POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Session Layer Bounded Latency in Wireless Mesh Networks

Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Author: Luca Conterio

Advisor: Federico Terraneo

Co-advisor: Prof. William Fornaciari

Academic year: 2021-2022

1. Introduction

During the last years, wireless networks gained increasing importance in humans life, becoming leading actors in many scenarios, especially thanks to their flexibility. Consider for example the incredible spread of the Internet of Things devices. In the consumer space several commercially successful wireless network protocols exist, which are usually targeted to either high performance or low power networks, such as WiFi or ZigBee. The mentioned technologies though are not suitable for real-time applications, in which guarantees on the expected latency bounds are required. Low power protocols based on the IEEE 802.15.4e [1] standard exist too. Nevertheless, to the best of the author's knowledge, none of the protocols built on top of the mentioned standard combines support to multi-hop network topologies with bounded latency. An example is LLDN [2], which provides latency guarantees but is limited to single-hop star topologies. Moreover, the packets delivery guarantees that some of these technologies provide are not compatible with the need for deterministic and predictable latency in real-time applications.
In this context, TDMH[1] (*Time Deterministic Multi-Hop*) [3] presents itself as a wireless communication stack that supports multi-hop mesh networks and provides low power medium access control (MAC). The TDMH protocol is organized in *tiles*, which in turn are divided into time slots. The first fraction of each tile is assigned either to the *uplink* or the *downlink phase*, while the remaining time slots are allocated to the *data phase*. The network topology is collected during the uplink phase, through the neighbors tables that each node transmits. *Topology collection* makes TDMH able to adapt to changes in the topology itself. Low power is achieved through *time synchronization* among all the network nodes, employing the FLOPSYNC-2 [5] algorithm, which enables the nodes to access the physical transmission medium only when required, in a TDMA (Time Deterministic Multiple Access) fashion, avoiding access contention and random exponential backoff [6]. The TDMH *session layer* provides to applications the *streams* abstraction to establish communication among two network nodes. Each stream is a data flow associated to a *period*, which is always guaranteed. At each period, a single data unit, called packet, can flow through a stream, which thus holds a single packet buffer. *Reliability* is achieved through redundant packets transmissions, which can also follow different paths across the network, that

[1] `https://github.com/fedetft/tdmh`

do not rely on packet queues or acknowledgments. The network coordination relies on a centralized streams routing and *scheduling* algorithm, executed on the master node. Scheduling takes place every time the network topology changes and is necessary for the TDMA medium access. Schedules are flooded to all the nodes in the network during the downlink phase. During the data phase, the schedule is reproduced managing transmissions and receptions. The described features make TDMH suitable for real-time wireless applications, running on battery-powered devices and targeting industrial control use cases. In such applications. guarantees on the expected latency bounds for all the packets flowing through the network is crucial. In the following, the author's contribution is presented, including the strategies implemented to enforce guarantees on the latency bounds, not only at the physical layer but also up to the application one, together with their validation in a feedback control loop scenario.

2. Problem Statement

As mentioned, TDMH targets real-time processing, which fails if tasks are not completed within a specified deadline. In a network, latency measures the time it takes for some data to get from its source to its destination across the network. Thus, for real-time tasks, low latency is desirable, but a consistent and bounded latency is usually more important.
Being TDMH a TDMA network protocol, each time slot is dedicated to the transmission of specific streams. Despite the usage simplicity of the current streams API, which is based on the two *read* and *write* primitives, it leaves to the application the task of synchronizing the data packets generation and transmission request with the actual transmission slot. If the application unconditionally generates data packets one after the other, the session layer buffer may remain occupied while a new packet is produced. Since TDMH always complies with streams' periods, successive transmissions are delayed in the future (i.e. to the next stream period), increasing packets latency. After very few iterations the latency converges to a value that is as high as two times the stream period, as shown in Fig. 1. Right now latency is a multiple of the stream period, which should only represent the worst-case

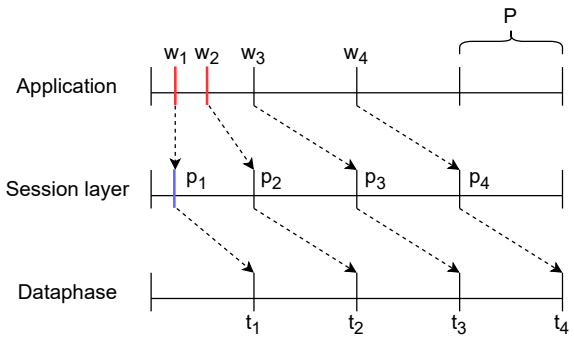

Figure 1: Undesired behavior in which *write* operations w_i are delayed by two periods. p_i represent ready for transmission packets and t_i the transmission slots, with period P.

maximum latency. Latency itself should only depend on the active schedule and the stream redundancy level. Redundancy directly affects the number of time slots needed to deliver a packet. As a consequence, each stream latency has to be, not only measurable but also predictable by knowing the current schedule.

