TTWiFi: Time-Triggered Communication over WiFi

Carl Lusty*

carl.lusty@alumni.griffithuni.edu.au
School of ICT, Griffith University
Nathan, Queensland, Australia

Vladimir Estivill-Castro[†]

vladimir.estivill@upf.edu
DTIC, Universitat Pompeu Fabra
Barcelona, Catalunya, Spain

René Hexel*

r.hexel@griffith.edu.au
School of ICT, Griffith University
Nathan, Queensland, Australia

ABSTRACT

WiFi is prevalent in mobile robotics and the Internet of Things (IoT) despite the fact that many applications in these environments require real-time communication. WiFi does not guarantee timeliness and often introduces unbounded communication delays. We present TTWiFi, a time-triggered protocol for fault-tolerant real-time communication between devices situated in a broadcast domain, that only requires minor changes to IEEE 802.11 parameters and remains standards-compliant in critical aspects. We implemented TTWiFi on a representative platform and benchmarked our implementation with TCP and UDP over unmodified WiFi. Our results demonstrate that unlike the latter, TTWiFi has the ability to maintain communication in a congested environment while exhibiting fail-silent behaviour crucial for fault tolerance and real-time communication.

CCS CONCEPTS

• **Networks** → **Network reliability; Error detection and error correction; Network experimentation; Network measurement.**

KEYWORDS

Performance evaluation and modelling, Wireless network algorithms and protocols, Fault Tolerance, Fail-silence, Time-Triggered Communication, Correctness, survivability and reliability evaluation, Wireless measurement tools and experiences.

ACM Reference Format:

Carl Lusty, Vladimir Estivill-Castro, and René Hexel. 2021. TTWiFi: Time-Triggered Communication over WiFi. In *Proceedings of the 11th ACM International Symposium on Design and Analysis of Intelligent Vehicular Networks and Applications (DIVANet '21)*, November 22–26, 2021, Alicante, Spain. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3479243.3487298>

1 INTRODUCTION

The number of WiFi devices, has soared over the last several decades, surpassing twelve billion devices [19], making, Institute of Electrical and Electronics Engineers (IEEE) 802.11 one of the most widely utilised series of wireless communication standards in the world. WiFi is also common in mobile robotics and Internet of Things (IoT) devices for human environments, for example:

- The Nao [12] and Pepper [15] in education / social robotics.

This work is licensed under a Creative Commons Attribution International 4.0 License.

DIVANet '21, November 22–26, 2021, Alicante, Spain.

© 2021 Copyright is held by the owner/author(s).

ACM ISBN 978-1-4503-9081-1/21/11.

<https://doi.org/10.1145/3479243.3487298>

- The Giraff [3] and VGo [5] in teleoperation.
- The Roomba [16] line of domestic robots
- Many IoT devices such as Nest Thermostats [14].
- Beagle-bone Black and its Programmable Real-time Unit (PRU) by Texas Instruments [33].

These mobile robots for human environments use Wi-Fi because it is ubiquitous; however, many have noticed the serious consequences that communication delays and jitter produce in the corresponding applications. For example, the Robot Operating System (ROS) [4] is now a widely-adopted middleware for robotics systems that relies on TCPROS and the Simple Object Access Protocol (SOAP) (which in turn relies on a Transmission Control Protocol (TCP) connection) to create a modular architecture where processes are linked by client-service and publisher-subscriber patterns (which are abstractions of remote-procedure calls). Such an elaborate networking stack has proven to be prone to dangerous delay and jitter and remedies have been proposed, fundamentally replacing TCP by User Datagram Protocol (UDP) [44]. However, in the context of mobile robotics (as opposed to cloud services, where eventual consistency is usually sufficient), timely messaging is crucial. We argue that the core of the problem relies on networked nodes communicating over a shared medium. Their transmissions may overlap, resulting in collisions, where the payload of the message is ‘destroyed’ [31]. Given this fundamental issue in wireless communication, many coordination algorithms have been proposed [6, 31, 41–43, 47]. In particular, with the growing use of mobile robotics on wireless networks, there has also been work on reliability and Quality of Service (QoS) issues specifically for mobile robots and control systems [34, 35, 45], where timeliness is considered to be a challenging and open problem [26].

We propose the integration of a time-triggered communication protocol and IEEE 802.11 to improve wireless communication timeliness and reliability for mobile robots that operate in human environments. We show that it is feasible to achieve functional wireless time-triggered communication, despite the non-deterministic nature of IEEE 802.11 transmission timing. We describe a functional prototype implementation of our approach and demonstrate its reliability and performance in a series of experiments. The data from these experiments demonstrate that our time-triggered approach, called Time-Triggered WiFi (TTWiFi), is superior to standard wireless communication approaches at providing reliable wireless communication for mobile robots that operate in human environments.

Although various specification amendments have been approved and integrated into the base IEEE 802.11 standard, we can narrow these to IEEE 802.11n for our current study. IEEE 802.11ad and 802.11ay operate at higher frequencies and have short ranges, which can hinder the range of a mobile robot. IEEE 802.11ac operates in the 5GHz range and has more throughput, while IEEE 802.11n

can operate in the 2.4GHz band, resulting in a much better communication range. IEEE 802.11ax has increased throughput and will replace IEEE 802.11n and 802.11ac [24]; however, at this stage, it is not well supported on current robotics platforms. Moreover IEEE 802.11n also has several open-source firmware implementations available [1, 48].

The challenge of timely communication over wireless channels is so common that the literature has numerous examples of wireless protocols aiming to improve reliability on IEEE 802.11 networks for real-time communication; for example, RT-WMP [43], Sched-WiFi [35], Adaptive TDMA [40] and RT-WiFi [51]. While many of these protocols offer substantial reliability benefits, each also presents at least one major limitation. These limitations include issues such as range, fault tolerance, network configuration, non-deterministic transmission timing and an inability to operate within the range of standard IEEE 802.11 networks. Our goal is to demonstrate the possibilities with the most standard access and tools that do not require specialised hardware; our protocol can be implemented as broadly as IEEE 802.11n itself.

This paper is organised as follows. Section 2 discusses the design of our time-triggered communication protocol. Section 3 discusses how we have implemented our protocol on realistic mobile robotics hardware. Section 4 discusses the experimental validation of our approach. Section 5 contains our conclusions and final remarks.

