the network. TSTP MAC is derived from RB-MAC, handling collision avoidance, duty cycling and greedy, fully-reactive geographic routing without any exchange of control messages, enabling receivers to achieve ultra-low duty cycles (e.g. 0.5% for 265ms per-hop delay), but taking the toll on senders, which must occupy the channel with a relatively long packetized preamble before each transmission. We also show how TSTP MAC itself can leverage clock synchronization it makes possible to TSTP to make senders only transmit critical portions of the preamble by predicting when there will possibly be receivers awake. We show that this optimization can achieve a 70% reduction in preamble transmission even in a non-ideal case, while not introducing any control messages and not affecting receiver duty cycle.**

***Keywords*-Wireless Sensor Networks; MAC; Cross-Layer Communication Protocol; Geographic; Clock Synchronization**

I. Introduction

Wireless Sensor Networks (WSNs) have been the focus of intense research for well over a decade by now. Several physical layers have been proposed, along with a myriad of medium access and routing protocols. Such protocols have been made energy-aware; aggregation and fusion strategies have been employed; basic infrastructures have been enriched with location, timing and security protocols; operating systems have been designed to support higher-level abstractions, along with large-scale management systems designed to handle the produced data properly. We are currently seeing these networks being connected to the Internet of Things (IoT).

Extensive research is also being carried out on cross-layer optimizations for wireless communication [4] and WSN [9] protocols. These works focus on taking into account related information given by one layer to make decisions at a different layer in the communication stack, and have been proven to be greatly effective. Such efforts have been often carried out in such a way as to preserve the interface of the original individual protocols and maintain the modularity of traditional layered architectures. In our opinion, however, this approach misses a great opportunity to design a truly application-oriented protocol for WSNs and the IoT, in which services are intimately combined to optimize resources at the same time as they address real and specific application needs.

The *Trustful Space-Time Protocol* (TSTP) [14] is an application-oriented, cross-layer communication protocol initially developed for the Embedded Parallel Operating System (EPOS). TSTP was designed to resource-efficiently deliver authenticated, encrypted, timed, georeferenced, SI-compliant data communication support to IoT devices interacting with an IoT gateway. Actually, it reaches beyond a communication protocol as it defines a user interface inspired by the IEEE 1451 Smart Transducer concept of "transducer electronic data sheets". Applications simply declare interest in a given physical quantity inside a portion of space-time that is to be measured with a minimum precision and at a given frequency. Nodes matching the criteria periodically send the corresponding data that is selectively forwarded to the gateway.

It is a challenge to provide reliable and energy-efficient data delivery in such resource-constrained, dynamic networks. In this work, we describe in detail the design of the Medium Access Control (MAC) that is a foundation to enable TSTP's features, as well as its implementation on top of an IEEE 802.15.4 2450MHz DSSS PHY layer. TSTP MAC handles collision avoidance, ultra-low duty cycling, deadline-aware geographic routing, and implicitly causes useful information such as timestamps to be constantly transmitted throughout the network, enabling overhearing nodes to update local information (e.g. passively synchronize their clocks [15]) with no overhead in terms of control messages. We also show how TSTP MAC leverages clock synchronization given by TSTP to reduce energy spent during transmission by 70% in the average case.

II. Trustful Space-Time Protocol

Cross-layer designs have been shown to be very efficient in optimizing wireless networks [17]. Any design that breaks in some way the black-box characteristic of the classic TCP/IP stack can be considered cross-layer [4]. In practice, cross-layer designs usually work by taking information from one or more layers of a typical layered stack to optimize a set of parameters or make a decision in another layer or set of layers. From the myriad of cross-layer proposals, a minority involves the application layer, and even fewer encompass the complete

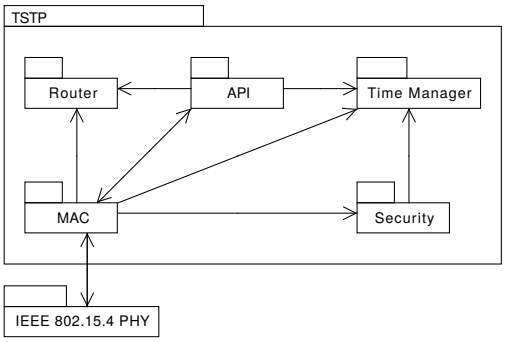


Figure 1: Interaction between TSTP components.

stack to present a truly application-oriented, domain-specific solution.

The *Trustful Space-Time Protocol* (TSTP) [14] is an application-oriented, cross-layer protocol for WSNs and the IoT. Instead of focusing on keeping the original protocol interfaces in a modular, layered architecture with shared data, TSTP focuses on efficiently delivering functionality recurrently needed by WSN applications: trusted, timed, geo-referenced, SI-compliant data that is resource-efficiently delivered to a sink. TSTP delivers this functionality directly to the application in the form of a complete communication solution, which allows the design of optimized, synergistic co-operation of protocols while eliminating the need for additional, heterogeneous software layers that come with an integration cost and often result in replication of data.