3. Proposed Solution

Two distinct, but interoperable, APIs to interface the application code with the underlying network stack layers are discussed and implemented. Some updates to existing TDMH modules have also been necessary.

3.1. Callbacks API

The first presented new API relies on a *callback functions* mechanism. After opening a stream, the application can specify two functions, addressed as send and receive callbacks. In order to retrieve the data to be sent from the application, the send callback is executed before the first redundant transmission of each packet. On the contrary, the receive callback is called after a packet is received (or missed) R times, where R is the redundancy level required by the mentioned stream. Indeed, if some data is received, it is delivered to the application only after the R redundant receptions (or misses) have completed. To allocate time for callbacks execution, their computation time ($T_{callback}$) has to be taken into account when computing the duration of each single TDMH slot. $T_{callback}$ is a single fixed value that the user can specify in the network configuration. The described API ex-

ploits the existence in TDMH of the data phase module, which is already in charge of managing transmissions and receptions from the opened streams at the correct time and slot. In this way, callbacks are executed in the same thread of the MAC, which has the highest possible priority. The drawback is that these functions implementation may affect the entire network stack. Moreover, the user has to provide a reasonable estimation of the callbacks execution time.

3.2. Write/Wait API

Although having a very low overhead, the callbacks API makes the network slot size dependent on the callbacks execution time. It is suitable for deeply embedded applications where the callbacks carry out simple operations, but is not flexible enough for more general computation. The second proposed API extends the existing one by adding the *wait* and *wakeup* primitives, which are intended for streams that have to transmit data. The *write* and *read* primitives remain available. The application code can specify a *time advance* value, based on the worst-case execution time, when opening a new stream and should put itself into a waiting state on the transmitting stream. The session layer internally manages the wake-up of the stream and, consequently, of the application, according to the specified time advance with respect to the assigned transmission slot. After waking up, the application can proceed to produce new data and to use the *write* primitive to trigger a transmission. The algorithm relies on an ordered list containing the wake-up time of all the streams that use the *write/wait* API. This data structure is computed at the end of the schedule distribution. At each schedule repetition, the algorithm traverses the mentioned list, waking up streams according to their specific time advance. The list also contains the end time of all the downlink slots in the schedule, since in correspondence of these slots it is needed to check if a newer schedule has been received. If it is the case, the algorithm enters a state in which both the current data structure and the one related to the newer schedule are compared when taking the next stream to be woken up. This is needed because streams may specify a time advance that makes their wake-up time belong to the previous tile with respect to the transmission one. Thus, the two lists may contain elements with overlapped wake-up times. One tile is the maximum allowed wake-up advance, thus it is required to receive schedules at least one tile before their own activation. When the new schedule activation time is reached, the older list is discarded and replaced by the newer one. Through the possibility of specifying a time advance, if correctly sized, the algorithm guarantees to the application to always find the session layer buffer empty when calling the *write* primitive. Therefore, the new packet transmission can take place in the first scheduled stream slot, without further delays. The described algorithm is implemented in a separate thread with respect to the TDMH MAC, due to implementation simplifications, and with lower priority. Moreover, the application may require some computation for generating a new packet, for which no additional and dedicated time is added to the slot duration, as it happens for the callback functions, which are also executed in the highest priority thread. For these reasons, the achieved latency jitter is in general lower when using the first API.
The two APIs behavior is exemplified in Fig. 2.

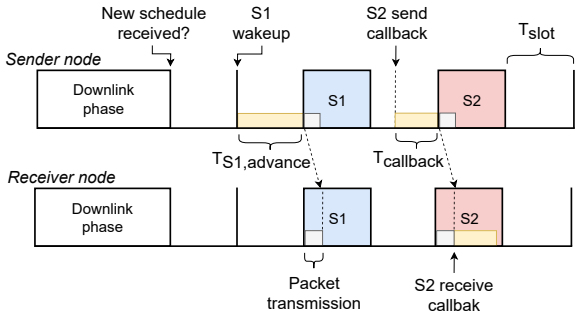

Figure 2: Example of tile composed by 8 slots. The downlink phase occupies 2 of them. Stream *S1* uses the *write/wait* API, while stream *S2* uses the callbacks one, with no redundancy.