2 METHOD

Within the IEEE 802.11 standards, there are two primary methods of coordinating access to the communication medium. The first method is the Distributed Coordination Function (DCF), a contention-based algorithm, based on Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA). The second, optional access method is the Point Coordination Function (PCF); where a centralised Point Coordinator (PC) node controls and coordinates access to the wireless medium [6, 50]. Further to these basic access methods, there are the QoS extensions Enhanced Distributed Channel Access (EDCA) and Hybrid Coordination Function (HCF) Controlled Channel Access (HCCA) [6, p.192]. These allow for adjustments to be made to transmission priorities or grant exclusive periods during which a transmitting node should, in principle, be uninterrupted by other transmissions. There are also non-standardised access methods that have been developed for WiFi networks (we already mentioned RT-WMP [43], SchedWiFi [35] and RT-WiFi [51]).

The DCF is the fundamental access method of IEEE 802.11, and its implementation is mandatory for all standards-compliant wireless nodes [6, p.1296]. Thus, CSMA/CA is the primary mechanism utilised by all nodes on an IEEE 802.11 LAN to determine when to transmit over the wireless medium. A key element of this is determining if the communication medium is currently idle or if another node is already transmitting. To determine the current state of the communication medium, IEEE 802.11 provides the Carrier Sense (CS) mechanism. The CS mechanism has two distinct methods of determining the state of the medium, the physical and virtual CS functions. The physical CS function listens to the communication medium in an attempt to detect any transmissions currently in progress. The virtual CS function utilises the Network Allocation Vector (NAV), which contains transmission predictions to

determine if another node is likely to be utilising the medium [6, p.1305]. One of the primary sources of NAV predictions are the Request-To-Send (RTS) and Clear-To-Send (CTS) mechanisms. The CS mechanism reports the medium as busy if either the physical or virtual CS functions determines that the medium is busy [6, p.1305].

When utilising the DCF, if the CS mechanism determines that the medium is busy, a node must wait before transmitting. The DCF implements this as a backoff timer, which only decrements when the CS mechanism determines the communication medium is idle [6, p.1325]. Nodes may only transmit when their CS mechanism determines that the communication medium is idle and once their backoff timer reaches zero [6, p.1326].

From a mobile robotics perspective, there are two main problems with utilising the DCF and its CSMA/CA based approach. The first is temporal delays when the communication medium is busy. A transmitting node must wait until the medium is no longer busy before transmitting [6, 25]; this results in potentially unbounded communication delays. The second issue is jitter in transmission timing; the duration of time it takes for a given transmission is dependent on network load, and thus the transmission behaviour of any other nodes utilising the same wireless frequency. Both of these can be highly dynamic, which leads to transmission delays that often fluctuate in duration. This fluctuation can lead to several problems, including inconsistent behaviour. For example, say we had a remotely controlled robot that received commands over the wireless network: random and time-varying delays could deteriorate its performance and result in an unstable system [29]. Another critical issue is ensuring reliability. Random transmission delays are unacceptable as they make timely operations [26] infeasible.

We propose TTWiFi to improve the reliability of wireless communication for mobile robots that operate in human environments. To maintain the requisite communication range and bandwidth that many of these mobile robots require, TTWiFi utilises IEEE 802.11n as a base communication layer and makes modifications to improve communication reliability. TTWiFi utilises time-triggered communication to reason about transmission timing and detect failed transmissions within a bounded time frame. A pre-defined, cyclic Time Division Multiple Access (TDMA) transmission schedule determines access to the wireless communication medium. Because of this, transmission arrival times can be derived from their scheduled transmission time and bounded delays, such as the various IEEE 802.11n Interframe Space (IFS) delays. This allows for the detection of missed transmissions in case they are not received at their expected arrival time. TTWiFi avoids as many single points of failure as possible to assist with fault tolerance; as such, TTWiFi operates using Ad-Hoc wireless communication rather than a centralised Access Point (AP) that could potentially fail. At present, TTWiFi assumes that all wireless nodes are within transmission range of each other; this avoids problems with hidden nodes and cliques.

Mobile robots for human environments often utilise a General Purpose Operating System (GPOS); as such, TTWiFi must also be compatible with this. The problem with utilising a GPOS is the scheduling of transmission processes. In a GPOS, scheduling is primarily based on a best-effort process scheduler that cannot make timing guarantees. As time-triggered communication intrinsically utilises the timing of transmissions, scheduler timing irregularities

are a problem. Within embedded systems, this problem is often resolved with the use of a Real-time Operating System (RTOS) or a dedicated processor for transmissions; however, utilising either of those approaches would severely limit compatibility with existing mobile robots for human environments.

To implement TTWiFi on a GPOS, we take several steps to reduce Operating System (OS) scheduling jitter. The first is to elevate the priority of the TTWiFi transmission process within the priority system of the OS scheduler (if such a facility is available). We also utilise an existing process scheduling library. The advantage of this is that many OS dependant mechanisms can often be utilised to improve schedulability; by utilising an existing library that supports multiple OSs, we retain compatibility with these systems while taking advantage of known scheduler features. An example of this type of library is `libdispatch`, which supports multiple platforms and allows for attempts at scheduling tasks at absolute points in time, which assists in adhering to our TDMA schedule.

While these strategies cannot eliminate the inherent unreliability of utilising a best-effort scheduler, they can minimise scheduling jitter on a GPOS. Consequently, we must factor scheduling jitter into how TTWiFi operates. In the ideal case, a time-triggered communication protocol can synchronise itself using only transmission times to infer the amount of clock drift each node has. This strategy is not applicable to TTWiFi due to the potential scheduling jitter of a GPOS. We therefore transmit a timestamp captured just prior to transmission to estimate clock drift relative to other nodes for clock synchronisation. This approach requires little extra communication bandwidth and minimises the reliance on scheduler accuracy, minimising the impact of scheduling jitter in a best-effort OS.

Communication in TTWiFi is based on a pre-defined, cyclic TDMA communication schedule. Clock synchronisation in TTWiFi utilises a two-phase approach. In the initial start-up phase, standard Network Time Protocol (NTP), Precision Time Protocol (PTP) or similar protocols synchronise the clocks of the ensemble of nodes. This step removes potential, initial extreme clock outlier values, which can occur, for example, if nodes had been fully powered down and lost all knowledge of the current time. Once the start-up phase is complete, TTWiFi is ready to begin operating and takes over clock synchronisation. Due to their potential for interference, NTP and PTP are disabled to avoid interference through uncontrolled wireless transmissions, which could result in wireless collisions.

Runtime synchronisation of TTWiFi measures the clock differences of all nodes running the protocol and calculates a distributed, global time for nodes to schedule their transmissions with, thus compensating for the problem of clock drift within individual nodes. To achieve this, each node measures the differences in system clock values of all other nodes running TTWiFi. The difference between node clocks is calculated using timestamp values that nodes capture just prior to transmitting and include within their transmissions. These timestamps, coupled with deterministic transmission timing and known IFS delays, provide the basis to estimate clock differences between nodes for synchronisation purposes. Each TTWiFi node then periodically recalculates its global clock value from its differences with all other TTWiFi nodes. The calculation of the global clock value is based on the Fault Tolerant Average [32], which reduces the impact of outlier clock values.