TSTP is composed of 5 parts closely integrated in a cross-layer architecture:

- **Time Manager:** Responsible for counting time and handling clock synchronization across the network;
- **Router:** Responsible for defining the format and semantics of network addresses and keeping them up to date;
- **Security:** Responsible for management of encryption, authentication and cryptographic keys;
- An **API**, through which applications can benefit from the implemented services;
- **MAC:** At the foundation, responsible for the actual transmission of messages and efficient dissemination of information from the other components across the network.

Figure 1 depicts the interactions between these components. In this work, we focus on the MAC component and its interaction with the Time Manager. We consider that the Router component uses spatial coordinates as network addresses and defines algorithms to localize the node in space [14] [12].

By including data from the time manager and router components in every message, TSTP devices are able to localize themselves in space and time passively, mostly by overhearing network traffic. The security mechanisms exploit temporal synchronization [13]. At the foundation, TSTP MAC handles duty cycles, passive acknowledgments, collision avoidance, and efficient dissemination of shared data throughout the components and across the whole network. The main concerns are with energy savings and ensuring that nodes can overhear useful traffic often enough, so that they don't need to exchange explicit synchronization messages.

Communication in TSTP occurs mostly between sensors and a specific node called *sink* or *gateway*. Sensors are nodes that can measure and report one or more types of data about the environment (e.g. temperature, luminosity) with a certain precision and maximum frequency. A sink is a node that is interested in such information. From the point of view of a sink, given that the information acquired about a desired space-time region is trustful and was measured with a certain precision, it does not matter which particular sensor measured it. TSTP is designed so that sinks announce what information they are interested in, and sensors deliver it if able.

A sink node announces interest in a physical quantity using an *Interest* message. This message specifies the space-time region in which the interest is valid (a sphere in space and a time interval), the periodicity of responses, SI unit and minimum required precision.

When receiving an Interest message, a sensor checks if it is inside the desired region and able to measure the requested quantity with the required precision. If so, the Interest is saved and the sensor automatically responds with a *Data* message every *period* of time until the Interest expires or it leaves the area of interest.

The units used in Interest and Data messages are base or derived units from the International System (SI), and their representation is inspired by the IEEE1451.0 standard [5]. Radian and Steradian, together with the seven SI base units, form the nine base units of the standard. Derived units are formed by the product of base units raised to a power (e.g. pressure is N/m^2), therefore, storing the exponents of each base unit is enough to represent any derived unit. In TSTP, 40 bits are necessary to represent any SI unit. Figure 2 shows the header that is included in every TSTP message.

A. Passive Time Synchronization

Time in computing systems is typically kept by counting cycles of a piezoelectric crystal oscillator. In the domain of energy-constrained IoT devices, no crystal or compensation mechanism can achieve perfect target frequencies under practical conditions. The IEEE 802.15.4 standard, for example, has a precision requirement for devices of $\pm$ 40 ppm, including temperature and aging variations [8]. Inaccuracy in frequency leads to the fact that two independent clocks, once synchronized, will drift apart without limit. Figure 3 shows the clock between nodes of out target IoT platform, EPOSMote III, drifting apart over time [15]. Because of this drift, a common measure of time between two independent systems requires a synchronization mechanism, which is itself subject to temporal inaccuracies and variations. In the case of wireless IoT devices, these variations arise from the message exchange process itself.

To achieve ϵ-precision time synchronization, a node needs [16]:

1) a high-resolution clock source with frequency f_0, and

Header Format									
Bits: 3	**1**	**2**	**2**	**8**	**3*sb**	**64**	**3*sb + tb**	***tb***	**0 or *sb***
Message Type	Time Request	Spatial Scale	Temporal Scale	Location Confidence	Last Hop x,y,z	Last Hop Time-stamp	Origin x,y,z,t	Deadline	Location Deviation

Figure 2: TSTP message header format. *sb* and *tb* are variables defined by the *Spatial Scale* and *Temporal Scale* codes [14].

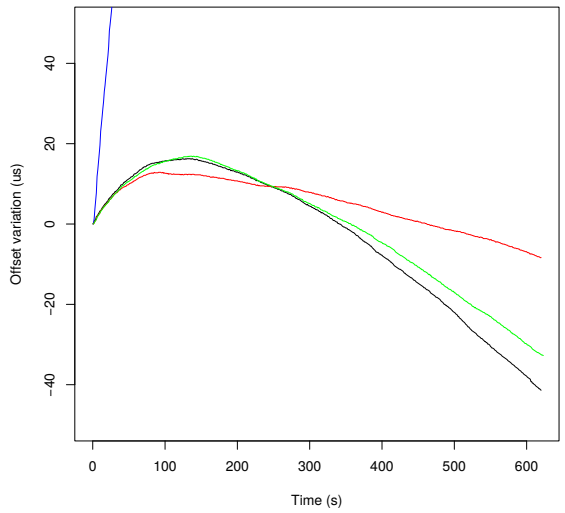


Figure 3: Clock drift of four different EPOSMote III devices in relation to a fifth. One of the motes (in blue) diverged particularly quick in this scenario.

2) message time-stamping with accuracy $\epsilon \pm 1/f_0$

TSTP is able to achieve the required time-stamping accuracy [15] by leaving the responsibility of updating the *Last Hop Time-stamp* field of every message to the MAC, which updates them as close as possible to the moment where the message is actually sent by the physical layer. With these premisses granted, nodes can passively synchronize their clocks to one another by simply overhearing network traffic from the neighborhood.