3.3. Data Phase

In the current data phase implementation, on the reception side, the received data is delivered to the application after all the redundant packet receptions (or misses) have been completed. When the last redundant packet is missed though, if some data was received during the previous transmissions, it is immediately delivered to the application, leading latency to fluctuate. To overcome this problem, the data

phase has to treat each packet as a maximum size one, whose transmission time is $T_{tx,max}$. The maximum packet size is limited by TDMH itself, so $T_{tx,max}$ is an upper bound of any real packet transmission time. The received data (if any) will always be delivered to the application only $T_{tx,max}$ after the last redundant slot start time. This mechanism slightly increases the end-to-end latency but avoids it to assume different values according to whether the last redundant packet is received or missed.

3.4. Schedule Expansion

At the end of the schedule distribution phase, the *schedule expansion* process takes place. It is needed to move from an implicit schedule form, that is flooded through the network, to an explicit one, in which every schedule element is associated to an action (e.g. transmission or reception). The expansion mechanism was updated to compute the data structures required by the *write/wait* API. Indeed, during the expansion, all the information about streams and their offset inside the schedule (and so their required wake-up time) is known. Since this process can become relatively long when the number of streams increases, it was also modified in order to be possible to split its execution over multiple downlink slots. During the schedule distribution, the master node computes the required number of downlink slots, also accounting for the fact that the *write/wait* API requires new schedules to be received by each node at least one tile before their own activation time.

4. Latency Computation

In order to compute a lower bound for a specific stream latency assume that the application computation for generating a new packet is null and that it takes place exactly in correspondence to the transmission slot start. By knowing the current schedule, the lower bound expression is:

$$L_{low} = (n_{slots} - 1)T_{slot} + T_{tx,max} \quad (1)$$

where T_{slot} is the time slot duration and n_{slots} is the number of slots spanned by the considered stream, including the intermediate ones, which may not be allocated to the same stream. As a consequence n_{slots} is always greater than or equal to the stream redundancy level.

The latency upper bound instead depends on the used API. When using the callbacks API, the latency upper bound can be computed in a scenario in which on the transmission side the application computation takes zero time, while on the reception side it takes the entire specified $T_{callback}$. Indeed, a new packet is produced as soon as the send callback is called, while the received data is delivered to the application as late as possible. In such a scenario the latency upper bound is:

$$L_{up} = L_{low} + 2T_{callback} \quad (2)$$

Now consider the case in which the *write/wait* API is used. Assuming that the *write* operation is called as soon as the application is woken up from its waiting state (null processing time), the latency can be derived as:

$$L_{up} = L_{low} + T_{advance} \quad (3)$$

In a real scenario, on the transmission side, the application may need some time to generate the new data, delaying it towards the transmission slot start. Moreover, a receive callback function may also need less time to be completed than the user-provided execution time. As such, when receiving a packet, it would be delivered to the application before the L_{up} time. Therefore, in both APIs scenarios, L_{up} represents the latency upper bound.
If cryptography is enabled, the time needed to encrypt and decrypt packets is non-null. Also, in a real-world device, the physical radio channel takes some time to startup every time it has to be used. Both the mentioned time amounts have to be considered for the computation of the latency bounds.
It is important to underline that, in any case, the value of the lower and upper bounds does not depend on the specific stream period, but only on the current schedule. Furthermore, any schedule change (e.g. stream scheduled in non-consecutive transmission slots) would affect both bounds by the same amount, through the value of n_{slots}. This is also the only variable factor on which L_{low} (so L_{up} too) depends. Thus, the bounds difference is kept constant across different schedules and so is also the maximum jitter with respect to the latency mean value. Only around new schedules activation a high variation on the latency value (and on its bounds too)

may be experienced, since the value of n_{slots} may change. This phenomenon is limited to a single packet and during all the plateaus between schedules the considerations made on the jitter are valid. Anyway, scheduling is intrinsic to the TDMA nature of TDMH and, thus, inevitable. Moreover, re-schedulings triggered by network topology changes are infrequent and, when new streams are opened, the scheduler tries to avoid overturning the already existing ones. In conclusion, changes in the schedule structure only affect the latency absolute value, not its variability.

5. Experiments

The conducted experiments are divided into two categories. The first ones aim at evaluating streams reliability and latency, by validating the implemented APIs and proving that they do not interfere with the overall network reliability. Successively, the implementation is evaluated in a control loop scenario over a wireless network. TDMH is designed to run on top of the *Miosix OS*[2] and on the *WandStem*[3] nodes, equipped with a single-core microcontroller. These are the devices used during all the experiments.