TTWiFi utilises broadcast UDP transmissions for all communication. UDP provides a minimalistic high-level interface for issuing transmissions and is well supported in OSs on current mobile robotics platforms for human environments. Unlike TCP or other communication protocols that utilise acknowledgements, UDP does not attempt retransmissions or any other significant data recovery mechanisms that could impact low-level transmission timing.

TTWiFi utilises broadcast transmissions to disseminate information to all nodes simultaneously, including synchronisation timestamps. Given the Ad-Hoc network configuration of TTWiFi, there is also no danger of AP broadcast retransmissions, which can often cause UDP packets to arrive late, multiple times, or out of order. Broadcast transmissions also avoid the timing problems of packet fragmentation at the Media Access Control (MAC) layer as IEEE 802.11 will not fragment a broadcast transmission.

TTWiFi also relies on modifications to the standard MAC behaviours of IEEE 802.11n. IFS timing is minimised to provide TTWiFi with priority access to the wireless medium. We disable the CS mechanism of the DCF for the deterministic timing requirements of TTWiFi. The reason for this is that by default, the IEEE 802.11 DCF mechanism results in an unbounded temporal delay before each transmission. This delay is due to the DCF waiting for the communication medium to be idle before attempting any transmission. As TTWiFi is fundamentally a time-triggered communication protocol that relies on deterministic transmission timing, the DCF must be modified to transmit without these delays. Disabling the CS mechanism, implies operation as if the medium were idle. While this removes some wireless collision protections from the DCF, for TTWiFi nodes, this is mitigated by the TDMA schedule which prevents wireless collisions between well synchronised nodes. Collisions between standard wireless nodes and TTWiFi nodes are still possible; however, they are minimised by the standard wireless collision avoidance mechanisms and these nodes will likely attempt retransmissions in this case. Their standard DCF implementations will wait until the TTWiFi transmissions pause before attempting retransmission. We account for this in the time allocated between TDMA slots.

As TTWiFi utilises an IEEE 802.11 Ad-Hoc network, we must consider the impact of network scanning. Nodes within an IEEE 802.11 Ad-Hoc network will periodically pause transmissions and change to another communication frequency, seeking other nodes that should be a part of the same Ad-Hoc network grouping. In part, this behaviour is for clique avoidance and Ad-Hoc network merging. This would pose a problem for TTWiFi, as during this time, nodes with a single transmitter and receiver cannot send or receive transmissions on the original communication frequency. Therefore, the scanning node would miss receiving some transmissions, and worse, many drivers queue outgoing transmissions until the scanning operation completes, severely impacting vital transmission timing.

Given that TTWiFi already pre-defines transmission timing, we therefore resolve this problem by pre-defining the communication frequency that TTWiFi nodes utilise. We can therefore disable Ad-Hoc network scanning within the MAC layer.

In order to schedule transmissions at specific times, TTWiFi nodes must agree to utilise the same temporal representation. A

Table 1: The processing capabilities of various mobile robotics platforms utilising IEEE 802.11.

Robot	OS	CPU Clock Speed	CPU Cores
Nao (Versions 4 and 5)	Linux [2]	1.6 GHz [9]	1
Nao (Version 6)	Linux [2]	1.91 GHz [39]	4
Pepper (1.8a)	Linux [2]	1.91 GHz [38]	4
Giraff	Windows [3]	2.1 GHz [27, p.31]	2
DARwin-OP	Linux [21]	1.6 GHz [11]	1
Oculus Prime	Linux [20]	3.2 GHz [20]	4
TUlip	Linux with Xenomai RT [28]	1 GHz [28]	1
TurtleBot3	Linux [23]	1.4 GHz [23]	4
ROBOTIS OP3	Linux or Windows [22]	Not listed in specification	2 [22]
Raspberry Pi 3 (Model B)	Linux	1.2 GHz [7]	4
TIAGo	Linux with RT Preempt [37]	Not listed in specification	
TALOS	Linux with RT Preempt [36]	Not listed in specification	

common approach within mobile robotics, and in general computing, is to utilise *Unix Time*, which provides the duration of time since a fixed point; an *Epoch* [46].

Utilising this fixed point in time allows us to use a simple mechanism for agreeing on the timing of slots in the TDMA round. We can have each node calculate the current communication slot by assuming that the TDMA cycle began running at the epoch time. The duration of our TDMA round is constant. Hence, a division of the current time by that period provides the number of rounds since *Epoch*, and a modulus provides the current offset into the TDMA round.

3 PROTOTYPE IMPLEMENTATION AND EXPERIMENT

3.1 Prototype implementation

Table 1, shows a list of common mobile robotics platforms, their Central Processing Unit (CPU) processing speeds, and the number of CPU cores available. In light of the similarities between the Raspberry Pi 3 and the other platforms in Table 1, we will utilise the Raspberry Pi 3 as our hardware platform for implementation. Using the Raspberry Pi 3 has several advantages as an implementation and test platform. The CPU clock speed of this version of the Raspberry Pi is well within the range of representative CPU speeds. The Pi has many exposed General-Purpose Input/Output (GPIO) pins, which can be useful for measuring and debugging purposes. Moreover, there is flexibility to select the Network Interface Card (NIC). We will use a NIC that is compatible with open-source firmware options (in particular, the Alfa Wireless Adapter AWUS036NHA [30]). It is worth noting that the Raspberry Pi 3 has more CPU cores than some other hardware in Table 1. To minimise the impact of this potential advantage over single-core platforms, we will constrain our implementation to a single Operating System (OS) process.

Similarly, the Raspberry Pi is a generic hardware platform, with a non-RTOS. This is also equivalent to the non-RTOS platforms that we often find for mobile robots in human environments. Therefore, and to ensure the most representative version and generic approach applicable to the robots in Table 1, our Raspberry Pi nodes will utilise the official Raspberry Pi distribution, Raspbian GNU/Linux 8 (jessie) [13], with version 4.4.38-v7+ of the Linux kernel. We stress that TTWiFi has no design dependence on a specific OS and uses the portable Portable Operating System Interface (POSIX) API (e.g., we used macOS for development and testing of the TTWiFi C/C++ code).

As discussed, scanning is a problem for reliable time-triggered wireless communication. Having unsynchronised, periodic times during which all communication essentially ceases is detrimental to our goal of temporally reliable wireless communication. Through the network driver, it is already possible to pre-define certain network parameters, such as the name and communication frequency of our Ad-Hoc network. This functionality makes scanning fundamentally unnecessary for TTWiFi to function. Our time-triggered approach already requires us to pre-define our communication schedule, fixing the communication frequency is only a minor additional step. Also, by disabling scanning, we avoid an unnecessary step and utilise more time for transmissions.