Let d_{TX} be the total time delay between insertion of the time-stamp at the sender and capturing of the time-stamp at the receiver, and $c_N(t_m)$ represent the value of the time-stamp counter of node N at physical time t_m. Because in our implementation of TSTP for EPOS d_{TX} is known and has a small enough jitter [15] [11], when node B receives a timestamp $c_A(t_1)$ from node A, it can trivially determine its clock offset ϕ:

$$\phi = c_A(t_1) - (c_B(t_1') + d_{TX}) \quad (1)$$

where $c_B(t_1')$ is the time recorded by node B upon reception of the message. The accuracy of this estimation is equal to the accuracy and jitter in d_{TX}.

Once the offset is corrected, node B is ready to estimate its clock drift in relation to A. After it receives a second message containing $c_A(t_2)$, the clock drift is given as [16]:

$$\hat{f_e} = \frac{(c_A(t_2) - c_B(t_2')) - (c_A(t_1) - c_B(t_1'))}{c_A(t_2) - c_A(t_1)} \quad (2)$$

Figure 4 shows the errors in clock corrections for nodes in a TSTP network under sparse traffic, applying (4.b) and not applying (4.a) the frequency error estimation (Equation 2). The error is measured as the difference between the offset estimation immediately before the reception of a new message and immediately after the offset is re-calculated given the information in that message.

III. TSTP MAC

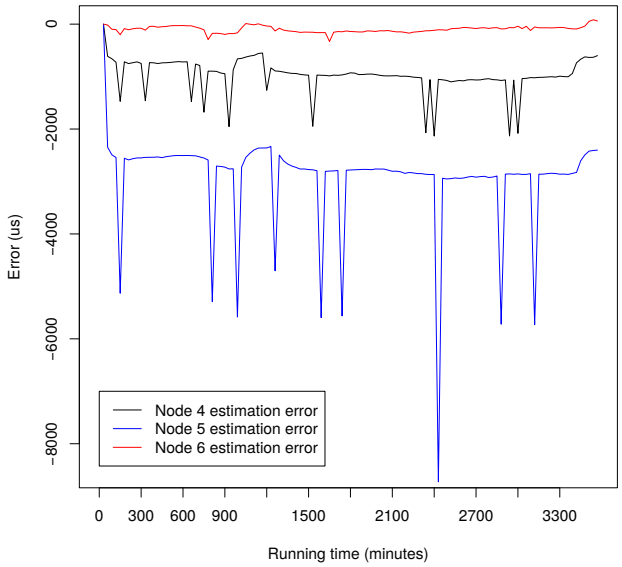
TSTP MAC follows the general principles of RB-MAC [1] (Section V): senders send a long preamble composed of Microframes before each message; sensor nodes sleep for most of the time and their distances to the message's destination is used to derive the back-off time when forwarding a message, resulting in a greedy, fully-reactive geographic routing. Actually, other metrics can be used instead of the distance, and in TSTP the Router defines this metric, which we call Hint. For example, it may take into account the remaining battery charge of the node to ensure that nodes in an optimal path are not going to have their batteries drained too quickly [10]. For the purposes of this work, we take the Hint as being the geographic distance to the destination.

In the next subsection, we detail the design and implementation of an asynchronous version of TSTP MAC. In subsection III-B, we show how we can modify only three states of the asynchronous version to save energy during transmission if clocks are synchronized across the network as TSTP provides. Figure 5 shows the Microframe format. The FCS field is used for error detection as defined in the IEEE 802.15.4 MAC layer specification. Figure 6 shows the activity diagram, which is explained in detail in this section. Both figures are valid to both variations of the design.

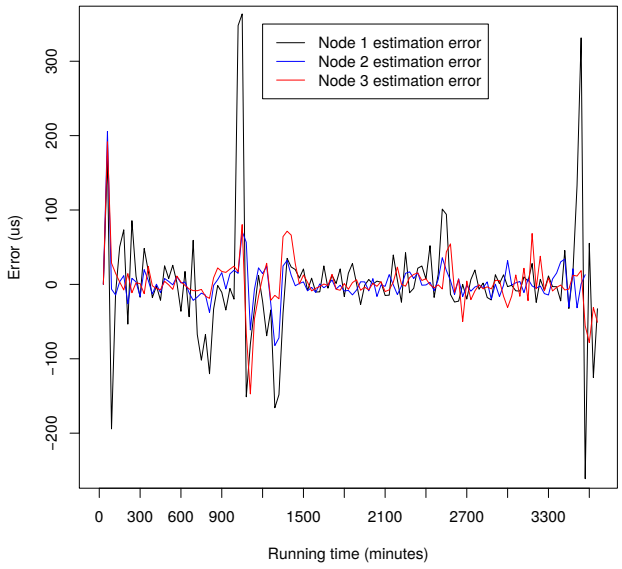
Throughout this work, we use the following definitions:

CI	Channel Checking Interval
$d \in (0, 1]$	Duty cycle during idle listening
$s_r = 62.5\frac{symbol}{ms}$	IEEE 802.15.4 Symbol Rate
$T_u = \frac{12}{s_r} = 0.192ms$	IEEE 802.15.4 Turnaround Time
$S_{MF} = 30symbol$	Microframe with PHY header size
$t_s = \frac{S_{MF}}{s_r} = 0.48ms$	Time to transmit a Microframe
$t_r = CI \times d$	Microframe listening time
$S = CI - t_r$	Receiver sleeping time
$N_{MF} \geq 1$	Number of Microframes
$t_i \geq T_u$	Time between Microframes

Each node maintains a list of messages called *TX Schedule*. Each entry in the list represents a message that is scheduled for transmission or retransmission. From any given entry in the TX Schedule, the following information shall be directly



(a) Error in the clock offset estimation for three nodes using only the last calculated instant offset.