5.1. Validation Experiments

During validation experiments, all nodes open a stream to the master (node 0), using triple spatial redundancy. The packet payload contains an incremental counter that allows evaluating packets. When creating a new packet, sender nodes also add to its payload the current network time timestamp. As soon as the packet is received by the master, it takes the reception timestamp and logs the elapsed time between the creation and the reception of the packet. Through master's logs, the history of each packet's latency can be reconstructed. All the available WandStem nodes were deployed on the first floor of the Building 21 of Politecnico di Milano, reproducing the node placement seen in the paper presented by Terraneo et al. at the RTSS 2018 conference [4], allowing to have a set of reference results. For both APIs, two experiments were conducted for several hours and under different electromagnetic interference conditions.
During these validation experiments, all the streams achieved a reliability higher than 99.50%, with half of them reaching a stable at 100%. For what concerns latency, streams had a worst-case jitter, with respect to the average latency value, in the order of one hundred microseconds, even in the described setup, in which the master node is highly loaded. The achieved jitter is multiple orders of magnitude lower than the streams period (1 s) and the low standard deviation underlines how the highest spikes are limited to very few samples. As anticipated, the callbacks API is able to reach a lower jitter.
Results about latency and network reliability, which was not worsened by the introduced features, are reported in Tab. 1.

	API	Reliability	Jitter	Std. Dev.
High interf.	Callbacks	99.88%	40.03 μs	1.47 μs
	Write/Wait	99.88%	107.53 μs	4.93 μs
Low interf.	Callbacks	99.94%	32.29 μs	0.82 μs
	Write/Wait	99.97%	125.21 μs	6.40 μs

Table 1: Network reliability with triple spatial redundancy, latency measured jitter and average standard deviation among all the streams, under high and low interference conditions.

5.2. Distributed Control Loop

An experiment in a real-world control loop scenario was also conducted, aiming at proving that a feedback control loop can be implemented through a wireless network, by using TDMH. The goal of this experiment is to control the temperature of a tube furnace for semiconductors synthesis. The network setup consists of a feedback *controller* running on the master node. The furnace electronics outputs temperature measures (*process variable*) on a serial port, which in turn are read by a *sensor* node and forwarded to the controller. The *actuator* is executed on another node that is connected to the furnace input serial port: after receiving the *control variable* (i.e. heating element PWM duty cycle) from the controller, it writes the value to the serial port. The sensor node transmits temperature data using the *write/wait* API and also forwards the sampling timestamp, to enable the actuator to measure the end-to-end latency. Other nodes are connected to the network in order to force data packets to traverse multiple

[2] `https://miosix.org`
[3] `https://miosix.org/wandstem.html`

hops to close the control loop. The complete deployment topology is shown in Fig. 3. The temperature sampling and the control period are set to 1 s, as well as the streams period.

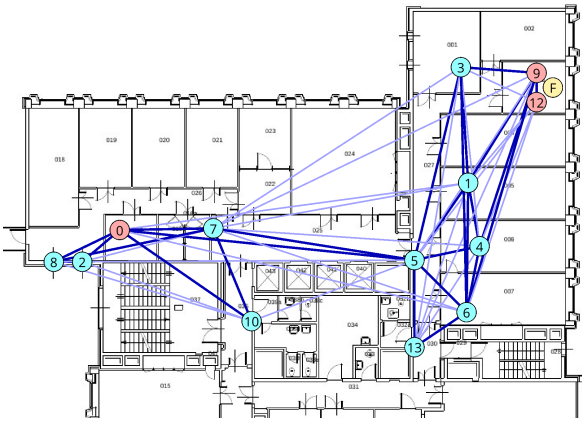

Figure 3: Control loop experiment network setup. In orange the controller (0), sensor (12) and actuator (9) nodes. The furnace (F) is shown in yellow. Dark blue lines are strong links, while light blue indicates the weak ones.

The controller was designed through the *direct synthesis for setpoint tracking* method, in which the design is based on the process model and the desired closed-loop transfer function, respectively represented by $P(z)$ and $F(z)$ in discrete time:

$$P(z) = \frac{\mu T z}{(T_s + \tau)z - \tau} \qquad F(z) = \frac{1-\alpha}{z(z-\alpha)} \tag{4}$$

where $\mu = 1250$, $\tau = 200$, $T_s = 1\ s$ and $\alpha = 0.9985$. Knowing the transfer function, the process model and that $F(z) = \frac{R(z)P(z)}{1+R(z)P(z)}$, the controller expression $R(z)$ can be found. Finally, knowing that $R(z) = \frac{u(k)}{e(k)}$, the expression for the control variable $u(k)$ is derived:

$$\begin{aligned} u(k) = \alpha u(k-1) + (1-\alpha)u(k-2) + \\ (1-\alpha)\frac{T_s+\tau}{\mu T_s}e(k-2) \\ -(1-\alpha)\frac{\tau}{\mu T_s}e(k-3) \end{aligned} \tag{5}$$

where $e(k)$ is the error between the required set point and the process variable: $e(k) = \bar{y} - y(k)$. During the experiment, the controller reached the setpoint of 1000 $°C$ without no further oscillations (Fig. 4) and showed to be tolerant to packets losses caused by the distributed wireless setting.