In the past, researchers have utilised the flexible and open-sourced MadWifi [1] driver for experimenting with changes to IEEE 802.11; but the MadWifi driver was declared legacy in 2007. ModWifi [48] is a forked project from the open-source Qualcomm Atheros 802.11n driver and firmware for their AR7010 and AR9271 networking chips [8]. ModWifi exposes configurable options for IEEE 802.11n which we can set from a Command Line Interface (CLI) utilising *debugfs* and custom registers. The configurable parameters include changing Interframe Space (IFS) values, timeout durations for Acknowledgement (ACK) and Clear-To-Send (CTS) periods, backoff queue lengths and more. The most useful option that ModWifi provides is the ability to ignore the 'idle' state of the communication medium when Carrier Sense (CS) is performed. Table 2 contains a summary of our ModWifi parameters.

Table 2: The ModWifi parameters utilised in our implementation.

Register	Value	Units	Register Description
time_slottime	0	μ s	aSlotTime
force_channel_idle	1	bool	Ignore channel sense
ifs_cwmin_queue0	0	aSlotTime	Minimum backoff slots to wait
ifs_cwmax_queue0	1	aSlotTime	Maximum backoff slots to wait
time_eifs	1	μ s	Extended Interframe Space (EIFS) time
time_sifs	1	μ s	Short Interframe Space (SIFS) time
ifs_aifs_queue0	0	aSlotTime	Arbitration Interframe Space (AIFS)
inject_noack	1	bool	Do not wait for ACK
time_ack_timeout	1	μ s	ACK timeout
ignore_virt_cs	1	bool	Ignore RTS/CTS
time_cts_timeout	1	μ s	CTS timeout

To schedule transmitting IEEE 802.11n packets in a deterministic and bounded time frame (at the absolute points in time, defined by our time-triggered communication schedule) we adjust OS process scheduling and priorities to ensure our implementation can gain the most precision possible from the non-RTOS scheduler. A popular and well-known method for adjusting process priorities in Linux is altering a processes' *nice* attribute. This impacts only SCHED_OTHER and SCHED_BATCH processes. Linux also offers several real-time scheduling policies, which have better timing precision and more priority control. Utilising a preemptive real-time scheduling algorithm would allow the time-triggered communication process to seize the CPU when necessary to perform timely message transmissions. The preemptive real-time scheduling policy options include SCHED_RR, SCHED_FIFO, and SCHED_DEADLINE which take priority over SCHED_RR and SCHED_FIFO [10]. We selected SCHED_FIFO as the time-triggered communication process can take priority over SCHED_OTHER

Figure 1: Experiment setup.

tasks, and if necessary, the priority level attribute can be manipulated to manage integration with other real-time scheduled processes.

3.2 Experiments

To experimentally test and evaluate the feasibility and expected reliability of TTWiFi, we use the prototype implementation and contrast with the two commonly used communication protocols in mobile robotics for human environments; that is, Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). We test three crucial aspects of TTWiFi against the baseline of TCP and UDP. First, to establish feasibility on non-RTOS systems, we test the accuracy of the Operating System (OS) scheduler configuration and verify that it can reasonably schedule the required Time Division Multiple Access (TDMA) transmission slots. The data provided by this experiment facilitates calculating the maximum number of TDMA slots that the prototype scheduler configuration can reasonably support, which, when combined with packet sizes and transmission bitrates, is indicative of the prototype's maximum communication bandwidth.

The second set of experiments will test the synchronisation mechanism to assess its ability to synchronise the clocks of nodes within these experiments. These synchronisation experiments also demonstrate the impact of not utilising a form of synchronisation on the throughput and overall reliability of TTWiFi.

The last, and most important series of experiments is on wireless interference and its impact on the communication reliability of TTWiFi and the comparison communication protocols. These experiments test three different levels of wireless interference; the first is a baseline test against the ambient background interference at the experiment location. The next interference experiments will be with an artificially increased wireless interference level generated by several external sources. The final interference experiments will be against an extreme amount of artificial wireless interference. The purpose of extreme interference testing is to evaluate the reliability of TTWiFi within even the most hostile of wireless networking environments.

Throughout all experimentation, the physical transmitter layout and location is maintained. Figure 1a illustrates the approximate machine and transmitter positioning, and Figure 1b shows an image of the experiment configuration.

In order to collect timing measurements without making temporally expensive system calls for memory allocation or printing to the console, each experiment machine preallocates a large array of structs during startup, with each struct corresponding to a temporal data collection point. Calculating the number of array elements for data collection is not difficult, given that the experiment iterations have a fixed number of TDMA slots. Preallocating data collection structures prevents the need to request more memory from the system during execution, which would otherwise result in additional delays and potentially jeopardise the validity of the collected data. Once the predefined number of TDMA cycles for each experiment has elapsed, all of the data collected within the struct arrays is printed to the console and saved into a file. This approach avoids executing costly Input/Output (I/O) operations during experiments when data is being collected, thus reducing the probe effect when measuring TTWiFi.

Our experiments have the following configuration parameters in common.

Communication Schedule.

The test machines utilise a predefined, periodic communication schedule. Transmission time slots on the communication medium are equally divided between the number of nodes, at the defined frequency and with the defined packet size. The Maximum Transmission Unit (MTU) value shown here is the size of the 'data' field of each packet and does not include UDP or Internet Protocol (IP) headers. Here is an example of the schedule configuration file:

```
Hz: 100
Max Members: 5
MTU: 65
```

Ad-Hoc Wireless Network.

Each experiment machine joins a specific Ad-Hoc wireless network and utilises a pre-defined communication frequency. Specifying the communication frequency is necessary as TTWiFi requires that IEEE 802.11 Scanning be disabled. The physical transmission distances to each node are static and are well within the range of the transmitters. As such, the experiments avoid the issue of hidden nodes.

Clock Synchronisation.

Before starting each experiment, each machine's clock is synchronised to an NTP server located on the Local Area Network (LAN). Unless otherwise noted, while each experiment is running, a clock offset is periodically calculated to manage clock drift.

Bitrate.

On Linux, the default broadcast transmission bitrate for 802.11n is set within the driver supplied by each hardware manufacturer. In this case, the hardware manufacturer has set the bitrate field to '10', in the driver file 'ath9k/common-init.c'. This value is defined in units of 100 kilobit per second (Kbit/s), within the structure 'ieee80211_rate' in the file 'include/net/cfg80211.h'. Therefore, each node in this experiment defaults to transmitting broadcast packets at approximately 1 megabit per second (Mbit/s), in line with the standard IEEE 802.11n broadcast bitrate.