(b) Error in the clock offset estimation for other three nodes using the last calculated instant offset and the last estimation of clock drift.

Figure 4: Six EPOSMote III devices synchronizing with a seventh. Messages with time stamp are received once every 30 minutes, directly from the synchronizer node. Nodes compensating for detected clock drift (b) estimate their offsets with far better accuracy than nodes calculating only instant offset (a). Both cases are far from the maximum possible error of 144ms.

Microframe					
Bits: 1	**11**	**12**	**32**	**16**	9 octets
All Listen	Count	ID	Hint	FCS	

Figure 5: TSTP MAC Microframe format.

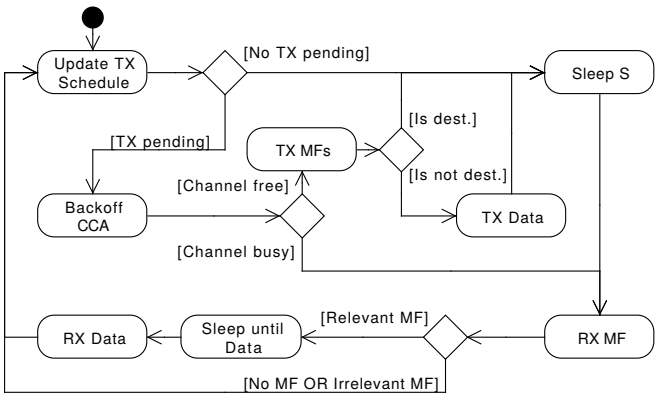


Figure 6: TSTP MAC activity diagram.

accessible or derivable:

New	An indication of whether this is a new message (generated by this node) or a forwarding;
T_{tx}	The time when this message shall be transmitted;
T_{lim}	The *deadline* for the message to reach its destination;
ID	The message's identification code;
Msg	The message itself;
Bkf	The back-off time;
Dst	The destination's coordinates.

New messages (e.g. generated by the application running on top of TSTP MAC) are added to this list with the *New* indication. The *ID* is generated during insertion of new messages according to some criterion and kept the same for the entire lifetime of the message across the network.

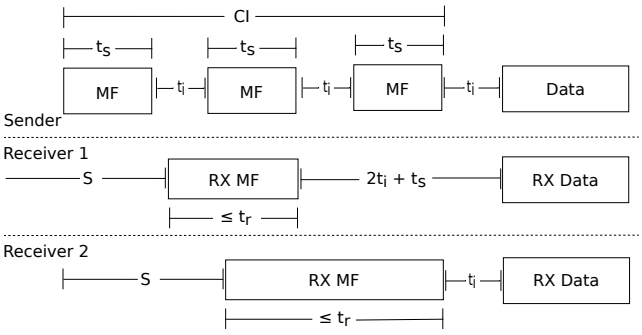


Figure 7: Asynchronous TSTP MAC transmission example with $N_{MF} = 3$. Receiver 2 might wake up late due to clock drift and miss the first Microframe, but it receives the second one if $t_r \geq 2t_s + t_i$ (Figure 8).

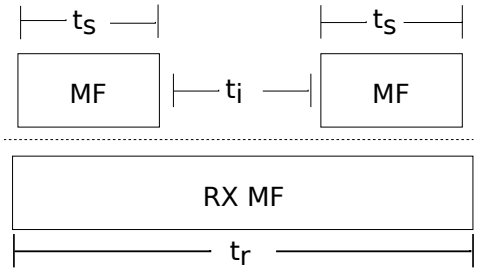


Figure 8: Microframe timing. If $t_r \geq 2t_s + t_i$, it is guaranteed that receiving nodes will have enough listening time to detect and receive at least one Microframe.

In the following subsections, we detail the actions taken at each state of the state machine shown in Figure 6.

A. Asynchronous Design

Check TX Schedule: Any message with an expired T_{lim} is removed from the list and dropped. Then, check if there are messages with an expired T_{tx}. If any, pick one (according to some scheduling criterion) and proceed to transmit it ([TX pending] transition in Figure 6); Otherwise, proceed to sleep for S ([No TX pending]).

Sleep S: The node turns its radio off and may enter low-power mode for a period S.

RX MF: Put the radio in receive mode for t_r time, or until a Microframe is successfully received. If a Microframe is received, search the list for any message with the same ID of this Microframe. If found, remove that message's entry from the table – because it has been already handled by another node –, then check if the Microframe is relevant and take the corresponding transition. If no Microframe is received, take the [No MF] transition.