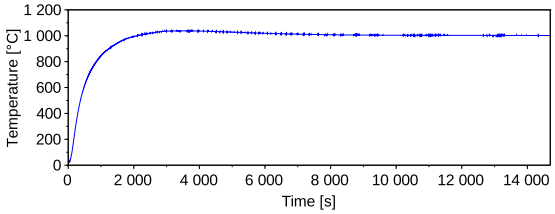

Figure 4: Process variable plot.

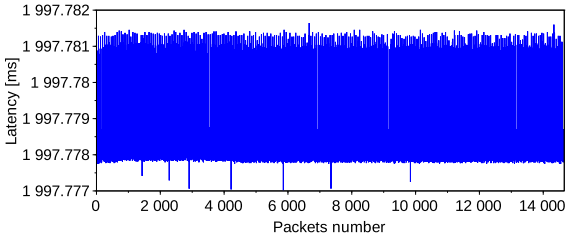

Figure 5: Measured sensor-actuator latency.

The measured latency maximum deviation from the mean value on the sensor-actuator path is 3.34 μs. The measured latency throughout the entire experiment is reported in Fig. 5. It may seem noisy, but the maximum measured peak-to-peak difference is 4.62 μs, a lower value than the one experienced during the previously presented validation experiments.

6. Conclusions

The two presented APIs offer different ways of synchronizing the application with the underlying layers of TDMH, a low-power wireless network stack designed for real-time applications over wireless mesh networks, also providing guarantees on the end-to-end latency bounds. The design was driven by the necessity of bounded latencies, which is a key aspect of real-time applications. The latency bounds provided by TDMH are deterministic and predictable. Moreover, their difference is constant, independently of the schedule structure. Experiments were conducted using the existing WandStem nodes and show that a latency having a very low standard deviation is achieved, without drawbacks to the network reliability provided by TDMH. The distributed temperature control experiment also confirms that the provided latency guarantees and high network reliability make TDMH suitable for real-time applications, such as industrial control, over a wireless network.

7. Acknowledgements

I'd like to thank both my advisor and co-advisor, Federico Terraneo and professor William Fornaciari, for allowing me to work on this great project. I'm especially grateful to Federico Terraneo for assisting me during the entire thesis process, teaching me countless things, a great part of which are not even related to the thesis itself. A different type of gratitude goes to my family, my friends, my classmates and my girlfriend, who supported me and pushed me to always do better. Finally, a special thanks goes to all the members of Skyward Experimental Rocketry, without whom the university experience would not have been the same.

References

[1] IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 1: MAC sublayer. *IEEE Std 802.15.4e-2012 (Amendment to IEEE Std 802.15.4-2011)*, pages 1–225, 2012.

[2] Domenico De Guglielmo, Simone Brienza, and Giuseppe Anastasi. IEEE 802.15.4e: A survey. *Computer Communications*, 88:1–24, 2016.

[3] Federico Terraneo, Federico Amedeo Izzo, Alberto Leva, and William Fornaciari. TDMH: a communication stack for real-time wireless mesh networks, 2020.

[4] Federico Terraneo, Paolo Polidori, Alberto Leva, and William Fornaciari. TDMH-MAC: Real-Time and Multi-hop in the Same Wireless MAC. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 277–287, 2018.

[5] Federico Terraneo, Luigi Rinaldi, Martina Maggio, Alessandro Vittorio Papadopoulos, and Alberto Leva. FLOPSYNC-2: Efficient Monotonic Clock Synchronisation. In *2014 IEEE Real-Time Systems Symposium*, pages 11–20, 2014.

[6] Qi Wang, Katia Jaffrès-Runser, Yongjun Xu, Jean-Luc Scharbarg, Zhulin An, and Christian Fraboul. TDMA versus CSMA/CA for wireless multi-hop communications: A comparison for soft real-time networking. In *2016 IEEE World Conference on Factory Communication Systems (WFCS)*, pages 1–4, 2016.