Table 3: Scheduler experiment timing statistics. (all values are in μs).

(a) 100Hz experiment data					(b) 500Hz experiment data				
100Hz					500Hz				
Node	Min	Max	Mean	Std Dev	Node	Min	Max	Mean	Std Dev
ttw1	32	346	65.68	28.64	ttw1	-163	296	64.28	29.29
ttw2	32	574	66.84	37.26	ttw2	-189	260	44.12	36.26
ttw3	32	298	68.43	37.18	ttw3	-187	363	48.08	19.76
ttw4	31	289	68.72	37.40	ttw4	-175	215	46.58	16.44
ttw5	32	346	77.01	42.83	ttw5	-189	415	51.87	21.82

3.3 Operating System Scheduler Experiments

For these scheduler experiments, each node has a copy of the TDMA schedule and must transmit within their allocated time slots during each TDMA round. For five nodes with a 100Hz TDMA round frequency and no TDMA slot padding, this means scheduling their transmission process execution within $2,000\mu\text{s}$ time slots. For 500Hz TDMA, in the same configuration, each time slot is only $400\mu\text{s}$. Also, each machine runs TTWiFi and records the system clock value at each TDMA slot starting point. The clock synchronisation mechanism has been disabled for this experiment to avoid node synchronisation impacting the results. Thus, the only variable factor within these experiments is the number of TDMA rounds per second. These experiments have a run time of one minute, and a set of scheduler timing data is collected from each node. Figure 2 and Table 3 show the results of comparing the recorded timestamps at the beginning of each TDMA slot, for each node against their expected ideal starting time. The ideal starting time is calculated from the static schedule and the start time of the experiment.

As we can see from the results summaries above, the scheduler error margins stayed relatively consistent between the 100Hz and 500Hz tests. This consistency is evident in the similarity between the mean and standard deviation of scheduler error, shown in Table 3. Both of the TDMA frequencies tested performed reasonably well in the average case, with mean values well within the size of one TDMA slot. The 100Hz worst-case measurement of $574\mu\text{s}$ was well within the bounds of its single $2000\mu\text{s}$ slot time. On the other hand, the 500Hz worst-case measurement of $415\mu\text{s}$ exceeded the size of its single TDMA slot time of $400\mu\text{s}$. This overflow means that some of the TDMA slots for the 500Hz experiment could not be scheduled in time by the OS scheduler.

3.4 Clock Synchronisation Experiments

To demonstrate the importance of clock synchronisation we performed two experiments to show the impact of disabling clock synchronisation. Figure 3 shows the occurrences of packets arriving at one microsecond intervals from the ideal time that each packet should have arrived. The more clustered the transmissions are towards the centre, the more synchronised the system is. The white background area shows the TDMA slot bounds, transmissions occurring outside of their slots are erroneous. In Figure 4, we can see transmission data as box plots; and observe that the timing errors in unsynchronised nodes tend to cluster towards a given direction, logically, this shows the direction of bias in each nodes individual clock. The results of these experiments show that clock synchronisation is necessary for running TTWiFi.

(a) The 100Hz experiment **(b) The 500Hz experiment****Figure 2: Accuracy of the OS scheduler at starting TDMA slots at the correct time.**

3.5 Interference Experiments

To help ensure fair testing conditions between communication protocols all experimentation was conducted in the same physical location, without any movement or change to the transmission antennas. We also conducted the experiments with as minimal delays between executions as practically possible (only a minute or two between experiment runs). For each experiment we restarted the Raspberry Pis to reduce the OS impacts of one run on another; that is, we eliminate effects such as Random Access Memory (RAM) fragmentation by always starting experiments with freshly booted nodes.

We conducted three separate sets of interference experiments, with each set consisting of a different level of artificial external wireless interference. Each set of experiments tests TTWiFi (abbreviated to *ttw* in the figures), a vanilla UDP configuration and a vanilla TCP configuration. The first set of experiments is to measure the baseline performance of each protocol at the experiment location. The level of wireless interference present is simply the background level of interference at the testing location. After the initial baseline experiments, we move on to artificially creating a more interference-heavy wireless environment. This interference is generated by three external nodes, operating standard IEEE 802.11n connections, which generate as much traffic as they can over the same wireless frequency as our experiment. Finally, the last set of experiments utilises ModWifi to generate an extreme amount of wireless interference, showing how each protocol behaves in some

(a) Synchronised

(b) Not Synchronised

Figure 3: Transmission occurrences over time. Each node, in both charts, is shown separately. The white background area shows the TDMA slot boundary.

of the worst-case wireless environments. For all experiments, the TDMA round frequency is 100Hz, the Max Bitrate is 1 Mbit/s, and the data size for packets is 65 bytes.

For our experimentation, there are protocol incompatibilities that required addressing to make our comparisons fair. For the TTWiFi and UDP experiments, each node broadcasts its packets to all other nodes; for this, nodes utilise group addressed packets for transmissions. The recovery mechanisms utilised by TCP, such as ACKs and retransmission, make TCP an inherently unicast transmission protocol. This difference makes evaluating the protocols solely by the number of received packets unfair to TCP. To allow for this limitation in the TCP experiments, we can utilise a round-robin style transmission strategy for TCP. For example, node one transmits its packets to node two and node two transmits to node three, while keeping their individual TCP connections open. When following this approach, the TCP experiments attempt the same number of transmissions as the other protocols and all nodes are still involved in receiving transmissions; there simply are fewer transmissions received overall.

The second protocol incompatibility that requires addressing is how to maintain synchronised clocks between nodes. The mechanism that TTWiFi uses to calculate the global time and synchronise transmissions relies on the time it takes to send a transmission being deterministic. This determinism comes from the firmware modifications to the 802.11 CSMA/CA backoff periods. To keep the UDP and TCP experiments as close to standard as possible, we need

(a) Synchronised

(b) Not Synchronised

Figure 4: Timing error distribution for the clock synchronisation experiment.

Table 4: Baseline results.

Protocol	Good Timing	Bad Timing	Lost Packets	% of Good Packets	Accrued Latency	Mean Duration	Std Dev
TTWiFi	102560	327	17113	85.47% (SD: 1.23%)	174.44s	14.88ms	13.60ms
UDP	34058	76383	9559	28.38% (SD: 24.93%)	859.42s	82.71ms	1569.15ms
TCP	19520	10479	1	65.07% (SD: 17.65%)	1004.80s	186.18ms	3157.78ms

to utilise unmodified firmware for the NIC, which leaves the problem of synchronisation open. Utilising no clock synchronisation mechanism works poorly. NTP is a commonly utilised synchronisation mechanism that could fill this role during the UDP and TCP experiments. The main issue with utilising NTP is that the process running it may choose to perform synchronisation transmissions concurrently with some of the experiment transmissions; however, this is the case for almost any typical use-case of NTP, as users often run it as a daemon. Each node in a UDP or TCP experiment iteration runs an NTP daemon configured for Orphan mode [18]. Orphan mode allows the nodes to synchronise against each other without requiring an external NTP server.