A Microframe is relevant if and only if at least one of the following conditions is true: 1) the *All Listen* bit is set; 2) any TSTP component needs Header information to update its local information as soon as possible (e.g. the Time Manager detects that it might be about to drift more than the acceptable limit); 3) given the Hint, the Router indicates that this node is a *forwarding candidate* (e.g. makes positive geographic progress towards the destination) for this message.

Sleep until Data: The node turns its radio off and may enter low-power mode for a period S':

$$S' = t_i + Count \times (t_i + t_s) \tag{3}$$

where $Count$ is taken from the Microframe just received in the RX MF state.

RX Data: Put the radio in receive mode and wait for a message (discard any Microframes eventually received in this state). When the message is received, calculate its back-off value Bkf (Equation 6) and add it to the TX Schedule for immediate retransmission. Share the message with the other TSTP components so they can extract the information they need (e.g. time stamps for clock synchronization).

Backoff / CCA: The back-off mechanism serves to avoid collisions and to select the best forwarder to a message. Before transmitting a message, a sensor backs off for a period and checks for channel activity. If the message to be sent is new (i.e. not a forwarding), the back-off Bkf shall be random and a multiple of the time gap g between a transmitter sensing an available channel and finishing transmission of enough symbols for other nodes to sense a busy channel (8 symbols in IEEE 802.15.4). Bkf shall not exceed S, to make sure that receivers will not miss the Microframes of an ongoing transmission:

$$g = T_u + 8/s_r \tag{4}$$

$$Bkf = random\left(\left\lfloor \frac{S}{g} \right\rfloor\right) \times g \tag{5}$$

where $random(n)$ is a function that returns an integer in the interval $[0, n]$ picked at random. Note that the probability of collisions is inversely proportional to S in this case.

If the message is not new, this sensor is currently trying to forward it, and the back-off serves to elect the forwarder among all the candidates [1]. Instead of delaying randomly, the sensor sets the back-off as being directly proportional to its distance to the destination, according to Equation 6:

$$Bkf = \left\lfloor \frac{|D - (D_{msg} - R)|}{gR/S} \right\rfloor \times g \tag{6}$$

where D is the node's distance to the message's destination, D_{msg} is the distance to the destination from the last node to transmit this message (taken from the Microframe) and R is a fixed network parameter representing the nodes' radio range. This value is calculated when the message is received, and remains the same for this message until it is dropped or successfully sent.

After backing off for Bkf, the node senses the channel and takes the transition corresponding to its perceived state (free or busy).

TX MFs: In this state, the node transmits N_{MF} Microframes with an interval of precisely t_i between every two consecutive Microframes (Figure 7). At each Microframe, it updates its *Count* value with how many Microframes are left after this one, and with a Hint from the Router (in this work, we consider that the Hint is always this node's distance to the destination. In Section VI we point out as future work the possibility of including Hints from different components in different Microframes).

After transmitting the Microframes, the node checks if it is the final destination of the message. If so, the Microframes served as an ACK to the last forwarder and any other nodes competing to retransmit, and there is no need to transmit the message itself. If it is not the final destination, it proceeds to transmit the message.

TX Data: The node updates the *Last Hop Time-stamp* field of the message with the local time-stamp as close as possible to the moment the message is actually sent by the physical layer, then transmits the message and reinserts it in its TX Schedule with a new $T_{tx} < T_{lim}$. Messages are only removed

from the schedule in the RX MF and Check TX Schedule states, when nodes overhear transmission of an equal ID or when T_{lim} expires.

B. Synchronized Clocks Optimization

By taking advantage of TSTP's network-wide clock synchronization (Section II-A), we can assume that nodes participating in communication will have their clocks synchronized with a minimum accuracy of ϵ units of time with high probability. With this observation, it is possible to align the periods of every node in a way that they wake up to check the channel at the same time with a simple change in the *Sleep S* state, so that senders know exactly when to send a Microframe and don't need to occupy the channel for a whole period. However, nodes that are competing to retransmit a message may wake up at any time during a period to check if they have won the contention, and the first one that does so must occupy the channel until the end of the period for the other candidates to detect that they lost. This problem calls for a slight change of the contention mechanism, and to solve it we alter two other states previously described in Section III-A (everything else remains valid):

Sleep S: To align the channel checks, instead of always sleeping for a period S, nodes sleep for $t_{now} \mod S$ (t_{now} being the node's current local time).

Backoff / CCA: Nodes execute this state just as in the asynchronous design, but with one addition when (and only when) $Bkf > S/2+t_r$: the node will mark two channel check moments, one at $C_1 = t_{now} + S/2$ and the usual one at $C_2 = t_{now} + Bkf$. If the first check indicates a busy channel, the node takes the [Channel busy] transition. Otherwise, it stays in this state and proceeds just as it would in the asynchronous design, checking the channel at the second mark and taking the corresponding transition. Note that in many cases (when there is at least one forwarding candidate with $Bkf \leq S/2 + t_r$) this incurs no overhead in listening time, since candidates that lose the contention will perform a single channel check, just as in the asynchronous design. In the worst case, it will result in one extra channel check per forwarding candidate.