3.5.1 Baseline Interference. Results for this setting where no explicit interference was added appear in Table 4. The benefits of each communication protocol ought to be considered relative to the desired usage scenario. For the TCP experiment; most packets were received; however, the time that they were received varies dramatically. Figure 5c shows that the delivery timing for TCP packets tended to cluster together with remarkably similar delays. These clustered delays could be the result of retransmissions or throttling

by TCP, either way, the TCP packets were delayed. The UDP experiment had similar issues to the TCP experiment when it came to transmitting within the bounds of TDMA slots. When compared, Figures 5b and 5c show that UDP is much closer to transmitting within its slot bounds than TCP. That said, most UDP experiment packets missed their TDMA slot transmission windows. Both the UDP and TCP experiments performed poorly at transmitting within their transmission windows, particularly when compared to the ability of our time-triggered protocol to transmit within TDMA slot bounds. Also, further experiments demonstrated that with our firmware changes (still utilising NTP, however, with our IEEE 802.11n backoff changes active), the UDP transmissions resulted in much better timing accuracy. The remaining difference between the timing results of the UDP and TTTWiFi experiments is due to the chosen synchronisation mechanism, further demonstrating the importance of accurate clock synchronisation among time-triggered nodes.

For this setting, our protocol, TTTWiFi, displayed excellent performance of its synchronisation mechanism. Figure 5 shows that TTTWiFi has a much tighter grouping of transmission occurrences within its TDMA slot boundaries. One of the downsides shown here is the lack of wireless collision prevention. As such, TTTWiFi had the highest number of lost packets in this experiment set; however, the synchronisation improvement resulted in TTTWiFi also having the highest number of packets arriving within the correct TDMA slot. Table 4 lists the exact number of successful packets per protocol, the percentage of packets that arrived at the correct time and the standard deviation of that percentage among nodes in the experiment. The standard deviation value in this table is particularly interesting, as it shows that the performance of TTTWiFi was remarkably consistent among nodes in the experiment. Depending on the application and fault tolerance requirements, many time-triggered communication mechanisms classify late transmissions as fundamentally the same as lost transmissions. Each packet that does not arrive at the correct time represents a loss of communication between the sending and receiving nodes. The duration of this lost period of communication has a minimum time of one TDMA round, and no maximum bound if transmissions continue to be lost. At 100Hz, with five nodes in the schedule, each lost transmission in these experiments represents 10,000 microseconds of missed communication time. Table 4 shows the sum of that lost communication time, and more importantly, it shows the mean duration of periods of lost communication, when multiple transmissions were ‘lost’ sequentially. The mean duration of lost periods of communication is a vital measurement of the responsiveness of these communication protocols. The duration of a period of lost communication can be the difference between a system continuing to operate or entering into a failure mode. Based on the mean durations in Table 4, we can see that our protocol (TTWiFi) responded much more quickly than either UDP or TCP and importantly, the standard deviations show that delays for our protocol were consistent in duration.

3.5.2 Three Additional Flooded Networks In Proximity. In this set of experiments, we explore the impact of increased wireless interference on each protocol. To generate this additional wireless traffic, three ESP8266 [17] wireless microcontrollers each created a wireless network and transmitted continually. These new wireless networks

Table 5

Protocol	Good Timing	Bad Timing	Lost Packets	% of Good Packets	Accrued Latency	Mean Duration	Std Dev
TTWiFi	104033	1034	14933	86.70% (SD: 1.46%)	159.79s	11.82ms	5.19ms
UDP	28671	79115	12214	23.89% (SD: 17.05%)	913.29s	66.48ms	1200.97ms
TCP	15458	14539	3	51.53% (SD: 36.11%)	1045.42s	329.47ms	4112.96ms

operated on the same communication frequency. By introducing additional transmitters, there is more contention for access to the wireless medium and more potential for wireless collisions. The level of TDMA synchronisation for all protocols is similar to the baseline measurements. The percentage of successful transmissions for UDP and TCP are slightly worse than the baseline experimentation (refer to Table 5). This slight degradation of performance is an expected consequence of higher wireless contention.

The mean durations and their standard deviations continue to show TTTWiFi outperforming the other protocols. This superior performance is in part due to the synchronisation issues with standard UDP, caused by medium access delays, as previously discussed. The standout change in this experiment compared to the baseline is that TCP has more protracted periods of lost communication than the other protocols. In Table 5, TTTWiFi and UDP have remained relatively steady since the baseline measurements; however, the mean duration of lost communication time for TCP has almost doubled.

3.5.3 Extreme Wireless Interference. This experiment consists of a single additional machine operating what is essentially a Denial of Service (DoS) attack on the same wireless frequency that the communication protocol is running on. Specifically, the additional machine runs the Constant Jamming attack from the ModWifi project [48]. This attack bypasses the standard 802.11 CS mechanism, backoff mechanism, IFSS, ACKs and queues a large number of packets for transmission [49, p.5].

We made one modification to the Constant Jamming attack, configuring the *time_slottime* variable to 100 μ s, which corresponds to *aSlotTime* in IEEE 802.11. This change is necessary or else none of the protocols can transmit anything. This setting corresponds to a worst-case transmission environment.

Perhaps unsurprisingly, standard UDP and TCP could not manage to transmit anything. Their 802.11 backoff periods prevent them from accessing the medium before the next *jamming* packet is transmitted. The point of this experiment is not to show standard UDP and TCP failing; the point is to validate that TTTWiFi is resilient even to extreme interference.

As expected, Table 6 shows that TTTWiFi has a severely lowered ability to successfully be received. Nevertheless, our protocol TTTWiFi still manages to transmit approximately half the number of successful packets compared to its baseline experiment. Despite a large number of packets lost due to wireless collisions, Figure 6 shows that TTTWiFi is synchronising reasonably well. There are few transmissions received outside the bounds of its TDMA slots. In Table 6, the mean duration of missed transmission periods has almost doubled from the baseline measurement. This doubling correlates to the halving of transmission rates.

Figure 5: Baseline experiment. The white bands indicate the TDMA slot boundary, packets received within this period are acceptable, all other occurrences are nodes encroaching on another nodes transmission slot. Tighter grouping of occurrences, within the white band, demonstrate a well synchronised and functional system.

Figure 6: Our protocol TTwifi behaves resilient under the worst channel denial attack.

Table 6

Protocol	Good Timing	Bad Timing	Lost Packets	% of Good Packets	Accrued Latency	Mean Duration	Std Dev
TTwif	54747	626	64627	45.62% (SD: 0.39%)	652.59s	38.61ms	23.27ms
UDP	0	0	120000	0.00% (SD: 0.00%)	1200.00s	60000.00ms	0.00ms
TCP	0	0	30000	0.00% (SD: 0.00%)	1200.00s	60000.00ms	0.00ms

4 ANALYSIS AND DISCUSSION

The most important information shown by our baseline experiment is the ability of our time-triggered approach to outperform both UDP and TCP with regard to timely transmissions. In the case of UDP, we can attribute our superior performance in timely delivery to two main factors. The first are our MAC settings for TTwifi, which ensured short, deterministic transmission times. The second is our synchronisation mechanism, enabled by our MAC changes. We verified the impact of our MAC changes to the UDP results by re-running the UDP experiment with our MAC changes in place. This change demonstrated a marked improvement in UDP transmission timing. The TCP results are more complex, as the TCP

experiment resulted in fewer lost communications than our time-triggered approach. Unsurprisingly, this result was mainly due to TCP retransmitting lost packets, impacting on the timing of reception. The TCP transmission timing results showed noticeable groupings of transmission times, which generally diminished in size the further they were from the original transmission deadline.

Our second set of interference experiments introduced three additional busy networks into the environment. The impact of the additional interference to our time-triggered approach was minimal; both the number of packets transmitted within TDMA slot bounds and the accrued latency of missed transmission time did not notably vary from our baseline measurements. The UDP experiment showed a loss of around 5% in the number of packets transmitted within TDMA slot bounds. The most notable difference in results for this experiment came from the TCP data. While TCP continued to deliver all of its data, its temporal delays grew significantly worse. The TCP experiment showed a loss of approximately 13.5% in well-timed packets, compared to its baseline result. However, the mean duration periods where communication was lost due to sequential poorly timed or lost transmissions nearly doubled from a baseline value of 186.18ms to 329.47ms. Compared to our time-triggered protocol's missed communication period mean of 11.82ms, the TCP results are over twenty-seven times worse.

Our last experiment set consisted of testing each protocol against an extreme level of wireless interference. With the modified wireless jammer machine running, both UDP and TCP could not communicate at all. This result is not surprising, given how the wireless jammer operates; in essence, it bypasses the standard MAC delays to transmit over the medium rapidly, keeping the communication medium busy. Our approach is tolerant of this type of wireless jamming because it also utilises shorter than normal MAC delays. The important finding from this experiment is that even with extreme levels of wireless interference, the mean period of lost communication for TTwifi nodes was only 38.61ms, which is even less than TCP and UDP during the baseline experiment that did not include additional interference.

5 FINAL REMARKS

The approach described here gives developers an alternative communication strategy viable on typical, current mobile robotics platforms that are deployed where IEEE 802.11n is present. We have demonstrated that our communication protocol, TTWiFi, facilitates real-time communication between mobile robots and IoT devices while maintaining the property of fail-silence that is crucial to fault tolerance in real-time applications. We achieve this with only minor modifications to 802.11n, largely maintaining compatibility and minimising interference, allowing co-existence with devices communicating over standard WiFi. The results of our experimentation demonstrate that our approach, TTWiFi, is robust to wireless interference in even the most hostile of wireless networking environments, and significantly outperforms standard protocols.