TX MFs: Nodes that win the contention must transmit Microframes for long enough to ensure that other contending nodes will detect it, and also at the moment when every node will be awake to perform the periodic channel check and receive the message. To account for a maximum clock drift of $\pm\epsilon$, at least M_{mf} Microframes must be transmitted:

$$M_{mf} = 2 \cdot \left\lceil \frac{\epsilon}{t_s + t_i} \right\rceil \tag{7}$$

To make other contending nodes detect the winner's preamble, it must send a number of Microframes F to fill the time until the next channel check, which is given by:

$$F = \begin{cases} \left\lceil \frac{S/2 - Bkf}{t_s + t_i} + \frac{M_{mf}}{2} \right\rceil, & Bkf \leq \frac{S}{2} - \epsilon \\ max\left(M_{mf}, \left\lceil \frac{S - Bkf}{t_s + t_i} + \frac{M_{mf}}{2} \right\rceil\right), & Bkf > \frac{S}{2} - \epsilon \end{cases} \tag{8}$$

***CI* (ms)**	2	10	12	24	116	150	231	1153
***d* (%)**	62	11.6	9.64	4.8	0.99	0.77	0.49	0.09

Table I: Best possible idle listening duty cycles for selected values of CI (Equation 9).

IV. Analysis

We let the user define CI and then find the best possible value of d. We start by considering $t_i = T_u$ (the minimum possible value of t_i), then find a (likely) fractional N'_{MF}. Since N_{MF} needs to be integral, we use $N_{MF} = \lfloor N'_{MF} \rfloor$ (so that $t_i \geq T_u$ will still hold) and, finally, find t_i, t_r and d from CI and N_{MF}:

$$N_{MF} = \left\lfloor 1 + \frac{CI - t_s}{T_u + t_s} \right\rfloor \quad (a) \qquad t_r = 2t_s + t_i \quad (c)$$
$$t_i = \frac{CI - t_s}{N_{MF} - 1} - t_s \quad (b) \qquad d = \frac{t_r}{CI} \quad (d) \tag{9}$$

Table I lists a few values of CI and d. It is possible to achieve low latency (2ms per hop) at the cost of very high idle listening duty cycle. We can achieve 1% duty cycle with a latency of 116ms, and 0.5% for 231ms.

Figure 9 shows the percentage of Microframes expected to be saved by the clock synchronization optimization as a function of the maximum tolerable clock drift. In Figure 4, we show a scenario where a TSTP network can keep the clocks synchronized with an accuracy $\epsilon < 400\mu s$ under sparse network traffic (synchronization tends to be better with more traffic [15]). In that scenario, $\epsilon \leq 0.001 \cdot CI$ when $CI \geq 400ms$, and $\epsilon \leq 0.01 \cdot CI$ when $CI \geq 40ms$, which means that even under sparse traffic TSTP is realistically able to deliver the clock synchronization accuracy needed for high savings in Microframe transmission while still keeping modest-to-low latency. If we instead took into account only the maximum possible drift defined in IEEE 802.15.4 ($\pm$ 40ppm), we would need to consider $\epsilon = 144ms$ for the scenario shown, which would require a CI of more than 2 minutes to achieve the same energy savings during transmission, a prohibitive latency for many applications. Although it is true that two clocks *might* momentarily drift close to the theoretical limit, for the MAC this would in the worst case only result in one node missing messages. Because of the anycast nature of TSTP MAC, in many cases other nearby nodes would be able to pick the message and normally carry on the transmission.

In Section V we continue the analysis by pointing to results from previous works, as well as comparing TSTP MAC to related protocols.

V. Related Work

TSTP MAC is an adaptation and implementation of Receiver-Based MAC (RB-MAC) [1] under the TSTP context. RB-MAC defines the cooperative, geographic forwarder selection and passive acknowledgment mechanisms used in TSTP MAC. It does not consider time synchronization and the authors do not discuss implementation details, as we do in Section III. The authors analyze the protocol in terms of

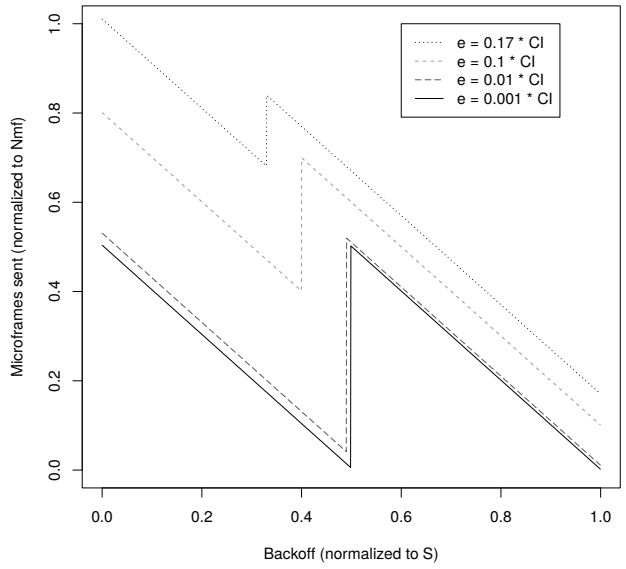


Figure 9: Savings in Microframe transmission by the clock synchronization optimization (Equation 8). If $\epsilon \geq 0.17 \cdot CI$, compensating for clock drift might start resulting in more Microframes being sent than in the asynchronous design. For smaller values of ϵ, it will often save more than 70% of transmission cost.

energy consumption and latency, comparing it to 1-hopMAC, where senders define a specific receiver at each hop. They show that RB-MAC is in general more energy efficient and causes less latency because its anycast nature makes it much more resilient to lossy links [1]. Furthermore, not defining a receiver allows for dynamic, energy-aware contention metrics (e.g. a node may opt to act as if it is in a worse route if its energy is currently low, avoiding nodes in an optimal path from having their batteries drained much quicker [10]). In previous work [18], we implemented both RB-MAC and B-MAC under a Configurable MAC framework in the Embedded Parallel Operating System (EPOS), and showed that RB-MAC far outperforms B-MAC for varying channel conditions. In another work, we also analyzed the impact of the latency caused by RB-MAC in TSTP [14]. Since we did not change any of the parts of RB-MAC relevant to these analyses, they hold for TSTP MAC, with the difference that the synchronous clock optimization can provide high energy savings for senders, as shown in Section IV. RB-MAC and TSTP MAC do not explicitly treat the hidden terminal problem. We consider that in the event of collisions, either ACKs will not be received and the message will be retransmitted, or other nodes that were contention candidates for the collided message will carry on the transmission after the collision.

In ContikiMAC [2], senders transmit the data packet repeatedly for a full period instead of using Microframes. If the packet is a unicast, the receiver cuts the transmission by sending back an acknowledgment as soon as it wakes up and receives the packet. If this happens, the sender and receiver lock their sleeping phases, resulting in much less repetitions of the data packet in the next unicast transmission involving the same two nodes. Although this results in savings in latency and transmission cost in specific cases, it only works for unicast, which is the case where RB-MAC's resilience to lossy links becomes advantageous [1]. Thus, TSTP MAC's clock synchronization optimization is a more reliable solution for reducing the cost of transmission, because it applies to every transmission while not losing the anycast nature. For idle listening, ContikiMAC reaches a duty cycle of roughly 1% for CI values around 125ms, a little greater than TSTP MAC's 116ms (Table I). The author provides the technical details of transmission/reception timings and gaps between frames that inspired the same analyses in Sections III and IV of the present work.

WiseMAC [3] is an early MAC protocol that uses the idea of clock synchronization to send packets only when the receiver is likely to be awake. WiseMAC assumes a structured network, in which a central node C without energy constraints can reach every node under its supervision and learns those nodes' sleeping schedules by listening to the channel all the time. Then, when the traffic goes from C to one of its children c, C can start sending the preamble only when c will be awake (compensating for the maximum possible clock drift), thus occupying the channel for much less time, providing better throughput and energy savings [3]. Its main drawbacks in relation to TSTP MAC's use of clock synchronization are that it only works for traffic in one direction, and WiseMAC compensates for the maximum possible clock drift since it does not assume a more accurate compensation mechanism. As Figure 4 shows, there is a big difference between the maximum possible drift and the values that are likely to occur in practice, even without drift compensation.

Although TSTP MAC is concerned with dealing with a lossy and dynamic environment, optimizing the number of message exchanges, and saving energy, one of its main drawbacks is in latency. Because it benefits from overhearings and uses time as a way to define contention winners, it is not trivial to cut the preamble and transmit the data as soon as a receiver signals that it is available. The DMAC [7] protocol uses a *staggered schedule* concept, which leverages a known network structure to greatly reduce latency in the uplink direction (traffic flowing from sensors to the sink). The idea is making nodes skew their duty cycles such that nodes n-hops away from the sink will wake up right after nodes $(n + 1)$-hops away, so nodes in the farther region will start transmission and have a receiver available right away. This design drastically reduces latency all the way from sensors to the sink [7]. Since TSTP provides nodes with their spatial locations [14], a similar idea might be employable. It is not directly applicable, however, because of the same problems mentioned in the start of this paragraph, and we leave it as future work.

The CMAC [6] protocol uses the same routing idea as RB-MAC, and it solves the latency problem. Messages can be sent in unicast or anycast modes. In anycast, when a receiver closer to the destination detects a Microframe (called RTS in

CMAC), it backs off for a period proportional to their distance to the destination. Then, if the channel is free, it sends back a CTS, and the sender immediately transmits the data packet. The main difference from TSTP MAC in this aspect is that this back-off will be at most the gap between two RTS packets – as opposed to the whole sleeping period –, meaning that not many possible forwarding candidates will have a chance to detect the preamble and passively benefit from information in the data packet if needed. If the receiver that replied with a CTS is in a good enough region, the sender locks in and will start unicasting to this node in future transmissions until one of them fails; if this happens, the sender returns to anycast mode. During unicast mode, nodes may synchronize and use a staggered schedule to optimize latency. When the forwarder is not in a good enough region, there is a period in which the nodes stay in receive mode to detect if in the next transmission from the same sender a better node is found.

The authors present a detailed analysis, with numerical results, simulations and real-world implementation. They optimize various parameters considering current perceived network traffic, which may result in a complex implementation, but yields impressive latency and energy performance results. TSTP MAC is more concerned about energy than latency, and opts to not use strategies such as the continuous listening period, which may make a set of nodes spend energy unnecessarily. Also, just as mentioned for previous approaches, CMAC loses some reactivity to lossy links during the unicast phase and deprives many nodes from getting a chance to overhear transmissions, since the first available listener will cut the preamble of the sender in anycast, and in unicast mode data may be sent right at the wake-up time of the intended receiver.