REFERENCES

- [1] 2007. madwifi-project.org - Trac. Retrieved 2020-10-31 from <http://madwifi-project.org>
- [2] 2012. Nao Documentation. Retrieved 2020-04-26 from <http://www.aldebaran-robotics.com/documentation/index.html>
- [3] 2014. For Researchers | Giraff. Retrieved 2020-03-16 from <http://www.giraff.org/for-researchers/?lang=en>
- [4] 2014. ROS.org | Powering the World's Robots. Retrieved 2020-02-8 from <http://www.ros.org>
- [5] 2014. VGo Robotic Telepresence for Healthcare, Education and Business. Retrieved 2020-03-16 from <http://www.vgocom.com>
- [6] 2016. IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)* (2016), 1–3534.
- [7] 2017. BCM2837 - Raspberry Pi Documentation. Retrieved 2020-11-1 from <https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2837/README.md>
- [8] 2017. The firmware for QCA AR7010/AR9271 802.11n USB NICs. Retrieved 2020-12-05 from <https://github.com/qca/open-ath9k-htc-firmware>
- [9] 2017. Motherboard - Aldebaran 2.1.4.13 documentation. Retrieved 2020-11-1 from http://doc.aldebaran.com/2-1/family/robots/motherboard_robot.html
- [10] 2017. sched(7) - Linux manual page. Retrieved 2020-11-6 from <http://man7.org/linux/man-pages/man7/sched.7.html>
- [11] 2018. DARwin-OP. Retrieved 2020-4-9 from <http://support.robotis.com/en/product/darwin-op.htm>
- [12] 2018. Discover Nao, the little humanoid robot from SoftBank Robotics | SoftBank Robotics. Retrieved 2018-09-6 from <https://www.softbankrobotics.com/emea/en/robots/nao>
- [13] 2018. FrontPage - Raspbian. Retrieved 2020-4-9 from <https://www.raspbian.org>
- [14] 2018. Nest Thermostats | Keep You Comfortable and Help Save Energy | Nest. Retrieved 2018-09-6 from <https://nest.com/thermostats/>
- [15] 2018. Pepper, the humanoid robot from SoftBank Robotics, a genuine companion | SoftBank Robotics. Retrieved 2020-09-6 from <https://www.softbankrobotics.com/emea/en/robots/pepper>
- [16] 2018. Roomba Robot Vacuum | iRobot. Retrieved 2018-09-6 from <https://www.irobot.com.au/Home-Robots/Vacuum-Cleaning>
- [17] 2019. ESP8266 Overview | Espressif Systems. Retrieved 2019-07-31 from <https://www.espressif.com/en/products/hardware/esp8266ex/overview>
- [18] 2019. OrphanMode < Support < NTP. Retrieved 2019-09-3 from <http://support.ntp.org/bin/view/Support/OrphanMode>
- [19] 2019. Wi-Fi® device shipments to surpass 15 billion by end of 2016 | Wi-Fi Alliance. Retrieved 2019-09-24 from <https://www.wi-fi.org/news-events/newsroom/wi-fi-device-shipments-to-surpass-15-billion-by-end-of-2016>
- [20] 2020. Oculus Prime SLAM Navigator Mobile Robot Platform. Retrieved 2020-02-25 from <http://www.xaxxon.com/oculusprime/slamnavigator>
- [21] 2020. ROBOTIS OP. Retrieved 2020-8-18 from <https://emanual.robotis.com/docs/en/platform/op/development>
- [22] 2020. Robotis OP3. Retrieved 2020-01-24 from <http://emanual.robotis.com/docs/en/platform/op3/introduction>
- [23] 2020. TurtleBot3. Retrieved 2020-12-14 from <https://emanual.robotis.com/docs/en/platform/turtlebot3/features>
- [24] B. Bellalta. 2016. IEEE 802.11ax: High-efficiency WLANS. *IEEE Wireless Communications* 23, 1 (2016), 38–46. <https://doi.org/10.1109/MWC.2016.7422404>
- [25] C. Chaudet, D. Dhoutaut, and I. G. Lassous. 2005. Performance issues with IEEE 802.11 in ad hoc networking. *IEEE Communications Magazine* 43, 7 (2005), 110–116.
- [26] Marisol García-Valls, António Casimiro, and Hans P. Reiser. 2017. A few open problems and solutions for software technologies for dependable distributed systems. *Journal of Systems Architecture* 73 (2017), 1 – 5. <https://doi.org/10.1016/j.sysarc.2017.01.007> Special Issue on Reliable Software Technologies for Dependable Distributed Systems.
- [27] GiraffPlus. 2012. Combining social interaction and long term monitoring for promoting independent living. (2012).
- [28] D. Hobbelen, T. de Boer, and M. Wisse. 2008. System overview of bipedal robots Flame and Tulip: Tailor-made for Limit Cycle Walking. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2486–2491.
- [29] Peter F. Hokayem and Mark W. Spong. 2006. Bilateral teleoperation: An historical survey. *Automatica* 42, 12 (2006), 2035 – 2057. <https://doi.org/10.1016/j.automatica.2006.06.027>
- [30] Alfa Network Inc. 2009. AWUS036NHA 802.11b/g/n Long-Range USB Adapter. (2009).
- [31] L. Kleinrock and F. Tobagi. 1975. Packet Switching in Radio Channels: Part I - Carrier Sense Multiple-Access Modes and Their Throughput-Delay Characteristics. *IEEE Transactions on Communications* 23, 12 (1975), 1400–1416.
- [32] Hermann Kopetz and Wilhelm Ochsenreiter. 1987. Clock Synchronization in Distributed Real-Time Systems. *IEEE Trans. Comput.* C-36, 8 (Aug 1987), 933–940. <https://doi.org/10.1109/TC.1987.5009516>
- [33] D. Molloy. 2014. *Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux*. Wiley.
- [34] Pangun Park, Sinem Coleri Ergen, Carlo Fischione, Chenyang Lu, and Karl Henrik Johansson. 2018. Wireless Network Design for Control Systems: A Survey. *IEEE Communications Surveys Tutorials* 20, 2 (2018), 978–1013. <https://doi.org/10.1109/COMST.2017.2780114>
- [35] G. Patti, G. Alderisi, and L. Lo Bello. 2015. SchedWiFi: An innovative approach to support scheduled traffic in ad-hoc industrial IEEE 802.11 networks. In *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*. 1–9. <https://doi.org/10.1109/ETFA.2015.7301460>
- [36] PAL Robotics. 2020. TALOS Technical Specifications. (2020).
- [37] PAL Robotics. 2020. TIAGo Technical Specifications. (2020).
- [38] SoftBank Robotics. 2017. Pepper Datasheet 1.8a. (2017).
- [39] SoftBank Robotics. 2018. Nao6 PRELIMINARY MARKETING DATASHEET. (2018).
- [40] Frederico Santos. 2004. An Adaptive TDMA Protocol for Soft Real-Time Wireless Communication Among Mobile Computing Agents. In *Proceedings of the Workshop on Architectures for Cooperative Embedded Real-Time Systems (satellite of RTSS 2004)*. 5–8.
- [41] J. L. Sobrinho and A. S. Krishnakumar. 1999. Quality-of-service in ad hoc carrier sense multiple access wireless networks. *IEEE Journal on Selected Areas in Communications* 17, 8 (Aug 1999), 1353–1368. <https://doi.org/10.1109/49.779919>
- [42] J. Song, S. Han, A. Mok, D. Chen, M. Lucas, M. Nixon, and W. Pratt. 2008. WirelessHART: Applying Wireless Technology in Real-Time Industrial Process Control. In *2008 IEEE Real-Time and Embedded Technology and Applications Symposium*. 377–386.
- [43] Danilo Tardioli. 2010. *Real time communications in wireless ad-hoc networks. The RT-WMP protocol*. Ph.D. Dissertation. Universidad de Zaragoza.
- [44] Danilo Tardioli, Ramviyas Parasuraman, and Petter Ögren. 2019. Pound: A multi-master ROS node for reducing delay and jitter in wireless multi-robot networks. *Robotics and Autonomous Systems* 111 (2019), 73 – 87. <https://doi.org/10.1016/j.robot.2018.10.009>
- [45] D. Tardioli and J. L. Villarreal. 2007. Real Time Communications over 802.11: RT-WMP. In *2007 IEEE International Conference on Mobile Adhoc and Sensor Systems*. 1–11. <https://doi.org/10.1109/MOBHOC.2007.4428607>
- [46] Ken Thompson and Dennis M Ritchie. [n.d.]. *unix Programmer's Manual*.
- [47] F. Tobagi and L. Kleinrock. 1975. Packet Switching in Radio Channels: Part II - The Hidden Terminal Problem in Carrier Sense Multiple-Access and the Busy-Tone Solution. *IEEE Transactions on Communications* 23, 12 (1975), 1417–1433.
- [48] M. Vanhoef. 2014. modwifi. Retrieved 2020-12-05 from <https://github.com/vanhoefm/modwifi>
- [49] Mathy Vanhoef and Frank Piessens. 2014. Advanced Wi-Fi attacks using commodity hardware. In *Proceedings of the 30th Annual Computer Security Applications Conference*. ACM, 256–265.
- [50] J. Villalon, P. Cuenca, and L. Orozco-Barbosa. 2005. Limitations and capabilities of QoS support in IEEE 802.11 WLANs. In *PACRIM. 2005 IEEE Pacific Rim Conference on Communications, Computers and signal Processing*, 2005. 633–636.
- [51] Y. Wei, Q. Leng, S. Han, A. K. Mok, W. Zhang, and M. Tomizuka. 2013. RT-WiFi: Real-Time High-Speed Communication Protocol for Wireless Cyber-Physical Control Applications. In *2013 IEEE 34th Real-Time Systems Symposium*. 140–149.