VI. Conclusion

In this work, we presented the MAC foundations that support the Trustful Space Time Protocol (TSTP). We show how TSTP MAC orchestrates the interaction between several TSTP components such as time and space synchronization, while keeping low energy consumption and taking advantage of overhearings to introduce explicit synchronization messages only when strictly necessary. TSTP MAC reaches 1% idle listening duty cycle with a 116ms per-hop latency. We also show how, in the other way, TSTP MAC benefits from the clock synchronization it makes possible in TSTP. By assuming a maximum clock drift (that TSTP can realistically grant) across the network, we are able to cut transmission costs by about 70% on average even in far-from-ideal scenarios such as a network with sparse traffic.

We recognize that as a consequence of the current solutions found to provide the functionality TSTP MAC pursues, it is not the best option in terms of latency. As future work, we wish to investigate further the idea of staggered schedules as a possible direction to improve end-to-end latency. Also, we wish to analyze exactly how we can reduce even more transmission costs by moving non-critical information from the message Header to the Microframes in an alternating way.

References

[1] M.R. Akhavan, T. Watteyne, and A.H. Aghvami. Enhancing the performance of RPL using a Receiver-Based MAC protocol in lossy WSNs. In *IEEE ICT*, pages 191–194, May 2011.

[2] Adam Dunkels. The contikimac radio duty cycling protocol, 2011.

[3] A. El-Hoiydi and J. D. Decotignie. Wisemac: an ultra low power mac protocol for the downlink of infrastructure wireless sensor networks. In *Computers and Communications, 2004. Proceedings. ISCC 2004. Ninth International Symposium on*, volume 1, pages 244–251 Vol.1, June 2004.

[4] Bo Fu, Yang Xiao, Hongmei Deng, and Hui Zeng. A survey of cross-layer designs in wireless networks. *Communications Surveys Tutorials, IEEE*, 16(1):110–126, First 2014.

[5] IEEE Instrumentation and Measurement Society. 1451.0 - IEEE Standard for a Smart Transducer Interface for Sensors and Actuators—Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats. online, 2007.

[6] S. Liu, K. W. Fan, and P. Sinha. Cmac: An energy efficient mac layer protocol using convergent packet forwarding for wireless sensor networks. In *2007 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pages 11–20, June 2007.

[7] G. Lu, B. Krishnamachari, and C. S. Raghavendra. An adaptive energy-efficient and low-latency mac for data gathering in wireless sensor networks. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pages 224–, April 2004.

[8] A. M. Mehta and K. S. J. Pister. Frequency offset compensation for crystal-free 802.15.4 communication. In *International Conference on Advanced Technologies for Communications (ATC)*, pages 45–47, Aug 2011.

[9] Lucas D.P. Mendes and Joel J.P.C. Rodrigues. A survey on cross-layer solutions for wireless sensor networks. *Journal of Network and Computer Applications*, 34(2):523 – 534, 2011. Efficient and Robust Security and Services of Wireless Mesh Networks.

[10] Alexandre Massayuki Okazaki and Antônio Augusto Fröhlich. ADHOP: an Energy Aware Routing Algorithm for Mobile Wireless Sensor Networks. In *IEEE SENSORS 2012*, 2012.

[11] Peterson Oliveira, Alexandre Massayuki Okazaki, , and Antônio Augusto Fröhlich. Sincronização de Tempo a nível de SO utilizando o protocolo IEEE1588. In *Brazilian Symposium on Computing System Engineering*, 2012.

[12] Ricardo Reghelin and Antônio Augusto Fröhlich. A Decentralized Location System for Sensor Networks Using Cooperative Calibration and Heuristics. In *9th ACM/IEEE MSWIM*, pages 139–146, 2006.

[13] Davi Resner and Antônio Augusto Fröhlich. Key establishment and trustful communication for the internet of things. In *4th SENSORNETS*, February 2015. To be published.

[14] Davi Resner and Antônio Augusto Fröhlich. Design Rationale of a Cross-layer, Trustful Space-Time Protocol for Wireless Sensor Networks. In *20th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2015).*, pages 1–8, Luxembourg, Luxembourg, September 2015.

[15] Davi Resner, Antônio Augusto Fröhlich, and Lucas Francisco Wanner. Speculative Precision Time Protocol: submicrosecond clock synchronization for the IoT. In *21th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2016). To appear*, Berlin, Germany, August 2016.

[16] Thomas Schmid, Prabal Dutta, and Mani B. Srivastava. High-resolution, low-power time synchronization an oxymoron no more. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, IPSN '10, pages 151–161, New York, NY, USA, 2010. ACM.

[17] V. Srivastava and M. Motani. Cross-layer design: a survey and the road ahead. *Communications Magazine, IEEE*, 43(12):112–119, Dec 2005.

[18] Rodrigo Vieira Steiner, Mohammad Reza Akhavan, Antônio A. Fröhlich, , and A. Hamid Aghvami. Performance evaluation of receiver based mac using configurable framework in wsns. In *IEEE WCNC*, pages 884–889, 2013.